Writing a Thesis:
Guidelines for Writing a Master’s Thesis in Computer Science

Keith Andrews

Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science

Keith Andrews B.Sc.

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's Degree Programme: Computer Science

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dr. Keith Andrews
Institute of Interactive Systems and Data Science (ISDS)

Graz, 10 Nov 2021

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The document uploaded to TUGRAZonline is identical to the present thesis.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Dokument ist mit der vorliegenden Arbeit identisch.

Date/Datum

Signature/Unterschrift

Abstract

Writing a thesis is a vast, overwhelming endeavour. There are many obstacles and false dawns along the way. This thesis takes a fresh look at the process and addresses new ways of accomplishing this daunting goal.

The abstract should concisely describe what the thesis is about and what its contributions to the field are (what is new). Market your contributions to your readership. Also make sure you mention all relevant keywords in the abstract, since many readers read *only* the abstract and many search engines index *only* title and abstract.

This thesis explores the issues concerning the clear structuring and the academic criteria for a thesis and presents numerous novel insights. Special attention is paid to the use of clear and simple English for an international audience, and advice is given as to the use of technical aids to thesis production. Two appendices provide specific local guidance.

Contents

Contents i

List of Figures iii

List of Tables v

List of Listings vii

Acknowledgements ix

Credits xi

1 Introduction 1

1.1 Motivation . . . 1

1.2 Research Problems and Goals . . . 1

1.3 Structure . . . 2

2 Related 3

3 Preliminaries 5

3.1 Mealy Machines . . . 5

3.2 Automata Learning . . . 5

3.2.1 L^* . . . 6

3.2.2 KV . . . 7

3.3 Fuzzing . . . 8

3.4 IPsec . . . 8

4 Learning 11

5 Evaluation 13

6 Conclusion 15

Bibliography 17

List of Figures

3.1 IKEv1 between two parties 9

List of Tables

List of Listings

3.1 L^* algorithm 6
3.2 *KV* algorithm 8
3.3 IKE Keying 10

Acknowledgements

I am indebted to my colleagues at the ISDS and the Know-Center who have provided invaluable help and feedback during the course of my work. I especially wish to thank my advisor, Keith Andrews, for his immediate attention to my questions and endless hours of toil in correcting draft versions of this thesis.

Special mention goes to Christian Gütl, Irene Isser, and Josef Moser for their help in translating the thesis abstract into German and to Bernhard Zwantschko and Didi Freismuth for ample supplies of coffee. Please remember to replace this tongue-in-cheek acknowledgements section with your own version!

Last but not least, without the support and understanding of my wife, pleasant hours with my girlfriend, and the tolerance of my friends, this thesis would not have been possible.

Keith Andrews

Graz, Austria, 10 Nov 2021

Credits

I would like to thank the following individuals and organisations for permission to use their material:

- The thesis was written using Keith Andrews' skeleton thesis [Andrews 2021].

Chapter 1

Introduction

1.1 Motivation

Virtual Private Networks (VPNs) are used to allow secure communication over an insecure channel. They function by creating a secure encrypted tunnel through which users can send their data. Example use cases include additional privacy from prying eyes such as Internet Server Providers, access to region-locked online content and secure remote access to company networks. The importance of VPN software has increased dramatically since the beginning of the COVID-19 pandemic due to the influx of people working from home [Abhijith and Senthilvadivu 2020]. This makes finding vulnerabilities in VPN software more critical than ever. IPsec is a popular VPN protocol and most commonly uses the Internet Key Exchange (IKE) protocol to share authenticated keying material between involved parties. Therefore, IKE and IPsec are sometimes used interchangeably. We will stick to the official nomenclature of using IPsec for the full protocol and IKE for the key exchange only. IKE has two versions, IKEv1 and IKEv2, with IKEv2 being the newer and recommended version [Barker et al. no date]. However, despite IKEv2 supposedly replacing its predecessor, IKEv1, sometimes also called Cisco IPsec, is still in widespread use today. This is reflected by the company AVM to this day only offering IKEv1 support for their popular FRITZ!Box routers [GmbH 2022]. Additionally, IKEv1 is also used for the L2TP/IPsec, one of the most popular VPN protocols according to NordVPN [Ferguson and Schneier 2021]. The widespread usage of IPsec-IKEv1, combined with its relative age and many options makes it an interesting target for security testing.

