MODELLING AND VERIFICATION OF EXTENSIBLE AUTHENTICATION PROTOCOL USING SPIN MODEL CHECKER

Manu S. Hegde, Jnanamurthy HK and Sanjay Singh

Department of Information and Communication Technology,
Manipal Institute of Technology, Manipal University, Manipal-576104, India
sanjay.singh@manipal.edu

ABSTRACT

The Extensible Authentication Protocol (EAP) is a framework for transporting authentication credentials. EAP offers simpler interoperability and compatibility across authentication methods. EAP supports multiple authentication methods. In this paper, we have modelled the Extensible Authentication Protocol as a finite state machine. The various entities in our model are Authenticator, EAP Server, User and User Database. The messages exchanged between various entities are modelled as transitions. The model is represented in PROMELA. The model is checked for conformance with its specifications to detect possible flaws using SPIN model checker.

KEYWORDS

Model checking, SPIN, Promela, EAP, Extensible Authentication Protocol

1. INTRODUCTION

Many security protocols have been shown to have design flaws, even a long time after their publication. For this reason, several automatic verification methods for security protocols have been proposed and they have been proved to be very useful [13].

Extensible Authentication Protocol was originally proposed for the Point-to-Point Protocol (PPP) for an optional authentication phase after the PPP link has been established. It is also a general purpose authentication protocol. EAP supports multiple authentication methods, such as token card, Kerberos (IETF RFC 1510), one-time password, certificate, public key authentication and smart card [11].

Extensible Authentication Protocol is an authentication framework defined in RFC 3748 and was updated by RFC 5247. The Extensible Authentication Protocol (EAP) is a framework for transporting authentication credentials. EAP offers simpler interoperability and compatibility across authentication methods [1].

In EAP, the party demanding proof of authentication is called the authenticator and the party being authenticated is called the supplicant. EAP defines four types of packet: request, response, success and failure. Request packets are issued by the authenticator and they solicit a response packet from the supplicant. Any number of request-response exchanges may be used to complete the authentication. If the authentication is successful, a success packet is sent to the supplicant; if not, a failure packet is sent. The basic EAP packet format is simple. A type field indicates the type of packet, such as a response or a request. An Identifier field is used to match requests and responses. Response and request packets have two further fields. The first,

DOI : 10.5121/ijnsa.2012.4606

confusingly called ‘type’, indicates the type of data being transported (such as an authentication protocol), and the second, type-data, consists of that data. Note that EAP method is synonymous with type, and both are used frequently [11].

Rest of the paper is organized as follows. Section 2 gives the detail of related work. Section 3 describes the theoretical background about EAP protocol and model checking. Section 4 explains the proposed modelling and validation technique. Section 5 discusses the simulation results obtained and finally section 6 concludes this paper.

2. RELATED WORK

There have been a number of attempts to model and verify the Extensible Authentication Protocol. In [7], the security properties of EAP-AKA authentication process are analysed and verified. The proof result shows that the authentication process of EAP-AKA can guarantee the security of wireless communication. In [8], the correctness of the extensible authentication protocol for Transport Layer Security (TLS) is verified using SPIN. The paper concentrates on the modelling and verification of the EAP-TLS authentication model. In [6], the Secure Electronic Transactions (SET) protocol is studied and the model checking tool SPIN is used to model and verify the purchasing process protocol. This paper uses LTL to describe the attributes that the protocol should satisfy and carry out the model checking [5].

Protocol modelling and verification are conducted using two approaches: (a) the conventional method in which a series of tests will be applied to the simulation model and (b) formal verification techniques that will allow us to establish safety and liveness properties with respect to the verification model [4]. In 3G mobile networks, EAP-AKA is the Extensible Authentication Protocol mechanism for authentication and key distribution using the Authentication and Key Agreement (AKA) mechanism. In [7], a model of the EAP-AKA based on the authentication tests and state space model is used for security protocol verification. Based on the EAP-AKA and its specification, this paper verifies the security properties of EAP-AKA with improved authentication tests. The proof shows that EAP-AKA can achieve the authentication property and session key distribution between the peer and server [3].

The EAP protocol allows a PPP peer to take advantage of the protected cipher-suite negotiation, mutual authentication and key management capabilities of the TLS protocol. TLS provides support for mutual authentication, integrity-protected cipher-suite negotiation and key exchange between two endpoints. In [8], the EAP-TLS authentication processing model specifies how the conversation between the authenticating peer and the authenticator takes place. The EAP-TLS authentication process model itself doesn't maintain any state but performs correlation and coordination between messages. The correctness of the EAP-TLS is expressed as the property of reachable states (Safety) and as the property of sequence of states (Liveness) [4].

3. THEORETICAL BACKGROUND

This section describes the EAP protocol and model checking concept.

3.1. EAP Authentication Sequence

The EAP specification defines three basic authentication EAP types and three non-authentication types (Identity, Nak and Notification). The three ‘basic’ authentication types are not considered secure for typical use, particularly in wireless environments. The Identity type is used by the authenticator to request the user name claimed by the supplicant, and is typically the first packet transmitted. The Nak type is used by the peer to indicate that a type proposed by the authenticator is unacceptable (for example, the authenticator has proposed an authentication

protocol that is unsupported by the peer, or policy forbids its use). If this happens then the authenticator may choose to try another, thereby allowing supplicant and authenticator to negotiate a mutually acceptable authentication protocol. The Notification type, which is rarely used, returns a message that must be displayed to the user [10]. The basic operation of the EAP is depicted in Fig 1.

