Analysis of IEEE 802.11i WLAN Security Protocol

Elankayer Sithirasenan, Saad Zafar, and Vallipuram Muthukkumarasamy, *Member, IEEE*

***Abstract* — As wireless LAN (WLAN) deployments increase, so does the challenge to provide these networks with adequate security. Business organizations, educational institutions and government co-operations are becoming more and more concerned about their e-security. The need for reliable and robust security mechanisms for WLANs is increasing. The latest WLAN security protocol IEEE 802.11i guarantees robust security with improved authentication, authorization and key distribution mechanisms. However, though the newest security protocol assures dependable communication sessions, the credibility of the three connection phases; security policy selection, authentication and key distribution needs further investigation. The loosely coupled state machines of the participating components can pave the way to security breaches. In this study we have investigated the integrity of these three phases. The analysis is carried out in two stages. Initially, the three phases are modeled using Genetic Software Engineering (GSE) methodology and then formally verified with Symbolic Analysis Laboratory (SAL) tools. We established several Linear Temporal Logic (LTL) formulas to model check our models. We have also examined and analyzed possible security threats due to various issues arising from software implementations and intruder behaviors.**

***Index Terms* — Wireless LAN Security, Formal Methods, Model Checking, Behavior Trees, IEEE 802.11i, Genetic Software Engineering (GSE).**

I. Introduction

WLANs have gained vast popularity over the last couple of years due to its ease of use. Unlike the wired network the wireless networks provides connection from any where without the need for a direct physical connection. This flexibility attracts many mobile users to opt for wireless connectivity. In contrast, this very nature of the wireless communication has also enabled easy means of breaking into organizational networks through a range of gaps in wireless connectivity.

The first wireless security solution for 802.11 based networks, the Wireless Equivalency Protocol (WEP), received a great deal of coverage due to various technical failures in the protocol [1]. Standard bodies and industry organizations are spending enormous amount of time and money in developing and deploying next-generation solutions that address growing wireless network security problems. The IEEE 802.11i standard [2] provides much-improved authentication, authorization, and encryption capabilities. The Wi-Fi Protected Access (WPA) standard [3], a subset of the 802.11i, created by the Wi-Fi Alliance, addresses the weaknesses of 802.11 data privacy by incorporating Temporal Key Integrity Protocol (TKIP), a much stronger implementation of the RC4 encryption algorithm, plus a sophisticated keying system that ties together the data privacy and authentication functions. IEEE 802.1X [4] was introduced to specifically address the authentication functions in the network environment. The IEEE 802.1X standard enhances the security capabilities of the IEEE 802.11i standard with its powerful authentication, authorization and key management functions.

The strong security mechanisms introduced by the standard bodies and other organizations must be correctly interpreted and comprehend by Software Engineers for proper implementation. A naïve implementation of security protocols can lead to the similar security breaches as in the case of technical flaws. As such, the primary aim of this study is to build a complete and consistent software model for the IEEE 802.11i security protocol. This is achieved by first identifying the requirements of the security protocol and then modeling it using the GSE methodology [5]. Once the system is modeled it is then formally verified using the SAL [6] model checker. Thereafter, we modeled and verified a possible intruder behavior in the WLAN environment. In this paper, we have presented a complete and consistent model for the IEEE 802.11i security protocol highlighting the possible security loopholes. We have also suggested feasible improvements to the model to address the security issues discussed in the IEEE 802.11i standard. The results presented give sufficient information for Software Engineers to implement the security protocol more rigidly. To the best of our knowledge this is the first ever analysis to be reported on the security association process proposed in IEEE 802.11i. Furthermore, the process used in the analysis is also innovative.

Manuscript received May 6, 2005. This work was supported in part by the Institute for Integrated and Intelligent Systems, Griffith University, Australia.

Elankayer Sithirasenan is with the Institute for Integrated and Intelligent Systems, Griffith University, Brisbane, Australia (phone: +61-7-387-56603; e-mail: Elankayer.Sithirasenan@student.griffith.edu. au).

Saad Zafar is with Software Quality Institute, Griffith University, Brisbane, Australia. (phone: +61-7-387-56603; e-mail: Saad.Zafar@ student.griffith.edu.au).

Vallipuram Muthukkumarasamy is with the School of Information and Communication Technology, Faculty of Engineering, Griffith University, Gold Coast, Australia (phone: +61-7-555-28256; e-mail: v.muthu@ griffith.edu.au).

We present related work in Section II and our intuition in Section III. An overview of the IEEE 802.11i protocol is presented in Section IV. The modeling and model checking details are explained in Section IV and V. The analysis is presented in Section VI and Section VII concludes the paper.

