A Methodology for Testing IPSec-based Systems

Uljana Boiko*, Alberto Ferrante†, Antonietta Lo Duca*, and Vincenzo Piuri†
* ALaRI Institute, University of Lugano
Lugano, Switzerland
Email: {boiko, loduca}@alari.ch
† Department of Information Technologies,
University of Milan
Milano, Italy
Email: {ferrante, piuri}@dti.unimi.it

***Abstract*— IPSec is a suite of protocols adding security to communications at the IP level. This suite of protocols is becoming more and more important as it is included as mandatory security mechanism in IPv6.**
This paper focuses on a methodology for testing IPSec implementations. A UML model of the IPSec suite of protocols was developed. Test cases were obtained applying a coverage method on the same model.

I. Introduction

IPSec is mainly composed of two protocols, Authentication Header (AH) and Encapsulating Security Payload (ESP). The former allows authentication of each IP datagram's headers or – depending on the operational mode that has been selected – of the entire IP datagram. The latter allows encryption – and optionally authentication – of the entire IP datagram or of the IP payload, depending on the operational mode that has been selected, namely the transport and the tunnel modes. The former was designed for being used in host machines, while the latter is for secure gateways. In tunnel mode the entire original IP datagram is processed; the result becoming the data payload of a new IP datagram with a new IP header. In transport mode only parts of the original IP datagram are processed (e.g. the data payload for the ESP protocol) and the original IP header is kept with some small modifications. Through encryption, authentication, and other security mechanisms included in IPSec (e.g. anti-reply), data confidentiality, data authentication, and peer's identity authentication can be provided [1], [2], [3]. In each of the protocols within the IPSec suite, many choices for cryptographic algorithms are available (for example AES, DES, Triple-DES, and many others can be used within the ESP protocol). The concept of Security Association (SA) is fundamental to IPSec. A Security Association is a simplex "connection" that afford security services to the traffic carried by it [4]. To secure typical bi-directional communication between two peers, two SAs (one in each direction) are required. Security services are afforded to a SA by the use of AH, or ESP, but not both. Security association establishment can be performed through a protocol named Internet Key Exchange (IKE) [3]. IKE is a two-phase protocol: in the first phase a bidirectional SA – named IKE SA – is established; in the second phase this SA is used to negotiate the parameters (protocol, protocol settings, keys, . . .) for the IPSec SAs to be created. The use of IKE is not mandatory for IPSec and SAs can be established either manually or through other suitable protocols. IKE is quite complex since it provides many operational modes and options. A second version of this protocol, IKEv2, is about to be released. At some extent this version will be simpler than the previous one [5]. Two databases are involved in processing IP traffic relative to security associations. These two databases are the Security Policy Database (SPD) and the Security Association Database (SAD). The former specifies the policies that determine the disposition of all IP traffic. The latter contains parameters that are associated with each SA. For each packet traversing the IP communication layer, the SPD needs to be queried. If, in conformance with the SPD, an IP datagram needs to be processed by IPSec, the SAD needs also to be queried to discover the parameters of the considered SA. Information about whether a SA has already been created or not are contained in the SPD. If a suitable SA for the IP datagram to be processed does not exist, it needs to be established, for example through IKE.

It is clear that IPSec is a complex suite of protocols (even when not considering IKE), therefore developing a test methodology and, most of all, writing a suitable set of test cases is not trivial. For network protocols, not only is it important to perform functional tests, but also to perform interoperability tests. The first ones are used to verify that an implementation works in a correct way (i.e. in conformance with its specifications), the others to verify that a specific protocol implementation can correctly interoperate with different implementations of the same protocol. Interoperability tests are described in some websites and papers like, for example, [6], [7]. By using high-level and implementation-independent functional testing, different implementations of the same protocol – being them done in software, hardware, or mixed hardware-software – can be easily tested. In this paper we therefore focus on this kind of testing. Theoretically, being able to guarantee the conformance of an implementation to its specifications (i.e. the RFCs), should also guarantee its interoperability with all the other different implementations of the same protocol, but actually none of the existing implementations of IPSec completely conforms to RFCs. Therefore interoperability tests

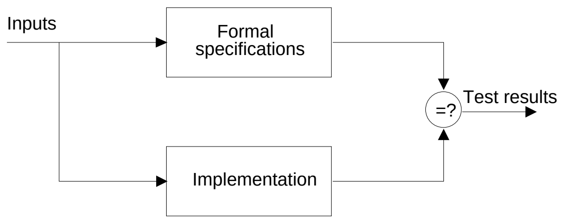

Fig. 1. Testing of a system using its specifications

need to be performed anyway. In order to develop a functional test for IPSec, many different test cases need to be considered even for a limited set of possible protocol configurations. Therefore a methodology for finding a good set of test cases needs to be put in place.

