Incremental Fingerprinting in an Open World

Loes Kruger
Radboud University
Nijmegen, The Netherlands
loes.kruger@ru.nl

Paul Kobialka
University of Oslo
Oslo, Norway
paulkob@ifi.uio.no

Andrea Pferscher
University of Oslo
Oslo, Norway
andreapf@ifi.uio.no

Einar Broch Johnsen
University of Oslo
Oslo, Norway
einarj@ifi.uio.no

Sebastian Junges
Radboud University
Nijmegen, The Netherlands
sebastian.junges@ru.nl

Jurriaan Rot
Radboud University
Nijmegen, The Netherlands
jurriaan.rot@ru.nl

***Abstract*—Network protocol fingerprinting is used to identify a protocol implementation by analyzing its input-output behavior. Traditionally, fingerprinting operates under a closed-world assumption, where models of all implementations are assumed to be available. However, this assumption is unrealistic in practice. When this assumption does not hold, fingerprinting results in numerous misclassifications without indicating that a model for an implementation is missing. Therefore, we introduce an open-world variant of the fingerprinting problem, where not all models are known in advance. We propose an incremental fingerprinting approach to solve the problem by combining active automata learning with closed-world fingerprinting. Our approach quickly determines whether the implementation under consideration matches an available model using fingerprinting and conformance checking. If no match is found, it learns a new model by exploiting the structure of available models. We prove the correctness of our approach and improvements in asymptotic complexity compared to naive baselines. Moreover, experimental results on a variety of protocols demonstrate a significant reduction in misclassifications and interactions with these black-boxes.**

***Index Terms*—Fingerprint recognition, active learning, conformance testing.**

I. Introduction

Fingerprinting is the problem of identifying a system. In this paper, we study how to identify a (network) protocol that is running on a black-box system. That is, we want to know the version of the protocol that is used on a device. We identify the protocol by the input-output behavior observed while interacting with the device, thus we consider *behavioral fingerprinting*. Concrete instances of the behavioral fingerprinting problem are to identify the Bluetooth chip used in car keys or the SSH version used in a doorbell camera [1], [2]. Determining the protocol version allows us to conclude whether the system is subject to (known) security vulnerabilities.

The need to fingerprint. Bluetooth communication is ubiquitous in modern internet-of-things (IoT) systems. In 2024 alone, 4.9 billion Bluetooth devices were shipped.[1] Given its security-critical role, for example, in the locking system of cars [2], the Bluetooth protocol demands rigorous implementation. This urgency was reinforced by recent vulnerabilities that allowed attackers to forcibly pair earbuds with Google Fast Pair without user consent [3], [4] or enabled unpaired attackers in proximity to hijack headphones that internally used an Airoha chip [5]. In the latter, many headphone and earbud vendors build on top of these vulnerable chips and many of them were unaware that these chips were used in their devices. Fingerprinting methods allow for determining whether a given Bluetooth device is susceptible to such attacks.

Fingerprinting Finite State Machines. In our work, we consider protocols that can be represented by Finite State Machines (FSMs). FSMs can represent various security-relevant network protocols, including TLS [6], [7], SSH [8] and Bluetooth Low Energy (BLE) [9]. The standard assumption for behavioral finterprinting is to assume access to all protocol versions [6], [8], [10], [7]. This assumption is called a *closed-world assumption* and is overly optimistic: There is no exhaustive list of BLE chips or SSH implementations. In this paper, we propose studying fingerprinting without a closed-world assumption, that is, *we study fingerprinting in an open world.* Section II demonstrates that wrongly assuming a closed world leads to a significant number of misclassifications.

Fingerprinting in a closed world. In a closed world, we are given a *reference set* of (known) FSM *models* and a set of black-box devices (from here onwards: *implementations*). This is a variation on behavioral fingerprinting of a single device and is also known as *group matching* [11]. Behavioral fingerprinting techniques similar to [12] use the reference set to compute a *fingerprint*, a small set of so-called *separating sequences* that together identify any model in the reference set. Fingerprinting an implementation in a closed world is easily done by executing the fingerprint on them. The result will match any black-box implementation with exactly the matching reference model.

Fingerprinting in an open world. In an open-world setting, instead, the reference set cannot be assumed to be complete. As a result, closed-world fingerprinting techniques may not match with any model (*no match*) or with the wrong model (*a misclassification*). In Section II, we show that they tend to misclassify an implementation instead of indicating that there is

This research is partially supported by the NWO grant No. VI.Vidi.223.096.
[1] https://www.bluetooth.com/2025-market-update/

no matching reference. Misclassification due to missing models can be avoided by learning a model of each implementation from scratch using, e.g., Active Automata Learning (AAL), see below. However, we demonstrate in Section II that this requires an excessive amount of interactions with the black-box protocols. This raises a key challenge:

> How can we limit the number of needed interactions, while keeping the number of misclassifications to a minimum?

Active Automata Learning. To address the above challenge, our approach tightly integrates fingerprinting and AAL to deal with the open-world setting. AAL is a well-established technique for constructing models of black-box systems. In fact, if the reference models are not known in advance during closed-world fingerprinting, they are often learned using AAL as a preprocessing step [12], [13], [10], [7]. These AAL algorithms learn a behavioral model of the so-called System Under Learning (SUL) through interactions [14], [15]. In this work, we consider AAL algorithms designed for deterministic, data-less FSMs equipped with a reliable reset mechanism. The state of the art in AAL for such FSMs includes algorithms such as L^* [16], TTT [17] and $L^{\#}$ [18]. They construct a hypothesis by targeted interactions with the SUL, where a hypothesis is an FSM. Then they extensively test the hypothesis using conformance checking against the SUL. The conformance check is the main bottleneck as it typically requires millions of interactions for real-world systems. Expert knowledge is often used to terminate the conformance check and the resulting model is assumed correct.

Our approach. We present INFERNAL, a novel and incremental fingerprinting algorithm to accurately and efficiently identify black-box devices under an open-world assumption. INFERNAL matches implementations with any previously learned reference model *and then tests whether this match is not a misclassification*. Only if the implementation cannot be matched, we learn its model. Instead of learning a new model from scratch, we use adaptive AAL, which ensures amortized costs by considering similar reference models [19], [20]. After learning a new model, we add it to the reference set. Compared to closed-world fingerprinting, incremental fingerprinting is proven to correctly classify all implementations and to only learn a model of an implementation when it is distinct from all previous models given an adequate conformance testing oracle. We illustrate its effectiveness on a motivating example and in the empirical evaluation using various common network protocols.

Contributions. In summary, this paper presents:

1) Formalization of the open-world fingerprinting problem for FSMs and showing the necessity to solve it.
2) The incremental fingerprinting algorithm, called INFERNAL, that solves open-world fingerprinting problems.
3) Proofs of the correctness of INFERNAL (Thm. 1) and improved query complexity w.r.t. baselines (Thm. 2).
4) The efficiency and robustness of INFERNAL in exhaustive experiments demonstrated on several network protocols.

Outline. Section II motivates incremental fingerprinting in an open world by example. The incremental fingerprinting problem is formally introduced in Section III and the state of the art is presented in Section IV. Section V details our novel INFERNAL algorithm, which is then evaluated in Section VI. Sections VII, VIII, and IX discuss the approach, situate it with respect to related work, and conclude with final insights and future work.

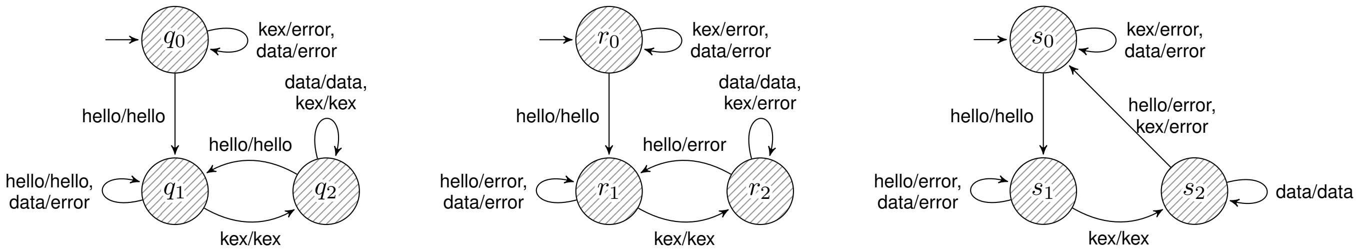
II. OVERVIEW

In this section, we motivate the necessity to account for an open world in fingerprinting, even when most models are already known. Additionally, we show that simply learning every model from scratch is prohibitively expensive. We formalize our models as Finite State Machines (FSMs). We introduce FSMs and separating sequences by example and refer to Section III-A for formal definitions.

Example 1 (Finite State Machines)**.** *Fig. 1 depicts three FSMs* $\mathcal{M}_0$, $\mathcal{M}_1$ *and* $\mathcal{M}_2$ *representing simplified TLS protocols. The models use input alphabet* $I = \{\text{hello}, \text{kex}, \text{data}\}$ *representing a hello message, key exchange and data sending, respectively. These inputs can generate outputs from the alphabet* $O = \{\text{hello}, \text{kex}, \text{data}, \text{error}\}$*. The FSMs behave differently with respect to handling* hello *messages at unexpected times. For example, FSMs* $\mathcal{M}_0$ *and* $\mathcal{M}_1$ *are distinct as* $\mathcal{M}_0$ *responds to* $\text{hello} \cdot \text{hello}$ *with* $\text{hello} \cdot \text{hello}$*, while* $\mathcal{M}_1$ *responds with* $\text{hello} \cdot \text{error}$*. The input sequence* $\text{hello} \cdot \text{hello}$ *is therefore called a* separating *sequence for* $\mathcal{M}_0$ *and* $\mathcal{M}_1$*.*

A common approach to fingerprinting with a set of reference models, represented as FSMs, is to compute a *fingerprint*, i.e., a set of separating sequences that together uniquely identify any of the original FSMs (e.g., [7]). When such a fingerprint for a set of reference models is executed on an implementation that is *not* already represented in the set, two outcomes are possible. Ideally, the considered implementation disagrees with each reference model on at least one separating sequence in the fingerprint, thereby indicating that the implementation is new. For example, $\{hello \cdot kex \cdot hello \cdot hello\}$ is a fingerprint for the set of models $\{\mathcal{M}_0, \mathcal{M}_1\}$ (Fig. 1). If we regard $\mathcal{M}_2$ as an implementation and execute this fingerprint on it, we find that $\mathcal{M}_2$ is a new implementation. However, for a different fingerprint, the considered implementation may match a reference model on all fingerprint sequences, even though it represents a distinct system. For instance, if we take instead the fingerprint $\{hello \cdot hello\}$ for $\{\mathcal{M}_1, \mathcal{M}_2\}$, then $\mathcal{M}_2$ is incorrectly matched with $\mathcal{M}_1$.

Closed-world fingerprinting in an open world. We conduct a motivational experiment to assess how often closed-world fingerprinting methods detect that an implementation behaves differently from all the references. We use 596 implementations of the TLS protocol with 22 underlying FSMs from [13, Section 6.4] and assume that 21 of the 22 FSMs that underlie the 596 implementations are known. Then, we evaluate the performance of closed-world fingerprinting using separating sequences (see Section IV for details) based on the number of misclassifications and implementations that could not be matched to a reference. We average over 10 runs, each with



(a) $\mathcal{M}_0$ (b) $\mathcal{M}_1$ (c) $\mathcal{M}_2$

Fig. 1: Finite state machines representing simplified TLS protocols.

one random model removed and shuffling the references before constructing the fingerprint. We observe that 4.5% of the implementations are misclassified and all implementations can be matched to some model. The number of misclassifications and implementations that cannot be identified grows with the number of missing models. When presented with 11 out of 22 models, 45.9% of the matches are misclassified and 2.2% cannot be matched. These misclassification rates suggest that closed-world fingerprinting methods lack robustness when the references are incomplete. In such cases, they are more likely to misclassify a model rather than recognize the absence of a match.

Learning models from scratch. As an alternative approach, the state-of-the-art AAL algorithm $L^{\#}$ [18] can be used to learn a model of each implementation and evaluate the correctness of the learned models. We find that $L^{\#}$ requires roughly 2.6 million interactions to learn a model of each black-box protocol implementation. However, this results in an incorrect model for 75.9% of the implementations. The high number of incorrectly learned implementations originates from the conformance check being inadequate. Even with a more exhaustive conformance check, where learning the model set requires a total of 14.3 million iterations, this still leads to 32.6% of the implementations being incorrectly learned.

INFERNAL. Algorithmically, we suggest two key adaptations over learning a model of each implementation from scratch. First, to avoid relearning previous models, we test whether the implementations match a model that we have previously seen. Second, to enable the reuse of previous models instead of learning from scratch, we consider adaptive AAL [19]. Adaptive learning reuses existing reference models in order to reduce the number of required interactions. These adaptions lead to a two-step approach: First, identify a likely match with the previously seen models. If such a match is found, check conformance between the black-box and this hypothesis model. If no match is found or the match fails the conformance check, learn the model, guided by previously learned models. We exemplify this approach below.

Example 2 (Incremental Fingerprinting (INFERNAL))**.** *We continue Example 1 and consider fingerprinting four black-box implementations, $\mathcal{I}_0$, $\mathcal{I}_1$, $\mathcal{I}_2$ and $\mathcal{I}_3$ such that*

$$\mathcal{I}_0 \sim \mathcal{M}_0, \qquad \mathcal{I}_1 \sim \mathcal{M}_1 \sim \mathcal{I}_3, \qquad \mathcal{I}_2 \sim \mathcal{M}_2$$

where $\sim$ denotes behavioral equivalence (see Section III-A). Initially, the set of reference model $\mathbb{M}$ is empty, therefore, we learn model $\mathcal{M}_0$ representing $\mathcal{I}_0$ and add it to $\mathbb{M}$.

Subsequently, we consider $\mathcal{I}_1$. We forego fingerprinting, as there is only one model in $\mathbb{M}$ and use a conformance check to determine whether $\mathcal{M}_0 \sim \mathcal{I}_1$. When testing hello · hello*, the outputs produced by $\mathcal{M}_0$ and $\mathcal{M}_1$ show that these implementations are distinct. This indicates the need to learn a model of $\mathcal{I}_1$, leading to the inclusion of $\mathcal{M}_1$ in $\mathbb{M}$.*

Next, we consider $\mathcal{I}_2$ and derive $\mathcal{M}_0 \nsim \mathcal{I}_2$ using fingerprint $\{\text{hello} \cdot \text{hello}\}$*. Observing that both $\mathcal{M}_1$ and $\mathcal{I}_2$ generate the output* hello · error*, we initially hypothesize that $\mathcal{M}_1 \sim \mathcal{I}_2$. However, a conformance check containing sequence* hello · kex · hello · hello *refutes this hypothesis. The learning process is triggered and learned model $\mathcal{M}_2$ is added to $\mathbb{M}$.*

Finally, when considering $\mathcal{I}_3$, we apply the fingerprint $\{\text{hello} \cdot \text{kex} \cdot \text{hello} \cdot \text{data}\}$*, which leads to hypothesis $\mathcal{I}_3 \sim \mathcal{M}_1$. Because this hypothesis is correct, we terminate after a passed conformance check and conclude $\mathcal{I}_3 \sim \mathcal{M}_1$.*

Misclassifications are absent in this example but tend to arise in larger models when conformance checks, either after fingerprinting or during learning, fail to cover the complete behavior. When using incremental fingerprinting in the motivational experiments, no misclassifications are produced by the algorithm when starting with 21 reference models. When starting with 11 reference models, incremental fingerprinting beats learning from scratch both in the misclassification rate (0.7% instead of 32.9%) and in the number of interactions (3.5 million instead of 14.2 million).

III. PROBLEM STATEMENT

In this section, we introduce the preliminaries required to formalize both the fingerprinting problem in a closed world as described in the literature, and the new open world variation.

A. Preliminaries

Throughout this paper, we fix a finite set I of inputs and a finite set O of outputs.

Definition 1. *A* Finite State Machine *is a tuple $\mathcal{M} = (Q, q_0, \delta, \lambda)$ with finite set Q of* states, initial state *$q_0 \in Q$,* transition function *$\delta \colon Q \times I \to Q$ and* output function *$\lambda \colon Q \times I \to O$.*

The transition and output functions are extended to input sequences of length $n \in \mathbb{N}$ as functions $\delta\colon Q \times I^n \to Q$ and $\lambda\colon Q \times I^n \to O^n$. We use superscript $\mathcal{M}$ to refer to elements of an FSM, e.g., $Q^{\mathcal{M}}$ and $\delta^{\mathcal{M}}$. We denote the concatenation of inputs and outputs for two sequences $v, v' \in I^*$, respectively O^*, as $v \cdot v'$. We write $\lambda(w)$ instead of $\lambda(q_0, w)$ for some $w \in I^*$. We denote the size of an object S, such as a set or list, by $|S|$. Given an FSM $\mathcal{M}$, $|\mathcal{M}|$ refers to $|Q^{\mathcal{M}}|$.