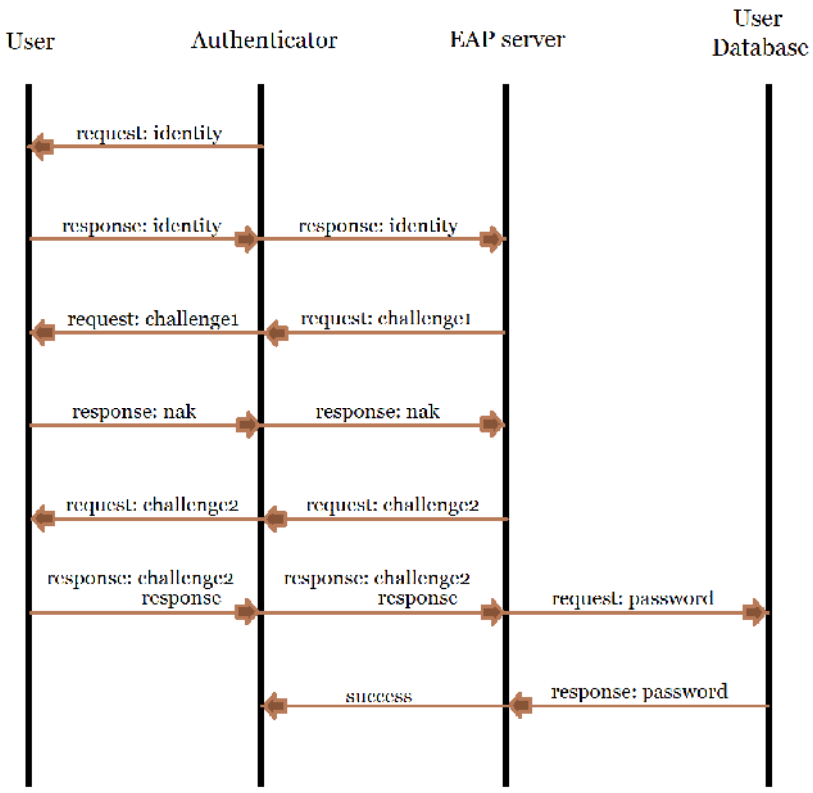


Figure 1 An Example of Authentication using EAP

The authenticator sends a request to authenticate the peer. The request has a Type field to indicate what is being requested. The peer sends a response packet in reply to a valid request. As with the request packet, the response packet contains a Type field, which corresponds to the Type field of the Request. The authenticator sends an additional request packet, and the peer replies with a response. The sequence of Requests and Responses continues as long as needed. The conversation continues until the authenticator cannot authenticate the peer, in which case the authenticator transmits an EAP Failure message. Alternatively, the authentication conversation can continue until the authenticator determines that successful authentication has occurred, in which case the authenticator transmits an EAP Success message [7].

3.2. Model Checking

Model checking has proved to be a very successful approach to analysing security protocols. The basic approach is to produce a model of a small system running the protocol(for example,

with one initiator and one responder), together with a model of the most general intruder/user who can interact with the protocol, and to use a state exploration tool to discover if the system can enter an insecure state [15].

Model Checking is a formal verification technique for verifying that a system satisfies a temporal logic formula. It works on finite state systems and proceeds by viewing the system as a structure for interpreting temporal logic and by evaluating the formula on that structure. It is much simpler than temporal deductive proofs and can be easily and effectively implemented. The properties of the system are expressed as temporal logic formulae. The model checking algorithm checks whether all the executions of the model satisfy the formula [6].

Formal verification techniques can be thought of as a sequence of three major activities [14]:

- a framework for modelling systems, typically a description language of some sort
- a specification language for describing the properties to be verified
- a verification method to establish whether the description of a system satisfies the specification

The Extensible Authentication Protocol may have flaws in its specification. This can be found by modelling the system as an automaton, and using some model checking tool to check the correctness of the operation of the system. We use the SPIN model checking tool to check the operation of the EAP, by modelling it as a finite state machine.

One of the most powerful formal methods is model checking. In principle model checking is trivial: simply generate all possible states of a program and check that the correctness specifications hold in each state. Furthermore, generating states and checking specifications can be done mechanically by a software tool. In practice, sophisticated algorithms based upon automata theory and logic are needed to perform model checking on nontrivial programs which have billions or trillions of states.

3.3 SPIN Model Checker

SPIN is a general tool for verifying the correctness of distributed software models in an automated fashion. Systems to be verified are described in PROMELA (Process Meta Language) whereas SPIN stands for “Simple Promela Interpreter”. Programs in Promela are composed of a set of processes. In addition to model checking, SPIN also acts as a simulator, following one possible execution path through the system and presenting the resulting execution trace to the user [8].

The only way to verify whether a program is correct is to systematically check that the correctness specifications hold in all possible computations and that is what model checkers like SPIN are designed to do. In a deterministic program (with no input), there is only one possible computation, so a single random simulation will be enough to demonstrate the correctness of a program. For a concurrent or nondeterministic program, checking all possible computations involves executing the program and backtracking over each choice of the next statement to execute. One of the ways that SPIN achieves efficiency is by generating an optimized program called a verifier for each PROMELA model [9].

Verification in SPIN is a three-step process and is as represented in Fig 2:

- Execute the verifier using a C compiler.

- Generate the verifier from the PROMELA source code. The verifier is a program written in C.

- Compile the verifier. The result of the execution of the verifier is a report that all computations are correct or else that some computation contains an error.

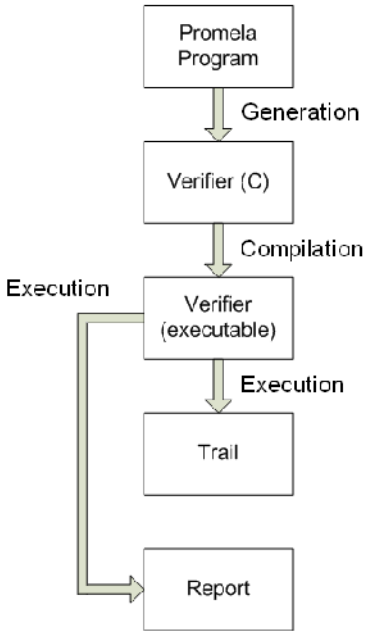


Figure 2 Architecture of Spin Model Checker

The SPIN model checking tool requires that the system be modelled using the Promela modelling language. Initially, the system has to be represented as a finite state machine (FSM). The entities that interact within the system are modelled as states. The communications between these entities are modelled as transitions between those states. Then, the finite state machine is translated into Promela code. Each state is represented in Promela as a function. The interaction between these functions models the working of the system.