Being our work focused on high-level testing, we chose to model IPSec through the Unified Modelling Language (UML). In fact UML is a widely used language allowing a high-level and implementation-independent description of systems. This modelling language groups together static and dynamic diagrams that allow to describe structure and behavior of systems.

In [8] UML is used as a formal method for describing a protocol named PROFISafe. [9] describes how to generate test cases from finite state machine descriptions of protocols; in this paper external states of the protocol are considered for generating test patterns. In our work internal protocol states are used instead. Therefore the two papers cited above were only taken as starting point for our work. Some other papers like [10], [11], [12] describe other methodologies for protocol testing. [13] describes a tool for validating IPSec configurations and implementations.

Aim of this paper is to describe our test methodology and to prove that building an UML model of IPSec and using it to generate test cases is a viable solution. Therefore this paper focuses only on a part of IPSec: ESP in tunnel mode.

In the next sections we will describe the methodology we have used for functional testing, the test coverage method we have used, and the results we have obtained.

II. Testing Methodology

Our testing methodology is mainly based on using a model obtained from the IPSec specifications. This model can be used both for generating test sequences and for testing the considered implementation. Test sequences are ordered lists of inputs, i.e., protocol settings plus input patterns. Values taken from these sequences are given as input both to the protocol model and to the protocol implementation; the obtained results are then compared. Correct implementations must always give the same output as the model (supposing the model itself is correct and validated). Figure 1 shows in general how testing can be performed. In this diagram the implementation is intended to be run on a suitable machine, this means that obtained results may also depend on the software running on that machine (mostly on the operating system in case of an IPSec implementation).

As for any other complex system, obtaining a good test case coverage is not an easy task. Most of all, covering all the possible test cases, possibly only once, requires a suitable methodology. Describing the system that needs to be tested – IPSec in this case – by the means of a formal language, helps in obtaining a good test coverage. The unambiguous specifications given by a formal language, may also help obtain the test sequences automatically. The same model for the test execution phase can obviously be used to generate test sequences.

In this sections the obtained specifications and the methodology for generating test sequences are described.

A. IPSec Modelling

UML was chosen for its flexibility as modeling language. As stated above, UML is composed of different languages which allow to describe both static and dynamic aspects of the same system. *Class diagrams* are fit for describing static aspects of systems (or parts of them). A general view of the main components and of their connections is described by these diagrams. This representation was therefore chosen in our work to describe the main structure of IPSec. *Statecharts* diagrams allow to formally describe the dynamic behavior of systems (or parts of them). This language is an evolution of the *finite state machine* one. System behavior is represented through its states and possible transitions between them. One of the key advantages of statecharts on finite state machines is their capability to provide hierarchical representations. Statecharts can in fact be nested inside states of other higher-level statecharts. This is one of the main reasons why we decided to use this modeling language for describing the behavior of some parts of IPSec. Here follows a description of the main diagrams we have drawn.

1) Class Diagram of IPSec: Figure 2 shows the class diagram representing the parts of IPSec involved in IP packet processing and their connections. IPSec is composed of two protocols, AH and ESP that are represented by the *AHProtocol* and by the *ESPProtocol* classes. As both AH and ESP can be used for transport and tunnel mode, they are specialized to these modes through inheritance. *TransportAH* and *TunnelAH* are the specializations of *AHProtocol* in transport and in tunnel mode, respectively. *TransportESP* and *TunnelESP* are the specializations of *ESPProtocol* for transport and tunnel mode, respectively. *HeaderESP* and *TrailerESP* are classes implementing the ESP Header and Trailer generation. ESP header and trailer are special fields added to the normal IP headers and at the end of the IP datagram when ESP is used. The *newIPHeader* class is used in tunnel mode both for AH and ESP to generate the new IP headers that are needed in this mode.