Definition 2. *Given a language $L \subseteq I^*$ and FSMs $\mathcal{M}_0$ and $\mathcal{M}_1$, two states $p \in Q^{\mathcal{M}_0}$ and $q \in Q^{\mathcal{M}_1}$ are* L-equivalent, *written as $p \sim_L q$, if $\lambda^{\mathcal{M}_0}(p, w) = \lambda^{\mathcal{M}_1}(q, w)$ for all $w \in L$.*

FSMs $\mathcal{M}_0$ and $\mathcal{M}_1$ are L-equivalent, written $\mathcal{M}_0 \sim_L \mathcal{M}_1$, if $q_0^{\mathcal{M}_0}$ and $q_0^{\mathcal{M}_1}$ are L-equivalent. The states p and q are *equivalent*, written $p \sim q$, if they are I^*-equivalent. Analogously, $\mathcal{M}_0$ and $\mathcal{M}_1$ are equivalent if $q_0^{\mathcal{M}_0} \sim q_0^{\mathcal{M}_1}$. Intuitively, separating sequences witness the inequality of FSMs:

Definition 3. *Given FSMs $\mathcal{M}_0$ and $\mathcal{M}_1$, a sequence $\sigma \in I^*$ is a* separating sequence *for $\mathcal{M}_0$ and $\mathcal{M}_1$ if $\lambda^{\mathcal{M}_0}(\sigma) \neq \lambda^{\mathcal{M}_1}(\sigma)$. A set of sequences $L \subseteq I^*$ is a* fingerprint *for a set of models $\mathbb{M}$ if every pair of non-equivalent models $\mathcal{M}_0, \mathcal{M}_1 \in \mathbb{M}$ has a separating sequence in L, and every $\sigma \in L$ separates some pair of models $\mathcal{M}_0, \mathcal{M}_1 \in \mathbb{M}$.*

B. Formal Problem Statement

We define the *fingerprinting problem under a closed-world assumption*, also known as *group matching fingerprinting* [21].

Closed-World Fingerprinting

Given a set of inequivalent models $\mathbb{M}$ and a list of black-box implementations $\mathbb{I} \subseteq \mathbb{M}$, compute a map $\mu\colon \mathbb{I} \to \mathbb{M}$ s.t. for all $\mathcal{I} \in \mathbb{I},\ \mathcal{M} \in \mathbb{M}$: $\mu(\mathcal{I}) = \mathcal{M}$ iff $\mathcal{I} \sim \mathcal{M}$.

By the inclusion $\mathbb{I} \subseteq \mathbb{M}$, we indicate that for every $\mathcal{I} \in \mathbb{I}$ there exists $\mathcal{M} \in \mathbb{M}$ such that $\mathcal{I} \sim \mathcal{M}$. The assumption $\mathbb{I} \subseteq \mathbb{M}$ ensures that every implementation is equivalent to a reference model. In Section II, we saw that this assumption is often too strict. This motivates the open world variation of fingerprinting, where the set of models is not known a priori. Thereby, building the set $\mathbb{M}$ becomes a part of the problem.

Open-World Fingerprinting

Given a list of black-box implementations $\mathbb{I}$, compute a set of models $\mathbb{M}$ and a map $\mu\colon \mathbb{I} \to \mathbb{M}$ s.t. for all $\mathcal{I} \in \mathbb{I},\ \mathcal{M} \in \mathbb{M}$: $\mu(\mathcal{I}) = \mathcal{M}$ iff $\mathcal{I} \sim \mathcal{M}$.

If an initial set of inequivalent models is available, the problem can easily be adapted to support a "warm start".

IV. State of the Art

In this section, we discuss the state of the art for fingerprinting, conformance checking and active automata learning.

A. Fingerprinting Algorithms

Given an implementation, a fingerprinting algorithm returns a potentially matching model from a fixed set of reference models. An algorithm executes a sequence by running it on the implementation. Contrary to standard definitions, we assume that the fingerprinting algorithm also returns the set of all executed sequences.

Definition 4. *A* fingerprinting algorithm *requires an implementation $\mathcal{I}$ and a set of models $\mathbb{M}$ as inputs. The algorithm executes a subset $L_F \subseteq I^*$ of the fingerprint for $\mathbb{M}$. It returns L_F and $\mathcal{M}$ if there is a model $\mathcal{M} \in \mathbb{M}$ which is the only model that satisfies $\mathcal{I} \sim_{L_F} \mathcal{M}$; otherwise, it returns L_F and* None.

If the closed-world assumption holds, i.e., for each $\mathcal{I} \in \mathbb{I}$ there exists $\mathcal{M} \in \mathbb{M}$ such that $\mathcal{I} \sim \mathcal{M}$, fingerprinting algorithms always return exactly one model. To solve the fingerprinting problem under a closed-world assumption, we execute a fingerprinting algorithm to each implementation $\mathcal{I} \in \mathbb{I}$ such that a reference model $\mathcal{M} \in \mathbb{M}$ is found for which $\mathcal{I} \sim \mathcal{M}$ holds. *Static fingerprinting algorithms* compute a set of separating sequences and run these sequences on implementation $\mathcal{I}$ in an arbitrary order, terminating once at most a single model remains [7], [21]. Because each separating sequence distinguishes a pair of models in $\mathbb{M}$, and the fingerprint includes one separating sequence for every such pair, executing all sequences on $\mathcal{I}$ guarantees the elimination of at least $|\mathbb{M}| - 1$ models. *Dynamic fingerprinting algorithms* also compute a set of separating sequences but determine on-the-fly which sequence to run next. A simple approach is to only select separating sequences that are guaranteed to rule out at least one of the remaining models. This can be accomplished through, e.g., *adaptive distinguishing graphs* (ADGs) [13]. An alternative implementation of ADGs might order the separating sequences based on the expected number of inequivalent models after applying the sequence, following the definition of adaptive distinguishing sequences [18]. We use the latter ADG interpretation throughout this paper.

Example 3. *Suppose we want to construct a fingerprint of references $\mathcal{M}_0, \mathcal{M}_1$ and $\mathcal{M}_2$ from Fig. 1. The separating sequence* hello $\cdot$ hello *distinguishes the pairs $(\mathcal{M}_0, \mathcal{M}_1)$ and $(\mathcal{M}_0, \mathcal{M}_2)$, whereas* hello $\cdot$ kex $\cdot$ hello $\cdot$ hello *separates $(\mathcal{M}_1, \mathcal{M}_2)$. A static fingerprinting algorithm begins by applying the separating sequence for $(\mathcal{M}_0, \mathcal{M}_1)$; if the implementation is not equivalent to $\mathcal{M}_0$, a second sequence is required. The ADG approach described above observes that* hello $\cdot$ kex $\cdot$ hello $\cdot$ hello *distinguishes all models, and thus only requires one sequence during application.*

B. Conformance Checking

Conformance checking [22], [23] studies whether a given model coincides with a black-box implementation. Under a closed-world assumption, a fingerprint is sufficient evidence to conclude that the implementation coincides with a model. However, in an open world, the selected model may not actually be equivalent to the implementation. A *conformance query* (CQ) checks if a given implementation $\mathcal{I}$ and model $\mathcal{M}$ conform, using a conformance checking algorithm.

Definition 5. *A conformance checking algorithm requires an implementation $\mathcal{I}$ and a model $\mathcal{M}$ as inputs. The algorithm* *returns* $L_{CQ} \subseteq I^*$ *along with a Boolean outcome:* true *if* $\mathcal{I} \sim_{L_{CQ}} \mathcal{M}$ *and* false *otherwise.*

We consider three categories of conformance checking: (a) algorithms with strong guarantees, (b) lightweight algorithms, and (c) algorithms providing a trade-off between them.

Wp (in cat. a) is an established conformance checking algorithm [24]. It creates a test suite of input sequences by: (1) accessing all states in the model, (2) performing k input steps from each state, and (3) checking whether the expected state has been reached using a separating sequence. Wp guarantees that if the implementation has at most k more states than the model, then the test suite contains a separating sequence if the implementation and model are inequivalent. Wp is expensive as it is exponential in k.

RandomWord (in cat. b) generates random input sequences of a specified length and compares the outputs from the model and implementation [16]. While random sequences are very cheap to generate, only statistical guarantees can be given.

RandomWp (in cat. c) combines Wp and RandomWord based on ideas described in [25]. It visits all states in the model, performs a *random walk*, and then checks whether the expected state has been reached. It does not give the same guarantees as Wp, but is very effective in finding separating sequences with few queries [26], [27].

C. Learning Algorithms

Active automata learning (AAL) aims to learn models of black-box systems by systematically providing inputs and observing outputs. AAL algorithms are a natural candidate to learn implementations and incrementally build the reference set in our setting. We refer the reader to the surveys by Howar and Steffen [28] and Vaandrager [15] for an overview of AAL.

Definition 6. *An AAL algorithm requires an implementation* $\mathcal{I}$ *as input and returns a model* $\mathcal{M}$ *such that* $\mathcal{I} \sim \mathcal{M}$.

AAL algorithms are often implemented within Angluin's *Minimally Adequate Teacher* (MAT) framework [16]. In this framework, the learning algorithm has access to a teacher who has perfect knowledge of the *System Under Learning* (SUL). The teacher can answer two types of queries: *Output Queries* (OQs) and *Equivalence Queries* (EQs). When asked an OQ with a given input sequence w, the teacher returns the output sequence as produced by SUL $\mathcal{M}$, i.e., $\lambda^{\mathcal{M}}(q_0, w)$. When asked an EQ, the teacher answers whether a provided hypothesis model $\mathcal{H}$ is equivalent to SUL $\mathcal{M}$, i.e., whether $\mathcal{H} \sim \mathcal{M}$ holds. If the provided hypothesis is incorrect, the teacher returns a counterexample that witnesses the behavioral difference. The counterexample can be used by the learner to refine the hypothesis.

Example 4. *Suppose we want to learn a model of* $\mathcal{M}_0$ *from Fig. 1. A learning algorithm might start by posing OQs:* hello, kex *and* data*. Based on the teacher's responses, we construct initial hypothesis* $\mathcal{H}_0$ *shown in Fig. 2. Next, we pose an EQ with* $\mathcal{H}_0$*. The teacher might respond with counterexample* hello $\cdot$ kex*,*

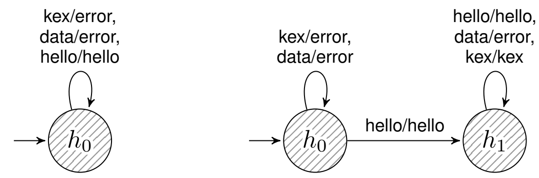


Fig. 2: Hypotheses $\mathcal{H}_0$ and $\mathcal{H}_1$ for SUL $\mathcal{M}_0$.

which produces hello $\cdot$ error *in* $\mathcal{H}_0$ *but* hello $\cdot$ kex *in* $\mathcal{M}_0$*. This discrepancy reveals the existence of a second state.*

To construct the next hypothesis, we need to identify transitions that lead to different states. This can be done using a separating sequence, such as kex.[2] *For instance, to find the target of the transition* hello$\cdot$kex*, we pose OQ* hello$\cdot$kex$\cdot$kex*. If this path reaches the initial state, then* kex *should yield* error*; if it reaches the second state, it should return* kex*. Using* kex *as separating sequence to identify all transitions, we build refined hypothesis* $\mathcal{H}_1$*. This hypothesis can be refuted by counterexample* hello $\cdot$ kex $\cdot$ data*, which discovers the last state. After identifying each transition once more, we arrive at the correct hypothesis which is equivalent to* $\mathcal{M}_0$*. Posing an EQ at this point confirms its correctness, and the learning process terminates.*

We consider the state-of-the-art algorithm $L^\#$ [18]. This algorithm efficiently learns a model from scratch by using an efficient data structure to store all interactions with the SUL. $L^\#$ outperforms the automata learning algorithm L^* [16] and is competitive with algorithms like TTT [17]. Additionally, we consider *adaptive* active automata learning [19], [29], in particular the algorithm $AL^\#$ [20] built on top of $L^\#$. Adaptive learning algorithms reuse information from a set of reference models $\mathbb{M}$. If these models closely resemble the SUL, the learning process can speed up significantly. For open-world fingerprinting, the models learned so far can be used as reference models.

Approximating the EQ. Assuming a teacher with perfect knowledge of the SUL, $L^\#$ and $AL^\#$ learn the correct model using a number of queries polynomial in the number of states and inputs of the SUL, as well as the length of the longest counterexample. However, such a perfect teacher often does not exist as the SUL is a black-box implementation. Thus, a teacher is often implemented by executing OQs directly on the SUL and approximating the EQs using conformance checking, see Section IV-B. When using conformance checking, the learning algorithm returns a model $\mathcal{M}$ that is equivalent with the SUL w.r.t. all input sequences posed during learning and conformance checking.

Definition 7. *An adaptive AAL algorithm using conformance checking requires an implementation* $\mathcal{I}$*, a set of models* $\mathbb{M}$*, and* $L_F \subseteq I^*$ *as inputs. The algorithm returns a model* $\mathcal{M}$ *and* $L_L \subseteq I^*$ *such that* $\mathcal{I} \sim_{L_F \cup L_L} \mathcal{M}$.

[2]This example is consistent with the $L^\#$ algorithm which identifies states by distinguishing them from all but one state, in contrast to algorithms like L^* that aim to establish equivalence.

In this setting, we allow initialization of the data structure using a set of sequences L_F. Data structure initialization using logs is frequently used in AAL to speed up the learning process, as described in [30].

V. Incremental Fingerprinting

In this section, we introduce a framework for incremental fingerprinting, combining closed-world fingerprinting and automata learning to solve the open-world fingerprinting problem efficiently and accurately. We detail the algorithm, its correctness and complexity. We conclude this section with a recommended configuration of the algorithm, which we refer to as *the* incremental fingerprinting algorithm INFERNAL.

A. Incremental Fingerprinting Algorithm

Algorithm 1 lists INCREMENTALFINGERPRINTING which solves the open-world fingerprinting problem, i.e., it returns $\mathbb{M}$ such that for all $\mathcal{I} \in \mathbb{I}$ there is an equivalent model in $\mathcal{M}$. The algorithm iteratively builds a model set $\mathbb{M}$, and functions μ and γ. For each $\mathcal{I} \in \mathbb{I}$, μ stores an equivalent model $\mathcal{M} \in \mathbb{M}$, while γ records the input sequences $L \subset I^*$ used to identify $\mathcal{I}$. Upon termination, $\mu(\mathcal{I}) = \mathcal{M} \leftrightarrow \mathcal{I} \sim_{\gamma(\mathcal{I})} \mathcal{M}$ holds for each implementation $\mathcal{I} \in \mathbb{I}$. The workflow of INCREMENTALFINGERPRINTING is outlined in Fig. 3 and the algorithm is listed in Alg. 1.

In Lines 1–2 of Alg. 1, we initialize $\mathbb{M}$ and iterate over all implementations $\mathbb{I}$. In Line 3, we run the algorithm IDENTIFY-ORLEARN, discussed below, with the current implementation $\mathcal{I}$ and set of references $\mathbb{M}$. The IDENTIFYORLEARN algorithm either matches $\mathcal{I}$ with a reference in $\mathbb{M}$ or learns a new model. In Lines 4–7, the model $\mathcal{M}$ and language L returned by IDENTIFYORLEARN are used to update $\mathbb{M}$, μ and γ. When all $\mathcal{I} \in \mathbb{I}$ are considered, we return $\mathbb{M}$, γ and μ.

Before discussing IDENTIFYORLEARN, we explain the behavior of the LEARN algorithm used in Lines 1 and 7. LEARN requires an implementation $\mathcal{I}$, a set of models $\mathbb{M}$ and a language $L_F \subseteq I^*$ to initialize the learning data structure. Internally, it makes use of several CQs to approximate the EQ. Algorithm LEARN returns a model $\mathcal{M}$ and language L_L of OQs used during learning, for which $\mathcal{I} \sim_{L_L \cup L_F} \mathcal{M}$ holds.

IDENTIFYORLEARN, presented in Alg. 2, either identifies a matching model $\mathcal{M}$ in the set of references $\mathbb{M}$, or learns a new model. In Line 1, we handle the special case where $\mathbb{M}$ is empty and call LEARN with $\mathbb{M} = \emptyset$ and $L_F = \emptyset$. We return the result of LEARN: found model $\mathcal{M}$ and language L_L.

In Line 2, we use fingerprinting to identify the reference in $\mathbb{M}$ that is a candidate for equivalence with implementation $\mathcal{I}$. FINGERPRINTING requires an implementation $\mathcal{I}$ and a set of inequivalent models $\mathbb{M}$ as inputs. The algorithm returns the set of executed sequences L_F to avoid reposing the sequences during the CQ or when learning a new model. Additionally, a candidate reference $\mathcal{M} \in \mathbb{M}$ if $\mathcal{M}$ is the only reference in $\mathbb{M}$ for which $\mathcal{I} \sim_{L_F} \mathcal{M}$ holds and *None* otherwise.