1.2 Research Problems and Goals

State machines of protocol implementations are useful tools in state-of-the-art software testing. They have, e.g., been used to detect specific software implementations, or to generate test cases automatically [Pferscher and Aichernig 2021; Pferscher and Aichernig 2022]. Mealy machines are a type of state machine that can be used to describe the behavior a system when faced with external input. Often we are interested in testing software without knowing its exact inner workings. We call these systems black-box systems. However, despite lacking information on the inner structure of a black-box system, the state machine of the system can still be extracted. One method of generating the state machine of such a system is to use active automata learning. A notable example of an active automata learning algorithm is the L^* algorithm by Angluin [Angluin 1987a]. In L^*, a learner queries the System under Learning (SUL) and constructs an automaton describing the behavior of the SUL through its responses. This automaton is then compared with the SUL, adapting it if they show different behaviors. The resulting automaton then fully describes the behavior of the SUL.

By combining automata learning with fuzzing or similar software testing techniques, network protocols can be extensively and automatically tested without requiring access to their source code. Guo et al. Guo et al. [2019] tested IPsec-IKEv2 using automata learning and model checking, however so far, no studies have focused on IKEv1 in the context of automata learning. Therefore our goal was to black-box test the IPsec-IKEv1 protocol using automata learning in combination with automata-based fuzzing. We used the active automata learning framework AALpy Muškardin et al. [2022] with a custom mapper to learn the state machines of various IPsec-IKEv1 server implementations. We then further utilized the learned models for fuzzing and fingerprinting.

1.3 Structure

This thesis is structured as follows. Chapter 2 gives an overview of related and relevant literature. Chapter 3 introduces necessary background knowledge, covering the IPsec-IKEv1 protocol, Mealy machines, automata learning and fuzzing. Our learning setup, custom mapper and fuzzing methodology are presented in chapter 4. In chapter 5 we present and analyze the learned models and the results of the fuzzing tests. Finally we summarize the thesis in chapter 6 and discuss future work.

Chapter 2

Related

The aim of this chapter is to give a brief overview of related work, focusing mainly on automata learning and testing of secure communication protocols. 1987 The concept of learning through the means of membership and equivalence queries was introduced in 1987 by Angluin [Angluin 1987a]. Angluin presented an algorithm for learning regular languages from queries and counterexamples, called L^*. In it, a student questions a teacher and constructs a hypothesis based on its responses. The hypothesis is then tested through equivalence queries which check if the hypothesis correctly matches the regular language being learned. While the L* algorithm was originally designed to learn deterministic finite automata (dfa), it can be simply extended to work for Mealy machines by making use of the similarities between dfa and Mealy machines, as shown by Steffen et al. [Steffen et al. 2011]. Over time, many related and improved algorithms were published, such as the one proposed by Rivest and Schapire in 1993 in which homing sequences were used to infer finite automata [Rivest and Schapire 1993]. Another, more recent algorithm came in the form of a redundancy-free active automata learning approach titled TTT by Isberner et al [Isberner et al. 2014]. In this algorithm, essential data is stored in three tree data structures, stripping away unessential information.

Model learning network protocols for the purpose of testing is a more recent development, with models of protocols like SSH Fiterău-Broştean et al. 2017, or TCP Fiterău-Broştean et al. [2016] being learned and used for model checking. Both Novickis et al. Novickis et al. [2016] and Daniel et al. Daniel et al. [2018] learned models of the related OpenVPN protocol and used the learned models to perform protocol fuzzing. In a work by Pferscher and Aichernig Pferscher and Aichernig [2021], learned models were used to fingerprint Bluetooth Low Energy devices (BLE), showing yet another possible use case of automate learning. Guo et al. Guo et al. 2019 used automata learning to learn and test the IPsec-IKEv2 protocol. They used the LearnLib [1] library for automata learning and performed model checking of the protocol, using the learned state machine. In contrast, our work focuses on the IPsec-IKEv1 protocol, the predecessor of IPsec-IKEv2, which, to the best of our knowledge, has not yet been tested with methods utilizing automata learning. The protocols differ greatly on a packet level, with IKEv1 needing more than twice the amount of packets to establish a connection than IKEv2. Additionally we used the AALpy [2] library for automata learning and focused on fuzzing and fingerprinting as opposed to model checking.

