

Mining Digital Twins of a VPN Server

Andrea Pferscher^{1,*}, Benjamin Wunderling¹, Bernhard K. Aichernig¹ and
Edi Muškardin^{1,2}

¹*Institute of Software Technology, Graz University of Technology, Inffeldgasse 16b/II, 8010 Graz, Austria*

²*TU Graz - SAL DES Lab, Silicon Austria Labs, Sandgasse 34, 8010 Graz, Austria*

Abstract

Virtual private networks (VPNs) are widely used to create a secure communication mode between multiple parties over an insecure channel. A common use case for VPNs is secure access to company networks. Therefore, bugs in VPN software are often severe. The Internet Key Exchange protocol (IKE) is a protocol in the Internet Protocol Security (IPsec) protocol suite used in VPNs. There are two versions of IKE, IPsec-IKEv1 and the newer IPsec-IKEv2, with IPsec-IKEv1 still widely used in practice. While IPsec-IKEv2 has been investigated in the context of automata learning, no such work exists for IPsec-IKEv1. This paper closes the gap for the IPsec-IKEv1 protocol and shows the steps taken to learn a digital twin of an IPsec server using automata learning. We present and contrast two learned models of an IPsec server. Using learning, we also found security issues in encryption libraries.

Keywords

IPsec, VPN, active automata learning, digital twin, model mining

1. Introduction

Virtual Private Networks (VPNs) allow secure communication over an insecure channel. The importance of VPNs has increased dramatically since the COVID-19 pandemic due to the influx of people working from home [1]. This makes finding vulnerabilities in VPN software more critical than ever. In practice, VPNs are set up based on third-party components, which are usually closed source. In addition, multiple parties are involved in VPN communications. These challenges make it difficult to test possible security vulnerabilities in all scenarios.

Behavioral models are a useful tool for testing and verifying complex systems. A model can be viewed as a digital twin that simulates the behavior of the system. The availability of models, however, might be limited for the following reasons. First, the manual creation of a model can be a tedious and error-prone process. Second, the model must always be kept up to date.

Automata learning has proven itself as a useful technique for automatically generating behavioral models of various communication protocols, e.g., Bluetooth Low Energy [2], TLS [3], SSL [4], or MQTT [5]. Active learning techniques create a behavioral model by actively querying

FMDT'23: Workshop on Applications of Formal Methods and Digital Twins, March 13, 2023, Lübeck, Germany

*Corresponding author.


✉ andrea.pferscher@ist.tugraz.at (A. Pferscher); benjamin.wunderling@gmail.com (B. Wunderling);


aichernig@ist.tugraz.at (B. K. Aichernig); edi.muskardin@silicon-austria.com (E. Muškardin)

🌐 <https://apferscher.github.io/> (A. Pferscher); <https://aichernig.blogspot.com/> (B. K. Aichernig);

<https://emuskardin.github.io/> (E. Muškardin)

🆔 0000-0002-9446-9541 (A. Pferscher); 0000-0002-3484-5584 (B. K. Aichernig); 0000-0001-8089-5024 (E. Muškardin)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

the system to gain knowledge about it. In this way, not only a model is created, but also a stateful testing approach for black-box systems. The learned model represents a digital twin of the system under learning, which can then be used for other model-based techniques, like model-based test-case generation or model checking.

In this paper, we investigate the applicability of automata learning for mining security-critical components of a VPN. For this, we learn the behavioral model of the Internet Key Exchange protocol (IKE) protocol which is part of the Internet Protocol Security (IPsec) protocol suite. IKE is used to share authenticated key material between the involved parties. For learning, we use a VPN client to query the VPN server instantiation. This allows us to create a digital twin of the security-critical part of a VPN server without knowing its internals (black-box). The work is part of LearnTwins, a research project on learning digital twins¹.

The paper is organized as follows. Sect. 2 provides details about the used modeling-formalism, the learning algorithm, and the system-under-learning. Section 3 provides a comparison with related work. In Sect. 4, we present our learning setup for mining a digital twin of a VPN. In Sect. 5 we present our learned models and, finally, in Sect. 6 we draw our conclusions.

