IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON DISTRIBUTED COMPUTING INFRASTRUCTURE FOR CYBER-PHYSICAL SYSTEMS

Received November 6, 2019, accepted November 20, 2019, date of publication November 26, 2019,
date of current version December 11, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2956062

Model Learning and Model Checking of IPSec Implementations for Internet of Things

JIAXING GUO[1], CHUNXIANG GU[1,2], XI CHEN[1], AND FUSHAN WEI[1]

[1]Henan Key Laboratory of Network Cryptography Technology, Zhengzhou 450002, China
[2]State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China

Corresponding author: Jiaxing Guo (guojiaxing124lab@gmail.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61772548, and in part by the Foundation of Science and Technology on Information Assurance Laboratory under Grant KJ-17-001.

ABSTRACT With the development of Internet of Things (IoT) technology, the demand for secure communication by smart devices has dramatically increased, and the security of the IoT protocol has become the focus of cyberspace. Recently, some scholars have attempted to extend the IPSec protocol to IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN) to ensure end-to-end security, which makes it essential to analyze the vulnerability of the IPSec protocol to enhance the security of the IoT. In this study, we use a method combining model learning and model checking to analyze the dynamic vulnerability of IPSec protocol implementations. This method automatically infers the black-box model and compares it with the relevant specifications to expose the defects of the system implementation and search its logic vulnerabilities. We first employ model learning on three IPSec implementations to infer state machine models; then, we use model checking to verify that these models satisfy basic security properties and conform to the RFCs. Our analysis reveals three new security issues: a wrong interaction causing server exception and two violations of the standard.

INDEX TERMS IPSec protocol, model learning, model checking.

I. INTRODUCTION

Internet of things (IoT) is regarded as the third revolution of information technology industry development after the computer and Internet. Its ubiquitous network characteristics make it possible to connect everything. In recent years, with the wide application of IoT technology and the improvement of security requirements, the application of security protocols such as IPSec, DTLS, OSCORE, and so on, in IoT to enhance its security have been explored [1]. Despite the limitations of power consumption and performance, IPSec is still a feasible security protocol for IoT. After the expansion of IPv6 over Low-Power Wireless Personal Area Networks (6LoWPAN), IPSec can make each device node communicate confidentially, thus realizing end-to-end security [2]. Owing to the natural inheritance of IoT, malicious attacks against the Internet began to spread to the field of IoT. Therefore, the vulnerability analysis of the IPSec implementation still has a significance for improving the security of IoT.

The associate editor coordinating the review of this manuscript and approving it for publication was Zhen Ling.

Protocol vulnerability analysis methods include static analysis and dynamic analysis. Static analysis methods, such as traditional formal analysis, are less automated and often modeled manually based on documentation. In this way, some errors have been detected, such as in [3] and [4]. However, as observed in [5], the relationship between the manual model of the protocol and the corresponding standard is often ambiguous, thereby undermining the reliability and relevance of the verification results obtained. Besides, the implementation of the protocol often does not conform to its specifications, which means this model checking method can never capture implementation-specific errors.

The inferring protocol state machine is a critical technology of vulnerability analysis. It builds a model of protocol implementation based on the state machine derived by the model learning method. Compared with the traditional static method, this technology is a dynamic black-box test, which does not depend on the source code and documents and has a high degree of automation. Currently, this technology has been applied to infer multiple state machines of security protocol implementations. De Ruiter and Poll [6] used this method to perform black-box testing on nine TLS

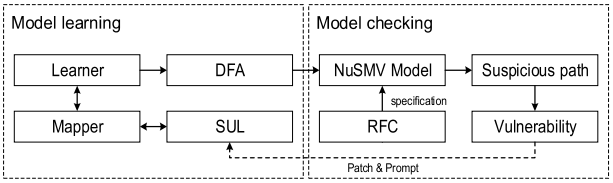

FIGURE 1. Process of the Protocol Test. DFA indicates the deterministic finite automaton and SUL demonstrates the system under learning.

implementations for inferring state machines and found three new security vulnerabilities. In [7], Stone *et al.* performed an automated analysis of seven Wi-Fi implementations and identified three new implementation flaws. In [8]–[10], protocol state fuzzing was also used to analyze OpenVPN, TLS1.3, and QUIC protocol implementations. These contributions show that this method is effective for analyzing vulnerability and can be migrated between different protocols, but the above studies use manual analysis of the model. Manual analysis is a time-consuming and laborious task, which requires a debugging personnel to master specific protocol knowledge to filter out abnormal test paths. To improve manual tests, model checking tools that aid in the analysis of models can be applied. In [11], Brostean first used the model checking tool to examine the learned model implemented by the TCP protocol, thus, avoiding manual testing. After that, Brostean further used the model checking method to analyze the learned SSH protocol model and pointed out the behavior of the implementation that did not conform to the specification [12].

In this paper, we combine model learning and model checking to perform vulnerability analysis on IPSec implementation for increasing the security of IoT. The specific test is divided into the following two steps:

The first step is model learning. Based on the model learning algorithm, using the classic Minimally Adequate Teacher (MAT) framework that relies on the interaction of learners, mappers, and the system under learning (SUL), we can automatically infer the state machine model of the specific protocol and obtain the corresponding deterministic finite automaton (DFA).

In the second step, based on the DFA inferred in the first step, we use the formal security criteria defined by the model checking tool and described by linear temporal logic to automatically compare the model with the relevant specifications, thereby acquiring counterexamples that guide us to search for the vulnerabilities in the implementation. Fig. 1 depicts the overall flow of the test.

To enhance the pertinence and practicability of the test, we analyze only the IKEv2 handshake protocol and ESP protocol in the IPSec protocol suite. IKEv2 is the latest version of the handshake negotiation protocol that overcomes the overcomplicated defects of the old version of IKEv1 and uses a more secure authentication method. The ESP provides confidentiality protection services and is more secure than AH; furthermore, it better meets the requirements of secure communication. We analyzed three widely used IPSec implementations (Strongswan, Libreswan, and Windows Server), obtained corresponding learned models and test results, and found three new defects in Libreswan. One of the flaws is the wrong interaction that caused the server to be abnormal (CVE-2019-12312 [13]), and the attacker only needs to send two packets to cause the server to restart. The other two defects violate the IKE SA management standard.