[1] https://learnlib.de/
[2] https://github.com/DES-Lab/AALpy

Chapter 3

Preliminaries

3.1 Mealy Machines

Mealy machines are finite state machines where each output transition is defined by the current state and an input. More formally, a Mealy machine is defined as a 6-tuple $M = \{S, S_0, \Sigma, \Lambda, T, G\}$, where S is a finite set of states, $S_0 \in S$ is the initial state, Σ is a finite set called the input alphabet, Λ is a finite set called the output alphabet, T is the transition function $G : S \times \Sigma \rightarrow \Lambda$ which maps a state and an element of the input alphabet to another state in S and G is the output function $T : S \times \Sigma \rightarrow S$ which maps a state-input alphabet pair to an element of the output alphabet Λ. We use Mealy machines to model the state of learned IPsec implementations.

3.2 Automata Learning

Automata learning refers to methods of learning the state model, or automaton, of a system through an algorithm or process. We differentiate between active and passive automata learning. In passive automata learning (PAL), models are learned based on a given data set describing the behavior of the SUL, e.g. log files. In contrast, in active automata learning (AAL) the SUL is queried directly. In this paper, we will focus on AAL and will, moving on, be referring to it as automata learning or AAL interchangeably.

One of the most influential AAL algorithms was introduced in 1987 through a paper by Dana Angluin, titled "Learning regular sets from queries and counterexamples" [Angluin 1987b]. In this seminal paper, Angluin introduced the L^* algorithm, variants of which are still used for learning deterministic automata to this day, for example by the AAL python library AALpy [Muškardin et al. 2022]. While the original L^* algorithm was designed to learn deterministic finite automata (DFA), the algorithm can be extended to learn Mealy machines [Niese 2003]. While many modern implementations, including AALpy use improved versions of L^*, fundamentally they still resemble the original algorithm by Angluin. The base L^* algorithm is briefly explained below. TODO: does AALpy version use homing sequences? –> Rivest1993Inference

Another popular algorithm in the field of AAL is the KV algorithm by Kearn's and Vazirani [Kearns and Vazirani 1994]. Published later than L^*, it boasts a more compact method of representing learned data called a classification tree. This, on average, leads to the KV algorithm requiring less membership queries than L^* to learn a system. Especially for learning internet protocols and other systems where communication with the SUL can be very time consuming, this can result in a significant performance increase.

3.2.1 L*

L^* uses a Minimally Adequate Teacher (MAT) model in which a learner queries a teacher in order to learn an unknown regular language L. Queries are built using a fixed input alphabet Σ where $L \subseteq \Sigma^*$ must hold. The teacher must respond to two types of queries posed by the learner, namely membership and equivalence queries. Membership queries consist of a word $s \in \Sigma^*$ and must be answered with either "yes" if $s \in L$, or "no" if not. In other words, membership queries are used to check if a given word is part of the language being learned. Equivalence queries on the other hand, consist of a regular language L_{prop}, proposed by the learner. The teacher must answer with "yes" if $L_{prop} \equiv L$, or returns a counterexample c proving the two languages are different, so $c \in L(S) \iff c \notin L$. In other words, equivalence queries are used to verify if the learner has successfully learned the target language L or if not, return a counterexample detailing the differences. The results of the membership queries are stored in an observation table $O = (S, E, T)$, where S is a prefix-closed set of strings representing candidates for states of L_{prop}, E a suffix-closed set of strings used to distinguish between candidates and T a transition function $(S \cup S \cdot \Sigma) \cdot E \rightarrow 0, 1$. Essentially, if visualized as a 2D array where the rows are labeled with elements in $(S \cup S \cdot \Sigma)$ and columns with elements in E, the entries in the table are ones, if the word created by appending the row-label to the column-label is accepted by L and zeros if not. The goal of L^* is to learn a DFA acceptor for L using the observation table. S-labeled rows correspond to states in the acceptor under construction. E-labeled columns represent individual membership query results. For the observation table to be transformable into a DFA acceptor, it must first be closed and consistent.

```
Initialization:
Set observation table O = (S, E, T) with S, E = {ϵ}.
populate(O).

repeat:
  while O is not closed or not consistent do
    if O is not closed then
      choose s₁ ∈ S, σ ∈ Σ such that
      row(s₁ · σ) ≠ row(s) ∀s ∈ S
      add s₁ · σ to S
      populate(O)
    end
    if O is not consistent then
      choose s₁, s₂ ∈ S, σ ∈ Σ and e ∈ E such that
      row(s₁) = row(s₂) and T(s₁ · σ · e) ≠ T(s₂ · σ · e)
      add σ · e to E
      populate(O)
    end
  end
  Construct L_prop from O and perform an equivalence query.
  if query returns a counterexample c then
    add all prefixes of c to S
    populate(O)
  end
until teacher replies "yes" to equivalence query L_prop ≡ L
return L_prop
```