2. Preliminaries

2.1. Mealy Machines

Mealy machines are a modeling formalism for reactive systems such as communication protocols. They are finite-state machines in which each transition is labeled with an input and the corresponding output action. We consider Mealy machines to describe deterministic behavior. More formally, a Mealy machine is defined as a 6-tuple $M = \{S, s_0, I, O, \delta, \lambda\}$, where S is a finite set of states, $s_0 \in S$ is the initial state, I is a finite set called input alphabet, O is a finite set called output alphabet, δ is the state-transition function $\delta: S \times I \rightarrow S$ that maps a state and an element of the input alphabet to another state in S and λ is the output function $\lambda: S \times I \rightarrow O$ that maps a state-input pair to an output in O .

2.2. Automata Learning

Automata learning algorithms generate a model that describes the behavior of the system under learning (SUL). We differentiate between active and passive automata learning. Passive learning creates a behavioral model based on a given data set, e.g., log files, while active learning directly queries the SUL. For learning a digital twin, we prefer active learning since active algorithms can query unusual or rare scenarios, whereas passive learning depends heavily on the provided data.

Today, many active learning algorithms are based on concepts derived from Angluin's L^* algorithm [6], which was designed to identify deterministic finite automata formalizing regular languages. In her seminal work, Angluin introduced the concept of the minimally adequate teacher (MAT), in which a learner queries a teacher about the SUL to iteratively generate a behavioral model. The teacher answers membership and equivalence queries posed by the learner regarding the SUL. Membership queries are used to check whether a word is accepted

¹<https://learntwins.ist.tugraz.at>

by the SUL. The learner then updates their model of the SUL based on the answers to their queries. Equivalence queries are used to check if a learned model exactly matches the behavior of the SUL. If the teacher provides a counterexample that represents the behavioral difference between the learned model and SUL, the learner improves the model by asking more membership queries. The MAT concept has been extended to facilitate other model formalisms like Mealy machines [7], where membership queries are renamed to output queries. Output queries are used to retrieve outputs to the corresponding input sequences.

The MAT concept is also used by other active learning algorithms. Kearns and Vazirani [8] propose an active learning algorithm that uses an underlying tree-based data structure to construct a model. We refer to their learning algorithm as *KV*. *KV*'s tree-based data structure can reduce the number of required output queries compared to the table-based technique used in the L^* algorithm.

2.3. Internet Protocol Security

Internet Protocol Security (IPsec) is a VPN Layer 3 protocol suite used to securely communicate over an insecure channel. It includes three sub-protocols: Internet Key Exchange protocol (IKE), the Authentication Header (AH), and the Encapsulating Security Payload (ESP) protocol. IKE is mainly used for authentication, and the secure exchange and management of keys. IKE has two versions, IKEv1 and IKEv2, with IKEv2 being the newer and recommended version [9]. Following a successful IKE round, either AH or ESP is used to send packets securely between parties. While AH only ensures the integrity and authenticity of messages, ESP also ensures their confidentiality through encryption.

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 [10] 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 [11] to replace IKEv1. Nevertheless, IPsec-IKEv1, RFC 2409 [12], is still in widespread use to this day, with the largest router producer in Germany, AVM, still only supporting IKEv1 in their routers [13]. We investigate IPsec-IKEv1 with ESP in this paper and focus on the IKE protocol.

The IKEv1 protocol works in two main phases, both relying on the Internet Security Association and Key Management Protocol (ISAKMP). A typical key exchange between two parties, an initiator and a responder, can be seen in Fig. 1. In phase one (Main Mode), the initiator sends a Security Association (SA) to the responder. A SA essentially details important security attributes 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