II. Related Work

The basic idea of analysing network and communication protocols is quite old, dating back to at least 1978 [7][8]. In recent decades, model checking has made significant progress in tackling the verification of complex, concurrent systems [9]. Tools such as SMV [10], SPIN [11], and Murphi [12] have been used to verify hardware and software protocols by exhaustively searching the state space. The drawback of traditional model checkers is that the system to be verified must be modeled in a particular description language, requiring a significant amount of manual effort that can easily be error prone.

Some formal verification tools have used the idea of executing and checking systems at the implementation level. Verisoft [13], for example, systematically executes and verifies actual code and has been used to successfully check communication protocols written in C. However, Verisoft does not store states and therefore, can potentially explore a state more than once. This problem is alleviated to some degree by partial order reduction, a sound state space reduction technique implemented in Verisoft that eliminates the exploration of redundant interleaving of transitions created by commutative operations. Nevertheless, this technique requires hints to be provided by the user and/or some static analysis of the code to determine dependencies between transitions.

Tools such as ESC [14], LCLint [15], and the MC Checker [16] have been used to check source code for errors that can be statically detected with minimal manual effort. Splint [17] reports a warning for any code path that fails to satisfy the storage-release obligation, because it causes a memory leak. Although memory leaks do not typically constitute a direct security threat, attackers can exploit them to increase a denial-of-service attack's effectiveness. Both stack and heap-based buffer overflow vulnerabilities are detected by Splint. The simplest detection techniques just identify calls to often misused functions; more precise techniques depend on function descriptions and program-value analysis.

The ESP [18] language uses processes to implicitly express state machines. These processes use channels to communicate with each other. In addition, ESP has a number of language features that simplify the task of writing device firmware. The ESP compiler can automatically extract models that can be used by a model checker to extensively test the program. ESP uses the SPIN model checker to verify correctness of software systems. It systematically explores the state space of the system and checks for violations of the specified property. The Spin models generated by the ESP compiler can be used together with programmer-supplied SPIN code to verify different properties of the system.

Gray and McLean propose encoding the entire protocol in terms of temporal logic [19]. Much like symbolic model checking, they describe the model by giving formulas that express the possible relationships between variable values in the current state and variable values in the next state. This makes their framework more formal than the others, but much more cumbersome as well. They provide a simple example and prove a global variant for this example. The few sub cases they consider are very straightforward but their technique demands very long proofs even for the extremely simple example they present. They argue that their technique could be automated but provide no tools for their system.

Woo and Lam propose much more intuitive model for authentication protocols [20]. Their model resembles sequential programming with each participating principle being modeled independently. There is an easy and obvious translation from the common description of a protocol as a set of messages to their model. Their models are also intuitive because they consider all possible execution traces instead of considering just the set of words obtainable by the intruder. They are concerned with checking of what they call secrecy and correspondence properties. The secrecy property is expressed as a set of words that the intruder is not allowed to obtain. The correspondence properties can express things of the form if principal A finishes a protocol run with principal B, then principal B must have started the protocol run with A. However, they do not provide a general logic in which to formalize security properties, nor do they provide an automated tool.

Dolev and Yao [21] proposed an approach to model network protocols by defining a set of states and a set of transitions that takes into account an intruder, the messages communicated back and forth, and the information known by each of the components. This state space is then traversed to check if some particular state can be reached or if some state trace can be generated. They developed an algorithm for determining whether or not a protocol is secure in their model. However, their model is extremely limited. They only consider secrecy issues, and they model only encryption, decryption, and adding, checking, or deleting a component name.

Medow [22] used an extension of the Dolev-Yao model in the PROLOG based model checker by giving a description of an insecure state to perform the verification. The model checker searches backwards in an attempt to find an initial state. In PROLOG, this can be achieved by unifying the

current state against the right hand side of a rule and deducing the state description for the previous state from the left hand side. If the initial state is found, then the system is insecure, otherwise an attempt is made to prove that that the insecure state is unreachable by showing that any state that leads to this particular state is also unreachable. This kind of trace can lead to infinite search where for an intruder to learn word A, he must learn word B, and in order to learn word B, he must learn word C, and so on. For these reason formal languages allows users to prove that no word in a set of words (or language) can be generated by the intruder. However, this PROLOG based model checker was still too limited, particularly it did not allow modeling of freshly generated nonces or session keys.

III. Motivation

The motivation for our work comes from the paper titled "Is this a Revolutionary Idea or not" [23] by Robert Glass. He has encouraged researchers to explore the claims made by Geoff Dromey - the inventor of GSE methodology who declares "GSE methodology is admired for its simplicity, traceability, ability to detect defects and its control of complexity". Dromey says that his methodology has been successfully used to analyze very large systems such as satellite control systems, air traffic control systems and like. Hence, as a novel attempt we have used the GSE methodology to model and analyze security protocols. Unlike the analysis techniques that are specific to security protocols, our approach is different since we first use the GSE methodology to model the security protocol from its requirement, then convert it into symbolic code and finally, verify it using temporal logic.