Listing 3.1: L^* algorithm

Closed is defined as for all $t \in S \cdot \Sigma$ there exists an $s \in S$ so that $row(t) = row(s)$. In other words, that no new information is gained by expanding the S-set by any word in Σ. If an observation is not closed, it is fixed by adding t to S and updating the table rows through more membership queries. Consistent means, that $\forall s_1, s_2 \,|\, row(s_1) = row(s_2) \implies \forall \sigma \in \Sigma \,|\, row(s_1 \cdot \sigma) = row(s_2 \cdot \sigma)$, or in other words, appending the same word to identical states should not result in different outcomes. If an observation table is inconsistent, it is made consistent again by adding another column to the table with the offending σ as its label and again updating the table rows through more membership queries.

Listing 3.1 shows the workings of the basic L^* algorithm by Angluin. The function $populate(O)$ extends T to $(S \cup S \cdot \Sigma) \cdot E$ by asking membership queries for all table entries still missing membership information. At the start of the algorithm, the observation table is initialized with $S = E = \{\epsilon\}$. Next, until a equivalence query succeeds, the observation table is repeatedly brought to a closed and consistent state by expanding the S and E sets respectively. Once both closed and consistent, L_{prop} is constructed from O and used in an equivalence query. If the equivalence query returns “yes", the algorithm terminates, returning the learned DFA. If not, the returned counterexample is used to update the observation table and the algorithm loops back to line 5.

3.2.2 KV

Another notable AAL algorithm is the KV algorithm published in 1994 by Kearns and Vazirani Kearns and Vazirani 1994. It is designed to work in the same learner-teacher framework as L^*, but was designed to minimize the amount of membership queries needed to learn a finite automaton M. The KV algorithm does this by organizing learned information in an ordered binary tree called a classification tree C_T as opposed to the table structure utilized by L^*. Intuitively, L^* must perform membership queries for every entry in the observation table to differentiate between possible states, whereas KV requires only a subset to distinguish them.

In the KV algorithm, learned data is stored in two sets called the access strings set S and the distinguishing strings set D. Every string $s \in S$ represents a distinct and unique state of the automaton M. In other words, any s when applied starting in the initial state of M leads to a unique state $M[s]$. The distinguishing strings set is defined as the set of strings $d \in D$ where for each pair $s, s' \in S, s \neq s'$ there exists a $d \in D$ such that either $M[s \cdot d]$ or $M[s' \cdot d]$ is an accepting state. D is used to ensure that their are no ambiguous states. The sets S, D are organized in a binary tree called the classification tree C_T where parent nodes are strings from D and the leaf nodes are strings from S. The root node is set to the empty string λ. For each node of the tree, starting from the root node, each right subtree contains access strings to accepting states while left subtrees contain access strings to rejecting states of M. Given a new string s', we simply start at the root nodes, then sift down the tree by executing a membership query for $s' \cdot \lambda_1$ and depending on if the query returns “yes" or “no" continuing with the left or right subtree until we reach a leaf node labeled with s. If $s' = s$ then the states are equivalent, otherwise the classification tree is updated to include another leaf node representing the newly learned distinct state s'. The main learning loop of the KV algorithm is shown in more detail in Listing 3.2. Following the initialization of the classification table, new states learned from counterexamples are repeatedly added until an equivalence query is successful. The $Update(C_T, c)$ function adds a new leaf to the C_T based on a counterexample c returned from an equivalence query.

```
Initialization:
  Set root node of C_T to ϵ.
  Perform membership query on ϵ to determine if the initial state is
     accepting or not.
  Construct hypothesis automaton M̂ consisting of only the initial state,
     with self-transitions for all other transitions.
  Add two access strings ϵ and the counterexample string c.

repeat:
  Construct hypothesis automaton M̂ from C_T.
  Equivalence query(M̂)
  if: query returns "yes" then
    return M̂
  end

  Update(C_T, c)
end
```