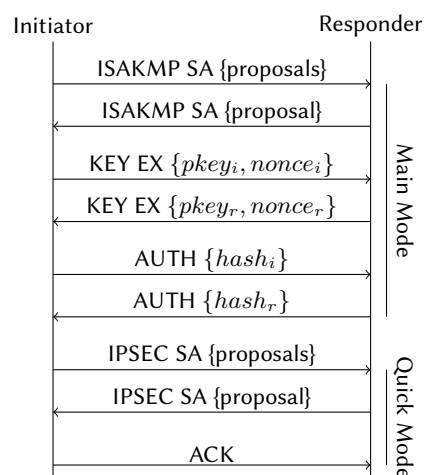


Figure 1: IKEv1 between two parties

possible security configuration. While the initiator can send multiple proposals to give the responder more options to choose from. In comparison, the responder must answer with only one proposal, provided it supports one of the suggestions. Subsequently, the two parties perform a Diffie-Hellman key exchange and exchange nonces to generate a shared secret key. This secret key is used as a seed key for all further session keys. Following a successful key exchange, all further messages are encrypted. Finally, both parties exchange hashed authentication material (usually pre-shared keys or certificates). If the hashes can be verified, a secure channel is created and used for phase two communication.

The shorter phase two (Quick Mode) begins with another SA exchange. This time, however, the SA describes the security parameters of the ensuing ESP/AH communication. This is followed by a single acknowledge message from the initiator to confirm the agreed upon proposal. After the acknowledgment, all further communication is done via ESP/AH packets.

3. Related Work

Model learning became a popular tool for creating behavioral models of various communication protocols, e.g., TLS [3], TCP [14], SSL [4], MQTT [5], 802.11 4-Way Handshake of Wi-Fi [15], or BLE [2]. The learned models reveal differences in the specification or provide a useful extension to the unspecified properties. In addition, the learned models serve as a basis for further techniques like model-checking or model-based security testing. In the VPN domain, Daniel et al. [16] learned a model of two OpenVPN implementations. The challenges of implementing a learning setup for OpenVPN were discussed by Novickis [17]. In contrast to our technique, they learned a more abstract model of the entire OpenVPN session, where details about the key exchange were abstracted in the learned model. Closely related to our work, Guo et al. [18] learned a model of the IPsec-IKEv2 protocol. The authors stress that IPsec-IKEv1 is more complicated to configure securely, which highlights the need to test the configuration of the older version as well, since it is still widely used [13]. With our work, we complete the learning approaches for all IKE versions.

4. Method

Environment Setup. As our SUL, we used a Linux Strongswan² US.9.5/K5.15.0-25-generic. Learning was done using two VirtualBox 6.1 virtual machines (VMs) running standard Ubuntu 22.04 LTS distributions. Both VMs were allotted 4GB of memory and one CPU core. All communication took place in an isolated virtual network to eliminate external influences. During learning, all power-saving options and similar potential causes of disruptions were disabled. The IPsec server was restarted before the start of each learning procedure to ensure identical conditions. We designated one VM as the initiator and one as the responder to create a typical client-server setup. The open-source IPsec implementation was installed on the responder VM and set to listen for incoming connections from the initiator VM. The Strongswan server was configured to use pre-shared keys for authentication and default recommended

²<https://www.strongswan.org/>

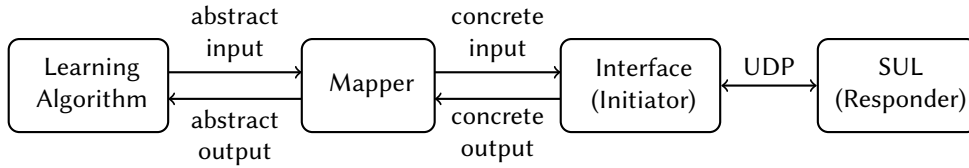


Figure 2: Automata learning setup for learning a model of IPsec-IKEv1.

security settings. Additionally, it was set up to allow unencrypted notification messages, which we used to reset the connection during the learning process. For learning, we used the Python library AALPY [19] version 1.2.9 in conjunction with the packet manipulation library Scapy [20], version 2.4.5. Significant effort was invested into expanding the ISAKMP Scapy module to support all packets required for IPsec.