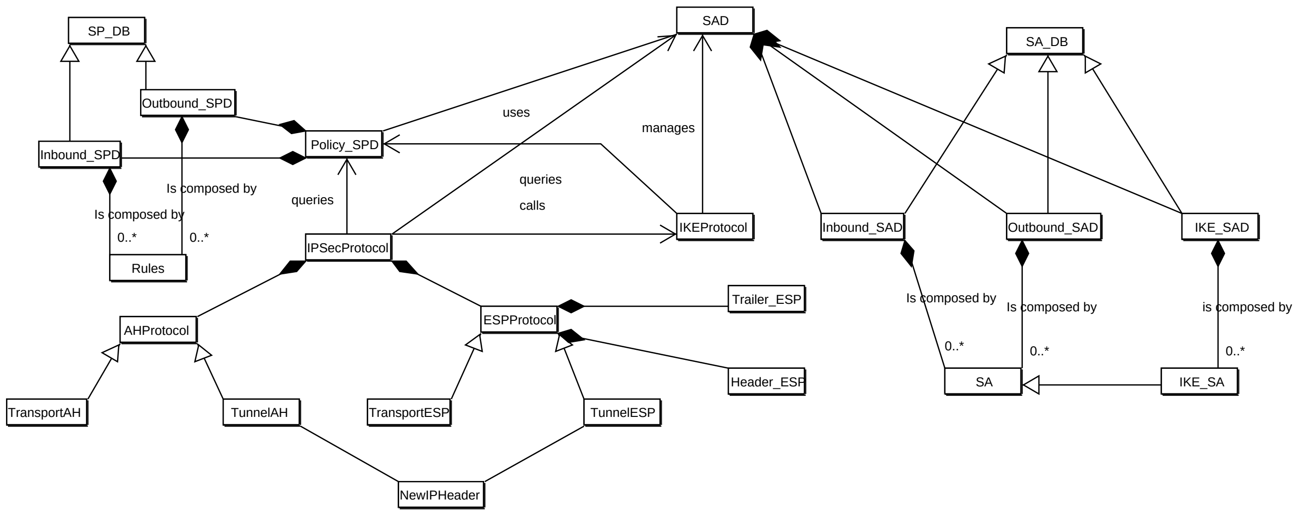

Fig. 2. UML class diagram of IPSec

The model shown here also includes the IETF IKE protocol. IKE was chosen, among all the other available ones, for implementing SA negotiation. In fact, if a security association has not yet been created and needs to be used - and this can be discovered querying the security policy database - a SA negotiation needs to take place. This functionality is here provided by the *IKEProtocol* class.

The *Policy_SPD* class manages the security policy databases, one for inbound and one for outbound traffic. These databases are contained in the *Inbound_SPD* and in the *Outbound_SPD*, respectively. inbound and outbound SPD databases are composed of records representing sets of rules for each connection (*Rules* class). These two databases are very similar and can be derived by a common class (*SP_DB*).

For each packet traversing the system, the security policy databases are queried through the *Policy_SPD class* to discover if any kind of IPSec processing needs to be applied on it. If it needs to and a suitable security association has already been created, the packet considered is processed by one of the specializations of the *AHProtocol* or of the *ESPProtocol* classes. Whether AH or ESP must be used is specified, along with all the needed parameters, in the security association database, managed by the *SAD* class. Also in this case two separate databases, *Inbound_SAD* and *Outbound_SAD*, need to be used for inbound and outbound SAs that have already been created. The *Inbound_SAD* and *Outbound_SAD* class can be derived by a base one, *SA_DB*. IKE SAs are kept in a separate database, represented by the *IKE_SAD* class. Records of this database are objects of the *IKE_SA* class. This class is derived from the *SA* one. In fact, IKE SAs are very similar to normal SAs; the main difference between these two types of SA is that IKE SAs are bidirectional, while normal SAs are monodirectional.

2) Statecharts: Statecharts were drawn for the main classes shown in the class diagram. So far only the ESP protocol in tunnel mode was completely modelled.

Figure 3 shows the statecharts diagram of the *TunnelESP* class. This diagram gives a behavioral description of the entire ESP protocol in tunnel mode. A similar diagram can be drawn for ESP in transport mode.

Figure 4 shows the statecharts diagram of the *Header_ESP* class. This diagram shows all the steps necessary to generate ESP headers.

Figure 5 shows the statecharts diagram of the *Trailer_ESP* class. This diagram shows all the steps necessary to generate ESP trailers. Our model contains other statecharts diagrams that are not shown here for space reasons.