The model in Promela is simulated using the SPIN tool and checked to verify whether it operates as expected. If not, the Promela code is edited and simulated till the model operates as expected. Now, since the model is ready, the next step is to check for the correctness of the model. This will be done in two phases namely assertions and linear temporal logic (LTL). Assertions are predicates that are inserted between any two statements in the Promela code, to check whether it is evaluated to true or false during simulation. LTL is used express the properties of the model that depend upon the evaluation of a predicate in a sequence of states. Some properties of the model that require the use of LTL to be expressed correctly include property of reachable states (Safety), property of sequence of states (Liveness) and absence of deadlocks. If the LTL specification is violated, then it is documented along with the associated

trace, leading to the violation. Finally, the results of applying the assertions and LTL specifications are documented.

3.4 Promela

PROMELA (Process or Protocol Meta Language) is a verification modelling language. The language allows for the dynamic creation of concurrent processes to model. In PROMELA models, communication via message channels can be defined to be synchronous or asynchronous. PROMELA models can be analysed with the SPIN model checker, to verify that the modelled system produces the desired behaviour. PROMELA is a process modelling language whose intended use is to verify the logic of parallel systems [2].

Given a program in PROMELA, SPIN can verify the model for correctness by performing random or iterative simulations of the modelled system's execution, or it can generate a C program that performs a fast exhaustive verification of the system state space. During simulations and verifications SPIN checks for the absence of deadlocks, unspecified receptions, and unexecutable code. The verifier can also be used to prove the correctness of system invariants and it can find non-progress execution cycles. Finally, it supports the verification of linear time temporal constraints; either with Promela never-claims or by directly formulating the constraints in temporal logic. Each model can be verified with Spin under different types of assumptions about the environment. Once the correctness of a model has been established with SPIN, then fact can be used in the construction and verification of all subsequent models [2].

Promela is designed for modelling a system, not for implementing one with an executable program. Typically, a model will be relatively small in size, so that it will be feasible to verify correctness properties by searching its state space. Programs in Promela are composed of a set of processes [8].

3.5 Methods for Verification

Assertions can be placed between any two statements of a program and the model checker will evaluate the assertions as part of its search of the state space. If, during the search, it finds a computation leading to a false assertion, either the program is incorrect, or the assertion does not properly express a correctness property that holds for the program. Assertions are statements consisting of the keyword assert followed by an expression. When an assert statement is executed during a simulation, the expression is evaluated. If it is true, execution proceeds normally to the next statement; if it is false, the program terminates with an error message. Assertions are limited in the properties that they can specify because they are attached to specific control points in the processes.

There are some correctness properties that simply cannot be expressed using assertions, because the properties cannot be checked by evaluating an expression in a single state of a computation. For such tasks, Linear Temporal Logic (LTL) is used for verification in SPIN. A correctness specification specified in LTL is translated by SPIN into a never claim, which is then used for verification. The never claim is executed together with the finite automaton that represents the PROMELA program. LTL may be used to check certain properties of the system like safety, liveliness and absence of deadlocks.

Linear temporal logic is a temporal logic, with connectives that allows us to refer to the future. It models time as a sequence of states, extending infinitely into the future. In general, the future is not determined, so we consider several paths, representing different possible futures, any one of which might be the actual path that is realized [14].

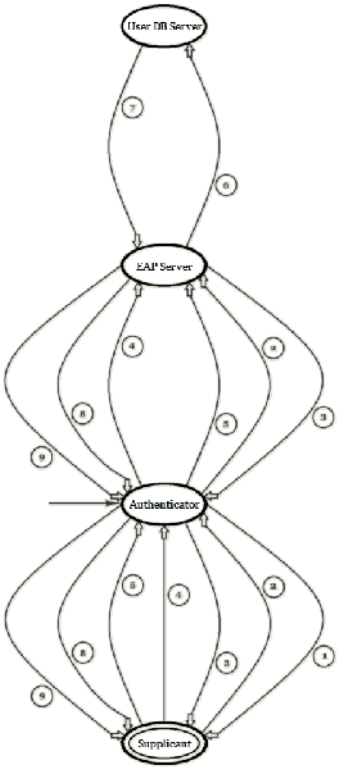
A formula of LTL is built from atomic propositions and from operators that include the operators of the propositional calculus as well as temporal operators. The temporal operators are as shown in Table 1.

Table 1 Temporal Operators

Operator	Math	SPIN
always	□	[]
eventually	◊	<>
until	υ	U

Where □ and ◊ operators are unary and the υ is a binary operator.

4. PROPOSED METHOD



Transitions

1 Request Identity
2 Response Identity
3 Request challengeX
4 Response nak
5 Response challengeX
6 Request password
7 Password
8 Success
9 Failure

Figure 3 FSM modelling the EAP

For the Extensible Authentication Protocol, we have designed a finite state machine that fully describes the working of the authentication protocol. The message types that are used by the protocol form the input alphabet. The states are the initial state, Authenticator, User, EAP and Database. The FSM takes as input the messages and transitions to one of the states. The FSM is a deterministic automaton. It is as shown in Fig 3. To check the correctness of the EAP, we automate the system by SPIN and check for all the possible reachable states as SPIN shows the sequence diagram as which all the sequence of the states it has been passed. By using appropriate LTL specifications we check the EAP model for potential problems in the operation of the protocol.

The modelling and verification of the Extensible Authentication Protocol has been carried out using the SPIN model checking tool as shown in Fig 4. The SPIN model checking tool requires that the system be modelled using the Promela modelling language. Initially, we represent the EAP as a finite state machine with 4 states namely Authenticator, Supplicant, EAP Server and User Database Server. The messages passed between these entities are modelled as transitions between the states. The possible messages that are exchanged between the entities are request identity, response identity, request challenge, response challenge, response nak, request password, password, success and failure. Then, the finite state machine is translated into Promela code. Each state is represented in Promela as a function. The interaction between these functions models the working of the EAP.