In Lines 3–6, we test the candidate reference $\mathcal{M}$ using a CQ if fingerprinting resulted in a non-*None* model $\mathcal{M}$. CONFQUERY requires an implementation $\mathcal{I}$ and a model $\mathcal{M}$ as

Algorithm 1 INCREMENTALFINGERPRINTING$_\mathcal{C}$

Require: implementations $\mathbb{I}$, initial references $\mathbb{M}_0$

1: Initialize $\mathbb{M} \leftarrow \mathbb{M}_0$
2: **for** $\mathcal{I} \in \mathbb{I}$ **do**
3: $\quad \mathcal{M}, L = $ IDENTIFYORLEARN$_\mathcal{C}(\mathcal{I}, \mathbb{M})$
4: $\quad \mathbb{M} \leftarrow \mathbb{M} \cup \{\mathcal{M}\}$
5: $\quad \gamma(\mathcal{I}) = L$ ▷ For Thm. 1
6: $\quad \mu(\mathcal{I}) = \mathcal{M}$
7: **return** $\mathbb{M}$, γ, μ

Algorithm 2 IDENTIFYORLEARN$_\mathcal{C}$

Require: implementation $\mathcal{I}$, references $\mathbb{M}$

1: **if** $\mathbb{M} = \emptyset$ **then return** LEARN$_\mathcal{C}(\mathcal{I}, \mathbb{M}, \emptyset)$
2: $\mathcal{M}, L_F \leftarrow$ FINGERPRINTING$_\mathcal{C}(\mathcal{I}, \mathbb{M})$ ▷ Section IV-A
3: **if** $\mathcal{M}$ is not *None* **then**
4: $\quad b, L_{CQ} \leftarrow$ CONFQUERY$_\mathcal{C}(\mathcal{I}, \mathcal{M})$ ▷ Section IV-B
5: $\quad L_F \leftarrow L_F \cup L_{CQ}$
6: $\quad$ **if** b **then return** $\mathcal{M}$, L_F
7: $\mathcal{M}, L_L \leftarrow$ LEARN$_\mathcal{C}(\mathcal{I}, \mathbb{M}, L_F)$ ▷ Section IV-C
8: **return** $\mathcal{M}$, $L_F \cup L_L$

inputs. CONFQUERY returns L_{CQ} and a Boolean b set to *true* if $\mathcal{I} \sim_{L_{CQ}} \mathcal{M}$ and *false* otherwise. After execution, we update L_F to include the input sequences L_{CQ} posed during the CQ. If the conformance check passes, i.e., $b =$ *true*, we return the found model $\mathcal{M}$ and the language L_F. Lines 7–8 handle the case where $\mathcal{I} \nsim \mathcal{M}$ for all $\mathcal{M} \in \mathbb{M}$ after fingerprinting or conformance checking. The algorithm LEARN in Line 7 receives the set of models $\mathbb{M}$ and the language $L_F \cup L_{CQ}$ as inputs. We return the found model $\mathcal{M}$ and the language $L_F \cup L_L$.

Alg. 1 and 2 are parameterized by a tuple $\mathcal{C}$ which contains the algorithms for fingerprinting (FINGERPRINTING$_\mathcal{C}$), fingerprinting CQ (CONFQUERY$_\mathcal{C}$), learning (LEARN$_\mathcal{C}$), and learning CQ (LEARNINGCONFQUERY$_\mathcal{C}$).

B. Correctness of Incremental Fingerprinting

Incremental fingerprinting is correct if $\mathcal{I} \sim_{\gamma(\mathcal{I})} \mu(\mathcal{I}) \rightarrow \mathcal{I} \sim \mu(\mathcal{I})$ holds for any implementation $\mathcal{I}$, thus, if we can correctly generalize from the constructed language $\gamma(\mathcal{I})$. We prove the correctness of INCREMENTALFINGERPRINTING w.r.t. $\mathcal{I} \sim_L \mathcal{M}$ by formalizing the contracts of IDENTIFYORLEARN and INCREMENTALFINGERPRINTING. Additionally, we prove that INCREMENTALFINGERPRINTING is correct when CONF-QUERY has complete knowledge.

Lemma 1. IDENTIFYORLEARN$_\mathcal{C}$ *(Alg. 2) requires an implementation $\mathcal{I}$ and a set of inequivalent models $\mathbb{M}$ as inputs. After execution, a model $\mathcal{M}$ and a language $L \subseteq I^*$ are returned such that $\mathcal{I} \sim_L \mathcal{M}$ and there is at most one $\mathcal{M}' \in \mathbb{M}$ for which $\mathcal{I} \sim_L \mathcal{M}'$. Additionally, if* CONFQUERY *and* LEARNINGCONFQUERY *in $\mathcal{C}$ are perfect teachers, then $\mathcal{I} \sim \mathcal{M}$.*

With a perfect teacher, INCREMENTALFINGERPRINTING precisely solves the open-world fingerprinting problem.

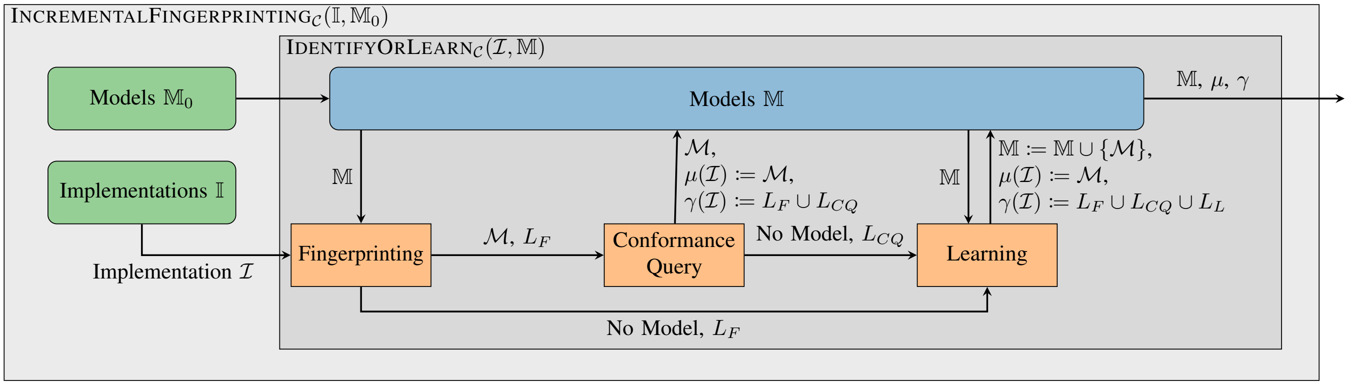


Fig. 3: Overview of Incremental Fingerprinting.

Theorem 1. INCREMENTALFINGERPRINTING$_{\mathcal{C}}$ *(Alg. 1) requires a list of implementations $\mathbb{I}$ and a set of distinct models $\mathbb{M}_0$ as inputs. The algorithm returns $\mathbb{M}$, γ and μ such that for $\mathcal{I} \in \mathbb{I}$ there exists a $\mathcal{M} \in \mathbb{M}$ for which $\mathcal{I} \sim_{\gamma(\mathcal{I})} \mathcal{M}$ iff $\mu(\mathcal{I}) = \mathcal{M}$. Additionally, if* CONFQUERY *and* LEARNINGCONFQUERY *in $\mathcal{C}$ are perfect teachers, then $\mathcal{I} \sim \mathcal{M}$.*

C. Complexity of Incremental Fingerprinting

The complexity of learning algorithms is often measured in number of queries used to learn the SUL [28], [15]. For $L^\#$, learning a model $\mathcal{M}$ has asymptotic query complexity $\mathcal{O}(kn^2 + n \log l)$ where $n = |Q^{\mathcal{M}}|$, $k = |I|$ and l is the length of the longest counterexample [18]. Additionally, we can prove that at most n EQs are required. We now analyze the complexity of learning a set of models $\mathbb{M}$ representing a list of implementations $\mathbb{I}$ when a perfect teacher is available. We first consider repeated application of $L^\#$ ($RL^\#$) as a baseline and INCREMENTALFINGERPRINTING using $L^\#$ as LEARN algorithm.

Theorem 2. *Let $m = |\mathbb{M}|$, $i = |\mathbb{I}|$. Assume that a perfect teacher is available and that all $\mathcal{M} \in \mathbb{M}$ have at most n states, k inputs and counterexamples of length at most l. $RL^\#$ learns the the correct set of models $\mathbb{M}$ within $\mathcal{O}(i(kn^2+n \log l))$ OQs and at most in EQs.* INCREMENTALFINGERPRINTING *with SepSeq and $L^\#$ learns the correct set of models $\mathbb{M}$ within $\mathcal{O}(m(kn^2 + n \log l) + im^2)$ OQs and at most $mn + i$ EQs.*

Clearly, if i is significantly larger than m, our approach asymptotically outperforms $RL^\#$ when considering the learning queries. An overview of the asymptotic complexities is presented in Table I and proven in App. A.

D. Misclassifications

CQs may yield misclassifications, i.e., they can wrongly conclude $\mathcal{I}$ is equivalent to some $\mathcal{M}$. As CQs are used within INCREMENTALFINGERPRINTING, our approach may also yield misclassifications. A misclassification for INCREMENTALFINGERPRINTING occurs when a model $\mu(\mathcal{I}) = \mathcal{M}$ while $\mathcal{I} \nsim \mathcal{M}$, caused by either the CQ on Line 4 of Alg. 2 or the final learning CQ on Line 1 and 8. However, if incremental fingerprinting starts with a complete set of references, as in the closed-world scenario, no misclassifications can occur.

Theorem 3. *Let $\mathbb{I}$ be a list of implementations and $\mathbb{M}_0$ a set of inequivalent models such that $\mathbb{I} \subseteq \mathbb{M}_0$. Executing* INCREMENTALFINGERPRINTING$_{\mathcal{C}}$ *with initial references $\mathbb{M}_0$ and implementations $\mathbb{I}$ returns $\mathbb{M}$ and μ such that $\mathbb{M}_0 = \mathbb{M}$ and for $\mathcal{I} \in \mathbb{I}$, $\mu(\mathcal{I}) = \mathcal{M}$ iff $\mathcal{I} \sim \mathcal{M}$ for some $\mathcal{M} \in \mathbb{M}$.*

This implies that once the algorithm has learned a set of models such that all implementations can be correctly identified, i.e., the reference set is complete, no further misclassifications occur. The same statement holds for repeated $AL^\#$ but cannot be proven for repeated $L^\#$ because no information from previously learned models is used by $L^\#$.

E. INFERNAL *Algorithm*

The incremental fingerprinting algorithm can be instantiated with different algorithms for fingerprinting, conformance checking, learning and conformance checking during learning, as indicated by parameter $\mathcal{C}$. This also exhibits the algorithm's flexibility; components are easily replaced when improved algorithms are designed. We use INFERNAL to refer to the incremental fingerprinting algorithm instantiated with *ADG* for fingerprinting, *RandomWp* to implement both conformance checks, and $AL^\#$ for learning, as these are all state of the art algorithms. The effects of different possible instantiations will be investigated in the experimental evaluation of RQ2 in Section VI.

VI. EXPERIMENTAL EVALUATION

In this section, we empirically evaluate the performance of our incremental fingerprinting algorithm INFERNAL (Section V), which solves the open-world fingerprinting problem (Section III). The source code and benchmarks are available online [31].[3] In our experiments, we focus on fingerprinting network protocols. We aim to answer the following research questions:

RQ1: How does INFERNAL compare against baselines?
RQ2: What algorithmic design choices are crucial for the performance of INFERNAL?
RQ3: To what extent do misclassifications produced by learning CQs influence misclassifications generated by the fingerprinting CQ?

[3] https://github.com/lkruger27/IncrementalFingerprintingOpenWorld

TABLE I: Query complexity under a perfect teacher with $m = |\mathbb{M}|$, $i = |\mathbb{I}|$, $n \leq |\mathcal{M}|$ for all $\mathcal{M} \in \mathbb{M}$, $k = |I|$ and counterexamples of length at most l. We assume $i > m$ when fingerprinting black-box implementations of the same protocol.

Algorithm	Maximum OQs	Maximum EQs
Repeated $L^{\#}$	$i(kn^2 + n \log l)$	in
IncrementalFingerprinting with $L^{\#}$	$m(kn^2 + n \log l) + im^2$	$mn + i$
Repeated $AL^{\#}$	$i(kn^2 + kmn^2 + n^3m^2) + mn \log l$	$mn + i - m$
IncrementalFingerprinting with $AL^{\#}$	$m(kn^2 + kmn^2 + n^3m^2 + n \log l) + im^2$	$mn + i$

A. Benchmarks

We consider several benchmarks representing network protocol implementations, see App. B for details.

TLS. Transport Layer Security (TLS) is a well-known security protocol. We use the 596 implementations and 22 underlying models of mbedTLS and OpenSSL learned and fingerprinted by [13]. These models range between 6 and 14 states.

SSH. The Secure Shell Protocol (SSH) is a prominent security protocol of which three implementations have been learned [8] using AAL. These models have between 17 and 66 states. We create 17 additional models by applying mutations [20], such as diverting transitions, removing states, adding states, and changing transition outputs. We consider 100 implementations, where each of the 20 models occurs 5 times.

BLE. Bluetooth Low Energy (BLE) is a low-power variant of the Bluetooth communication protocol. BLE devices have previously been learned and fingerprinted [10]. For our evaluation, we use 4 of their models (CC2650, CC2652R1, CYW43455, nRF52832). We extend the BLE model suite by a model of BLE car access systems of a Tesla Model 3 [32] and, additionally, we use models of 3 devices: (1) a proof-of-concept version of CYBLE-416045-02, (2) an updated version of CC2652R1, and (3) Apollo3 Blue. These last three models were learned for this paper; learning details can be found in App. B. The 8 models range between 2 and 16 states and use the same 7 inputs. We copy each model 5 times, leading to 40 implementations.

BLEDiff. Furthermore, we consider 6 BLE models learned using the black-box protocol noncompliance checking framework BLEDiff and provided by the authors of [33]. We make the model input-complete by adding self-loops with a unique output ϵ for undefined transitions. The 6 models range between 5 and 8 states and have 32 inputs. Due to a difference in the input names and number of inputs, we do not merge BLE and BLEDiff. We copy each model 5 times, resulting in 30 implementations.

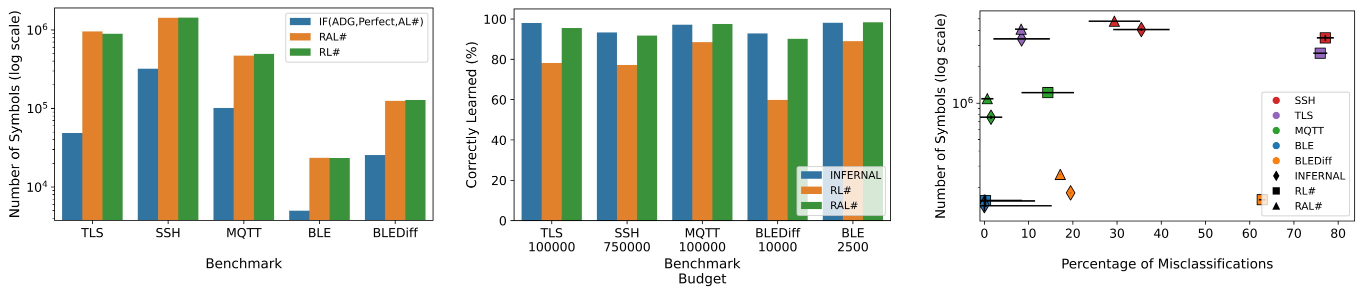
MQTT. Message Queuing Telemetry Transport (MQTT) is a publish/subscribe protocol often used in IoT [34]. MQTT has been learned and fuzz tested [35]. We consider an extended set of MQTT brokers containing: HiveMQ 1.3.5, emqx 5.8.6, ejabberd 25.3.0, VerneMQ 2.0.1, Eclipse Mosquitto 2.0.11, and mochi 2.7.9 with a broadened alphabet. The models range between 7 and 53 states. Many of the available MQTT clients include broker logic to improve communication and implement broker behavior already on the client side, e.g., responses to invalid requests. Therefore, to communicate with the broker, we utilize a custom Java client to test the broker behavior in isolation. We use the *Py4J* library to connect the incremental fingerprinting setup, which is written in Python, to the custom Java client. Implementations are learned using $L^{\#}$ with a state prefix oracle with 10 walks per state and a walk length of 12. The state prefix oracle is similar to RandomWp but does not append a separating sequence after the random walk. We copy each model 5 times to model 30 implementations.

B. Experiment Set-up

We implement IncrementalFingerprinting using the automata learning library AALpy [36], using its $AL^{\#}$ and $L^{\#}$ algorithms. When learning a specific implementation, we rely on a cache containing all OQ responses for fingerprinting, conformance checking, and learning.