The rest of this article is organized as follows. A background on model learning, model checking, and the IPSec protocol is provided in Section 2. Section 3 shows the specific methods and results of model learning. In Section 4, we use model checking to analyze the resulting model. Finally, the full paper is summarized in Section 5, and future work is presented.

II. PRELIMINARIES

In this section, we describe the basic principles of model learning and model checking methods and introduce the basics of the IPSec protocol.

A. MODEL LEARNING

Model learning is a dynamic analysis method for learning the black box system behavior model. It provides an accurate and reliable model to restore the actual internal structure and operation of the system. Ordinarily, we need to make the following three reasonable assumptions:

1) The final result is a finite state machine.
2) The learner knows all valid input and output of the target system.
3) The target system can answer any query and give a definite answer to each query.

These three assumptions ensure that we can obtain the final state diagram through specific methods. Next, we shall introduce the method of describing the finite state machine and the main framework of model learning.

1) MEALY MACHINES

We use the Mealy machine to describe the state machine of the system.

Definition 1: A Mealy machine is a tuple $M = (I, O, Q, q_0, \delta, \lambda)$, where I is a finite set of inputs, O is a finite set of outputs, Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times I \rightarrow O$ is a transition function, and $\lambda : Q \times I \rightarrow O$ is an output function.

The Mealy machine is used to represent the internal state machine of the system implementation. Here, we only introduce the necessary information needed to explain the methods and results of this paper. For further descriptions, please refer to [14], which discusses the relationship between the Mealy machine and the inferred state machine in more detail.

The Mealy machine corresponds to a graphical representation that typically describes the state as a node and the state transition as an edge. Fig. 2 shows a graphical example of a simple Mealy machine. For example, when the Mealy machine is in a state $q_0 \in Q$ and receives as input $X \in I$,

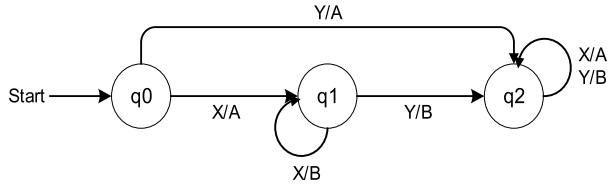

FIGURE 2. A simple DFA.

the state transition function operates $\lambda(q_0, X) = A$ and creates output $\delta(q_0, X) = q_1$. The corresponding Mealy machine is shown in Fig. 2 with the edge from q_0 to q_1 and the label X/A, which means that inputting X in q_0 results in output A with a changing current state to q_1.

Furthermore, the output function λ can be extended to a function with a sequence as input: $\lambda(q, i\sigma) = \lambda(q, i)\lambda(\delta(q, i), \sigma)$ and $\lambda(q, \varepsilon) = \varepsilon$, here $q \in Q$, $i \in I$, non-empty sequence $\sigma \in I^*$, and an empty sequence ε. The behavior of Mealy machine M is defined by $A_M : I^* \to O^*$, which means $A_M(\sigma) = \lambda(q_0, \sigma), \sigma \in I^*$. In particular, Mealy machines M and N are equivalent if and only if $A_M = A_N$, denoted as $M \approx N$. Meanwhile, the sequence $\sigma \in I^*$ is distinguished between M and N if and only if $A_M(\sigma) \neq A_N(\sigma)$.

The Mealy machine is a deterministic finite state machine with up to one corresponding conversion and output for the each combination of state and input. It has some key features that are useful for modeling our state machines:

1) Transition function: The Mealy machine uses both the current state and the input to determine the output, which emphasizes that different states affect the output of the implementation.
2) Deterministic: The same state and input sequence always results in the same output, which emphasizes that the behavior of the target should be consistent.
3) Finite set of states: The number of input symbols and states corresponding to the Mealy machine is limited. This point emphasizes that the state that appears in the target can only be a finite number; that is, the final result must be a finite state machine.

2) MINIMALLY ADEQUATE TEACHER FRAMEWORK

The normal method for inferring state machine is Angluin's L* algorithm [14], which corresponds to the Minimally Adequate Teacher (MAT) framework [15], [16]. The executor of the algorithm consists of two parts: the learner and Oracle (or teacher). The learner is the executor of the algorithm and knows the input and output symbol sets I and O. The Oracle acts as an interface to perform SUL (system under learning) and can respond to the learner's query, which is why the Oracle is called a minimally adequate teacher.

The algorithm is divided into two steps:

1) The membership query. In the first phase, the learner sends the string $\sigma \in I^*$ as a query to the system, and Oracle responds with the corresponding output string $A_M(\sigma)$. After each round of complete queries, the learner will send a reset query and ask Oracle to reset the SUL to the start state q_0 for ensuring the consistency of subsequent operations. The L* algorithm can decide which queries to send to the SUL as needed, and through the membership query, the learner establishes the hypothesis H for the state machine in the SUL.
2) The equivalent query. The learner asks Oracle to determine whether the hypothesis H is equivalent to the Mealy machine of the actual system by a particular method, that is $H \approx M$. If the Oracle's answer is yes, the algorithm will terminate and the state machine H will be given, otherwise Oracle will return a counterexample $\sigma \in I^*$ to satisfy $A_H(\sigma) \neq A_M(\sigma)$, which means the counterexample as the input will cause H and M to get different outputs. The learner uses the counterexample to improve the hypothesis and continue the learning process until an acceptable one is obtained. It should be noted that the learner can only make an approximate equivalence check on the hypothesis because Oracle cannot access the internal implementation of SUL (black box) and can only perform a limited number of test cases. As a result, the output state machine may only have a subset of SUL behavior.

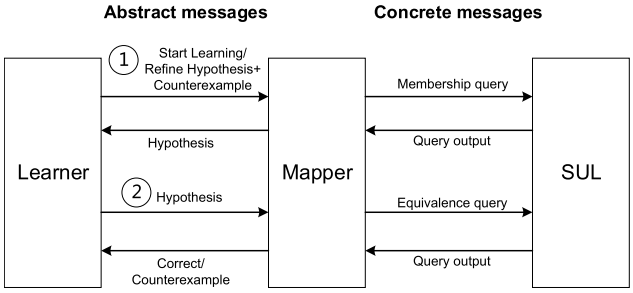

FIGURE 3. MAT Framework.