The model in Promela is simulated using the SPIN tool and checked to verify whether it operates as expected. If not, the Promela code is edited and simulated till the model operates as expected. Now, since the model is ready, the next step is to check for the correctness of the model. This will be done in two approaches namely assertions and linear temporal logic (LTL).

Assertions are predicates that are inserted between any two statements in the Promela code, to check whether it is evaluated to true or false during simulation. If the Assertion is violated, then it is documented along with the associated trace, leading to the violation. Some properties cannot be checked by evaluating a predicate in a single state of computation. In such cases, LTL is used express the properties of the model. Some properties of the model that require the use of LTL to be expressed correctly include property of reachable states(Safety), property of sequence of states(Liveness) and absence of deadlocks. If the LTL specification is violated, then it is documented along with the associated trace, leading to the violation. Finally, the results of applying the assertions and LTL specifications are documented.

The finite state machine shown in Fig 3 represents the operation of the Extensive Authentication Protocol. This FSM is modelled using the Promela language and executed using the SPIN model checker. The results are as shown in the Fig 5-8. Fig 5, 6, 7 and 8 depict the SPIN generated state diagrams for the entities Authenticator, User, EAP and Database respectively.

Promela code for Authenticator module:

```
Authenticator(msg)
1 /* precondition: The authenticator receives a message */
2 /* postcondition: The authenticator responds to */
3 /* the message */
4 switch
5 case msg = null :
6 return user(RequestIdentity)
7 case msg = ResponseIdentity :
8 return EAP(ResponseIdentity)
```

```
9 case msg = RequestChallenge :
10 return EAP(RequestChallenge)
11 case msg = ResponseNak :
12 return EAP(ResponseNak)
13 case msg = ResponseChallenge :
14 return EAP(ResponseChallenge)
15 case msg = Success :
16 return user(Success)
17 case msg = Failure :
18 return user(Failure)
```