Listing 3.2: *KV* algorithm

3.3 Fuzzing

3.4 IPsec

Virtual Private Networks (VPN) are used to extend and or connect private networks across an insecure channel (usually the public internet). They can be used e.g. to gain additional privacy from prying eyes such as Internet Server Providers, access to region-locked online content or secure remote access to company networks. Many different VPN protocols exit, including PPTP, OpenVPN and Wireguard. IPsec or IP Security, is a VPN layer 3 protocol used to securely communicate over an insecure channel. It is based on three sub-protocols, the Internet Key Exchange (IKE) protocol, the Authentication Header (AH) protocol and the Encapsulating Security Payload (ESP) protocol. IKE is mainly used to handle authentication and to securely exchange as well as manage keys. Following a successful IKE round, either AH or ESP is used to send packets securely between parties. The main difference between AH and ESP is that AH only ensures the integrity and authenticity of messages while ESP also ensures their confidentiality through encryption.

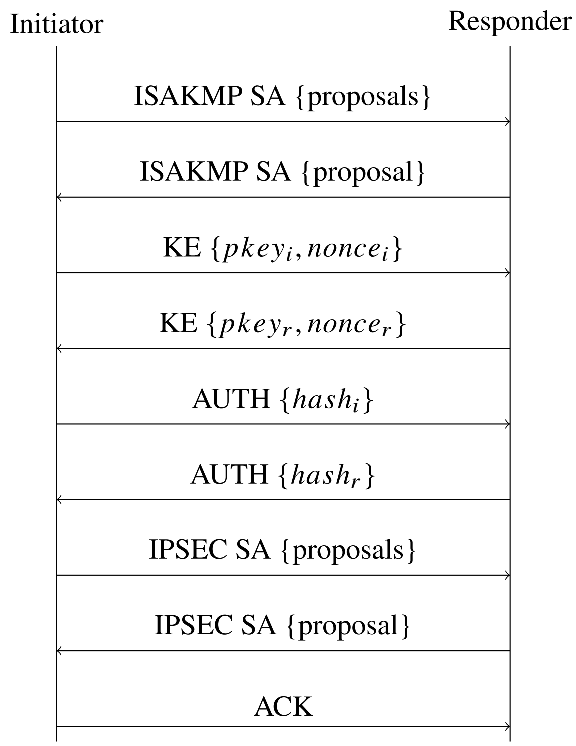

Figure 3.1: IKEv1 between two parties

The IKEv1 protocol works in two main phases, both relying on the Internet Security Association and Key Management Protocol (ISAKMP). Additionally, phase one can be configured to proceed in either Main Mode or Aggressive Mode. A typical exchange between two parties, an initiator and a responder, using Main Mode for phase 1, can be seen in Figure 3.1. In phase one (Main Mode), the initiator begins by sending a Security Association (SA) to the responder. A SA essentially details important security attributes required for a connection such as the encryption algorithm and key-size to use, as well as the authentication method and the used hashing algorithm. These options are bundled in containers called proposals, with each proposal describing a possible security configuration. While the initiator can send multiple proposals to give the responder more options to choose from, the responder must answer with only one proposal, provided both parties can agree upon one of the suggested proposals. This initial communication is denoted as *ISAKMP SA* in Figure 3.1. Subsequently, the two parties perform a Diffie-Hellman key exchange, denoted as *KE*, and send each other nonces used to generate a shared secret key *SKEYID* as detailed in Listing 3.3. PSK refers to the pre-shared key, Ni/Nr to the initiator/responder nonce and CKY-I/CKY-R to the initiator/responder identifier cookie. Note that IKEv1 allows using various different authentication modes aside from PSK, including public key encryption and digital signatures. *SKEYID* is used as a seed key for all further session keys *SKEYID_d*, *SKEYID_a*, *SKEYID_e*, with g^{xy} referring to the previously calculated shared Diffie-Hellman secret and prf to a pseudo-random function (in our case, HMAC). Following a successful key exchange, all further messages of phase one and two are encrypted using a key derived from *SKEYID_e* and *SKEYID_a* for authentication. Finally, in the last section of phase one *AUTH*, both parties exchange and verify hashes to confirm the key generation was successful. Once verification succeeds, a secure channel is created and used for phase two communication. If phase one uses Aggressive Mode, then only three packets are needed to reach phase two. While quicker, the downside of Aggressive Mode is that the communication of the hashed authentication material