Learning Setup. Figure 2 gives an overview of the learning setup, adapted from Tappler et al. [5]. The learning algorithm sends abstract inputs chosen from the input alphabet to the mapper, which converts them to concrete inputs. The concrete inputs, which are actual IPsec packets, are then sent to the SUL, by means of a UDP communication interface. The interface represents the initiator whereas the SUL is the responder represented by a Strongswan server instance. This separation between abstract and concrete inputs and outputs allows us to learn a generic model in a reasonable amount of time.

The abstract inputs consist of the initiator-to-responder messages: `isakmp_sa`, `key_ex`, `auth`, `ipsec_sa` and `ack`. The responder-to-initiator outputs from Fig. 1 were extended by `NONE` and `ERROR`, where `NONE` signifies a lack of response from the SUL and `ERROR` is used as a collection of received error notifications. We use the KV [8] algorithm for learning with the improved counterexample-processing of Rivest and Schapire [21]. Since a perfect equivalence oracle cannot be assumed in practice, we substitute the equivalence oracle with model-based conformance testing between the intermediate learned model and the SUL. The conformance tests provide state-coverage combined with randomness.

Our mapper implements translation methods for each communication step in a typical IPsec-IKEv1 exchange, as described in Sect. 2. We use the Python library Scapy to construct IKEv1 packets. This approach allows us to change the fields and values of generated packets at will, opening up the possibility of fuzz testing these fields in future work as shown by Pferscher and Aichernig [22]. Parsing was made more difficult by the fact that Scapy does not support all the packets required by IPsec-IKEv1. To solve this problem, we implemented the missing packets in the Scapy ISAKMP class and used this modified version.

The IPsec packets generated by the mapper are then passed on to our interface for the SUL that handles all incoming and outgoing UDP packets. Additionally, it converts responses from the SUL into valid Scapy packets and passes them on to the mapper. The mapper class then parses the responses received from the interface and returns an abstract output representing the received data to the learning algorithm. Since part of communication is encrypted, we require a framework that correctly handles the en/decryption of messages. For each request, we store the base key and the responses for use in the next message, and update affected key material

as needed. Most notably, the initialization vectors (IVs) are updated in almost every request and differ between messages. Informational requests also handle their IVs separately. For each request that we send, if available, we try to parse the response, decrypting it if necessary and resetting or adjusting internal variables as required to match the server.

Automata learning requires that the SUL can be reset to an initial state after each query. We implement this using a combination of the ISAKMP delete request and general ISAKMP informational error messages. While delete works for established connections in phase two of IKE, we require informational error messages to trigger a reset in phase one. Implementation was hindered at times by unclear RFC specifications, but this was overcome by manually comparing packet dumps and Strongswan logs to fix encryption errors.

Combating Non-determinism. During learning, the server occasionally exhibited non-deterministic behavior. For this, we implemented two methods of counteracting it. In our first method, we simply repeat the output query if we observe non-deterministic behavior. In this case, we select the output that occurs most often. Additionally, using timed waits after each input also helped to further decrease the number of non-determinism errors. However, learning still failed occasionally with these mitigation mechanisms for non-deterministic behavior.

A closer examination of the remaining non-deterministic behavior led to the discovery that it is caused by so-called retransmissions. Essentially, the IKE specification allows for previous messages to be retransmitted if deemed useful by the server. A possible trigger could be the final message of an IKE exchange being skipped/lost. For example, if instead of an AUTH message, the server receives a phase two IPSEC SA message, the server would not know if it missed a message or if there was an error on the other party's side. In this case, the Strongswan server reacts by retransmitting the previous message, prior to the missing one in an attempt to signal to the other party, that they should resend the missing message. To counteract this behavior, we implemented checks in our mapper to allow for the ignoring of retransmissions. If a repeated message ID is found, it is flagged as a retransmission. With this addition, the IPsec server behaved deterministically. The downside of this method is that it completely ignores the retransmissions, which could be a good source of information for fingerprinting different IPsec servers.