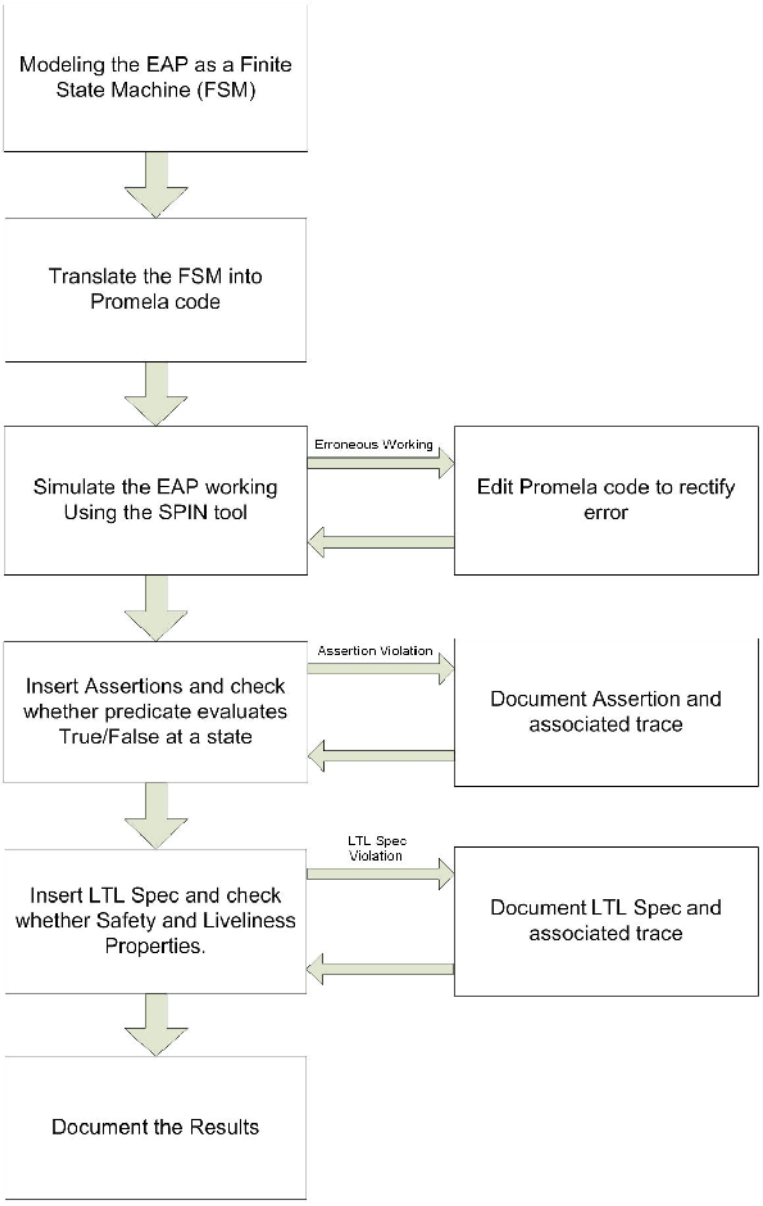


Figure 4 Modelling and Verification Technique

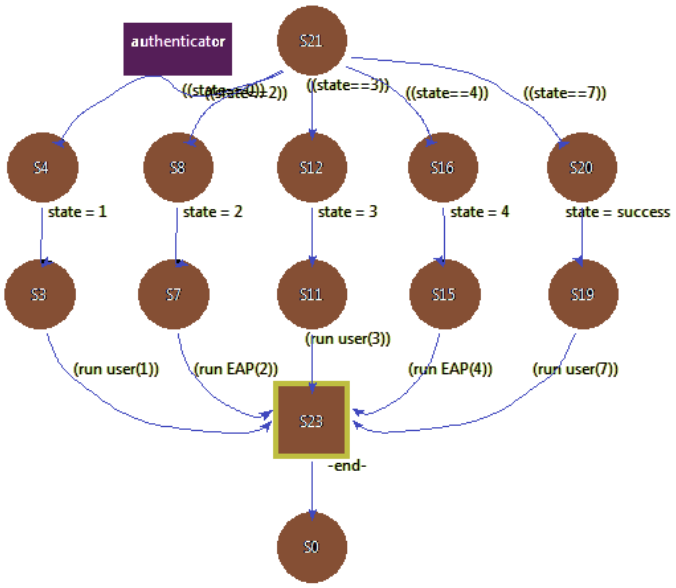


Figure 5 SPIN generated state diagram for Authenticator module

The Authenticator module models the Authenticator in the EAP authentication sequence. This module accepts messages sent to the Authenticator from the User and EAP server entities. It responds to the received messages with appropriate messages to the User and EAP server entities. Initially, the Authenticator sends a request identity message to the User which it wants to authenticate. When the Authenticator receives a response identity message, it knows that the message has been sent by the User and it sends a response identity message to the EAP server. When the Authenticator receives a request challenge message, it knows that the message has been sent by the EAP server and it sends a request challenge message to the User. When the Authenticator receives a response nak message, it knows that the message has been sent by the User and it sends a response nak message to the EAP server. When the Authenticator receives a response challenge message, it knows that the message has been sent by the User and it sends a response challenge message to the EAP server. When the Authenticator receives a success/failure message, it knows that the message has been sent by the EAP server and it sends a success/failure message to the User.

Promela code for User module:

```
User(msg)
1 /* precondition: The user receives a message */
2 /* postcondition: The user responds to the message */
3 switch
4 case msg = RequestIdentity :
5 return Authenticator(ResponseIdentity)
6 case msg = RequestChallenge :
7 return Authenticator(ResponseNak)
```

8 case msg = Success :
9 return Success
10 case msg = Failure :
11 return Failure

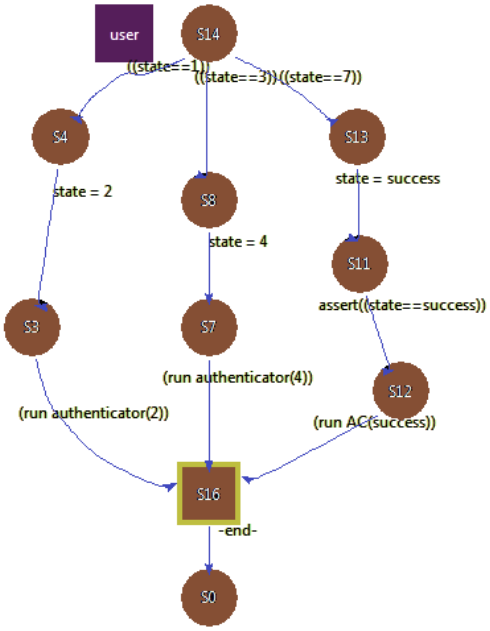


Figure 6 SPIN generated state diagram for User module

The User module models the User in the EAP authentication sequence. This module accepts messages sent to the User from the Authenticator. It responds to the received messages with appropriate messages to the Authenticator. When the User receives a request identity message, it knows that the message has been sent by the Authenticator and it sends a response identity message to the Authenticator. When the User receives a request challenge message, it knows that the message has been sent by the Authenticator and it sends a response nak or response challenge message to the Authenticator. When the User receives a success/failure message, it knows that the message has been sent by the Authenticator. Depending upon whether it received a success or failure message it decides whether it has been successfully authenticated by the Authenticator or not.

Promela code for EAP module:

EAP(msg)
1 /* precondition: The EAP server receives a message */
2 /* postcondition: The EAP server responds to the message */
3 switch
4 case msg = ResponseIdentity :
5 return Authenticator(RequestChallenge)
6 case msg = ResponseNak :

7 return Authenticator(RequestChallenge)
8 case msg = Password :
9 return Authenticator(Success)
10 case msg = ResponseChallenge :
11 return Database(RequestPassword)

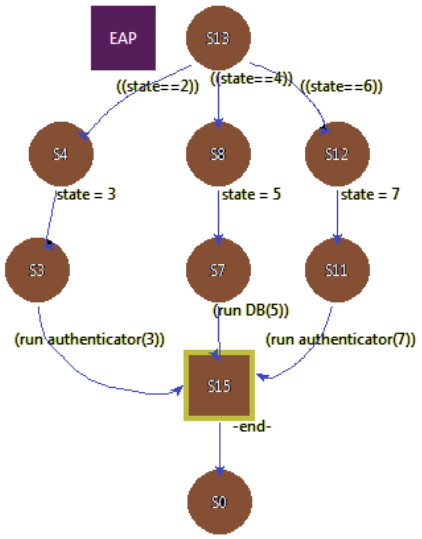


Figure 7 SPIN generated state diagram for EAP module

The EAP module models the EAP server in the EAP authentication sequence. This module accepts messages sent to the User from the Authenticator and User Database server. It responds to the received messages with appropriate messages to the Authenticator and User Database server entities. When the EAP server receives a response identity message, it knows that the message has been sent by the Authenticator and it sends a request challenge message to the Authenticator. When the EAP server receives a response nak, it knows that the message has been sent by the Authenticator and it sends a request challenge message to the Authenticator. When the EAP server receives a response challenge, it knows that the message has been sent by the Authenticator and it sends a request password message to the User Database server. When the EAP server receives a password message, it knows that the message has been sent by the User Database server and sends a success message to the Authenticator.

Promela code for Database module:

Database(msg)
1 /* precondition: The Database server receives a message */
2 /* postcondition: The Database server responds to the message */
3 switch
4 case msg = RequestPassword :

5 return EAP(Password)

The Database module models the User Database server in the EAP authentication sequence. This module accepts messages sent to the User Database server from the EAP server and responds to the received messages with appropriate messages to the EAP server. When the User Database server receives a request password message, it knows that the message has been sent by the EAP server and it sends a password message to the EAP server.