B. Generation of Test Sequences

A formal language along with a coverage method can be used for generating test sequences [8], [12]. We used statecharts as formal language and the *transition coverage* method as coverage criterion. In this method all possible combinations of transitions are considered. Selecting a transition means selecting the necessary combinations of inputs allowing to move from the current state to one of the following states. These inputs correspond both to protocol settings and to input data patterns. Therefore this method allows to test all the combinations of parameters of the protocols providing changes in their internal states. Test patterns can be represented as follows:

$$S_I \, i_1, \, i_2, \, i_3, \, \ldots, \, i_n; \quad o_1, \, o_2, \, \ldots, \, o_m \, S_E$$

where:

- S_I is the initial state.
- $i_1, i_2, i_3, \ldots, i_n$ are the values of the inputs to be applied.
- $o_1, o_2, \ldots, o_m$ are the values of the obtained outputs.
- S_E is the final state.

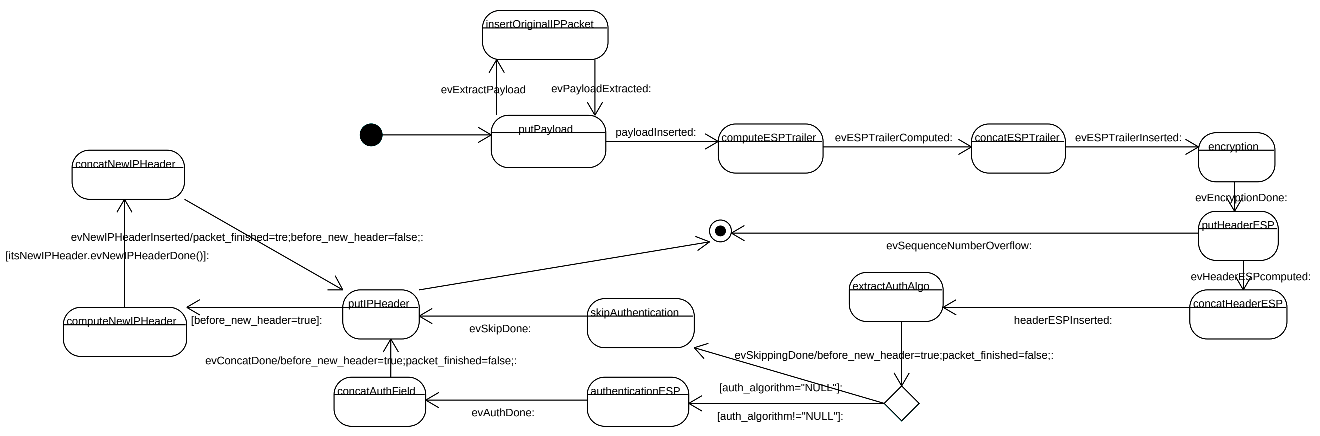

Fig. 3. Statechart of the *TunnelESP* class.

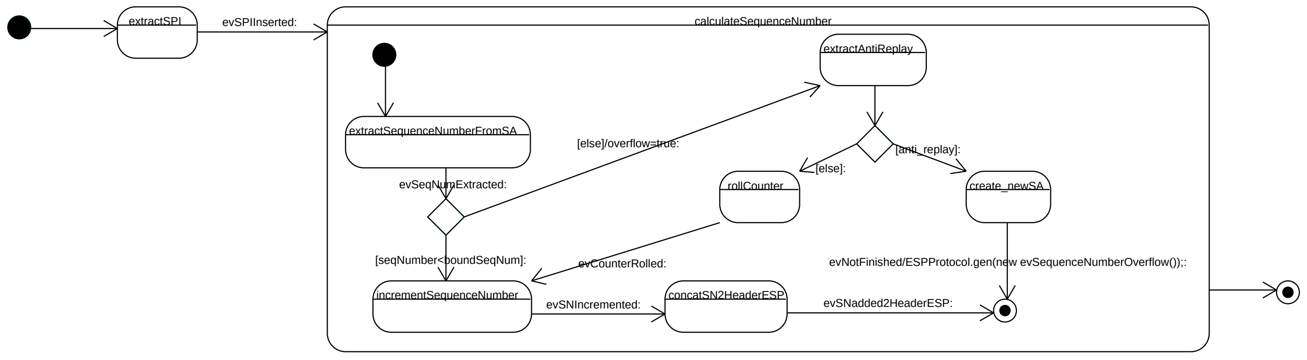

Fig. 4. Statechart of the *Header_ESP* class.