5. Evaluation

In our evaluation, we used two different learning setups to learn a model of the Linux Strongswan server. The first setup considers retransmitted messages from the server as outputs, whereas the second setup filters out any retransmitted messages. We included the model with retransmitted messages in the Appendix A, see Fig. 4. The model without retransmission is shown in Fig. 3. To enhance the readability of the models, we simplified the depicted models. The complete models are available as supplementary material [23].

For learning the models, we compared the active automata learning algorithms: L^* and KV. Both algorithms use an improved counterexample processing following Rivest and Schapire [21]. We prefer the usage of KV since it required fewer queries and less time to learn a model. Learning with KV was about one-fifth faster than L^* and required less than half the output queries. The

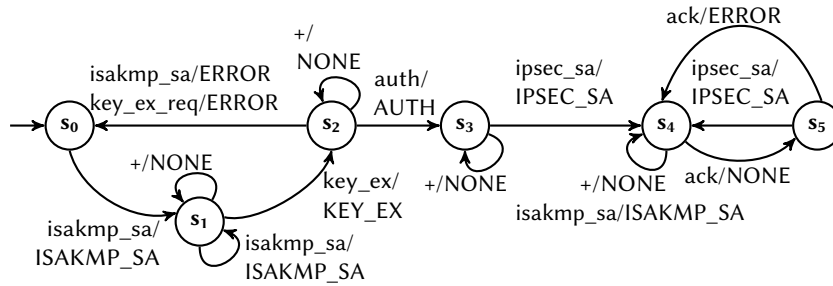


Figure 3: Simplified model of Strongswan server learned without retransmitted messages. The ‘+’-symbol is an abbreviation for all inputs that are not explicitly shown.

model with retransmission (Fig. 4) has 13 states, whereas the model without retransmissions (Fig. 3) has only six states. For readability, we group together several different error messages in the output ERROR. The models also include NONE responses, which were used in cases where the input misses sensible information to establish an encrypted communication. While observing the behavior of the server when exposed to completely non-sensible input is interesting from a security testing standpoint, as all specifications state that the encryption requires a prior keying procedure, we decided to ignore those few cases. However, for future work in the field of fuzzing, these edge-cases should be considered as well.

We state the learning results of learning without retransmissions, since learning was more reliable in this case. We repeated each experiment five times and averaged the results. KV required approximately 38 minutes (2296 seconds) to learn, 79 output queries and 60 conformance tests were performed in 753 and 991 steps respectively. Learning performed four equivalences queries. The learning results for the other model can be found in the Appendix A.

Examining both models, especially Fig. 4, we can clearly see the separation between the two phases of the IKEv1 protocol. Phase one (Main Mode) completes in state s_3 , and phase two (Quick Mode) begins right thereafter in state s_4 . Comparing the models, we see that for the states s_0 to s_3 both models are identical. This is likely due to the fact, that most differences were caused by retransmissions which only occur in phase two. The model depicted in Fig. 3 shows streamlined behavior that fits our reference IKE exchange (see Fig. 1) almost perfectly. The clean automaton makes it easy to see, that after a connection has been established, we can still create new connections or reestablish existing ones by sending another IPSEC SA message and then acknowledging the response.

The synthesis of a digital twin from the Linux Strongswan server not only provides interesting insight into the behavioral aspects of the implementation, especially regarding unexpected retransmitted messages. It also helped to test the general environment that is used to establish a VPN. A notable finding was the discovery of a very niche bug in a used Python Diffie-Hellman (DH) key exchange library³. The bug was very elusive and only found thanks to the exhaustive number of packets sent by the learning algorithm. It turns out there was a very niche bug in the library where, if the most significant byte (MSB) was a zero, it would be omitted from the response, causing the local result to be one byte shorter than the value calculated by the SUL. Regardless, it could compromise the security of affected systems and therefore the maintainer

³<https://github.com/TOPDapp/py-diffie-hellman>

of the library has been notified of the problem. Due to the elusive nature of this bug, it would very likely not have been noticed without the exhaustive communication done by the model learning process and without seeing the resulting non-deterministic behavior of the SUL due to the truncated message.