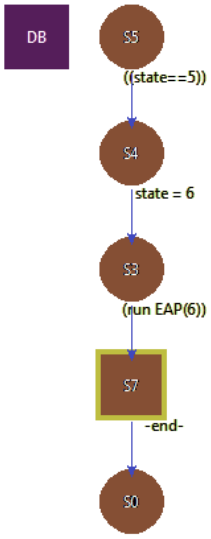


Figure 8 SPIN generated state diagram for Database module

5. SIMULATION RESULTS AND DISCUSSION

The FSM has to be checked to validate the system it has been modelled for. Various LTL specifications are used to validate the system's operation. In SPIN, verification is subdivided into two aspects: safety and liveness.

5.1. Safety Properties

By safety properties, we mean "nothing bad ever happens". For instance, invariant and deadlock freedom (the system never reaches a state where no actions are possible) are both safety properties. By default, SPIN will check a set of basic safety properties such as absence of deadlock and unreachable code. It will also check that any user-defined process assertions or invariants cannot be violated. SPIN check a safety property by trying to find a trace leading to the "bad" thing. If there is not such a trace, the property is satisfied [12].

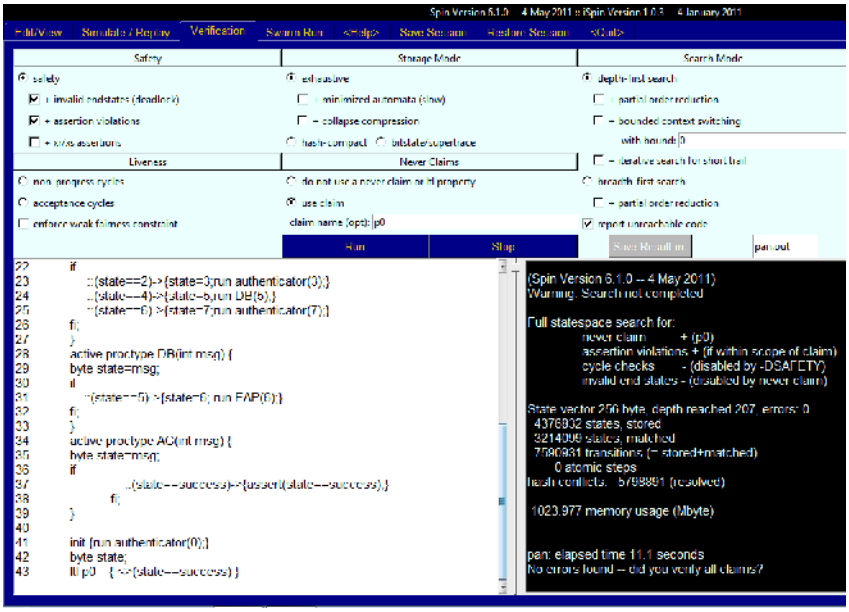


Figure 9 Checking for conformance with the LTL spec <>(state == success)

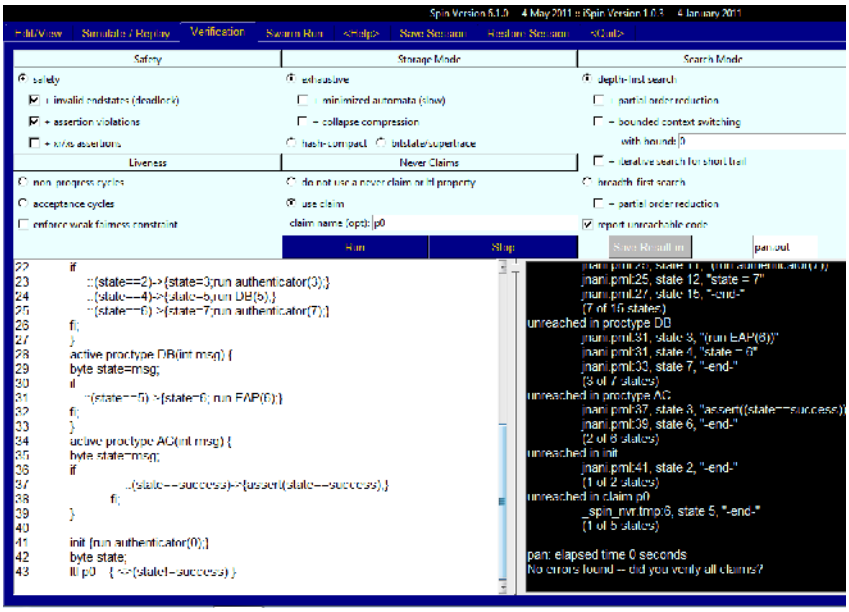


Figure 10 Checking for conformance with the LTL spec <>(state != success)

To check the absence of deadlocks, we have to check whether User is the only final state. That is, the finite state machine should not stop at any state other than the state where the User is authenticated. The FSM has to be checked to find whether the FSM will eventually enter a state where the authentication is successful. That is, the FSM traverses through several states where the authentication of the user is not yet successful, but later it enters a state where the authentication succeeds. The LTL specification used to check this is <>(state == success). The system conforms to this specification. The SPIN simulation result for this check is as shown in Fig 9.