Outputs here included are the ones needed to verify the correctness of the outputs of the implementation. In fact these outputs are the ones obtained by applying the test inputs to the model.

The following algorithm is applied to generate test patterns:

1) Go to the initial state S_I.
2) For each of the possible transitions apply inputs activating the transition to state S_{i+1}. This state is the next one following the chosen transition. Put the obtained inputs in an ordered list.
3) Apply recursively step 1 and 2 considering S_{i+1} as initial state, until final state S_E is reached.

This procedure has to be applied recursively to all the nested statecharts. Each test pattern represents a sequence of inputs (i.e. configuration and datagram parameters) that allow to pass through a certain set of internal states. The application of all the test cases obtained allows to go through all the possible cases that may happen when the protocol itself is used. As stated earlier, the outputs obtained need to be compared with the ones generated by the implementation under test when the same sequence of inputs is applied.

III. Results

Our methodology was applied to the model of the ESP protocol in tunnel mode. In this way the related test cases were obtained. Test cases were generated by hand just as a proof of concept, but, once the UML model is completed and validated, a suitable tool can be used for doing that. The test cases obtained are 65. Not only do these cases include different configurations for the ESP protocol, but also all the different input patterns that are needed for testing. These 65 test cases should be repeated for each considered combination of algorithms within the ESP protocol. In this protocol a symmetric-key cryptographic algorithm is usually combined with an authentication one. Commonly used symmetric-key cryptographic algorithms are AES, DES, and Triple-DES. Most widely spread authentication algorithms are HMAC-SHA-1 and HMAC-MD5 but also the new HMAC-SHA-2 is becoming widely used. Therefore we can assume that at least 3 symmetric key cryptographic algorithms and 3 authentication algorithms need to be considered, thus having 585 test cases (not considering possible different parameters for the algorithms). Each test case requires a different configuration of the ESP protocol and this implies a renegotiation of the related security associations.

IV. Conclusions and Future Work

A test methodology for IPSec has been introduced in this paper. A UML model of a part of IPSec along with a test pattern generation methodology based on statecharts have been