Based on the assumptions of previous model learning, we believe that a certain SUL state machine can be obtained after a limited number of queries. The number of queries required mainly depends on the complexity of the state machine to be inferred. The time complexity is $O(|\Sigma|mn^2)$, where $|\Sigma|$ corresponds to the size of the alphabet, m indicates the longest length of counterexample from Oracle, and n refers to the number of states in the state model of target SUL. It should be noted that the uncertain behavior in the operation of the algorithm may lead to an increase in state or result in the failure of the L* algorithm. Therefore, a corresponding detection of uncertain behavior needs to be added in the test.

To learn a specific protocol implementation system, a mapper needs to be placed between the learner and the target system. The mapper is used to record the state of the target system in real-time and implement the conversion of abstract messages and specific messages. Fig. 3 shows the logical relationship of MAT framework.

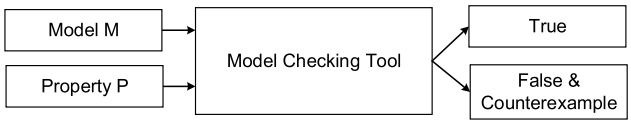

FIGURE 4. Model checking process.

Furthermore, we also need to fine-tune the timing parameters in the mapper by manual debugging to reduce the number of non-deterministic behavior and ensure the mapper cooperates with the system interaction in a better performance. Finally, after the above process, we can obtain the finite state machine model of the reaction target system.

B. MODEL CHECKING

Model checking is a property verification method based on the checking algorithm. It aims to determine whether a given behavior in a system class satisfies a design specification corresponding to a given temporal logic formula for a given finite state system.

As shown in Fig. 4, the input of model checking includes two parts: model M of the system to be verified and the description of property P for checking security attributes. If model M satisfies property P, the model checking tool outputs ''true''; otherwise a counterexample is given to explain why M does not satisfy property P. System modeling, description of property, and verification of the model checking tool constitute the three main parts of model checking. In this study, we chose linear temporal logic (LTL) to describe the specification.

1) LINEAR TEMPORAL LOGIC

Linear temporal logic uses a linear, discrete, and natural number-isomorphic time structure to describe all possible computational path attributes and takes the path (state sequence) as the proposition object. Its semantics and grammar are based on the Manna and Pnueli [17] temporal logic framework.

LTL grammar consists of three parts: propositional variables, logical operators, and temporal modal operators. Propositional variables p and q are used to describe the basic system attributes. Logical connectors include $\wedge$ (and), $\vee$ (or), $\neg$ (non), $\rightarrow$ (implication), etc. They are used to express complex conditional statements. Temporal modal operators include G (global), X (next), O (once), and so on. They represent the corresponding temporal logic. For example, if Gp is true, the next sequence must satisfy the condition p, and if Xp is true, the P holds in the next state. If Op is true, a previous state condition P holds.

The grammar of linear temporal logic can be defined recursively as follows:

1) Propositional constants {true, false} and atomic propositional variables (p, q, r. . .) are all linear temporal logic formulas.
2) If p and q are linear temporal logic formulas, then $\neg$p, p $\wedge$ q, and p $\vee$ q are also linear temporal logic formulas; Xp, Gp, and Op are also linear temporal logic formulas.
3) Each linear temporal logic formula can be obtained by using the finite number of the above constructions.

Formulas derived from recursive definitions can be used to express more complex temporal logic specifications. A model checking tool searches the transformation space of the transformation system in detail to check whether the corresponding specifications are satisfied. If not, counterexamples are generated.

C. IP SECURITY PROTOCOL

IP Security (IPSec) [20]–[23] is a security protocol running in the IP layer that can be used for end-to-end confidential communication. Given the security risks that the IP layer may face, IPsec provides three security services: confidentiality, integrity verification, and identity authentication, and resists replay attacks. IPSec provides many different modes and supports almost all encryption algorithms. Implementers can freely choose cryptographic algorithms to meet the corresponding needs and fulfil their needs in different scenarios.

IPSec is a protocol family that consists of three subprotocols: the Internet Key Exchange (IKE) protocol, the Authentication Header (AH) protocol, and the Encapsulating Security Payload (ESP) protocol. The IKE protocol is used for identity authentication, cryptographic algorithm selection, and key establishment, while AH and ESP are used for secure communication.

IKE is the signaling protocol of IPSec. It manages IPSec secure communication by securely maintaining Security Association (SA) among multiple parties. SA stores the identity and key information of both parties. The two parties first establish IKE SA, and then negotiate to generate IPSec SA in the corresponding encryption channel of IKE SA. IPSec SA is used for ESP or AH secure communication, while IKE SA is responsible for managing IPSec SA.

At present, IKE has two versions. The original design of IKEv1 [20] has been criticized for its complexity and its large number of options. Its successor IKEv2 [21] is simpler and can guarantee secure negotiation and authentication functions. Meanwhile, because the ESP protocol is applied in most practical scenarios, we choose the IKEv2 and ESP protocols as research objects.

IKEv2 mainly contains four types of message:

1) Initial Exchange (IKE_SA_INIT)
2) Authentication Exchange (IKE_AUTH)
3) Create Child SA Exchange (CREATE_CHILD_SA)
4) Informational Exchange (INFORMATIONAL)

The two sides of the communication first perform crypto parameter negotiation and identity authentication through IKE_SA_INIT and IKE_AUTH. At this time, the two parties respectively have an IKE SA and an IPSec SA, such that the parameters under the IPSec SA can be used for secure communication of the ESP protocol, and the session key can be rekeyed during the communication between the two parties. Finally, both parties delete the IPSec SA and IKE

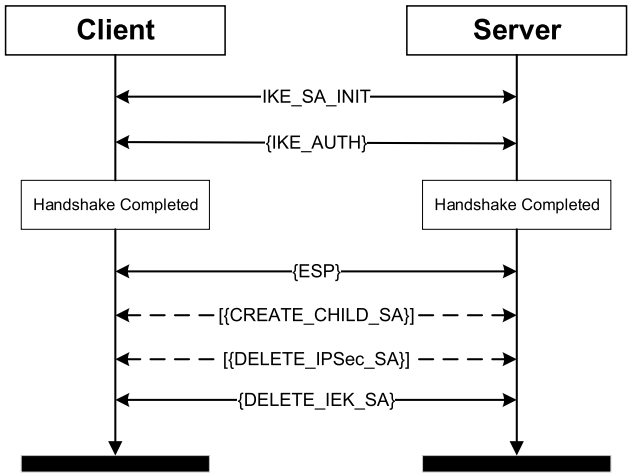

FIGURE 5. Regular IPSec session based on IKEv2. An encrypted message m is denoted as {m}. If message m is optional, this is indicated by [m].