6. Conclusion

Summary. We presented an automata learning framework that automates the generation of a digital twin of a VPN server. More precisely, the learned digital twin represents a behavioral model of the security-critical key-exchange procedure. We established a learning interface that enables active interaction with a VPN server. We found that the investigated VPN server occasionally retransmits previously sent messages, which violated our assumption about deterministic behavior. To deal with this problem, we adapted the learning interface to filter these retransmitted messages. With this adaptation, we could reliably mine a new digital twin within less than one hour.

Discussion. Digital twins should adequately simulate the behavior of their twinned systems. When mining digital twins this property becomes critical. Compared to passive automata learning that simply uses given data, active approaches enforce checking behavioral conformance by testing for behavioral equivalence. Even though this equivalence check is limited by the exhaustiveness of the applied model-based testing technique, we can at least state that the system conforms according to the test suite. However, applying active learning techniques requires the establishment of an interface that enables the interaction with the SUL. Albeit creating a learning interface for a VPN server was not straightforward, the interface has to be created only once.

This paper presented a methodology for learning digital twins of software components. Even though the twinned system does not represent a physical twin in the classical sense, digital twins of software systems still have many application areas. First, the learned behavioral model creates a common understanding of the system. This can be especially useful if access to the twinned system is limited, e.g., third-party components. Secondly, the model can be used for further testing and verification purposes. In our case study, already the learning procedure of the digital twin revealed security issues in the libraries used to establish a secure VPN. Finally, the twin models can be used to simulate VPN components in a network infrastructure. This can be useful if we want to simulate differences in behavior between distinct VPN servers. Instead of setting up each VPN server individually, the mined models can be used.

Future Work. In this paper, we discussed how digital twins can be automatically learned from a black-box system. In future work, we intend to use the digital twin to simulate behavior of the twinned system so that components in the network can be replaced by their twin model. In addition, we want to use the twin for model-based verification and testing purposes like stateful fuzz testing or model checking.

Acknowledgments

This work is supported by the LearnTwins project funded by FFG (Österreichische Forschungsförderungsgesellschaft) under grant 880852, and the “University SAL Labs” initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems.

References

- [1] A. Feldmann, O. Gasser, F. Lichtblau, E. Pujol, I. Poese, C. Dietzel, D. Wagner, M. Wichtlhuber, J. Tapiador, N. Vallina-Rodriguez, O. Hohlfeld, G. Smaragdakis, A year in lockdown: How the waves of COVID-19 impact internet traffic, *Commun. ACM* 64 (2021) 101–108. doi:10.1145/3465212.
- [2] A. Pferscher, B. K. Aichernig, Fingerprinting Bluetooth Low Energy devices via active automata learning, in: M. Huisman, C. S. Pasareanu, N. Zhan (Eds.), *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings*, volume 13047 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 524–542. doi:10.1007/978-3-030-90870-6_28.
- [3] J. de Ruiter, E. Poll, Protocol state fuzzing of TLS implementations, in: J. Jung, T. Holz (Eds.), *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, USENIX Association, 2015, pp. 193–206. URL: <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/de-ruiter>.
- [4] P. Fiterau-Brostean, T. Lenaerts, E. Poll, J. de Ruiter, F. W. Vaandrager, P. Verleg, Model learning and model checking of SSH implementations, in: H. Erdogmus, K. Havelund (Eds.), *Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Santa Barbara, CA, USA, July 10-14, 2017*, ACM, 2017, pp. 142–151. doi:10.1145/3092282.3092289.
- [5] M. Tappler, B. K. Aichernig, 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. doi:10.1109/ICST.2017.32.
- [6] D. Angluin, Learning regular sets from queries and counterexamples, *Information and Computation* 75 (1987) 87–106. doi:10.1016/0890-5401(87)90052-6.
- [7] M. Shahbaz, R. Groz, Inferring Mealy machines, in: A. Cavalcanti, D. Dams (Eds.), *FM 2009: Formal Methods, Second World Congress, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings*, volume 5850 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 207–222. doi:10.1007/978-3-642-05089-3_14.
- [8] M. J. Kearns, U. V. Vazirani, *An Introduction to Computational Learning Theory*, MIT Press, 1994. URL: <https://mitpress.mit.edu/books/introduction-computational-learning-theory>.
- [9] E. Barker, Q. Dang, S. Frankel, K. Scarfone, P. Wouters, *Guide to IPsec VPNs*, 2020. doi:10.6028/NIST.SP.800-77r1.
- [10] N. Ferguson, B. Schneier, *A cryptographic evaluation of IPsec* (1999).
- [11] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, T. Kivinen, *Internet Key Exchange Protocol*