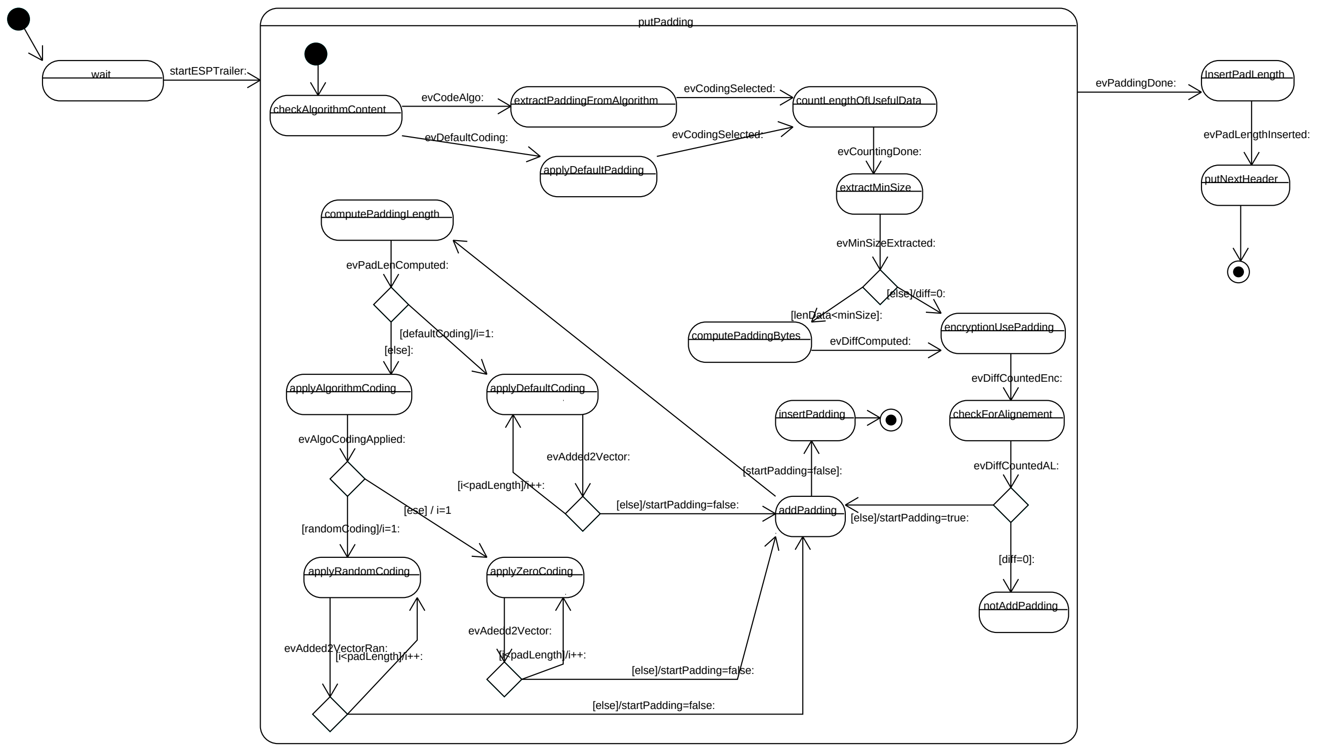

Fig. 5. Statechart of the *TrailerESP* class.

developed. The test pattern generation methodology allows to cover all the possible test cases for the protocols in the IPSec suite. So far ESP in tunnel mode is the only part of IPSec we have modeled by statecharts. Therefore the methodology we described in this document was applied on this protocol only. This allowed to show that our approach is viable. Giving a complete model of the IPSec suite of protocols is beyond the scope of this paper.

Future developments of this work include completing the UML model of IPSec and validating it. Test patterns for all the other parts of the system can be then generated. The coverage method described in this paper was applied to our model by hand (test cases were few enough to allow this approach), but once the model is validated, a tool for automatic test case generation can be conveniently used. Other coverage criteria may also be selected and the results obtained with these criteria can be compared with the ones obtained with the criteria adopted here. A study on how different algorithms may influence others within the same protocols, may help lowering the number of test cases that are needed. It may be, in fact, that not all the combinations of algorithms really need to be tested.

ACKNOWLEDGMENT

The authors would like to thank Microsoft Co. for having partially supported this research under the grant "Microsoft Embedded Systems RFP" awarded to the ALaRI Institute of the University of Lugano.

REFERENCES

[1] S. Kent and R. Atkinson, "IP Authentication Header – RFC2402," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[2] ——, "IP Encapsulating Security Payload (ESP) – RFC2406," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[3] D. Harkins and D. Carrell, "The Internet Key Exchange (IKE) – RFC2409," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[4] S. Kent and R. Atkinson, "Security Architecture For the Internet Protocol – RFC2401," IETF RFC, 1998. [Online]. Available: http://www.ietf.org/rfc.html

[5] C. Kaufman, "The internet key exchange (IKEv2) protocol," IETF Internet Draft, 22 Mar. 2004, expires: Semptember 2004. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-13.txt

[6] IPSec WWW-based Interoperability Tester (IPsec WIT). NIST. [Online]. Available: http://w3.antd.nist.gov/Groups/ITG/IP_Security/WIT/wit.html

[7] IPSec interoperability tester: Testing paradigm. NIST. [Online]. Available: http://ipsec-wit.antd.nist.gov/newipsecdoc/paradigm.html

[8] H. Ural and B. Yang, "A test sequence selection method for protocol testing," *IEEE Transactions on Communications*, April 1994.

[9] D. P. Sidhu and T.-K. Leung, "Formal methods for protocol testing: A detailed study," *IEEE Transactions on Software Engineering*, pp. 413–426, April 1989.

[10] K. L. Mills, "Testing OSI protocols: NBS advances the state of the art," *Data Communications Magazine*, March 1984.

[11] B. Sarikaya, "Protocol test generation, trace analysis and verification techniques," *Comput. Networks ISDN Syst.*, pp. 285–297, March 1988.

[12] R. Malik and R. Mühlfeld, "A case study in verification of UML statecharts: tre PROFIsafe protocol," *Journal of Universal Computer Science*, vol. 9, pp. 138–151, 2003.

[13] R. Sailer, A. Acharya, M. Beigi, R. Jennings, and D. Verma, "IPSec validate – a tool to validate IPSec configurations," in *Proc. LISA, 15th System Administration Conference*, San Diego, California, December 2001.