Measurements. The performance of the algorithms is measured based on the *number of symbols* (sum of the number of inputs and number of resets) sent to the SUL, as is standard in active automata learning [18]. Interacting with a black-box system usually requires more time than computing the next OQ, indicating that the number of interactions accurately represents performance. For example, setting up the connection and waiting for network packets to arrive takes a considerable amount of time when interacting with a BLE device. Additionally, we measure the *percentage of misclassifications* while learning a list of implementations, defined as the ratio of misclassifications to implementations. To determine whether a misclassification occurs, we check bisimilarity between the ground truth model for an implementation $\mathcal{I}$ and $\mu(\mathcal{I})$. Unlike conventional automata learning experiments, we do not abort the learn process or CQ based on any side information. In some experiments, we set a maximum symbol budget. The symbol budget is the maximal number of symbols that may be used during learning and testing. In this case, the learning process can stop either when all tests pass or when the budget runs out. If the budget is exceeded during learning, we return the previous hypothesis. If it happens during testing, we return the hypothesis currently under evaluation. In this scenario, a model is learned correctly if the returned hypothesis is equivalent to the implementation. We run all experiments with 30 seeds.

Configuration $\mathcal{C}$. We call the naive separating sequence fingerprinting approach from Section IV-A *SepSeq* and use *ADG* to refer to the second discussed ADG implementation. Inside Infernal, we use either $AL^{\#}$ or $L^{\#}$ as discussed in Section IV-C. We use the following specific settings in our experiments concerning the CQ implementations, see Section IV-B:



(a) RQ1A: Perfect teacher. (b) RQ1B: Maximum symbol budget. (c) RQ1C: Testing until all tests pass.

Fig. 4: Comparison of INFERNAL, $RL^{\#}$ and $RAL^{\#}$ for Experiment 1. The colors of bars indicate the algorithms for Figs. 4a and 4b. For Fig. 4c, the colors indicate the benchmarks, and the markers indicate the algorithms. The results are averaged over all seeds in all plots. For Fig. 4c, the black lines indicate the standard deviations.

- Wp: The test suite of the Wp method grows exponentially with its parameter k. We use $k = 2$ in line with conclusions from [37] and as also done in [27].
- RandomWord: We execute 1000 sequences with lengths ranging from 10 to 30.
- RandomWp: The number of walks per state is 100 unless indicated otherwise. Each walk has a random length between 1 and 5 sampled from a uniform distribution.
- Budget RandomWp: The random walk length is set to a geometric distribution with minimum length 3 and expected length 8, see [25].

Baselines. We assume that no models are available at the start of an experiment, which rules out comparisons with closed-world fingerprinting. The automata learning algorithm L^* is often used to learn a set of models that is later used in closed-world fingerprinting [7], [9] We use $RL^{\#}$ (repeated $L^{\#}$) and $RAL^{\#}$ (repeated $AL^{\#}$) as baselines instead of repeated RL^* as $L^{\#}$ outperforms L^* [18]. In $RL^{\#}$, we maintain a set $\mathbb{M}$ and a mapping from implementations to models. For every $\mathcal{I}$, we run $L^{\#}$, add the model if it is not equivalent to any in $\mathbb{M}$ and update the mapping. $RAL^{\#}$ follows the same procedure, using the current $\mathbb{M}$ as the reference set. We note that $RAL^{\#}$ can be seen as a simplification of INFERNAL that omits the fingerprint and subsequent CQ.

C. Experiment 1: INFERNAL vs Baselines $RAL^{\#}$ and $RL^{\#}$

To answer RQ1, we compare the performance of INFERNAL to baselines $RAL^{\#}$ and $RL^{\#}$. Given the teacher's significant impact on the performance of INFERNAL and the baselines, we examine RQ1 across three scenarios. To this end, we address the following subquestions: How does INFERNAL compare against baselines when using

1) A perfect teacher,
2) RandomWp and a maximum symbol budget, and
3) RandomWp until all tests pass?

Results. The results for Experiment 1 are depicted in Fig. 4, per subquestion. Fig. 4a shows the number of symbols (log-scaled) sent to the SUL while learning the set of models using a perfect teacher; lower is better. Fig. 4b shows the percentage of correctly learned models when learning is terminated after exceeding the budget which is included on the x-axis; higher is better. Finally, Fig. 4c shows both the number of interactions (log-scaled) and the number of misclassifications. The interactions in Fig. 4c are not limited by a budget, instead the experiment ends when the CQ indicates that all tests passed. This last plot can be used to determine the trade-off between misclassifications and number of interactions; low and left is best. We perform an additional experiment for RQ1c varying the number of walks per state in RandomWp for SSH and TLS, displayed in Fig. 7 and 8 (App. E).

Answer RQ1

When a perfect teacher is available, INFERNAL performs almost an order of magnitude better than the baselines across all benchmarks. When a maximum budget is used, INFERNAL and $RAL^{\#}$ are more accurate than $RL^{\#}$. When testing until all tests pass, INFERNAL and $RAL^{\#}$ provide a better trade-off between few interactions and few misclassifications compared to $RL^{\#}$.

On $RL^{\#}$. The baseline $RL^{\#}$ is not a reasonable solution to open-world fingerprinting as it leads to an excessive number of interactions with the system under a perfect teacher compared to INFERNAL (Fig. 4a). Moreover, $RL^{\#}$ leads to an exceptionally high misclassification rate compared to INFERNAL and $RAL^{\#}$ when using a maximal budget or when testing until convergence (Fig. 4b). The high misclassification rate is especially noticeable in Fig. 4c, $RL^{\#}$ has a misclassification of 75.9% for TLS while INFERNAL and $RAL^{\#}$ have a misclassification rate under 9%.

On INFERNAL vs $RAL^{\#}$ When considering a perfect teacher (Fig. 4a), INFERNAL clearly outperforms $RAL^{\#}$. However, when such a teacher is not available (Figs. 4b and 4c), the trade-off between INFERNAL and $RAL^{\#}$ is more nuanced: INFERNAL often requires fewer symbols than $RAL^{\#}$ while $RAL^{\#}$ often produces fewer misclassifications.

Additionally, it can be observed that the same CQ implementation for all benchmarks leads to widely different misclassification rates in Fig. 4c. This shows that to obtain an accurate model, it is essential that the CQ is well-configured.

We hypothesize that $RAL^{\#}$ has fewer misclassifications because relearning is more effective at revealing distinct implementations than some CQ configurations. Therefore, we performed additional tests with a more exhaustive CQ. In Fig. 7 for SSH, the difference in misclassification for INFERNAL and $RAL^{\#}$ diminishes. Moreover, the same experiment for TLS (Fig. 8) reveals that INFERNAL outperforms $RAL^{\#}$ for TLS when the CQ is exhaustive enough. Thus, the trade-off between INFERNAL and $RAL^{\#}$ is benchmark and resource specific.

D. Experiment 2: Ablation Study

To answer RQ2, we perform an ablation study to check whether other configurations for INFERNAL are better performing and to gain insights into the effect of the individual algorithms on the overall performance. We compare all combinations of the algorithms mentioned in Section IV:

- FINGERPRINTING: *SepSeq* and *ADG*,
- CONFQUERY: *Wp2*, *RandomWp100* and *RandomWord1000*,
- LEARN: $L^{\#}$ and $AL^{\#}$.

Results. Fig. 5 contains subplots for the TLS, SSH, MQTT, and BLEDiff benchmarks. The results for BLE are presented in Table XII in App. D, as all BLE models are well differentiable. The axis interpretation matches Experiment 1c: lower and left is better. Fig. 9 in App. E depicts an additional experiment where fingerprinting and learning use different CQ implementations.

Answer RQ2

The selected CQ algorithm has the biggest impact on the performance, with *RandomWp100* providing the best trade-off between misclassifications and number of symbols. Additionally, we conclude that *ADG* outperforms *SepSeq* and $AL^{\#}$ outperforms $L^{\#}$. These results justify the choices for INFERNAL.

On ADG vs SepSeq. ADG usually leads to fewer misclassifications; the circle marker is more to the left compared to the triangle marker. However, in BLEDiff *SepSeq* outperforms *ADG*. The number of interactions is largely dictated by CQ, rendering the fingerprinting algorithm's influence negligible.

On $AL^{\#}$ vs $L^{\#}$: Misclassifications. We observe that using $AL^{\#}$ during learning leads to fewer misclassifications compared to $L^{\#}$; the opaque color is more to the left. We hypothesize that this is because $AL^{\#}$ tests whether states in the known models are also present in the current implementation, leading to bigger and more accurate models. This explanation is in line with Experiment 1c.

On $AL^{\#}$ vs $L^{\#}$: Interactions. The number of interactions does not seem to be impacted as much when using $AL^{\#}$ over $L^{\#}$. This can be explained by the fact that learning only occurs when the implementation does not match any of the models in $\mathbb{M}$, which happens 27% of the time for TLS and 20% of the time for the other benchmarks. When only considering learning symbols, $AL^{\#}$ usually requires fewer symbols than $L^{\#}$ (Table XII, App. D). For example, incremental fingerprinting with $L^{\#}$, *RandomWp100* and *ADG* requires 25.6% more learning symbols compared to $AL^{\#}$ with the same fingerprinting and CQ settings on average.

On the CQ algorithm. RandomWp100 has fewer misclassifications compared to *RandomWord1000* while using the same number of interactions, indicating that *RandomWp100* should be preferred over *RandomWord1000*. *Wp2* produces almost no misclassifications but requires significantly more interactions. The data points for *Wp2* with INFERNAL-$AL^{\#}$ are hidden behind the data points for INFERNAL-$L^{\#}$. If the number of misclassifications should be minimal, *Wp2* may be preferable over to *RandomWp100*.

On the number of RandomWp tests. In Fig. 9 (App. E), we experiment with combinations of *RandomWp25*, *RandomWp50* and *RandomWp100* during the fingerprinting CQ (FCQ) and learning CQ (LCQ). As expected, using *RandomWp100* during the FCQ and LCQ produces the fewest misclassifications. Performing a less exhaustive CQ during either the FCQ or LCQ leads to more misclassifications but fewer symbols. The exact trade-off varies depending on the benchmark.

E. Experiment 3: Propagation of Misclassifications

To evaluate how misclassifications produced by learning affect misclassifications produced by fingerprinting (RQ3), we begin by distinguishing two categories of misclassifications, defined by the termination point of Alg. 2.

FCQ misclassifications: Terminating with an incorrect $\mathcal{M}$ on Line 6, i.e., with the CQ after fingerprinting.

LCQ misclassifications: Terminating with an incorrect $\mathcal{M}$ on Line 8, i.e., with the CQ within learning.

We consider a configuration of INFERNAL with $AL^{\#}$, *ADG* and *RandomWp100* for the FCQ. For the LCQ, we vary between *RandomWp25*, *RandomWp50* and *RandomWp100*. We deliberately use weak CQs to ensure the effect of LCQ misclassifications on FCQ misclassifications is visible.

Results. The influence of the LCQ misclassification rate on the FCQ misclassification rate is depicted in Fig. 6. Colors indicate the implementation of LEARNINGCONFQUERY. Misclassification rates are shown as stacked bars: LCQ misclassifications in solid color, FCQ misclassifications transparently.

Answer RQ3

Both CQs are essential for a low misclassification rate. A more exhaustive LCQ usually increases the number of correct models and may reduce the misclassifications during the FCQ.

Discussion. The results indicate that a more exhaustive LCQ influences both the LCQ misclassifications and its downstream effects on the FCQ misclassifications. First, we note that for MQTT and BLE the FCQ is sufficiently exhaustive to detect wrongly learned models, as seen by the non-existent FCQ misclassifications. For the BLEDiff benchmark, we observe a disproportional increase in correct models as FCQ misclassifications are prevented when fewer incorrect models are provided.

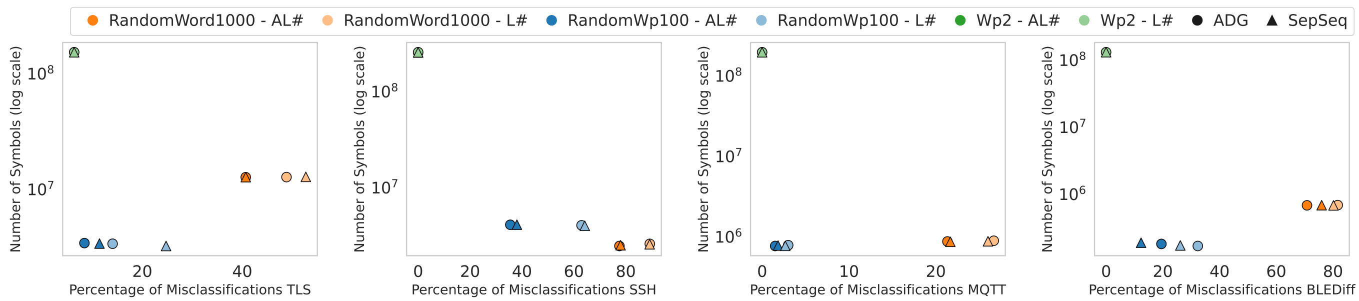


Fig. 5: Comparison of different algorithms for the components of incremental fingerprinting for Experiment 2. In each subplot, color shows CQ, opacity the learning algorithm, and marker the fingerprinting algorithm.

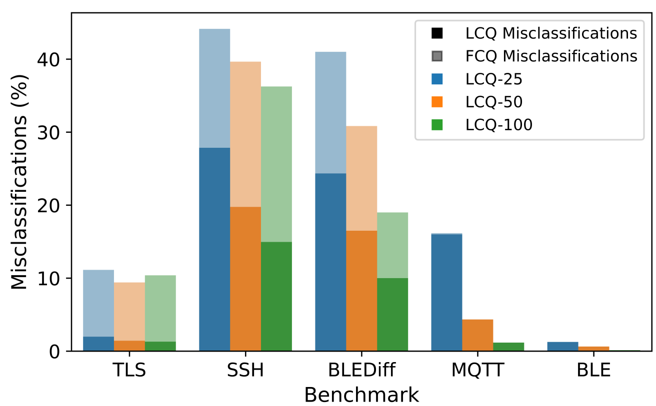


Fig. 6: Influence of the LCQ misclassifications on the FCQ misclassifications for Experiment 3. Colors show the LEARNINGCONFQUERY. LCQ errors are shown in opaque color and FCQ errors are layered transparently above in the stacked bars.

For SSH, we find that a less LCQ misclassification translates to a lower number of overall misclassifications, but increases the number of FCQ misclassifications; both CQs cannot uncover the wrong matches. In the TLS benchmark, there is no clear trend; the number of FCQ misclassifications fluctuates, while the LCQ misclassification rate marginally improves with more tests.

VII. Discussion

We discuss limitations of the proposed approach and our experiments, explore a commonly studied problem variation in machine learning, and emphasize the trade-off between misclassifications and performance.

Finite state machine learning. Our approach requires the ability to learn Finite State Machines (FSMs) representing implementations with active automata learning, which means we assume that interaction with the System Under Learning (SUL) is possible and that the SUL can be represented as an FSM. Such FSMs abstract away from data and timing information. Many network protocols can be represented as FSMs in theory but exhibit non-determinism in practice due to, e.g., packet loss. It would be interesting to use non-deterministic learning algorithms during incremental fingerprinting. However, the currently available non-deterministic automata learning algorithms are rather pragmatic, inefficient or make assumptions on the type of non-determinism, see e.g. [38], [39], [40]. However, when better suited non-deterministic learning algorithms or efficient algorithms for richer types of automata get developed, such as extended FSM which allow (user-entered) data values or FSMs with timers, they can easily be integrated into our incremental fingerprinting framework. While tools such as Nmap fingerprinting [41] and ssh-audit [42] can identify implementations with only a few probing sequences, they rely on a closed-world assumption and are typically tailored to specific protocol families. In contrast, state-machine learning is agnostic to both the protocol and its implementation.

Black-box learning. We do not use side information in the form of source code, documentation, or standards in our approach; we assume a completely black-box scenario. On the one hand, this means we do not rely on side information which allows us to learn models of proprietary software like BLE. On the other hand, many protocol implementations are open-source and, thus, their source code is freely available and could have been used to improve the performance of automata learning and the CQ [43]. In future work, we plan to investigate gray-box open-world fingerprinting, where, for example, the source code for a subset of the implementations is accessible, or the source code is available but the run-time configuration remains unknown.

Acceptable misclassification percentage. Misclassifying a black-box implementations leads to the false belief that it behaves like a certain reference model. Using this flawed reference model to assess vulnerabilities can then lead to false conclusions, such as assuming the implementation is secure when it is not. Thus, preferably an incremental fingerprinting algorithm leads to a 100% accuracy rate. We aimed for 90% accuracy when using INFERNAL in Experiment 1b, as 100% typically requires significantly more symbols (see Experiment 2, 99% with Wp).

Duplicate distribution. For all benchmarks except TLS, no realistic distribution of the number of implementations per unique model is available. As we focused on using INFERNAL for learning a list of implementations with diverse models and duplications, we included five copies of each model in the benchmark sets. When repeating Experiment 1c with different numbers of copies per unique model of MQTT, we find that the gain of INFERNAL increases with the number of copies, see Table XV in App. E. However, this experiment uses a constant number of copies per unique model while in TLS, some of the 22 models have no duplicates while other have 99 duplicates. It would be interesting to gain further insights into the distribution of duplicates and their likeliness in real-world legacy systems and investigate their impact of the performance of INFERNAL.