Version 2 (IKEv2), RFC 7298, RFC Editor, 2014. URL: <https://www.rfc-editor.org/rfc/rfc7296.txt>.

- [12] D. Harkins, D. Carrel, The Internet Key Exchange (IKE), RFC 2409, RFC Editor, 1998. URL: <https://www.rfc-editor.org/rfc/rfc2409.txt>.
- [13] A. C. V. GmbH, Connecting the FRITZ!Box with a company's VPN, 2023. URL: <https://en.avm.de/service/vpn/connecting-the-fritzbox-with-a-companys-vpn-ipsec/>, accessed: 2023-03-03.
- [14] P. Fiterau-Brostean, R. Janssen, F. W. Vaandrager, Combining model learning and model checking to analyze TCP implementations, in: S. Chaudhuri, A. Farzan (Eds.), Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part II, volume 9780 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 454–471. doi:10.1007/978-3-319-41540-6_25.
- [15] C. M. Stone, T. Chothia, J. de Ruiters, Extending automated protocol state learning for the 802.11 4-way handshake, in: J. López, J. Zhou, M. Soriano (Eds.), Computer Security - 23rd European Symposium on Research in Computer Security, ESORICS 2018, Barcelona, Spain, September 3-7, 2018, Proceedings, Part I, volume 11098 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 325–345. doi:10.1007/978-3-319-99073-6_16.
- [16] L.-A. Daniel, E. Poll, J. de Ruiters, Inferring OpenVPN state machines using protocol state fuzzing, in: 2018 IEEE European Symposium On Security And Privacy Workshops (EuroS&PW), IEEE, 2018, pp. 11–19.
- [17] T. Novickis, E. Poll, K. Altan, Protocol state fuzzing of an OpenVPN, Ph.D. thesis, PhD thesis. MS thesis, Fac. Sci. Master Kerckhoffs Comput. Secur., Radboud Univ, 2016.
- [18] J. Guo, C. Gu, X. Chen, F. Wei, Model learning and model checking of ipsec implementations for internet of things, *IEEE Access* 7 (2019) 171322–171332. doi:10.1109/ACCESS.2019.2956062.
- [19] E. Muskardin, B. K. Aichernig, I. Pill, A. Pferscher, M. Tappler, AALpy: an active automata learning library, *Innov. Syst. Softw. Eng.* 18 (2022) 417–426. doi:10.1007/s11334-022-00449-3.
- [20] R. R. S, R. R, M. Moharir, S. G, Scapy - a powerful interactive packet manipulation program, in: 2018 International Conference on Networking, Embedded and Wireless Systems (ICNEWS), 2018, pp. 1–5. doi:10.1109/ICNEWS.2018.8903954.
- [21] R. L. Rivest, R. E. Schapire, Inference of finite automata using homing sequences, *Inf. Comput.* 103 (1993) 299–347. doi:10.1006/inco.1993.1021.
- [22] A. Pferscher, B. K. Aichernig, Stateful black-box fuzzing of Bluetooth devices using automata learning, in: J. V. Deshmukh, K. Havelund, I. Perez (Eds.), NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings, volume 13260 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 373–392. doi:10.1007/978-3-031-06773-0_20.
- [23] A. Pferscher, B. Wunderling, Supplemental material “Mining Digital Twins of a VPN Server”, 2023. doi:10.6084/m9.figshare.21953222.v1, accessed: 2023-01-25.