SA by the notification payload to close the communication. Fig. 5 shows the general flow of protocol interaction.

The Initial Exchange (IKE_SA_INIT) is used for version and cryptographic algorithm negotiation, and the seed key is generated by the public key cryptographic algorithm negotiation (DH or ECDH) to generate a subsequent session key.

The Authentication Exchange (IKE_AUTH) is used for identity authentication. Certificate and pre-shared key modes are used for authentication. When using certificate authentication, both sides need to extract the private key of their certificate to sign the ID data from ID payload and other information. The respondent uses the corresponding public key to verify the signature to complete authentication. Generally, the ID is taken from the user field information of the certificate, but the relevant specifications allow the ID to be inconsistent with the information from the field of certificate. In this round of interaction, both sides will produce an IPSec SA for secure communication of application data.

After the first four messages, the two sides complete the handshake phase and can use the ESP protocol for secure communication. ESP provides security services such as confidential communication, data source authentication, anti-replay attack, and integrity verification that are the secure communication schemes adopted by most IPsec VPNs.

The Create Child SA Exchange (CREATE_CHILD_SA) can be used to generate new IKE SA and IPSec SA to update the key for forward secrecy. IKE SA is used to control the following sub-IPSec SA. After updating IKE SA, the original sub-IPSec SA should be passed on to the new IKE SA.

The Informational Exchange (INFORMATIONAL) is mainly used to deliver control messages, such as sending information of different error types, sending the delete_IKE message to terminate the connection, etc. Generally, deleting IKE SA implies deleting IPSec SA.

III. ADAPTING MODEL LEARNING FOR IPSec

In this section, we adapt model learning for the implementation of the IPSec protocol and discuss the overall framework and process. We also introduce specific methodology details and problems encountered and give the relevant learning results.

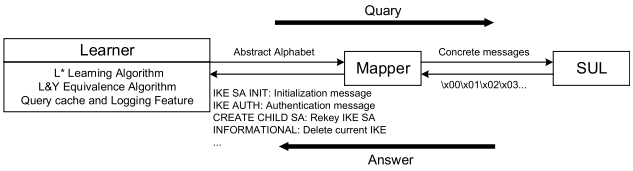

FIGURE 6. Setup model learning of IPSec implementations based on Learnlib.

A. LEARNING PROCESS

To infer the state machine implemented by the IPSec protocol, we use the information provided in Section 2 to build the following MAT learning framework with an L* learning algorithm as the core method. The framework consists of three parts:

The learner uses the LearnLib [22] library, which efficiently implements the L* algorithm and generates test messages for each round of learning of the Mealy machine. The mapper uses the Scapy [23] library, which is used to send and receive specific IPSec data packets to coordinate the learner's interaction with the SUL. The SUL is implemented as a specific IPSec server to respond to the learner's queries normally.

These three components communicate through sockets, as shown in Fig. 6.

We need to provide LearnLib with a list of messages that can be sent to the SUL (also known as the input alphabet) and a reset command to initialize the SUL to its initial state for ensuring query consistency. By sending multiple rounds of message sequences and reset commands, LearnLib attempts to make assumptions for the state machine based on the response received from the SUL. Once the L* algorithm completes its state machine hypothesis, LearnLib inputs it to the equivalent algorithm to check whether the hypothesis is equivalent to the actual state machine. If the models are not equal, a counterexample is returned, and LearnLib will use it to redefine its hypothesis.

As the actual state machine is unknown, an equivalence check is needed to test the correctness of the hypothesis, which is a model-based test. We use the algorithm based on Lee and Yannakakis [24] to find adaptive distinguishing sequences, which can search for counterexamples more efficiently and give a certain degree of confidence in the final model.

The learner expects the SUL to be deterministic; however, the implementation exhibits non-deterministic behavior. The response time is primarily a source of uncertainty. For example, if the response time of the SUL exceeds the packet waiting time of the mapper, the mapper will recognize that the SUL has no response; as a result, some behaviors may be recognized as receiving a reply, and some will be identified as not responding, thereby causing inconsistent behavior.

TABLE 1. Learning alphabet of IPsec protocol.

Number	Exchange type	Message content
1	IKE_SA_INIT	INIT_IKE
2	IKE_AUTH	IKE_AUTH
3	IKE_AUTH	IKE_AUTH_emptyCert
4	IKE_AUTH	IKE_AUTH_wrongID
5	CREATE_CHILD_SA	create_child_ESP_SA_over_current_IKE
6	CREATE_CHILD_SA	create_child_ESP_SA_over_old_IKE
7	CREATE_CHILD_SA	rekey_IKE
8	INFORMATIONAL	delete_current_ESP_over_current_IKE
9	INFORMATIONAL	delete_old_ESP_over_current_IKE
10	INFORMATIONAL	delete_current_ESP_over_old_IKE
11	INFORMATIONAL	delete_old_ESP_over_old_IKE
12	INFORMATIONAL	delete_current_IKE_SA
13	INFORMATIONAL	delete_old_IKE_SA
14	ESP	test_current_IKE_current_ESP
15	ESP	test_current_IKE_old_ESP
16	ESP	test_old_IKE_current_ESP
17	ESP	test_old_IKE_old_ESP

In practice, it is necessary to continuously debug to eliminate this behavior as much as possible and set a reasonable timeout according to the type of query message to improve the efficiency and accuracy of learning.

To eliminate non-determinism, we also use the SQLite-based tracking and response logs to extend learners. For any (sub)tracking, the learner checks if the SUL response matches the previous answer. In the case of non-deterministic situations, the program will throw an exception and then perform a manual check. At the same time, these trace records and response logs can speed up the learning. In other words, if the cache needs to query the messages and results, there is no need to call the mapper, which enables us to continue learning from where the last experiment stopped by loading the previous cache.

We use the alphabets in Table 1 to interact with the server. For more details on related messages, please refer to RFC 7296 [21].

We used certificate authentication as the authentication method because all implementations support this form of authentication. We provided three authentication messages, which correspond to normal certificate, empty certificate, and error ID (The ID data in the ID payload of the IKE_AUTH message is inconsistent with the information from the certificate in the certificate payload). The RFC 7296 allows for implementation with cached certificates to accept empty certificate authentication and allows the ID information to be different from the ID field of the certificate. The mapper maintains a simple data structure to test the rekey_IKE and create_ESP_SA message, allowing for new SAs to be created only when the old SA is empty, thus preventing the creation of too many SAs and causing state explosions.