Implementation clustering. Machine learning approaches for fingerprinting often consider a variation of the fingerprinting problem where equivalent implementations are clustered without building a model set, see e.g. [44], [45], [46], [47]. We call this variant the *implementation clustering* problem. A solution to the open-world fingerprinting problem trivially solves the implementation clustering problem. To compute whether implementations $\mathcal{I}$ and $\mathcal{I}'$ are equivalent, the returned mapping μ alone is sufficient as $\mu(\mathcal{I}) = \mathcal{M} = \mu(\mathcal{I}')$ implies $\mathcal{I} \sim \mathcal{I}'$. The reverse does not generally hold. To build the reference set $\mathbb{M}$ from a given set of equivalent implementations, a representative model from each set must be learned. Further, the set of models returned by open-world fingerprinting favors the explainability of the solution, as all models are individually available and can be analyzed, which is not possible when only implementation clusters are available.

Passive vs active. When active learning is not possible, supervised and unsupervised learning techniques can be used, depending on whether the problem is assumed to be closed or open world. In a closed world, the training set contains collected pairs of implementations and ground-truth models from which a supervised learner generalizes. In the open-world variant, there is no such representative training set, and unsupervised learning needs to be used to identify equivalent implementations [48]. Independent of the setting, the data used to feed those expensive a priori training processes often takes several months or years to collect [47], [49], [44]. In contrast, the most costly operation in incremental fingerprinting, the learning of a new model, is only triggered when such a model is actually encountered.

Incremental fingerprinting in practice. Given a list of black-box devices, incremental fingerprinting can be used to build an initial reference set. As new devices appear, they can be identified at a relatively low cost. Many IoT systems integrate BLE devices for which firmware updates can be pulled. In such cases, the incremental approach can be integrated into the IoT system to automatically maintain behavioral models for analysis and identify when potentially harmful devices connect.

Configuring the CQ. We evaluated algorithms on benchmarks with known ground-truth models, allowing misclassification rates to be computed. In practice, ground-truth models are unavailable due to black-box assumptions. When using incremental fingerprinting instantiated as INFERNAL, (*ADG* for fingerprinting, *RandomWp* for CQ, and $AL^{\#}$ for learning), it is essential that *RandomWp* is well configured as the performance of INFERNAL depend first and foremost on the CQ oracle used. We propose the following steps to configure the CQ: Initialize RandomWp with a walk length of 5 and 100 walks per state, based on the results from Experiment 1c. Pick one of the black-box implementations to be fingerprinted. Learn the implementation multiple times over different seeds and double the number of walks per state whenever the learned models vary until all runs stabilize and produce the same model. If behavior is clearly missing from the model, increment the walk length by one. Repeat this procedure for a few of the other black-box implementations, starting from the configuration that was deemed acceptable for the previous implementation. The CQ configuration leads to an empirically reasonable trade-off. To get fewer misclassifications, we recommend a more exhaustive CQ such as the Wp-method.

Exploitation potential. State machine learning has proven to be a useful tool for revealing deviations in implementations from the corresponding protocol standards [50], [8], [10]. For example, learned models of Bluetooth implementations could uncover logical errors in the pairing procedure, as was recently exploited in the forced pairing behavior of Google Fast Pair earbuds [3], [4], which allowed pairing with any nearby device. Furthermore, state machine learning has also been proposed for detecting security vulnerabilities [51]. This approach is often referred to in the literature as *protocol state fuzzing* and has successfully detected security issues in communication protocols such as TLS [52] and DTLS [6]. However, these approaches often rely on the manual analysis of the learned models. In our framework, we could flag models that have known vulnerabilities and check whether an implementation matches any of these models.

We could automate the detection of security vulnerabilities by analyzing learned models using testing or verification techniques. Based on the idea of differential testing for detecting security vulnerabilities [53], one technique could be to examine differences between the learned models as indicators for possible security vulnerabilities. Such integration can be incorporated directly into the CQ within the IDENTIFYORLEARN procedure (Alg. 2) of our INFERNAL framework. For example, we could include input sequences in our test suite that test for the acceptance of invalid hostnames in X.509 certificates. Another approach would be to apply model checking to verify specific properties of the learned models, similar to [54], [8]. These properties could verify whether the models contain a path that bypasses authentication or enables a protocol version downgrade. Note that this extension does not violate the open-world assumption, since test suites and properties only depend on the investigated protocols and are independent of the implementations. In addition, we can incorporate possible attacks into our learning procedure by applying concepts from learning-based fuzzing [55], [35] to capture the behavior of the implementation under unexpected inputs.

VIII. RELATED WORK

Fingerprinting is extensively researched for finding known security vulnerabilities in black-box systems. The survey by Sánchez *et al*. summarizes fingerprinting using statistical methods [48], and Alrabaee *et al*. fingerprinting for binary code [56]. Previous works by Li *et al*. and Wang [57], [58] highlight limitations of closed-world fingerprinting and propose open-world methods for Android apps and websites. These rely on passively captured traffic flows. In contrast, we focus on network protocol fingerprinting via active interactions with a black-box. In the sequel, we discuss related work on fingerprinting via active automata learning (AAL) and fingerprinting using machine learning.

The most relevant works use AAL to learn a model of the implementations under a closed-world assumption and then perform fingerprinting on the learned models. Fingerprinting using AAL was initially proposed by Shu *et al*. [21] as an efficient alternative to passive automata learning which requires massive logs [49]. In the past, several protocols have been fingerprinted, e.g. [9], [13], [7]. Pferscher *et al*. demonstrates that AAL can be used to learn models of BLE devices, revealing safety-critical behavior, and how these models can be used to fingerprint the learned BLE devices. Karim *et al*. present *BLEDiff*, a black-box compliance checking tool for BLE devices based on deviations between individually learned models [33]. Janssen demonstrates how TLS stacks can be learned with AAL and then compares different methods for generating fingerprints [13]. Further, Rasoamanana *et al*. demonstrates how the TLS stack can be efficiently fingerprinted [7]. All of those works use adapted versions of L^*, targeted at the specific protocol. Unlike our work, each new model, even if related to previously learned models, is learned again from scratch. Our work proposes incremental fingerprinting to solve the fingerprinting problem in an open world. Incremental fingerprinting only learns a new model if it is determined to be different from all previously seen models, and then performs adaptive learning to make optimal use of the knowledge stored in the previous model. Fingerprinting of closely related families of models has been studied by Damasceno *et al*. [59], who propose an efficient approach that models whole families as *Featured Finite State Machines* constrained over versions, instead of individual machines. By knowing which implementations are related, they assume a closed world.

Recent work utilizes machine learning and stochastic learning to fingerprint black-box implementations. Marzani *et al*. use passive automata learning and stochastic learning for fingerprinting versions of apps based on their network communication [44]. Further, Wang *et al*. fingerprint the platform of a video stream by using machine learning [47], and Msadek *et al*. fingerprint IoT devices [45]. Sabahi-Kaviani *et al*. use machine learning to compute the alphabet for automata learning algorithms to classify encrypted traffic [46]. All of these approaches can only solve the implementation clustering problem, they can not build a model set $\mathbb{M}$. In comparison, our work uses incremental fingerprinting with adaptive learning to detect equivalent implementations, but also supports further analysis by learning behavioral models of black-box systems.

IX. Conclusion

Identifying the network protocol version running on a device allows to assess whether the device is susceptible to known security flaws. Fingerprinting is often done under a closed-world assumption, which implies that all devices match one of a curated set of known reference models. In an open world, this set is not complete, and it is not known which models are missing. Learning these models on-the-fly poses challenges in terms of resources and accuracy. We formalized the problem of beavioral open-world fingerprinting and present incremental fingerprinting (INFERNAL) to address these challenges by integrating closed-world fingerprinting and active automata learning. The experiments show that INFERNAL improves significantly over the state of the art.

Future work. We want to evaluate INFERNAL on a more extensive set of network protocol implementations, as well as on models that go beyond software protocols, such as large legacy systems. Additionally, we plan to explore gray-box open-world fingerprinting by including side information to improve the learning performance. Side information, such as source code or logs, can help answer output queries without querying the system, or can be used to initialize the learning data structure. Finally, we will explore combining incremental fingerprinting with machine learning techniques from the literature to conclude equivalence of the fingerprinted version and protocol implementation faster.

References

[1] NCC Group, "Domestic iot nightmares: Smart doorbells," 2020, accessed: 2025-10-09. [Online]. Available: https://www.nccgroup.com/research-blog/domestic-iot-nightmares-smart-doorbells/

[2] ——, "NCC Group uncovers Bluetooth Low Energy (BLE) vulnerability that puts millions of cars, mobile devices and locking systems at risk," https://newsroom.nccgroup.com/news/ncc-group-uncovers-bluetooth-low-energy-ble-vulnerability-that-puts-millions-of-cars-mobile-devices-and-locking-systems-at-risk-447952/, 2022, accessed: 2025-09-18.

[3] S. Duttagupta, N. Antonijević, B. Preneel, S. Wyns, and D. Singelée, "Whisperpair: Hijacking bluetooth accessories using google fast pair," https://whisperpair.eu/, 2026, accessed: 2026-01-16.

[4] "Cve-2025-36911," https://www.cve.org/CVERecord?id=CVE-2025-36911, accessed: 2026-01-16.

[5] D. Heinze and F. Steinmetz. (2025) Security advisory: Airoha-based bluetooth headphones and earbuds. [Online]. Available: https://insinuator.net/2025/06/airoha-bluetooth-security-vulnerabilities/#more-15309

[6] P. Fiterau-Brostean, B. Jonsson, R. Merget, J. de Ruiter, K. Sagonas, and J. Somorovsky, "Analysis of DTLS implementations using protocol state fuzzing," in *USENIX Security Symposium*. USENIX Association, 2020, pp. 2523–2540.

[7] A. T. Rasoamanana, O. Levillain, and H. Debar, "Towards a systematic and automatic use of state machine inference to uncover security flaws and fingerprint TLS stacks," in *ESORICS (3)*, ser. Lecture Notes in Computer Science, vol. 13556. Springer, 2022, pp. 637–657.

[8] P. Fiterau-Brostean, T. Lenaerts, E. Poll, J. de Ruiter, F. W. Vaandrager, and P. Verleg, "Model learning and model checking of SSH implementations," in *SPIN*. ACM, 2017, pp. 142–151.

[9] A. Pferscher and B. K. Aichernig, "Fingerprinting and analysis of bluetooth devices with automata learning," *Formal Methods Syst. Des.*, vol. 61, no. 1, pp. 35–62, 2022.

[10] ——, "Fingerprinting bluetooth low energy devices via active automata learning," in *FM*, ser. Lecture Notes in Computer Science, vol. 13047. Springer, 2021, pp. 524–542.

[11] G. Shu and D. Lee, "A formal methodology for network protocol fingerprinting," *IEEE Trans. Parallel Distributed Syst.*, vol. 22, no. 11, pp. 1813–1825, 2011.

[12] G. Argyros, I. Stais, S. Jana, A. D. Keromytis, and A. Kiayias, "Sfadiff: Automated evasion attacks and fingerprinting using black-box differential automata learning," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October*

24-28, 2016, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 1690–1701.

[13] E. Janssen, "Fingerprinting TLS implementations using model learning," MSc thesis, Radboud University, 2021. [Online]. Available: https://www.sidnlabs.nl/downloads/2eEQaXhsxKO0Js3FKoV2Yt/3136d1f6e7d60a1712e8e032631f7aca/Fingerprinting_TLS_Implementations_Using_Model_Learning_-_Erwin_Janssen.pdf

[14] B. Steffen, F. Howar, and M. Merten, "Introduction to active automata learning from a practical perspective," in *SFM*, ser. Lecture Notes in Computer Science, vol. 6659. Springer, 2011, pp. 256–296.

[15] F. W. Vaandrager, "Model learning," *Commun. ACM*, vol. 60, no. 2, pp. 86–95, 2017.

[16] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, 1987.

[17] M. Isberner, F. Howar, and B. Steffen, "The TTT algorithm: A redundancy-free approach to active automata learning," in *RV*, ser. Lecture Notes in Computer Science, vol. 8734. Springer, 2014, pp. 307–322.

[18] F. W. Vaandrager, B. Garhewal, J. Rot, and T. Wißmann, "A new approach for active automata learning based on apartness," in *TACAS*, ser. Lecture Notes in Computer Science, vol. 13243. Springer, 2022, pp. 223–243.

[19] A. Groce, D. A. Peled, and M. Yannakakis, "Adaptive model checking," in *TACAS*, ser. Lecture Notes in Computer Science, vol. 2280. Springer, 2002, pp. 357–370.

[20] L. Kruger, S. Junges, and J. Rot, "State matching and multiple references in adaptive active automata learning," in *FM (1)*, ser. Lecture Notes in Computer Science, vol. 14933. Springer, 2024, pp. 267–284.

[21] G. Shu and D. Lee, "A formal methodology for network protocol fingerprinting," *IEEE Trans. Parallel Distributed Syst.*, vol. 22, no. 11, pp. 1813–1825, 2011.

[22] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Software Eng.*, vol. 4, no. 3, pp. 178–187, 1978.

[23] M. Vasilevskii, "Failure diagnosis of automata," *Cybernetics*, vol. 9, no. 4, pp. 653–665, 1973.

[24] S. Fujiwara, G. von Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. Software Eng.*, vol. 17, no. 6, pp. 591–603, 1991.

[25] W. Smeenk, J. Moerman, F. W. Vaandrager, and D. N. Jansen, "Applying automata learning to embedded control software," in *ICFEM*, ser. Lecture Notes in Computer Science, vol. 9407. Springer, 2015, pp. 67–83.

[26] B. K. Aichernig, M. Tappler, and F. Wallner, "Benchmarking combinations of learning and testing algorithms for active automata learning," in *TAP@STAF*, ser. Lecture Notes in Computer Science, vol. 12165. Springer, 2020, pp. 3–22.

[27] B. Garhewal and C. D. N. Damasceno, "An experimental evaluation of conformance testing techniques in active automata learning," in *MODELS*. IEEE, 2023, pp. 217–227.

[28] F. Howar and B. Steffen, "Active automata learning in practice - an annotated bibliography of the years 2011 to 2016," in *Machine Learning for Dynamic Software Analysis*, ser. Lecture Notes in Computer Science, vol. 11026. Springer, 2018, pp. 123–148.

[29] C. D. N. Damasceno, M. R. Mousavi, and A. Simao, "Learning from difference: an automated approach for learning family models from software product lines," in *SPLC (A)*. ACM, 2019, pp. 10:1–10:12.

[30] N. Yang, K. Aslam, R. R. H. Schiffelers, L. Lensink, D. Hendriks, L. Cleophas, and A. Serebrenik, "Improving model inference in industry by combining active and passive learning," in *SANER*. IEEE, 2019, pp. 253–263.

[31] L. Kruger, P. Kobialka, A. Pferscher, E. Johnsen, S. Junges, and J. Rot, "Incremental Fingerprinting in an Open World: Supplementary Material," Jan. 2026. [Online]. Available: https://zenodo.org/records/18374611

[32] A. Pferscher, "Automata learning for security testing and analysis in networked environments." Ph.D. dissertation, Graz University of Technology, 2023. [Online]. Available: https://apferscher.github.io/docs/phd-thesis.pdf

[33] I. Karim, A. A. Ishtiaq, S. R. Hussain, and E. Bertino, "Blediff: Scalable and property-agnostic noncompliance checking for BLE implementations," in *SP*. IEEE, 2023, pp. 3209–3227.

[34] A. Banks, E. Briggs, K. Borgendale, and R. Gupta, "MQTT version 5.0," OASIS Standard, Tech. Rep., Mar. 2019. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

[35] B. K. Aichernig, E. Muškardin, and A. Pferscher, "Learning-based fuzzing of iot message brokers," in *ICST*. IEEE, 2021, pp. 47–58.

[36] E. Muškardin, B. K. Aichernig, I. Pill, A. Pferscher, and M. Tappler, "Aalpy: an active automata learning library," *Innov. Syst. Softw. Eng.*, vol. 18, no. 3, pp. 417–426, 2022.

[37] L. Kruger, B. Garhewal, and F. W. Vaandrager, "Lower bounds for active automata learning," in *ICGI*, ser. Proceedings of Machine Learning Research, vol. 217. PMLR, 2023, pp. 157–180.

[38] K. El-Fakih, R. Groz, M. N. Irfan, and M. Shahbaz, "Learning finite state models of observable nondeterministic systems in a testing context," in *22nd IFIP International Conference on Testing Software and Systems*, 2010, pp. 97–102.

[39] A. Khalili and A. Tacchella, "Learning nondeterministic mealy machines," in *ICGI*, ser. JMLR Workshop and Conference Proceedings, vol. 34. JMLR.org, 2014, pp. 109–123.

[40] W. Pacharoen, T. Aoki, P. Bhattarakosol, and A. Surarerks, "Active learning of nondeterministic finite state machines," *Mathematical Problems in Engineering*, vol. 2013, no. 1, p. 373265, 2013.