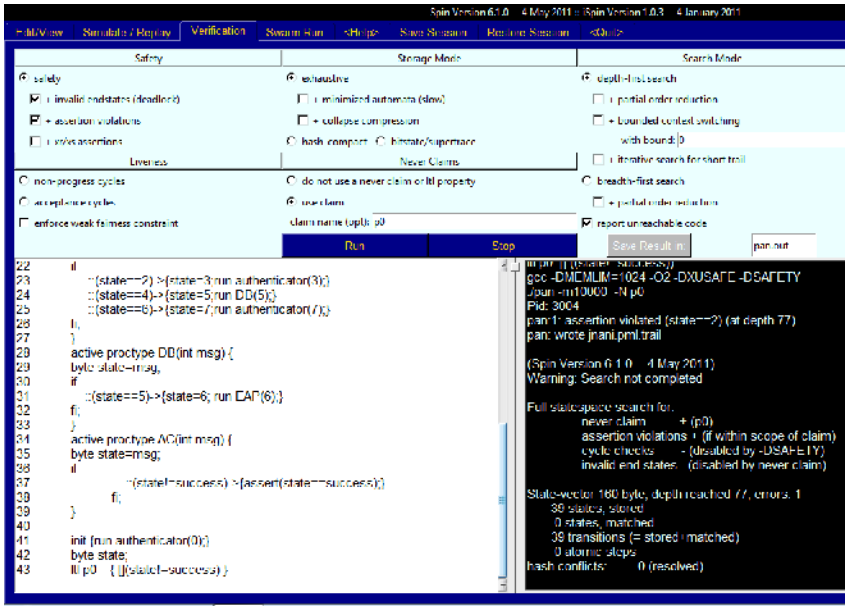


Figure 11 Assert statement to check Authentication status

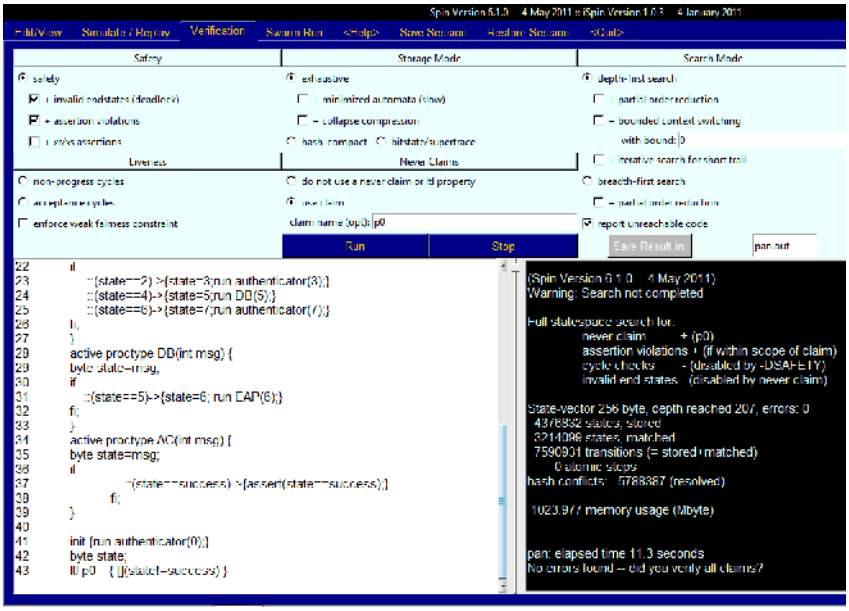


Figure 12 Assert statement to check Authentication status

To check the absence of unreachable code, we have to check whether the finite state machine will generate a failure message whenever the authentication is unsuccessful and a success message is generated whenever the authentication is successful. The FSM has to be checked to find whether the FSM will always enter a state where the authentication is successful when a failure message is received. That is, the FSM traverses through several states where the authentication of the user is not yet ascertained, but later it always enters a state where it receives a failure message but the authentication succeeds. The specification used to check this is assert (state == success) when failure message is received. The system does not conform to this specification. The SPIN simulation result for this check is as shown in Fig 11.

The FSM has to be checked to find whether the FSM will always enter a state where the authentication is successful when a success message is received. That is, the FSM traverses through several states where the authentication of the user is not yet ascertained, but later it always enters a state where it receives a success message and the authentication succeeds. The specification used to check this is assert (state == success) when success message is received. The system conforms to this specification. The SPIN simulation result for this check is as shown in Fig 12.

5.2. Liveness Properties

By liveness properties, we mean "something good will eventually happen". For instance, termination (the system will eventually terminate) and response are both liveness properties. By default, SPIN can check that the system can only terminate in user-defined valid end-states. The PROMELA language includes two types of labels that can be used to define two complementary types of liveness properties: acceptance and progress. When checking for acceptance cycles, the verifier will complain if there is an execution that visits infinitely often an acceptance state. When checking for non-progress cycles, the verifier will complain if there is an infinite execution that does not visit a progress state infinitely often. SPIN check a liveness property by trying to find an infinite loop in which the "good" thing does not happen. If there is not such a loop, the property is satisfied [12].

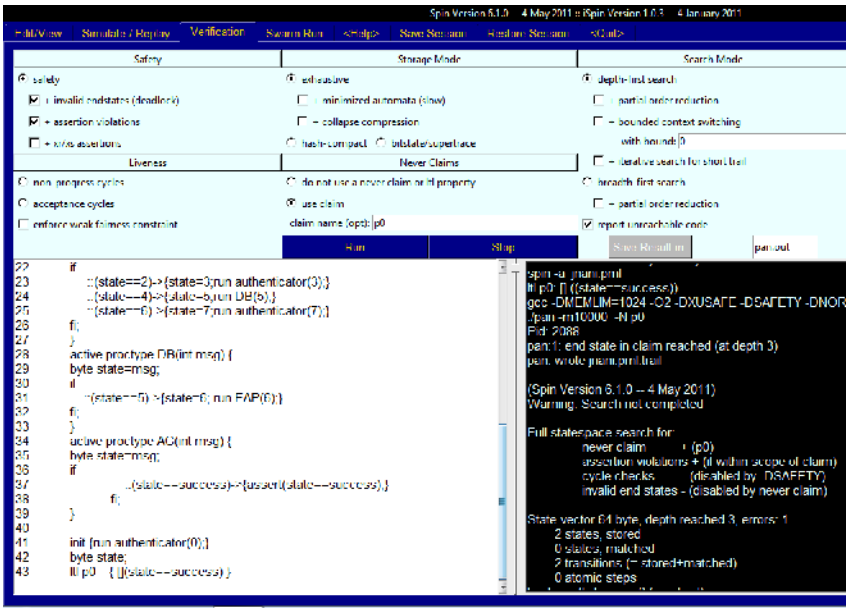


Figure 13 Checking for conformance with the LTL spec [ ](state == success)

The FSM has to be checked to find whether the FSM will always enter a state where the authentication is successful. That is, the FSM traverses through several states where the authentication of the user is not yet ascertained, but later it always enters a state where the authentication succeeds. The LTL specification used to check this is [](state == success). The system does not conform to this specification. The SPIN simulation result for this check is shown in Fig 13.