happens without encryption. This means, that using short pre-shared keys in combination with Aggressive Mode is inherently insecure, as the unencrypted hashes are vulnerable to brute-force attacks provided a short key-size [1]. The shorter phase two (Quick Mode) begins with another SA exchange, labeled with *IPSEC SA* in Figure 3.1. This time, however, the SA describes the security parameters of the ensuing ESP/AH communication and the data is sent authenticated and encrypted using the cryptographic material calculated in phase one. This is followed by a single acknowledge message, *ACK*, from the initiator to confirm the agreed upon proposal. After the acknowledgment, all further communication is done via ESP/AH packets, using *SKEYID_d* as keying material.

```
# For pre-shared keys:
SKEYID = prf(PSK, Ni_b | Nr_b)

# to encrypt non-ISAKMP messages (ESP)
SKEYID_d = prf(SKEYID, g^xy | CKY-I | CKY-R | 0)

# to authenticate ISAKMP messages
SKEYID_a = prf(SKEYID, SKEYID_d | g^xy | CKY-I | CKY-R | 1)

# for further encryption of ISAKMP messages in phase 2
SKEYID_e = prf(SKEYID, SKEYID_a | g^xy | CKY-I | CKY-R | 2)
```

Listing 3.3: IKE Keying

In addition to the packets shown in Figure 3.1, IKEv1 also specifies and uses so called ISAKMP Informational Exchanges. Informational exchanges in IKEv1 are used to send ISAKMP Notify or ISAKMP Delete payloads. Following the key exchange in phase one, all Informational Exchanges are sent encrypted and authenticated. Prior, they are sent in plain. ISAKMP Notify payloads are used to transmit various error and success codes, as well as for keep-alive messages. ISAKMP Delete is used to inform the other communication partner, that a SA has been deleted locally and request that they do the same, effectively closing a connection.

Compared to other protocols, IPsec offers a high degree of customizability, allowing it to be fitted for many use cases. However, in a cryptographic evaluation of the protocol, Ferguson and Schneier Ferguson and Schneier [1999] criticize the complexity arising from the high degree of customizability as the biggest weakness of IPsec. To address its main criticism, IPsec-IKEv2 was introduced in RFC 7296 to replace IKEv1 [Kaufman et al. 2014]. Nevertheless, IPsec-IKEv1 is still in wide-spread use to this day, with the largest router producer in Germany, AVM, still only supporting IKEv1 in their routers [GmbH 2022]. We use IPsec-IKEv1 with Main Mode and ESP in this paper and focus on the IKE protocol as it is the most interesting from an AAL and security standpoint.

[1]https://nvd.nist.gov/vuln/detail/CVE-2018-5389

Chapter 4

Learning

test

Chapter 5

Evaluation

test

Chapter 6

Conclusion

test

Bibliography

Abhijith, M and K Senthilvadivu [2020]. *Impact Of VPN Technology On It Industry During Covid-19 Pandemic*. IJEAST. 2020 (cited on page 1).

Andrews, Keith [2021]. *Writing a Thesis: Guidelines for Writing a Master's Thesis in Computer Science*. Graz University of Technology, Austria. 10 Nov 2021. https://ftp.isds.tugraz.at/pub/keith/thesis/ (cited on page xi).

Angluin, Dana [1987a]. *Learning regular sets from queries and counterexamples*. Information and computation 75.2 (1987), pages 87–106 (cited on pages 1, 3).

Angluin, Dana [1987b]. *Learning regular sets from queries and counterexamples*. Information and Computation 75.2 (1987), pages 87–106. ISSN 0890-5401. doi:https://doi.org/10.1016/0890-5401(87)90052-6. https://www.sciencedirect.com/science/article/pii/0890540187900526 (cited on page 5).