[41] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure.Com LLC, 2009.

[42] A. Joffe, "ssh-audit: Ssh server and client auditing tool," https://github.com/jtesta/ssh-audit, accessed: 2026-01-08.

[43] E. Elkind, B. Genest, D. A. Peled, and H. Qu, "Grey-box checking," in *FORTE*, ser. Lecture Notes in Computer Science, vol. 4229. Springer, 2006, pp. 420–435.

[44] F. Marzani, F. Ghassemi, Z. Sabahi-Kaviani, T. van Ede, and M. van Steen, "Mobile app fingerprinting through automata learning and machine learning," in *IFIP Networking*. IEEE, 2023, pp. 1–9.

[45] N. Msadek, R. Soua, and T. Engel, "Iot device fingerprinting: Machine learning based encrypted traffic analysis," in *WCNC*. IEEE, 2019, pp. 1–8.

[46] Z. Sabahi-Kaviani and F. Ghassemi, "An encrypted traffic classifier via combination of deep learning and automata learning," *Soft Comput.*, vol. 28, no. 23, pp. 13 443–13 460, 2024.

[47] Y. Wang, M. Lyu, and V. Sivaraman, "Characterizing user platforms for video streaming in broadband networks," in *IMC*. ACM, 2024, pp. 563–579.

[48] P. M. S. Sánchez, J. M. J. Valero, A. H. Celdrán, G. Bovet, M. G. Pérez, and G. M. Pérez, "A survey on device behavior fingerprinting: Data sources, techniques, application scenarios, and datasets," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 2, pp. 1048–1077, 2021.

[49] G. Celosia and M. Cunche, "Fingerprinting bluetooth-low-energy devices based on the generic attribute profile," in *IoT S&P@CCS*. ACM, 2019, pp. 24–31.

[50] M. Tappler, B. K. Aichernig, and R. Bloem, "Model-based testing iot communication via active automata learning," in *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*. IEEE Computer Society, 2017, pp. 276–287.

[51] K. Hossen, R. Groz, and J. Richier, "Security vulnerabilities detection using model inference for applications and security protocols," in *Fourth IEEE International Conference on Software Testing, Verification and Validation, ICST 2012, Berlin, Germany, 21-25 March, 2011, Workshop Proceedings*. IEEE Computer Society, 2011, pp. 534–536.

[52] J. de Ruiter and E. Poll, "Protocol state fuzzing of TLS implementations," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 193–206. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter

[53] S. Sivakorn, G. Argyros, K. Pei, A. D. Keromytis, and S. Jana, "Hvlearn: Automated black-box analysis of hostname verification in SSL/TLS implementations," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 521–538.

[54] P. Fiterau-Brostean, R. Janssen, and F. W. Vaandrager, "Combining model learning and model checking to analyze TCP implementations," in *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, S. Chaudhuri and A. Farzan, Eds., vol. 9780. Springer, 2016, pp. 454–471.

[55] A. Pferscher and B. K. Aichernig, "Stateful black-box fuzzing of bluetooth devices using automata learning," in *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, ser. Lecture Notes in Computer Science, J. V.

Deshmukh, K. Havelund, and I. Perez, Eds., vol. 13260. Springer, 2022, pp. 373–392.
[56] S. Alrabaee, M. Debbabi, and L. Wang, "A survey of binary code fingerprinting approaches: Taxonomy, methodologies, and features," *ACM Comput. Surv.*, vol. 55, no. 2, pp. 19:1–19:41, 2023.
[57] J. Li, H. Zhou, S. Wu, X. Luo, T. Wang, X. Zhan, and X. Ma, "FOAP: fine-grained open-world android app fingerprinting," in *USENIX Security Symposium*. USENIX Association, 2022, pp. 1579–1596.
[58] T. Wang, "High precision open-world website fingerprinting," in *SP*. IEEE, 2020, pp. 152–167.
[59] C. D. N. Damasceno and D. Strüber, "Family-based fingerprint analysis: A position paper," in *A Journey from Process Algebra via Timed Automata to Model Learning*, ser. Lecture Notes in Computer Science, vol. 13560. Springer, 2022, pp. 137–150.

Appendix

A. Proof of Theorems

Proof of Theorem 1. Before diving into the proofs, we reiterate the contracts of the components described in Section V.

FINGERPRINTING Algorithm FINGERPRINTING requires an implementation $\mathcal{I}$ and a set of models $\mathbb{M}$ as inputs. The algorithm executes a subset $L_F \subseteq I^*$ of the fingerprint for $\mathbb{M}$. It returns L_F and $\mathcal{M}$ if there is a model $\mathcal{M} \in \mathbb{M}$ which is the only model that satisfies $\mathcal{I} \sim_{L_F} \mathcal{M}$; otherwise, it returns L_F and *None*.

CONFQUERY Algorithm CONFQUERY requires an implementation $\mathcal{I}$ and a model $\mathcal{M}$ as inputs. The algorithm returns $L_{CQ} \subseteq I^*$ along with a Boolean outcome: *true* if $\mathcal{I} \sim_{L_{CQ}} \mathcal{M}$ and *false* otherwise.

LEARN Algorithm LEARN requires an implementation $\mathcal{I}$, a set of models $\mathbb{M}$ and $L_F \subseteq I^*$ as inputs. The algorithm returns a model $\mathcal{M}$ and $L_L \subseteq I^*$ such that $\mathcal{I} \sim_{L_F \cup L_L} \mathcal{M}$.

Under perfect teachers for CONFQUERY and LEARNINGCONFQUERY, we assume the following contracts:

CONFQUERY Algorithm CONFQUERY requires an implementation $\mathcal{I}$ and a model $\mathcal{M}$ as inputs. The algorithm returns $L_{CQ} = \emptyset$ and *true* if $\mathcal{I} \sim \mathcal{M}$ and *false* otherwise.

LEARN Algorithm LEARN requires an implementation $\mathcal{I}$ and a set of models $\mathbb{M}$ and $L_F \subseteq I^*$ as inputs. The algorithm returns a model $\mathcal{M}$ such that $\mathcal{I} \sim \mathcal{M}$ and $L_L \subseteq I^*$.

Note that the bookkeeping with L_F and L_L is not strictly necessary under a perfect teacher. Moreover, recall that our set of models only includes models that are distinct, i.e., all models are inequivalent.

Lemma 1. IDENTIFYORLEARN$_{\mathcal{C}}$ (Alg. 2) requires an implementation $\mathcal{I}$ and a set of inequivalent models $\mathbb{M}$ as inputs. After execution, a model $\mathcal{M}$ and a language $L \subseteq I^*$ are returned such that $\mathcal{I} \sim_L \mathcal{M}$ and there is at most one $\mathcal{M}' \in \mathbb{M}$ for which $\mathcal{I} \sim_L \mathcal{M}'$. Additionally, if CONFQUERY and LEARNINGCONFQUERY in $\mathcal{C}$ are perfect teachers, then $\mathcal{I} \sim \mathcal{M}$.

Proof. We follow the flow of Alg. 2 throughout the proof. First, we perform a case distinction based on the size of $|\mathbb{M}|$.

1) If $\mathbb{M} = \emptyset$, we return LEARN$(\mathcal{I}, \emptyset, \emptyset)$ on Line 1. Following the contract of LEARN, we know that a language $L_L \subseteq I^*$ and a model $\mathcal{M}$ are returned such that $\mathcal{I} \sim_{L_L} \mathcal{M}$. Additionally, since $\mathbb{M} = \emptyset$ it must hold that $\mathcal{M} \notin \mathbb{M}$. Moreover, $\mathcal{I} \sim_L \mathcal{M}$ because $L = L_L \cup L_F = L_L \cup \emptyset = L_L$. Thus, there exists a model $\mathcal{M}$ with $\mathcal{I} \sim_L \mathcal{M}$ which is not in $\mathbb{M}$. We return $\mathcal{M}$ and L_L.
2) If $\mathbb{M}$ is not empty, we perform FINGERPRINTING according to Line 2. FINGERPRINTING is guaranteed to return at most one $\mathcal{M} \in \mathbb{M}$ for which $\mathcal{I} \sim_{L_F} \mathcal{M}$ holds with $L_F \subseteq I^*$. We perform a case distinction based on whether a model is returned or not:
   a) One model $\mathcal{M}$ is returned and we enter the if-statement starting on Line 3. For this model, we know $\mathcal{I} \sim_{L_F} \mathcal{M}$ holds. Therefore, we execute CONFQUERY and obtain $L_{CQ} \subseteq I^*$ and boolean b with value *true* if $\mathcal{I} \sim_{L_{CQ}} \mathcal{M}$ and *false* otherwise. We set L_F to $L_F \cup L_{CQ}$ in Line 5 and perform a case distinction based on boolean b in Line 6.
      i) $b =$ *true*, which indicates $\mathcal{I} \sim_{L_{CQ}} \mathcal{M}$. When combining this with $\mathcal{I} \sim_{L_F} \mathcal{M}$, we derive $\mathcal{I} \sim_{L_F \cup L_{CQ}} \mathcal{M}$. We terminate the algorithm with $\mathcal{M}$, $L = L_F \cup L_{CQ}$ on Line 6. Thus, there exists exactly one $\mathcal{M} \in \mathbb{M}$ with $\mathcal{I} \sim_L \mathcal{M}$.
      ii) $b =$ *false*, which indicates $\mathcal{I} \nsim_{L_{CQ}} \mathcal{M}$. Because L_F is set to $L_F \cup L_{CQ}$, it holds that $\mathcal{I} \nsim_{L_F} \mathcal{M}$ for all $\mathcal{M}' \in \mathbb{M}$. Because we do not enter the if-branch in Line 6, we go to Line 7 and run LEARN. LEARN returns a model $\mathcal{M}$ and $L_L, L_{LCQ} \subseteq I^*$ such that $\mathcal{I} \sim_{L_F \cup L_L \cup L_{LCQ}} \mathcal{M}$. We terminate the algorithm on Line 8 with $\mathcal{M}$ and $L = L_F \cup L_L \cup L_{LCQ}$. Because $\mathcal{I} \nsim_{L_F} \mathcal{M}'$ for all $\mathcal{M}' \in \mathbb{M}$ and $\mathcal{I} \sim_{L_F \cup L_L \cup L_{LCQ}} \mathcal{M}$, it must be the case that $\mathcal{M} \notin \mathbb{M}$. Thus, there exists a model $\mathcal{M}$ with $\mathcal{I} \sim_L \mathcal{M}$ which is not in $\mathbb{M}$.
   b) *None* is returned after FINGERPRINTING, which implies that for all $\mathcal{M} \in \mathbb{M}$, $\mathcal{I} \nsim_{L_F} \mathcal{M}$. We go to Line 7 and use the reasoning from 2aii to conclude that there exists a model $\mathcal{M}$ with $\mathcal{I} \sim_L \mathcal{M}$ which is not in $\mathbb{M}$. We return $\mathcal{M}'$ and L_F.

The cases above cover all possible ways to terminate the algorithm IDENTIFYORLEARN. In all cases, we find a model $\mathcal{M}$ such that $\mathcal{I} \sim_L \mathcal{M}$. When terminating after the conformance check $\mathcal{M} \in \mathbb{M}$ holds, while terminating after learning guarantees $\mathcal{M} \notin \mathbb{M}$. Thus, in all cases, there is at most one $\mathcal{M} \in \mathbb{M}$ for which $\mathcal{I} \sim_L \mathcal{M}$ holds.

Next, we prove: if $\mathcal{C}$ contains perfect teachers CONFQUERY and LEARNINGCONFQUERY, then $\mathcal{I} \sim \mathcal{M}$.

- From the perfect teacher contracts, we know that LEARN using a perfect teacher terminates with $\mathcal{I} \sim \mathcal{M}$. Combining this information with case 1, 2aii and 2b, it follows that we terminate with $\mathcal{I} \sim \mathcal{M}'$ for some $\mathcal{M}' \notin \mathbb{M}$.
- For case 2ai, we remark that a perfect teacher for CONFQUERY terminates with $\mathcal{I} \sim \mathcal{M}$, proving that in this case there exist one $\mathcal{M} \in \mathbb{M}$ such that $\mathcal{I} \sim \mathcal{M}$.

Thus, in both cases $\mathcal{I} \sim \mathcal{M}$ holds if $\mathcal{C}$ contains perfect teachers for CONFQUERY and LEARNINGCONFQUERY. □

Theorem 1. INCREMENTALFINGERPRINTING$_{\mathcal{C}}$ (Alg. 1) requires a list of implementations $\mathbb{I}$ and a set of distinct models $\mathbb{M}_0$ as inputs. The algorithm returns $\mathbb{M}$, γ and μ such that for $\mathcal{I} \in \mathbb{I}$ there exists a $\mathcal{M} \in \mathbb{M}$ for which $\mathcal{I} \sim_{\gamma(\mathcal{I})} \mathcal{M}$ iff $\mu(\mathcal{I}) = \mathcal{M}$. Additionally, if CONFQUERY and LEARNINGCONFQUERY in $\mathcal{C}$ are perfect teachers, then $\mathcal{I} \sim \mathcal{M}$.

Proof. For each $\mathcal{I} \in \mathbb{I}$, IDENTIFYORLEARN is guaranteed to return a model $\mathcal{M}$ and $L \subseteq I^*$ such that $\mathcal{I} \sim_L \mathcal{M}$ and there is at most one such $\mathcal{M} \in \mathbb{M}$. Statement $\mu(\mathcal{I}) = \mathcal{M} \iff \mathcal{I} \sim_{\gamma(\mathcal{I})} \mathcal{M}$ holds by construction because we update $\mathbb{M}$, γ and μ according to the output of IDENTIFYORLEARN which guarantees $\mathcal{I} \sim_L \mathcal{M}$. □

Proof of Theorem 2. Let $m = |\mathbb{M}|$, $i = |\mathbb{I}|$. Assume that a perfect teacher is available and that all $\mathcal{M} \in \mathbb{M}$ have at most n states, k inputs and counterexamples of length at most l. $RL^{\#}$ learns the the correct set of models $\mathbb{M}$ within $\mathcal{O}(i(kn^2 + n \log l))$ OQs and at most in EQs. INCREMENTALFINGERPRINTING with $SepSeq$ and $L^{\#}$ learns the correct set of models $\mathbb{M}$ within $\mathcal{O}(m(kn^2 + n \log l) + im^2)$ OQs and at most $mn + i$ EQs.

Proof. First, we consider repeated application of $L^{\#}$. Because there are i implementations and learning one model using $L^{\#}$ requires $\mathcal{O}(kn^2 + n \log l)$ OQs and at most n CQs, it trivially holds that learning all implementations requires $\mathcal{O}(i(kn^2 + n \log l))$ OQs and in CQs.
Now, we consider INCREMENTALFINGERPRINTING. Fingerprinting one implementation requires at most m^2 OQs since the fingerprint contains at most one sequence to separate each pair of models. Fingerprinting all implementations, therefore, requires at most im^2 OQs. Under the perfect teacher assumption, we always learn correct models of implementations and only need to learn them if there is no $\mathcal{M} \in \mathbb{M}$ that is equivalent to the implementation. Therefore, we only need to learn $m = |\mathbb{M}|$ models. Thus, the output query complexity is $\mathcal{O}(m(kn^2 + n \log l) + im^2)$.

When considering the maximum number of CQs, we note that we only learn a model m times, each time requiring at most n CQs. Additionally, the CQ after fingerprinting occurs at most i times. Combining these results, we find that at most $mn + i$ CQs are required. □

Analogous Proof for AL$^{\#}$. First, we note that the complexity of $AL^{\#}$ is $\mathcal{O}(kn^2 + kno + no^2 + n \log l)$ where o is the number of equivalence classes over all reference models. In our case, o is at most mn as there are m models in $\mathbb{M}$ of each at most n states. Therefore, the complexity for $AL^{\#}$ is $\mathcal{O}(kn^2 + kmn^2 + n^3m^2 + n \log l)$.

Let $m = |\mathbb{M}|$, $i = |\mathbb{I}|$. Assume that a perfect teacher is available and that all $\mathcal{M} \in \mathbb{M}$ have at most n states, k inputs and counterexamples of length at most l. Repeated $AL^{\#}$ learns the the correct set of models $\mathbb{M}$ within $\mathcal{O}(i(kn^2 + kmn^2 + n^3m^2) + mn \log l)$ OQs and at most $mn + i - m$ CQs. INCREMENTALFINGERPRINTING with $SepSeq$ and $L^{\#}$ learns the correct set of models $\mathbb{M}$ within $\mathcal{O}(m(kn^2 + kmn^2 + n^3m^2 + n \log l) + im^2)$ OQs and at most $mn + i$ CQs.