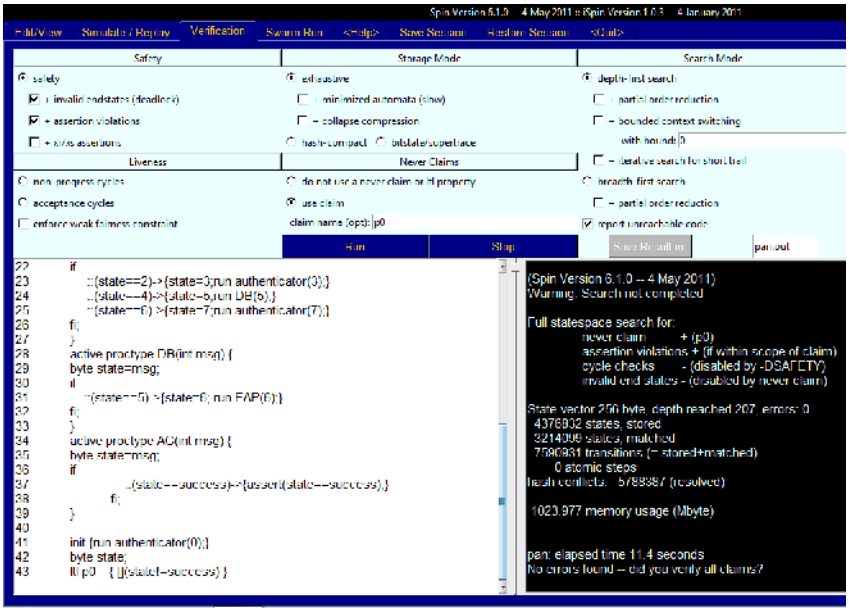


Figure 14 Checking for conformance with the LTL spec [ ](state != success)

The FSM has to be checked to find whether the FSM will always enter a state where the authentication is unsuccessful. That is, the FSM traverses through several states where the authentication of the user is not yet ascertained, but later it always enters a state where the authentication fails. The LTL specification used to check this is [](state != success). The system does not conform to this specification. The SPIN simulation result for this check is as shown in Fig 14.

6. CONCLUSION

The objective of this paper is to detect any possible erroneous states in the Extensible Authentication Protocol. The Extensible Authentication Protocol has been modelled using the Promela language. The operation of the EAP has been verified using the SPIN model checker. The conformance of the system to various specifications has been verified by the use of assertions and LTL specifications. The possible reachable states and all possible paths were shown in the automata generated by SPIN. So the final state will be reachable by all possible paths from the initial state.

Our future work will concentrate upon modelling and verification of the various extensions of the Extensible Authentication Protocol. The possible attacks on the Extensible Authentication Protocol can be modelled and analysed in the future as an extension of this work.

ACKNOWLEDGEMENTS

We express our sincere gratitude to anonymous reviewers and the editor for their valuable comments and suggestions to improve the manuscript.

REFERENCES

[1] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson and H. Levkowetz, (2004) “RFC3748 - Extensible Authentication Protocol (EAP)”, [Available Online] www.ietf.org/rfc/rfc3748.txt.

[2] Promela, (2012), [Available Online] http://en.wikipedia.org/wiki/Promela.

[3] Xiehua Li and Xiaohong Zhang, (2009) "Formal Verification for EAP-AKA Protocol in 3G Networks", *International Conference on Computational Intelligence and Software Engineering*, pp1-4.

[4] Ali, H.B. and Karim, M.R. and Ashraf, M. and Powers, D.M.W. (2010) "Modelling and verification of Extensible Authentication Protocol for Transport layer Security in Wireless LAN environment", *2nd International Conference on Software Technology and Engineering (ICSTE)*, Vol. 2, ppV2-41 -V2-45.

[5] Li Jing and Li Jinhua, (2009) "Model Checking the SET Purchasing Process Protocol with SPIN", *5th International Conference on Wireless Communications, Networking and Mobile Computing*, pp1-4.

[6] Godefroid, P. and Wolper, P., (1991) "A partial approach to model checking", *Proceedings of Sixth Annual IEEE Symposium on Logic in Computer Science*, pp406 -415.

[7] Seyed Amin Hoseini, Seyed Saed Rezaee and Hassan Taheri (2011) "Implementation of Extensible Authentication Protocol in OPNET Modeller*"*, *International Conference on Network Communication and Computer (ICNCC)*, pp99 -103.

[8] Mordechai Ben-Ari, (2008) *Principles of the Spin Model Checker*, Springer.

[9] Holzmann, G.J., (1997) "The model checker SPIN", IEEE Transactions on Software Engineering, Vol. 23, No. 5 pp279 -295.

[10] Qing Xu and Changsheng Wan and Aiqun Hu, (2008) "The Performance Analysis of Fast EAP Re-authentication Protocol", International Symposium on Computer Science and Computational Technology (ISCSCT), Vol. 1, pp99 -103.

[11] Jyh-Cheng Chen and Yu-Ping Wang (2005) "Extensible authentication protocol (EAP) and IEEE 802.1x: tutorial and empirical experience*"*, IEEE Communications Magazine, Vol. 43, No. 12, pp supl.26 - supl.32.

[12] Elisabeth A. Strunk, M. Anthony Aiello and John C. Knight, (2006) "A Survey of Tools for Model Checking and Model-Based Development", [Available Online] http://citeseerx.ist.psu.edu/.

[13] Panti, M. and Spalazzi, L. and Tacconi, S. and Pagliarecci, R., (2005) "Model checking the security of multi-protocol systems", Proceedings of the 2005 International Symposium on Collaborative Technologies and Systems, pp92 -99.

[14] Michael Huth and Mark Ryan, (2007) *Logic in Computer Science*, Cambridge.

[15] Lowe, G., (1998) "Towards a completeness result for model checking of security protocols", Proceedings. 11th IEEE Computer Security Foundations Workshop, pp96 -105.