After the learning process, the program will output a state diagram in the form of Graphviz. We write a python script to preprocess the state diagram, delete the meaningless edge, and merge the same edge of the target node to improve the readability of the state diagram. The complete and modified state machine have been uploaded to Github[1] for readers to access.

TABLE 2. Statistics for learning experiments.

SUL	Version	States	Hypotheses	Mem. Q.	Test Q.	Total time
Strongswan	5.8	28	3	8586	23481	15h
Libreswan	3.27	57	4	17460	45813	33h
Winserver	2019-17763	27	2	8280	17834	11h

B. LEARNING RESULTS

We used the model learning method to test three specific IPSec servers, i.e., Strongswan 5.8, Libreswan 3.27, and Windows Server 2019. These three are currently popular IPSec servers, and some teams actively manage them.

Table 2 describes the exact version of the analyzed system and the statistical data of model learning, including the number of states in the learned model, the number of assumptions established, the number of members and equivalent queries, and the total learning time.

The results revealed that Libreswan had more states, while Strongswan and Windows Server had similar states. This is because only Libreswan allows for the managing of IKE SA and IPSec SA on the old IKE in the three implementations. Multiple states lead to more queries and longer learning times. We checked the models and found that the three models were different and could be used for fingerprint identification.

In the following subsection, we will discuss the characteristics of the corresponding server and reveal the corresponding defects according to the test results. As the Windows Server model is very close to that of the Strongswan, we will not discuss it here; instead, we will focus on the first two models. Since the complete model is too large to be displayed in the paper, we will highlight only the critical parts of the model. The thick green line of each model represents the correct interaction path, and the dotted red line indicates the problematic path.

1) STRONGSWAN

Strongswan [25] was initially based on the frees/WAN project; then, it was completely rewritten. Now it is an IPSec implementation supporting multiple authentications and encryption methods that can be used in various operating systems.

As shown in Fig. 7, a part of the model of Strongswan is shown. The green path is a regular interaction process, starting with IKE_SA_INIT and IKE_AUTH, and ending

[1] https://github.com/rainingInIsland/DFA_of_IPsec_Server

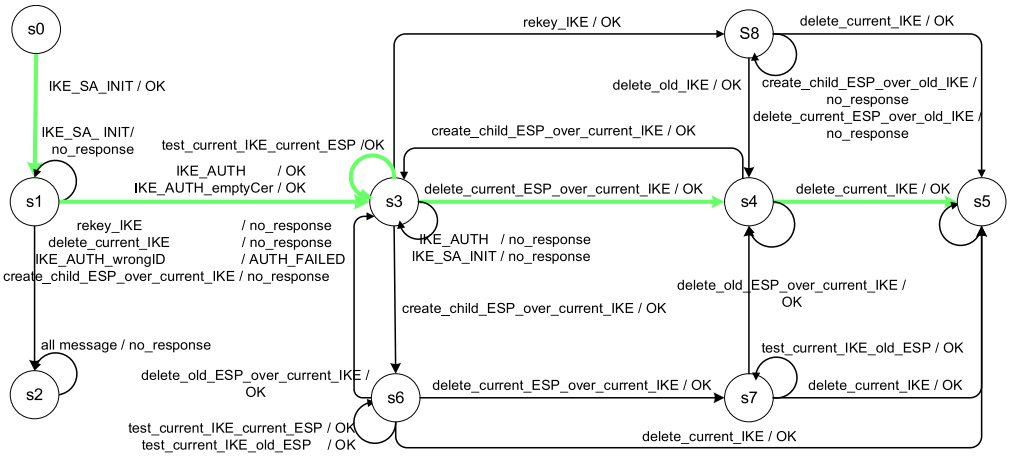

FIGURE 7. Simplified learned state machine model for Strongswan 5.8. The thick green edges highlight the happy flow. The label "i/o" close to its edge indicates the input and the responding output.

with delete_ESP_SA and delete_IKE. Strongswan's model did not have an abnormal path, and the interactive path was as expected. The transition from state 0 to state 1 is a regular IKE_SA_INIT interaction. The transition from state 1 to state 3 involves IKE AUTH and IKE AUTH_emptyCert interactions, which indicates that the Strongswan server supports empty certificate authentication after caching certificates.

State 3 is the state in which the handshake negotiation is completed. At this time, ESP message communication can be carried out without changing the state.

Deletion of the current ESP and IKE to disconnect occurs in state 3 through state 4 to state 5, and the subsequent state 5 conforms to what is expected with no output transition because this is the process designed to close the IKE SA connection.

As can be seen from State 3 and State 6 that deleting the ESP SA immediately after creation will result in the same state.

The Strongswan model also demonstrates the system's reasonable response to some anomaly tests. Resending IKE_SA_INIT in state 1, the system will ignore this message and will not cause a state transition. The transition from state 1 to state 2 is other than standard authentication, and the server enters a calm state, which is what we expect because the specification indicates that the IKEv2 interaction always begins with IKE SA INIT and IKE AUTH exchanges. At the same time, the server will reject interaction even if the regular authentication message is sent later.

According to state 8, the system does not respond to the creation and deletion of ESP packets on the old IKE channel. This indicates that Strongswan does not allow for the managing of sub-IPSec SAs on the old IKE SA.

2) LIBRESWAN

Libreswan does not rewrite the frees/WAN codebase but extends it. At present, it supports most IPSec related functions. Fig. 8 shows a part of Libreswan's model. The green path is a regular interaction process, starting with IKE_SA_INIT and IKE_AUTH, and ending with delete_ESP_SA and delete_IKE.

The transition from state 1 to state 3 involves IKE AUTH, IKE AUTH_emptyCert, and IKE AUTH_wrong, which indicates that the Libreswan server not only supports the empty certificate authentication after caching the certificate but also allows authentication in which the ID field is inconsistent with the certificate field.

Sending encrypted IKE_SA_INIT, rekey_IKE, and create_ESP in state 1 does not cause a state transition. From state 3 through state 4 to state 7 is the transition to delete the ESP and the current IKE; in this way, the communication is normally terminated.