Proof. First, we consider repeated application of $AL^{\#}$. Because we have a perfect teacher, we know that we have to learn a model m times and rebuild it $i - m$ times. Rebuilding may take $\mathcal{O}(kn^2 + kmn^2 + n^3m^2)$ OQs but is guaranteed to terminate with all required states (see Thm. 4.8 from [20]) and thus does not require counterexample processing. Therefore, the output query complexity is $\mathcal{O}(i(kn^2 + kmn^2 + n^3m^2) + mn \log l)$. For each $\mathcal{M}$, we may need up to n CQs and for each $\mathcal{I}$ that already has a learnt model in $\mathbb{M}$, we only perform a final CQ as rebuilding leads to the first hypothesis being correct. Thus, at most $mn + i - m$ CQs are required.
Now, we consider INCREMENTALFINGERPRINTING with $AL^{\#}$. Fingerprinting one implementation requires at most m^2 OQs since the fingerprint contains at most one sequence to separate each pair of models. Fingerprinting all implementations, therefore, requires at most im^2 OQs. Under the perfect teacher assumption, we always learn correct models of implementations and only need to learn them if there is no $\mathcal{M} \in \mathbb{M}$ that is equivalent to the implementation. Therefore, we only need to learn $m = |\mathbb{M}|$ models. Thus, the output query complexity is $\mathcal{O}(m(kn^2 + kmn^2 + n^3m^2 + n \log l) + im^2)$.

When considering the maximum number of output queries, we note that we only learn a model m times, each time requiring at most n CQs. Additionally, the CQ after fingerprinting occurs at most i times. Combining these results, we find that at most $mn + i$ CQs are required. □

Proof of Theorem 3. Let $\mathbb{I}$ be a list of implementations and $\mathbb{M}_0$ a set of inequivalent models such that $\mathbb{I} \subseteq \mathbb{M}_0$. Executing INCREMENTALFINGERPRINTING$_{\mathcal{C}}$ with initial references $\mathbb{M}_0$ and implementations $\mathbb{I}$ returns $\mathbb{M}$ and μ such that $\mathbb{M}_0 = \mathbb{M}$ and for $\mathcal{I} \in \mathbb{I}$, $\mu(\mathcal{I}) = \mathcal{M}$ iff $\mathcal{I} \sim \mathcal{M}$ for some $\mathcal{M} \in \mathbb{M}$.

Proof. For each $\mathcal{I} \in \mathbb{I}$, there must be a $\mathcal{M} \in \mathbb{M}_0$ such that $\mathcal{I} \sim \mathcal{M}$ because of assumption $\mathbb{I} \subseteq \mathbb{M}_0$. Additionally, we know that FINGERPRINTING returns at most one model and since there cannot be a separating sequence that shows that $\mathcal{I} \nsim \mathcal{M}$, the returned model must be $\mathcal{M}$. Next, a CQ is performed for $\mathcal{I} \sim \mathcal{M}$ and this must lead to output *true* as there does not exist a counterexample. Thus, exactly $\mathcal{M} \in \mathbb{M}_0$ with $\mathcal{M} \sim \mathcal{I}$ is returned for implementation $\mathcal{I}$. Additionally, $\mathbb{M} = \mathbb{M}_0$ because for each $\mathcal{I} \in \mathbb{I}$ there is a $\mathcal{M} \in \mathbb{M}_0$ that is equivalent to $\mathcal{I}$. Therefore, we never learn a new model that has to be added to $\mathbb{M}_0$. □

B. Benchmark Details

Model	States	Inputs	Copies	Learn Symbols	EQ Symbols
mbedtls/1.1.8/TLS11	6	11	20	1065	9818
mbedtls/1.2.7/TLS12	6	11	42	1074	9323
mbedtls/1.3.3/TLS12	6	11	18	1055	12244
mbedtls/2.1.5/TLS12	6	11	81	1028	12218
mbedtls/2.7.8/TLS12	6	11	72	603	2681
mbedtls/3.0.0p1/TLS12	8	11	99	936	5875
openssl/0.9.7d/TLS10	14	11	5	1629	1796151
openssl/0.9.8a/TLS10	14	11	33	1284	1833540
openssl/0.9.8l/TLS10	10	11	1	801	5359
openssl/0.9.8y/TLS10	14	11	13	1560	130666
openssl/0.9.8zh/TLS10	11	11	15	1109	71365
openssl/1.0.0g/TLS10	11	11	4	993	20260
openssl/1.0.0m/TLS10	13	11	5	1306	133092
openssl/1.0.0p/TLS10	11	11	5	1065	68804
openssl/1.0.1/TLS10	14	11	11	1548	167539
openssl/1.0.1/TLS12	13	11	8	1390	108994
openssl/1.0.1d/TLS12	13	11	2	1315	174942
openssl/1.0.1r/TLS12	11	11	33	1065	52090
openssl/1.0.2d/TLS12	10	11	39	932	124984
openssl/1.0.2m/TLS12	8	11	27	787	127132
openssl/1.1.0a/TLS12	8	11	39	629	106845
openssl/1.1.1g/TLS12	8	11	24	614	101873

TABLE II: Details on TLS models. All models originate form [13]. Learned with $L^{\#}$ and RandomWp set to stop early when the correct amount of states is reached, averaged over 30 seeds. For more information on the protocols, we refer the reader to https://github.com/Mbed-TLS/mbedtls and https://github.com/openssl/openssl.

Model	States	Inputs	Copies	Learn Symbols	EQ Symbols
DropBearOrig	17	13	5	3896	17705
DropBear20	20	13	5	4208	11520
DropBear22	22	13	5	3828	13488
DropBear24	24	13	5	4157	7604
DropBear26	26	13	5	4523	4901
OpenSSH26	26	13	5	5530	351107
OpenSSHOrig	27	13	5	6294	230825
OpenSSH28	28	13	5	5362	666117
OpenSSH29	29	13	5	6597	719682
OpenSSH31	31	13	5	5911	746095
OpenSSH34	34	13	5	6630	346615
OpenSSH36	36	13	5	7691	688311
BitVise39	39	13	5	10594	178497
BitVise45	45	13	5	19569	156627
BitVise47	47	13	5	16148	143216
BitVise54	54	13	5	23330	696630
BitVise57	57	13	5	22720	1165158
BitVise59	59	13	5	26440	1586899
BitVise63	63	13	5	27342	929043
BitViseOrig	66	13	5	31236	2681762

TABLE III: Details on SSH models. Model names ending in 'Orig' refer to models from [18] and available on https://automata.cs.ru.nl/, the other models are obtained by manually mutating the base models. Learned with $L^{\#}$ and RandomWp set to stop early when the correct amount of states is reached, averaged over 30 seeds.

Model	States	Inputs	Copies	Learn Symbols	EQ Symbols	url
HiveMQ	7	20	5	2478	1000	https://docs.hivemq.com/
emqx	24	20	5	15106	5079	https://www.emqx.com/en
Mosquitto	32	20	5	20900	7066	https://mosquitto.org/
VerneMQ	19	20	5	10735	3222	https://vernemq.com/
ejabberd	53	20	5	67734	10217	https://www.ejabberd.im/
mochi	8	20	5	3480	1165	https://github.com/mochi-mqtt/server

TABLE IV: Details on MQTT Learning. Learned with $L^{\#}$ and StatePrefixOracle with 10 walks per state and walk length 12.

Model	States	Learn Symbols	EQ Symbols
cc2652r1_new	6	4531	2533
cyble-416045-02_new	2	843.3	1695.3
explorable	2	654.3	1743.3

TABLE V: Details on new BLE models. Each model was learned *online* with $L^{\#}$ and RandomWp with a maximum of 100 test queries per EQ.

Model	States	Inputs	Copies	Learn Symbols	EQ Symbols
CYW43455	16	7	5	1710	941
cc2650	4	7	5	284	96
cc2652r1_new	6	7	5	568	253
cc2652r1_old	4	7	5	263	77
cyble-416045-02	2	7	5	108	27
explorable	2	7	5	101	31
nRF52832	5	7	5	260	312
tesla_model_3	10	7	5	813	456

TABLE VI: Details on BLE Learning from dot models (offline). Learned with $L^{\#}$ and RandomWp set to stop early when the correct amount of states is reached, averaged over 30 seeds.

Model	States	Inputs	Copies	Learn Symbols	EQ Symbols
M1	5	32	5	1675	833
M2	8	32	5	3726	6620
M3	8	32	5	3730	8289
M4	7	32	5	3180	3062
M5	8	32	5	4073	11112
M6	7	32	5	3188	6067

TABLE VII: Details on BLEDiff Learning from dot models provided by the authors of [33], subsequently made input complete and minimized. Learned with $L^{\#}$ and RandomWp set to stop early when the correct amount of states is reached, averaged over 30 seeds.

C. Motivational Experiment Results

Algorithm	|Initial Models|	Correct Models	Misclassifications	No Matches	Fingerprint Symbols	Conformance Symbols	Learn Symbols	Total Symbols
$RL^{\#}$: RandomWp100	0	143.7 - 24.1%	452.3 - 75.9%	0 - 0.0% 0	0	0	2606357	2606357
$RL^{\#}$: RandomWp500	0	401.6 - 67.4%	194.4 - 32.6%	0 - 0.0% 0	0	0	14266503	14266503
Fingerprint: SepSeq	11	309.4 - 51.9%	273.6 - 45.9%	13.0 - 2.2%	7275	0	0	7275
INFERNAL	11	591.7 - 99.3%	4.3 - 0.7%	0.0 - 0.0%	8415	3399028	100276	3507719
Fingerprint: SepSeq	21	568.9 - 95.5%	27.1 - 4.5%	0.0 - 0.0%	10388	0	0	10388
INFERNAL	21	595.8 - 100.0%	0.2 - 0.0%	0.0 - 0.0%	8270	3485626	7210	3501105
Fingerprint: SepSeq	22	596.0 - 100.0%	0.0 - 0.0%	0.0 - 0.0%	10371	0	0	10371
INFERNAL	22	596.0 - 100.0%	0.0 - 0.0%	0.0 - 0.0%	8287	3491649	0	3499936

TABLE VIII: Summarized results for the motivational experiment discussed in Section II. Conformance symbols are only the symbols used during the CQ directly after fingerprinting, the conformance symbols during learning are included in the learning symbols.

D. Experimental Evaluation Results

Benchmark	Algorithm	Fingerprint Symbols	Learn Symbols	Total Symbols
BLE	$RL^{\#}$	0	23530	23530
BLE	$RAL^{\#}$	0	23612	23612
BLE	INFERNAL	154	4843	4997
BLEDiff	$RL^{\#}$	0	127655	127655
BLEDiff	$RAL^{\#}$	0	125310	125310
BLEDiff	INFERNAL	169	25244	25413
MQTT	$RL^{\#}$	0	494490	494490
MQTT	$RAL^{\#}$	0	471005	471005
MQTT	INFERNAL	162	101031	101193
SSH	$RL^{\#}$	0	1433586	1433586
SSH	$RAL^{\#}$	0	1424811	1424811
SSH	INFERNAL	1476	319697	321173
TLS	$RL^{\#}$	0	892491	892491
TLS	$RAL^{\#}$	0	957179	957179
TLS	INFERNAL	8760	39677	48437

TABLE IX: Summarized results for Experiment 1a.

Benchmark	Algorithm	Budget	Correct Models
BLE	$RL^{\#}$	2500	35.6 - 89.0%
BLE	$RAL^{\#}$	2500	39.4 - 98.4%
BLE	INFERNAL	2500	39.2 - 98.1%
BLEDiff	$RL^{\#}$	10000	17.9 - 59.8%
BLEDiff	$RAL^{\#}$	10000	27.1 - 90.2%
BLEDiff	INFERNAL	10000	27.9 - 92.8%
MQTT	$RL^{\#}$	100000	26.6 - 88.5%
MQTT	$RAL^{\#}$	100000	29.2 - 97.5%
MQTT	INFERNAL	100000	29.1 - 97.2%
SSH	$RL^{\#}$	750000	77.2 - 77.2%
SSH	$RAL^{\#}$	750000	91.8 - 91.8%
SSH	INFERNAL	750000	93.3 - 93.3%
TLS	$RL^{\#}$	100000	465.6 - 78.1%
TLS	$RAL^{\#}$	100000	569.2 - 95.5%
TLS	INFERNAL	100000	584.1 - 98.0%

TABLE X: Summarized results for Experiment 1b.

Benchmark	Algorithm	Correct Models	Fingerprinting Symbols	CQ Symbols	Learning Symbols	Total
BLE	$RL^{\#}$	39.9 - 99.8%	0	0	155644	155644
BLE	$RAL^{\#}$	40.0 - 100.0%	0	0	154085	154085
BLE	INFERNAL	40.0 - 100.0%	176	109314	31154	140644
BLEDiff	$RL^{\#}$	11.2 - 37.2%	0	0	157645	157645
BLEDiff	$RAL^{\#}$	24.9 - 82.8%	0	0	255977	255977
BLEDiff	INFERNAL	24.1 - 80.5%	265	112316	67115	179695
MQTT	$RL^{\#}$	25.7 - 85.7%	0	0	1221350	1221350
MQTT	$RAL^{\#}$	29.8 - 99.3%	0	0	1086956	1086956
MQTT	INFERNAL	29.6 - 98.5%	171	499849	263114	763134
SSH	$RL^{\#}$	22.9 - 22.9%	0	0	3480605	3480605
SSH	$RAL^{\#}$	70.6 - 70.6%	0	0	4780912	4780912
SSH	INFERNAL	64.5 - 64.5%	1652	2496480	1579190	4077322
TLS	$RL^{\#}$	143.3 - 24.1%	0	0	2595294	2595294
TLS	$RAL^{\#}$	546.5 - 91.7%	0	0	4106272	4106272
TLS	INFERNAL	545.8 - 91.6%	9760	3204005	198114	3411879

TABLE XI: Summarized results for Experiment 1c.