Barker, Elaine, Quynh Dang, Sheila Frankel, Karen Scarfone, and Paul Wouters [no date]. *Guide to IPsec VPNs*. en. doi:https://doi.org/10.6028/NIST.SP.800-77r1 (cited on page 1).

Daniel, Lesly-Ann, Erik Poll, and Joeri de Ruiter [2018]. *Inferring OpenVPN state machines using protocol state fuzzing*. 2018 IEEE European Symposium On Security And Privacy Workshops (Euros&PW). IEEE. 2018, pages 11–19 (cited on page 3).

Ferguson, Niels and Bruce Schneier [1999]. *A cryptographic evaluation of IPsec* (1999) (cited on page 10).

Ferguson, Niels and Bruce Schneier [2021]. *The best VPN protocols* (2021). https://nordvpn.com/de/blog/protocols/ (cited on page 1).

Fiterău-Broştean, Paul, Ramon Janssen, and Frits Vaandrager [2016]. *Combining model learning and model checking to analyze TCP implementations*. International Conference on Computer Aided Verification. Springer. 2016, pages 454–471 (cited on page 3).

Fiterău-Broştean, Paul, Toon Lenaerts, Erik Poll, Joeri de Ruiter, Frits Vaandrager, and Patrick Verleg [2017]. *Model learning and model checking of SSH implementations*. Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software. 2017, pages 142–151 (cited on page 3).

GmbH, AVM Computersysteme Vertriebs [2022]. *Connecting the FRITZ!Box with a company's VPN*. https://en.avm.de/service/vpn/tips-tricks/connecting-the-fritzbox-with-a-companys-vpn/. 2022 (cited on pages 1, 10).

Guo, Jiaxing, Chunxiang Gu, Xi Chen, and Fushan Wei [2019]. *Model learning and model checking of IPsec implementations for Internet of Things*. IEEE Access 7 (2019), pages 171322–171332 (cited on pages 1, 3).

Isberner, Malte, Falk Howar, and Bernhard Steffen [2014]. *The TTT Algorithm: A Redundancy-Free Approach to Active Automata Learning*. International Conference on Runtime Verification. 2014, pages 307–322 (cited on page 3).

Kaufman, Charlie, Paul Hoffman, Yoav Nir, Pasi Eronen, and Tero Kivinen [2014]. *Internet key exchange protocol version 2 (IKEv2)*. Technical report. 2014 (cited on page 10).

Kearns, Michael J and Umesh Vazirani [1994]. *An introduction to computational learning theory*. MIT press, 1994 (cited on pages 5, 7).

Muškardin, Edi, Bernhard K Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler [2022]. *AALpy: an active automata learning library*. Innovations in Systems and Software Engineering (2022), pages 1–10 (cited on pages 2, 5).

Niese, Oliver [2003]. *An integrated approach to testing complex systems*. 2003 (cited on page 5).

Novickis, Tomas, Erik Poll, and Kadir Altan [2016]. *Protocol state fuzzing of an OpenVPN*. PhD Thesis. PhD thesis. MS thesis, Fac. Sci. Master Kerckhoffs Comput. Secur., Radboud Univ, 2016 (cited on page 3).

Pferscher, Andrea and Bernhard K Aichernig [2021]. *Fingerprinting Bluetooth Low Energy devices via active automata learning*. International Symposium on Formal Methods. Springer. 2021, pages 524–542 (cited on pages 1, 3).

Pferscher, Andrea and Bernhard K Aichernig [2022]. *Stateful Black-Box Fuzzing of Bluetooth Devices Using Automata Learning*. NASA Formal Methods Symposium. Springer. 2022, pages 373–392 (cited on page 1).

Rivest, Ronald L. and Robert E. Schapire [1993]. *Inference of Finite Automata Using Homing Sequences*. Information & Computation 103.103 (1993), pages 51–73 (cited on page 3).

Steffen, Bernhard, Falk Howar, and Maik Merten [2011]. *Introduction to Active Automata Learning from a Practical Perspective*. Jun 2011, pages 256–296. ISBN 978-3-642-21454-7. doi:10.1007/978-3-642-21455-4_8 (cited on page 3).