In addition to standard behavior, we found some abnormal paths. Sending the delete_IKE message in state 1 will cause the system to enter a silent state. After the related log review, we find that the wrong packet interaction triggers the null pointer exception, which causes the IKE daemon to restart. The flaw has been successfully submitted (CVE-2019-12312 [13]). The new version of Libreswan 3.28 fixed the vulnerability five days later after the flaw was submitted.

State 6 and State 10 highlight the same problem, i.e., deleting the current ESP immediately after creating a new IPSec SA causes the old ESP to fail to communicate properly(note that the old ESP has not been deleted), which violates the relevant specifications.

Besides, according to the path after state 8, it can be seen that, after updating the IKE SA, the server still allows for the creation and deletion of the sub IPSec SA over old IKE SA. However, the RFC 7296 in section 2.8 indicates that "an IKE SA so created inherits all of the original IKE SA's Child SAs, and the new IKE SA is used for all control messages needed to maintain those Child SAs," indicating that Libreswan did not comply with this specification.

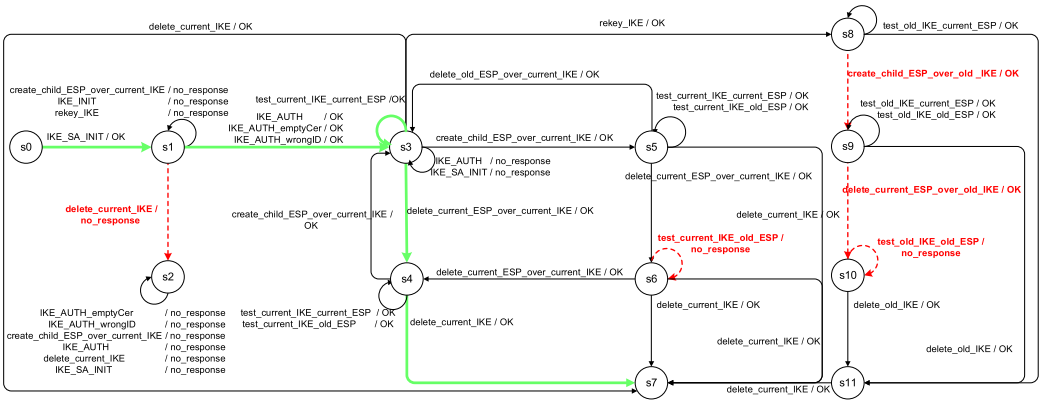

FIGURE 8. Simplified learned state machine model for Libreswan 3.27. The thick green edges highlight the happy flow and the dashed red transition indicates the function flaw.

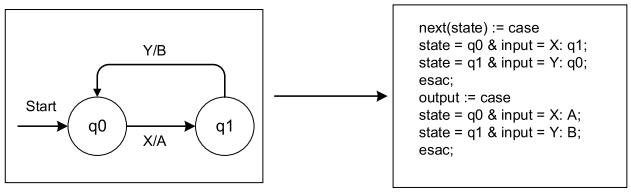

FIGURE 9. A simple example of DFA transformed to NuSMV model.

During the test, we also found that Libreswan's delete_IKE message syntax is different from that of Strongswan. This is because the related specifications do not strictly define the corresponding message construction, which may affect the communication between different system implementations.

IV. MODEL CHECKING

For the obtained state machine model, we use the LTL standard to describe the relevant security attributes and use the NuSMV [26] model checking tool to test the model. The NuSMV model can check the state of the model according to the defined specification. If an interaction path does not meet the specification, NuSMV will provide a counter example.

First, we need to convert the previous learned model Mealy machine into an NuSMV model for model checking tool identification. For example, a conversion $q_0 \rightarrow q_1$ containing the output and transition states in the Mealy machine will be mapped to two statements: $state = q_0$ & $input = X : q_1$ and $state = q_0$ & $input = X : A$ in the NuSMV model. As shown in Fig. 9, the conversion function of the Mealy machine can be easily described using the NuSMV related syntax.

Then, we use LTL to construct unique specifications for verifying the security attributes of the model. The security criterion formula consists of a set of finite propositional variable APs, logical operators, and time modal operators, and represents the corresponding meaning by a particular combination. The NuSMV can check the path of the security attribute that we need to verify. If non-conformance is found, the NuSMV will return a specific counterexample, which can guide us to find the corresponding vulnerability in the system implementation.

We use the model checking method to formalize the relevant attributes of the model and design the security specifications from three levels:

1) Fundamental security attributes, which are used to describe the basic security specifications of negotiation and interaction, to check the necessary authentication and confidentiality of the system, and to identify potential security risks such as man in the middle attack and DoS attack.
2) Rekeying attributes, which measure the status of creating new session keys to check the forward secrecy of the system and ensure the freshness of the key.
3) Functional attributes, which will be used to discuss the extra functional feature that may have an impact on safety according to RFC specification.

According to the above classification, we have specially designed 12 properties.

First, we define the following variables:

DEFINE InitFinished := (inp=init_IKE & out=ok);
DEFINE AuthReq:= (inp=IKE_AUTH | inp=IKE_AUTH_emptyCert | inp=IKE_AUTH_ wrongID);
DEFINE HasAuth := (AuthReq & out=ok);
DEFINE RekeyReq := (inp=rekey_IKE | inp=create_child_ESP_SA_over_current_IKE | inp=create_child_ESP_SA_over_old_IKE);
DEFINE DeleteIKEReq := (inp=delete_current_IKE_SA | inp=delete_old_IKE_SA);
DEFINE TestOldESP :=(inp=test_old_IKE_current_ESP | inp=test_old_IKE_old_ESP);
DEFINE TestOldIKE := (inp=create_child_ESP_SA_over_old_IKE | inp=delete_current_ESP_over_old_IKE | inp=delete_old_ESP_over_old_IKE);

According to RFC7296, communication using IKE always begins with IKE_SA_INIT and IKE_AUTH exchanges, and then CREATE_CHILD_SA and INFORMATIONAL

exchanges can occur. Thus, three specifications, 1–3, are formed.

Property 1: The remaining message negotiation must be performed after IKE_SA_INIT succeeds.

$$LTLSPEC\ NAME\ Init\_SA\ :=\ G\,(out\ !=\ None\ ->\ O\ InitFinished)$$

Property 2: IKE_AUTH must be completed following the right IKE_SA_INIT interaction.

$$LTLSPEC\ NAME\ Auth\_Sec\ :=\ G\,(HasAuth\ ->\ O\ (InitFinished\ \&\ out\ !=\ fail))$$

Property 3: ESP or INFORMATIONAL message communication must only be performed after successful authentication.

$$LTLSPEC\ NAME\ Connect\_Sec\ :=\ G\ ((inp = delete\_current\_IKE\_SA\ \&\ out = ok)|\ (inp = test\_current\_IKE\_current\_ESP\ \&\ out = ok)\ ->\ O\ HasAuth)$$

Properties 4–6 describe the situations after authentication.