A. Learned Models

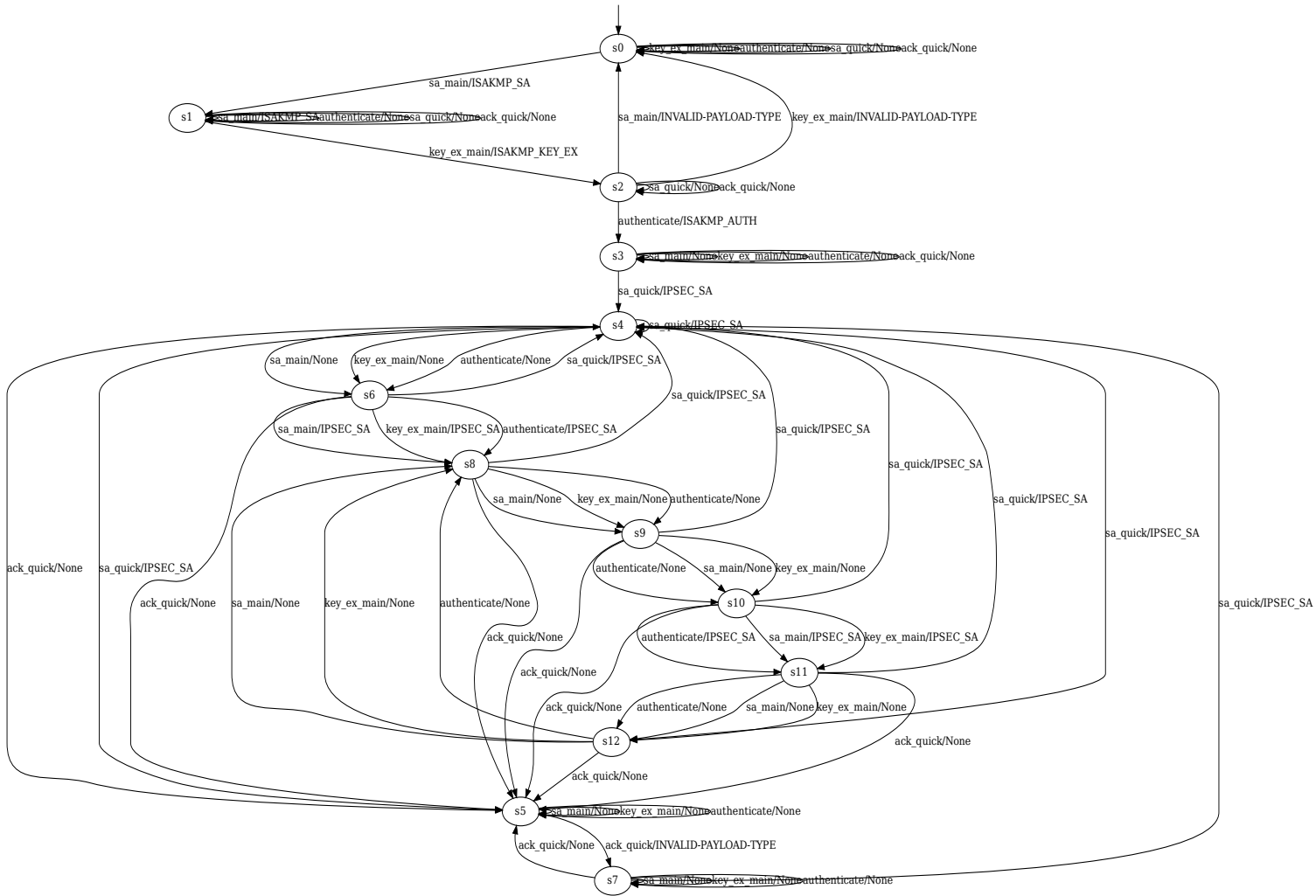


Figure 4: Strongswan Server automata with retransmissions. The model took approximately 200 minutes (12187 seconds) to learn with the L^* algorithm, spread over five learning rounds. During learning, 761 output queries were performed in 7135 steps. On average, three to four non-determinism errors were caught and fixed per learned model, arising from differing timings causing retransmissions to be sent at different points.