Benchmark	Components	Correct Models	Fingerprint Symbols	CQ Symbols	Learn Symbols	Total Symbols
BLE	ADG - RandomWord1000 - $RAL^{\#}$	40.0 - 100.0%	176	672146	173516	845838
BLE	ADG - RandomWord1000 - $RL^{\#}$	40.0 - 100.0%	176	672009	173161	845346
BLE	ADG - RandomWp100 - $RAL^{\#}$	40.0 - 100.0%	176	109314	31154	140644
BLE	ADG - RandomWp100 - $RL^{\#}$	40.0 - 100.0%	176	109300	30970	140446
BLE	ADG - WpK - $RAL^{\#}$	40.0 - 100.0%	176	1178870	294846	1473892
BLE	ADG - WpK - $RL^{\#}$	40.0 - 100.0%	176	1178870	294661	1473707
BLE	SepSeq - RandomWord1000 - $RAL^{\#}$	40.0 - 100.0%	228	672434	173226	845888
BLE	SepSeq - RandomWord1000 - $RL^{\#}$	40.0 - 100.0%	228	671884	173239	845351
BLE	SepSeq - RandomWp100 - $RAL^{\#}$	40.0 - 100.0%	228	109292	31141	140661
BLE	SepSeq - RandomWp100 - $RL^{\#}$	40.0 - 100.0%	228	109252	30957	140437
BLE	SepSeq - WpK - $RAL^{\#}$	40.0 - 100.0%	228	1178872	294833	1473933
BLE	SepSeq - WpK - $RL^{\#}$	40.0 - 100.0%	228	1178872	294568	1473668
BLEDiff	ADG - RandomWord1000 - $RAL^{\#}$	8.8 - 29.2%	263	459260	217166	676689
BLEDiff	ADG - RandomWord1000 - $RL^{\#}$	5.5 - 18.3%	270	459092	222335	681697
BLEDiff	ADG - RandomWp100 - $RAL^{\#}$	24.1 - 80.5%	265	112316	67115	179695
BLEDiff	ADG - RandomWp100 - $RL^{\#}$	20.3 - 67.7%	273	98970	68232	167474
BLEDiff	ADG - WpK - $RAL^{\#}$	30.0 - 100.0%	203	104112844	26047499	130160546
BLEDiff	ADG - WpK - $RL^{\#}$	30.0 - 100.0%	203	104112844	26040358	130153405
BLEDiff	SepSeq - RandomWord1000 - $RAL^{\#}$	7.2 - 24.0%	286	472071	207137	679494
BLEDiff	SepSeq - RandomWord1000 - $RL^{\#}$	6.0 - 19.8%	301	474918	199665	674883
BLEDiff	SepSeq - RandomWp100 - $RAL^{\#}$	26.3 - 87.7%	325	121525	65367	187217
BLEDiff	SepSeq - RandomWp100 - $RL^{\#}$	22.1 - 73.8%	294	105217	65003	170514
BLEDiff	SepSeq - WpK - $RAL^{\#}$	30.0 - 100.0%	224	104112848	26047927	130160999
BLEDiff	SepSeq - WpK - $RL^{\#}$	30.0 - 100.0%	224	104112848	26042524	130155596
MQTT	ADG - RandomWord1000 - $RAL^{\#}$	23.6 - 78.7%	267	444971	423227	868466
MQTT	ADG - RandomWord1000 - $RL^{\#}$	22.0 - 73.3%	294	423153	463513	886960
MQTT	ADG - RandomWp100 - $RAL^{\#}$	29.6 - 98.5%	171	499849	263114	763134
MQTT	ADG - RandomWp100 - $RL^{\#}$	29.1 - 97.0%	185	493004	287362	780552
MQTT	ADG - WpK - $RAL^{\#}$	30.0 - 100.0%	156	156252748	39304431	195557335
MQTT	ADG - WpK - $RL^{\#}$	30.0 - 100.0%	156	155831888	39179615	195011659
MQTT	SepSeq - RandomWord1000 - $RAL^{\#}$	23.5 - 78.3%	271	447660	413195	861126
MQTT	SepSeq - RandomWord1000 - $RL^{\#}$	22.2 - 74.0%	297	416989	455815	873102
MQTT	SepSeq - RandomWp100 - $RAL^{\#}$	29.4 - 98.2%	170	485930	286471	772571
MQTT	SepSeq - RandomWp100 - $RL^{\#}$	29.2 - 97.3%	175	503734	262112	766022
MQTT	SepSeq - WpK - $RAL^{\#}$	30.0 - 100.0%	156	156252748	39302856	195555760
MQTT	SepSeq - WpK - $RL^{\#}$	30.0 - 100.0%	156	155845715	39174490	195020360
SSH	ADG - RandomWord1000 - $RAL^{\#}$	22.4 - 22.4%	1420	1646509	802867	2450795
SSH	ADG - RandomWord1000 - $RL^{\#}$	10.8 - 10.8%	1671	1411471	1167240	2580381
SSH	ADG - RandomWp100 - $RAL^{\#}$	64.5 - 64.5%	1652	2496480	1579190	4077322
SSH	ADG - RandomWp100 - $RL^{\#}$	37.0 - 37.0%	1803	2076632	1943940	4022374
SSH	ADG - WpK - $RAL^{\#}$	100.0 - 100.0%	1356	202410117	51018700	253430172
SSH	ADG - WpK - $RL^{\#}$	100.0 - 100.0%	1356	203492342	51900170	255393868
SSH	SepSeq - RandomWord1000 - $RAL^{\#}$	22.0 - 22.0%	2189	1620057	872903	2495149
SSH	SepSeq - RandomWord1000 - $RL^{\#}$	10.8 - 10.8%	2511	1421740	1128794	2553044
SSH	SepSeq - RandomWp100 - $RAL^{\#}$	62.0 - 62.0%	2546	2505740	1574980	4083266
SSH	SepSeq - RandomWp100 - $RL^{\#}$	35.8 - 35.8%	2589	2052354	1922384	3977326
SSH	SepSeq - WpK - $RAL^{\#}$	100.0 - 100.0%	2086	202438306	51026351	253466743
SSH	SepSeq - WpK - $RL^{\#}$	100.0 - 100.0%	2086	202608001	51790165	254400252
TLS	ADG - RandomWord1000 - $RAL^{\#}$	353.4 - 59.3%	8824	12098388	561023	12668234
TLS	ADG - RandomWord1000 - $RL^{\#}$	304.9 - 51.1%	9808	11999406	681247	12690460
TLS	ADG - RandomWp100 - $RAL^{\#}$	545.8 - 91.6%	9760	3204005	198114	3411879
TLS	ADG - RandomWp100 - $RL^{\#}$	512.2 - 85.9%	10689	3109059	248937	3368686
TLS	ADG - WpK - $RAL^{\#}$	558.0 - 93.6%	8495	146698960	5773796	152481251
TLS	ADG - WpK - $RL^{\#}$	558.0 - 93.6%	8495	146545674	5757351	152311520
TLS	SepSeq - RandomWord1000 - $RAL^{\#}$	353.4 - 59.3%	9905	12125882	543444	12679230
TLS	SepSeq - RandomWord1000 - $RL^{\#}$	281.8 - 47.3%	10893	12012321	723714	12746928
TLS	SepSeq - RandomWp100 - $RAL^{\#}$	527.9 - 88.6%	10737	3170738	197752	3379227
TLS	SepSeq - RandomWp100 - $RL^{\#}$	448.4 - 75.2%	11312	2977946	231426	3220684
TLS	SepSeq - WpK - $RAL^{\#}$	558.0 - 93.6%	9898	146700891	5773722	152484511
TLS	SepSeq - WpK - $RL^{\#}$	558.0 - 93.6%	9898	146547624	5750312	152307834

TABLE XII: Summarized results for Experiment 2.

Benchmark	Fingerprinting - Learning	FCQ Wrong	LCQ Wrong	Correct Models	Entering FCQ	End with FCQ	End with LCQ	Fingerprint Symbols	Learn Symbols	Total Symbols
BLE	RandomWp100 - RandomWp25	0.0 - 0.0%	0.5 - 1.2%	39.5 - 98.8%	34	32	8	153	12114	119862
BLE	RandomWp100 - RandomWp50	0.0 - 0.0%	0.25 - 0.6%	39.8 - 99.4%	34	32	8	150	18516	127475
BLE	RandomWp100 - RandomWp100	0.0 - 0.0%	0.05 - 0.1%	40.0 - 99.9%	34	32	8	150	31135	140270
BLEDiff	RandomWp100 - RandomWp25	5.0 - 16.7%	7.3 - 24.3%	17.7 - 59.0%	26	17	13	252	40607	127737
BLEDiff	RandomWp100 - RandomWp50	4.3 - 14.3%	4.95 - 16.5%	20.8 - 69.2%	26	20	10	220	49448	148955
BLEDiff	RandomWp100 - RandomWp100	2.7 - 9.0%	3.0 - 10.0%	24.3 - 81.0%	27	21	9	211	67635	180710
MQTT	RandomWp100 - RandomWp25	0.05 - 0.2%	4.8 - 16.0%	25.1 - 83.8%	27	19	11	262	264869	609911
MQTT	RandomWp100 - RandomWp50	0.0 - 0.0%	1.3 - 4.3%	28.7 - 95.7%	26	23	7	198	234359	704225
MQTT	RandomWp100 - RandomWp100	0.0 - 0.0%	0.35 - 1.2%	29.6 - 98.8%	27	24	6	171	267846	767069
SSH	RandomWp100 - RandomWp25	16.3 - 16.3%	27.85 - 27.9%	55.9 - 55.9%	93	58	42	2004	927287	2962899
SSH	RandomWp100 - RandomWp50	19.9 - 19.9%	19.75 - 19.8%	60.4 - 60.4%	93	66	34	1934	1119367	3451109
SSH	RandomWp100 - RandomWp100	21.3 - 21.3%	14.95 - 14.9%	63.8 - 63.8%	93	70	30	1799	1520484	4032741
TLS	RandomWp100 - RandomWp25	54.55 - 9.2%	11.8 - 2.0%	529.6 - 88.9%	589	565	31	10433	81330	3244653
TLS	RandomWp100 - RandomWp50	47.6 - 8.0%	8.45 - 1.4%	540.0 - 90.6%	589	568	28	9813	119449	3314805
TLS	RandomWp100 - RandomWp100	54.2 - 9.1%	7.7 - 1.3%	534.1 - 89.6%	589	569	27	9535	198440	3383042

TABLE XIII: Summarized results for Experiment 3.

E. Additional Figures and Tables

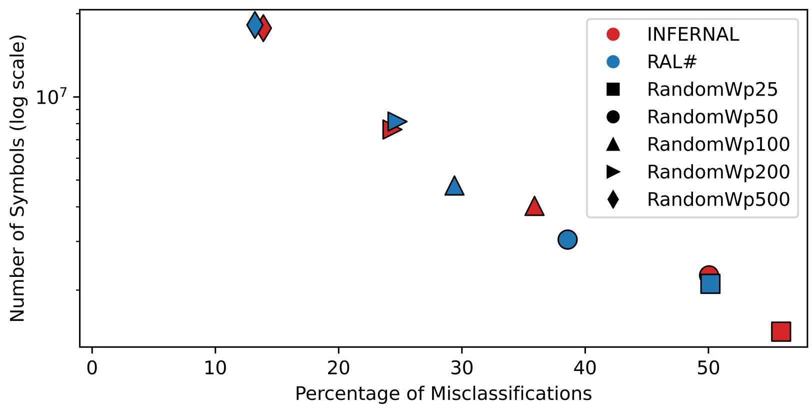


Fig. 7: Additional experiment for RQ1. Comparison of $RAL^{\#}$ and INFERNAL for SSH.

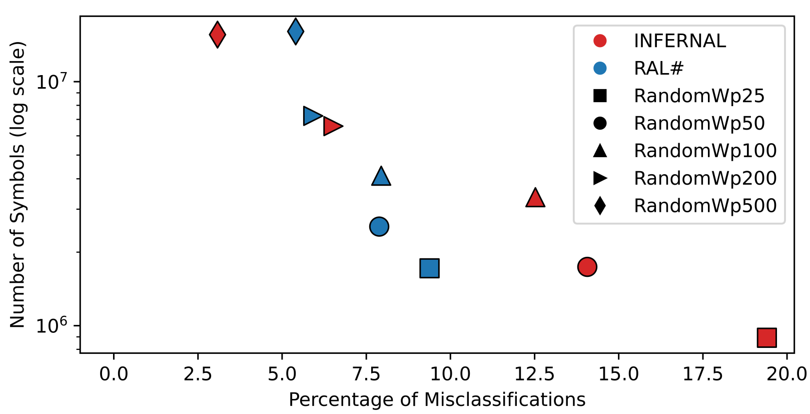


Fig. 8: Additional experiment for RQ1. Comparison of $RAL^{\#}$ and INFERNAL for TLS.

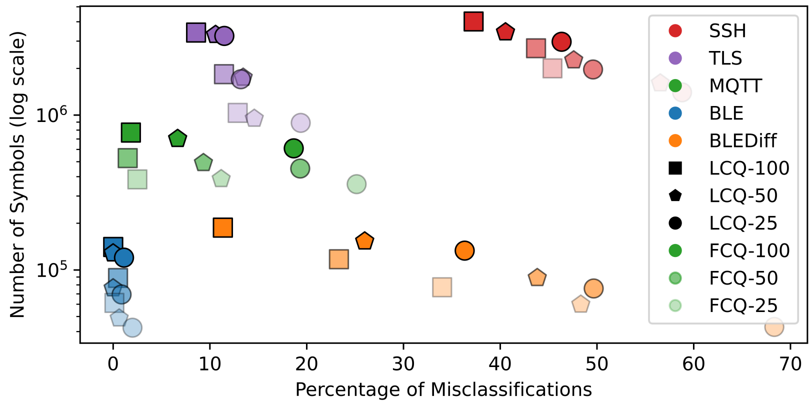


Fig. 9: Additional experiment for RQ2. Comparison of the performance of different CQs for fingerprinting and learning.

Benchmark	Fingerprinting - Learning	Correctly Learned	Fingerprinting	FCQ	Learning	Total
BLE	RandomWp25 - RandomWp25	39.2 - 98.0%	140	29367	12849	42356
BLE	RandomWp25 - RandomWp50	39.8 - 99.4%	133	30021	18564	48717
BLE	RandomWp25 - RandomWp100	40.0 - 99.9%	134	30038	31104	61276
BLE	RandomWp50 - RandomWp25	39.6 - 99.1%	137	57053	12221	69410
BLE	RandomWp50 - RandomWp50	40.0 - 100.0%	134	57550	18583	76267
BLE	RandomWp50 - RandomWp100	39.8 - 99.5%	133	57548	31129	88810
BLE	RandomWp100 - RandomWp25	39.5 - 98.9%	148	107559	12327	120033
BLE	RandomWp100 - RandomWp50	40.0 - 100.0%	134	109556	18484	128174
BLE	RandomWp100 - RandomWp100	40.0 - 100.0%	134	109331	31275	140739
BLEDiff	RandomWp25 - RandomWp25	9.5 - 31.7%	192	17223	25423	42838
BLEDiff	RandomWp25 - RandomWp50	15.5 - 51.7%	184	21878	37776	59839
BLEDiff	RandomWp25 - RandomWp100	19.8 - 66.0%	174	26458	50697	77329
BLEDiff	RandomWp50 - RandomWp25	15.1 - 50.3%	229	42122	33543	75893
BLEDiff	RandomWp50 - RandomWp50	16.9 - 56.2%	220	43853	44783	88855
BLEDiff	RandomWp50 - RandomWp100	23.0 - 76.7%	210	56299	60793	117302
BLEDiff	RandomWp100 - RandomWp25	19.1 - 63.7%	276	90770	42153	133198
BLEDiff	RandomWp100 - RandomWp50	22.2 - 74.0%	233	103560	49281	153073
BLEDiff	RandomWp100 - RandomWp100	26.6 - 88.7%	241	121551	65867	187659
MQTT	RandomWp25 - RandomWp25	22.4 - 74.8%	296	82515	275074	357885
MQTT	RandomWp25 - RandomWp50	26.6 - 88.8%	213	115601	270970	386784
MQTT	RandomWp25 - RandomWp100	29.2 - 97.5%	164	128308	256041	384513
MQTT	RandomWp50 - RandomWp25	24.2 - 80.7%	297	174681	276447	451425
MQTT	RandomWp50 - RandomWp50	27.2 - 90.7%	223	222894	267886	491003
MQTT	RandomWp50 - RandomWp100	29.6 - 98.5%	160	251157	276000	527317
MQTT	RandomWp100 - RandomWp25	24.4 - 81.3%	285	322632	286519	609437
MQTT	RandomWp100 - RandomWp50	28.0 - 93.3%	215	454233	248219	702667
MQTT	RandomWp100 - RandomWp100	29.4 - 98.2%	165	494615	276490	771270
SSH	RandomWp25 - RandomWp25	41.2 - 41.2%	1868	570473	829245	1401586
SSH	RandomWp25 - RandomWp50	43.5 - 43.5%	1792	628178	975705	1605675
SSH	RandomWp25 - RandomWp100	54.6 - 54.6%	1736	667866	1333819	2003422
SSH	RandomWp50 - RandomWp25	50.4 - 50.4%	1914	1104158	862602	1968675
SSH	RandomWp50 - RandomWp50	52.4 - 52.4%	1879	1216912	1041720	2260511
SSH	RandomWp50 - RandomWp100	56.3 - 56.3%	1770	1294165	1405565	2701500
SSH	RandomWp100 - RandomWp25	53.6 - 53.6%	2060	2062388	907471	2971920
SSH	RandomWp100 - RandomWp50	59.5 - 59.5%	1955	2360563	1066350	3428868
SSH	RandomWp100 - RandomWp100	62.8 - 62.8%	1855	2529312	1490268	4021435
TLS	RandomWp25 - RandomWp25	480.4 - 80.6%	8542	799210	82231	889983
TLS	RandomWp25 - RandomWp50	509.1 - 85.4%	8337	818179	122530	949046
TLS	RandomWp25 - RandomWp100	519.3 - 87.1%	8188	829912	194812	1032913
TLS	RandomWp50 - RandomWp25	517.4 - 86.8%	9083	1609942	83533	1702559
TLS	RandomWp50 - RandomWp50	515.9 - 86.6%	8739	1614072	123948	1746758
TLS	RandomWp50 - RandomWp100	527.8 - 88.5%	8667	1630764	198207	1837637
TLS	RandomWp100 - RandomWp25	527.5 - 88.5%	8760	3153798	79750	3242308
TLS	RandomWp100 - RandomWp50	532.9 - 89.4%	8940	3166015	123417	3298372
TLS	RandomWp100 - RandomWp100	545.0 - 91.4%	8570	3203933	197418	3409921

TABLE XIV: Additional details for Fig. 9.

Algorithm	Copies	Correct Models	Fingerprint Symbols	CQ Symbols	Learn Symbols	Total Symbols
$RL^{\#}$	0	4.8 - 80.0%	0	0	245147	245147
$RAL^{\#}$	0	5.6 - 93.3%	0	0	233369	233369
INFERNAL	0	5.6 - 93.3%	15	89	238834	238938
$RL^{\#}$	2	15.8 - 87.8%	0	0	732097	732097
$RAL^{\#}$	2	17.4 - 96.7%	0	0	658528	658528
INFERNAL	2	16.8 - 93.3%	101	222131	308863	531096
$RL^{\#}$	4	26.6 - 88.7%	0	0	1221398	1221398
$RAL^{\#}$	4	30.0 - 100.0%	0	0	1092100	1092100
INFERNAL	4	29.2 - 97.3%	215	493824	291786	785824
$RL^{\#}$	6	36.6 - 87.1%	0	0	1704670	1704670
$RAL^{\#}$	6	41.6 - 99.0%	0	0	1516457	1516457
INFERNAL	6	42.0 - 100.0%	243	768878	238334	1007455
$RL^{\#}$	8	46.4 - 85.9%	0	0	2173474	2173474
$RAL^{\#}$	8	53.4 - 98.9%	0	0	1943352	1943352
INFERNAL	8	53.6 - 99.3%	358	1023326	238891	1262575
$RL^{\#}$	10	58.0 - 87.9%	0	0	2680479	2680479
$RAL^{\#}$	10	64.6 - 97.9%	0	0	2367425	2367425
INFERNAL	10	65.4 - 99.1%	492	1270763	256849	1528103

TABLE XV: Results for MQTT when using a different number of copies per unique model. All algorithms use RandomWp100 and each experiment is repeated 5 times.