Property 4: IKE cannot be reinitialized on the current IKE channel after successful authentication.

$$LTLSPEC\ NAME\ Auth\_Post\ :=\ G\,((HasAuth)\ ->\ X\ G\,(!\ InitFinished))$$

Property 5: The effect of successful authentication can last until disconnection, and the server will reject the request for re-authentication.

$$LTLSPEC\ NAME\ reAuth\_Sec\ :=\ G\,((HasAuth)\ ->\ XG(AuthReq\ ->\ out\ !=\ ok))$$

Property 6: After normal authentication, the system must agree to the delete IKE_SA request and respond to it.

$$LTLSPEC\ NAME\ Delete\_IKE\_Sec\ :=\ G\,(\ HasAuth\ ->\ G\,(DeleteIKEReq\ ->\ (out! = fail\ \&\ out! = no\_response)))$$

In terms of rekeying attributes, RFC7296 states that the function of rekeying the SA is optional for implementation. After rekeying the SA, it should be noted that the new IKE_SA must inherit all the sub-SAs, thus forming the properties 7–10.

Property 7: It is allowed to rekey IKE_SA or create a child SA after authentication.

$$LTLSPEC\ NAME\ Rekey\_Sec\ :=\ G\,((RekeyReq\ \&\ out = ok)\ ->\ O\ HasAuth)$$

Property 8: Both parties can still communicate based on the IPSec_SA of the old IKE_SA.

$$LTLSPEC\ NAME\ Test\_Old\_ESP\ :=\ G\,(TestOldESP\ ->\ (out = ok\ |\ out = None))$$

Property 9: The new IKE_SA inherits the original IPSec_SA; that is, the old IKE cannot continue to manage the IPSec SA.

$$LTLSPEC\ NAME\ Test\_Old\_IKE\ :=\ G\,(TestOldIKE\ ->\ out! = ok)$$

Property 10: The new IKE_SA inherits the original IPSec_SA, and deleting an IKE SA implicitly closes any remaining Child SAs negotiated under it. This means that after deleting the current new IKE_SA, the ESP created by the old IKE should no longer be able to communicate.

$$LTLSPEC\ NAME\ Test\_Old\_ESP\_Post\ :=\ G\,(inp = delete\_current\_IKE\_SA\ \&\ out = ok\ ->\ G\,(TestOldESP\ ->\ out! = ok))$$

RFC7296 also points out that "this improves efficiency when the endpoints have certificate data cached and makes IKE less subject to DoS attacks." At the same time, "the Identification payload allow peers to assert an identity to one another. This identity may be used for policy lookup but does not necessarily have to match anything in the CERT payload." These instructions have produced properties 11–12.

Property 11: Allow caching of certificate data for authentication.

$$LTLSPEC\ NAME\ Test\_Empty\_Cert\ :=\ G\,(InitFinished\ ->\ X\,(inp = IKE\_AUTH\_emptyCert\ ->\ out\ =\ ok))$$

Property 12: Allow ID information to be inconsistent with the corresponding field of the certificate.

$$LTLSPEC\ NAME\ Test\_Wrong\_ID\ :=\ G\,(InitFinished\ ->\ X\,(inp = IKE\_AUTH\_wrongID\ ->\ out\ =\ ok))$$

We have checked the relevant safety specifications for the obtained model, and the results are summarized in Table 3.

The results reveal that the three servers generally follow the correct negotiation process and support the rekeying function in IPSec communication. It is worth noting that Libreswan allows IPSec SA to be managed on the old IKE channel(the new IKE SA does not thoroughly inherit the old IPSec SA), and sometimes, Libreswan does not reply to ESP test messages, thus not meeting properties 8–10 and affecting the system's forward secrecy and normal secure communication. These checking results also respond with the previous model learning results. In the forms of identity authentication, Strongswan does not allow the identity ID to be inconsistent with the client certificate, while Windows Server does not support the authentication method of the empty certificate. Only Libreswan supports the authentication in multiple situations.

TABLE 3. Model checking results.

Attribute	Property	Key word	Strongswan	Libreswan	Winserver
Fundamental security attributes	Prop. 1	MUST	√	√	√
	Prop. 2	MUST	√	√	√
	Prop. 3	MUST	√	√	√
	Prop. 4	MUST	√	√	√
	Prop. 5	MUST	√	√	√
	Prop. 6	MUST	√	√	√
Rekeying attributes	Prop. 7	OPTIONAL	√	√	√
	Prop. 8	MUST	√	×	√
	Prop. 9	MUST	√	×	√
	Prop. 10	MUST	√	×	√
Functional attributes	Prop. 11	OPTIONAL	√	√	×
	Prop. 12	OPTIONAL	×	√	√

V. CONCLUSION

In this study, we combined model learning and model checking to analyze three IPSec implementations. We performed a black box test on the target system based on model learning to infer the state machine model, then extracted the security attributes from the IPSec related specifications, and used the model checking method to automatically perform path detection. The entire process is almost completely automated, which increases the efficiency of the test. Our analysis found some vulnerabilities in the implementation of the IPSec system, including a false interaction causing server exceptions and some violations of the specification, thus demonstrating the effectiveness of our approach. The inferred state machine reveals the internal structure of the system, which helps us understand the functions inside the system and assists the developer in maintaining the system. As the model checking results can determine the difference between the implementation and the specification, the security problem may be discovered, and we can further improve it.

Further, this method can be extended to other different forms of authentication and encryption algorithms of the IKEv2 protocol to analyze the security of the system implementation under the corresponding protocol. In the future, the methods used in this paper can also be applied to analyze other protocol implementations, such as Zigbee and NBIoT, which is also a great advantage of inferring protocol state machine compared to other analysis methods. Besides, we can consider how to enrich the content of the model to detect more complicated security issues. In general, this study expands the application field of dynamic protocol tests and have made individual contributions in supplementing the security analysis methods of network security protocols.

REFERENCES

[1] S. Aragon, M. Tiloca, M. Maass, M. Hollick, and S. Raza, "ACE of spades in the IoT security game: A flexible IPsec security profile for access control," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Beijing, China, May/Jun. 2018, pp. 1–9.

[2] P. Varadarajan and G. Crosby, "Implementing IPsec in wireless sensor networks," in *Proc. 6th Int. Conf. New Technol. Mobility Secur. (NTMS)*, Dubai, United Arab Emirates, 2014, pp. 1–5.

[3] T. Ninet, A. Legay, R. Maillard, L.-M. Traonouez, and O. Zendra, "Model checking the IKEv2 protocol using Spin," in *Proc. 17th Int. Conf. Privacy, Secur. Trust (PST)*, Fredericton, NB, Canada, Aug. 2019, pp. 1–9.

[4] P. Fiterău-Bro tean, R. Janssen, and F. Vaandrager, "Learning fragments of the TCP network protocol," in *Formal Methods for Industrial Critical Systems*, vol. 8718, F. Lang and F. Flammini, Eds. Cham, Switzerland: Springer, 2014, pp. 78–93.

[5] H. Brinksma and A. H. Mader, *On Verification Modelling of Embedded Systems*. Enschede, The Netherlands: Univ. of Twente, Centre for Telematics and Information Technology, 2004.

[6] J. De Ruiter and E. Poll, "Protocol state fuzzing of TLS implementations," in *Proc. 24th USENIX Secur. Symp. (USENIX Secur.)*, 2015, pp. 193–206.

[7] C. M. Stone, T. Chothia, and J. de Ruiter, "Extending automated protocol state learning for the 802.11 4-way handshake," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, 2018, pp. 325–345.

[8] L.-A. Daniel, E. Poll, and J. de Ruiter, "Inferring OpenVPN state machines using protocol state fuzzing," in *Proc. IEEE Eur. Symp. Secur. Privacy Workshops (EuroS&PW)*, Apr. 2018, pp. 11–19.

[9] J. van Thoor, J. de Ruiter, and E. Poll, "Learning state machines of TLS 1.3 implementations," Bachelor thesis, Dept. Comput. Sci., Radboud Univ., Nijmegen, The Netherlands, 2018.

[10] A. Rasool, G. Alpár, and J. de Ruiter, "State machine inference of QUIC," Mar. 2019, *arXiv:1903.04384*. [Online]. Available: https://arxiv.org/abs/1903.04384

[11] P. Fiterău-Bro tean, R. Janssen, and F. Vaandrager, "Combining model learning and model checking to analyze TCP implementations," in *Computer Aided Verification*, vol. 9780, S. Chaudhuri and A. Farzan, Eds. Cham, Switzerland: Springer, 2016, pp. 454–471.

[12] P. Fiterău-Bro tean, T. Lenaerts, E. Poll, J. de Ruiter, F. Vaandrager, and P. Verleg, "Model learning and model checking of SSH implementations," in *Proc. 24th ACM SIGSOFT Int. SPIN Symp. Model Checking Softw. (SPIN)*, Santa Barbara, CA, USA, 2017, pp. 142–151.

[13] *CVE–CVE-2019-12312*. Accessed: Jul. 11, 2019. [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-12312

[14] D. Angluin, "Learning regular sets from queries and counterexamples," *Inf. Comput.*, vol. 75, no. 2, pp. 87–106, Aug. 1987, doi: 10.1016/0890-5401(87)90052-6.

[15] O. Niese, "An integrated approach to testing complex systems," Ph.D. dissertation, Tech. Univ. Dortmund, Dortmund, Germany, 2003.

[16] M. Shahbaz and R. Groz, "Inferring mealy machines," in *Proc. Int. Symp. Formal Methods*, 2009, pp. 207–222.

[17] B. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*. New York, NY, USA: Springer-Verlag, 1992.

[18] D. Piper, *The Internet IP Security Domain of Interpretation for ISAKMP*, document RFC 2407, 1998. [Online]. Available: https://tools.ietf.org/html/rfc2407

[19] D. Maughan and M. Schneider, *Internet Security Association and Key Management Protocol (ISAKMP)*, document RFC 2408, 1998. [Online]. Available: https://tools.ietf.org/html/rfc2408

[20] D. Carrel and D. Harkins, *The Internet Key Exchange (IKE)*, document RFC 2409, 1998. [Online]. Available: https://tools.ietf.org/html/rfc2409

[21] T. Kivinen, P. Hoffman, C. Kaufman, Y. Nir, and P. Eronen, *Internet Key Exchange Protocol Version 2 (IKEv2)*, document RFC 7296, 2014. [Online]. Available: https://tools.ietf.org/html/rfc7296

[22] H. Raffelt, B. Steffen, T. Berg, and T. Margari, "LearnLib: A framework for extrapolating behavioral models," *Int. J. Softw. Tools Technol. Transf.*, vol. 11, no. 5, p. 393, 2009, doi: 10.1007/s10009-009-0111-8.

[23] P. Biondi, "Scapy," 2011.

[24] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines—A survey," *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, Aug. 1996, doi: 10.1109/5.533956.

[25] A. Steffen. (2017). *StrongSwan—The OpenSource IPsec-Based VPN Solution*. Accessed: Mar. 31, 2012. [Online]. Available: http://www.strongswan.org

[26] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NUSMV 2: An OpenSource tool for symbolic model checking," in *Proc. Int. Conf. Comput. Aided Verification*, 2002, pp. 359–364.

JIAXING GUO is currently pursuing the M.S. degree in information security from the Henan Key Laboratory of Network Cryptography Technology, Zhengzhou, China. His recent research interests include protocol analysis, model learning, and model checking.

CHUNXIANG GU was born in 1976. He is currently a Professor and a Ph.D. Supervisor with the State Key Laboratory of Mathematical Engineering and Advanced Computing. His research interests include network security and cryptography.

XI CHEN was born in 1988. He received the bachelor's degree in computer science and technology from Tsinghua University and the master's degree from Information Engineering University. He is currently a Lecturer with the Henan Key Laboratory of Network Cryptography Technology. His research interests include cyberspace security and cryptography.

FUSHAN WEI received the M.S. and Ph.D. degrees in applied mathematics from the Zhengzhou Information Science and Technology Institute, China, in 2008 and 2011, respectively. He is currently an Associate Professor with the Henan Key Laboratory of Network Cryptography Technology, Zhengzhou, China. His research fields include cryptography and information security.