In GSE methodology each requirement is translated into its corresponding behavior tree (BT), which describes the behaviors that will result from that requirement. A BT is made up of components (the software pieces), states (that those components can take on), events and decisions/constraints (that are associated with the components), data that the components exchange, and the causal, logical, and temporal dependencies (associated with component interactions). It is the way these BTs are integrated that makes things different. To integrate, the BTs are placed together like a jigsaw puzzle, where clear points of intersection between the trees make the puzzle pieces to fit together. Using this approach, the software system is built "out of its requirements" rather than just "satisfying its requirements."

The whole point of this approach, Dromey says, is to master the complexity that accompanies building a significant software system. Complexity can be handled piecewise via integrating the localized behavior trees, rather than as one big global cognitively daunting task. This greatly reduces the strain on our short-term memory.

A benefit of doing integration is that just as a picture emerges when all the pieces of a jigsaw puzzle are put in their correct places, a similar thing happens as the behavior trees of functional requirements are integrated. Surprisingly, the picture in this case is the integrated component architecture of the system, along with the integrated behavior of each of the components in the system. Dromey claims requirements integration has the additional benefit of being a powerful way to find defects in a system early - only when a requirement is seen in the context where it is applied do we see its problems.

As a fringe benefit of this approach, Dromey claims, the traceability of the original requirements into the as-built software system becomes much easier. Even with the use of commercially available tools, it is well known that requirements traceability is a complex and barely manageable task due to the "requirements explosion" caused when the original requirements explode into the requirements for a design to satisfy those requirements. (Some researchers have found that explosion rate to be on the order of 50:1). With Dromey's approach, adding a new requirement is like adding a new piece to a jigsaw puzzle that was originally incomplete.

It is a well accepted fact in software engineering that fixing defects after the software has been implemented is costly both in terms of time and money. And in case of security, this can jeopardize an entire organization. The GSE methodology proposes a systematic approach for deriving the system implementation models from the requirements enabling conceptual errors to be detected much earlier in the development phase rather than waiting until the software is implemented. Most, if not all conventional modeling techniques are built by intuition to match the system requirements which results in implementation models representing the designers' mental replica. The resulting model will significantly depend on the designers experience and expertise. Further, conventional methodologies have several different approaches to describe the various aspects of the system. For example the UML modeling technique has almost nine different modeling tools to describe the various functionalities of a system.

The main aim in applying the GSE methodology to model and analyze the security protocol will be to reduce the amount of work required to go from system modelling to systematic verification. We presume that the analysis will not suffer from too many false positives since every scenario checked will be a valid execution path due to the inherent qualities of the BT models. Unlike the methods that are currently available for analysing communication protocols, this approach first delivers a complete and consistent model emerging from its requirements. Having derived a reliable model we analyse its credibility on the various security aspects specific to the wireless network environment. We presume that this novel approach will reveal greater scope to analyse security protocols compared to the existing techniques.

IV. THE 802.11I SECURITY PROTOCOL

The IEEE 802.11i standard defines two classes of security framework for IEEE 802.11 WLANs: RSN and pre-RSN. A station is called RSN-capable equipment if it is capable of creating RSN associations (RSNA). Otherwise, it is a pre-RSN equipment. The network that only allows RSNA with RSN-capable equipments is called an RSN security framework. The major difference between RSNA and pre-RSNA is the 4-way handshake. If the 4-way handshake is not included in the authentication/association procedures, stations are said to use pre-RSNA. The RSN, in addition to enhancing the security in pre-RSN defines a number of key management procedures for IEEE 802.11 networks. It also enhances the authentication and encryption mechanisms from the pre-RSN. The enhanced features of RSN are as follows:

Authentication Enhancement: IEEE 802.11i utilizes IEEE 802.1X for its authentication and key management services. The IEEE 802.1X incorporates two components namely, (a) *IEEE 802.1X Port* and (b) *Authentication Server (AS)* into the IEEE 802.11 architecture. The IEEE 802.1X port represents the association between two peers as shown in Fig. 1. There is a one-to-one mapping between IEEE 802.1X Port and association.

Key Management and Establishment: Two ways to support key distribution are introduced in IEEE 802.11i: *manual key management* and *automatic key management.* Manual key management requires the administrator to manually configure the key. The automatic key management is available only in RSNA. It relies on IEEE 802.1X to support key management services. More specifically, the 4-way handshake is used to establish each transient key for packet transmission as in Fig. 2.

Encryption Enhancement: In order to enhance confidentiality, two advanced cryptographic algorithms are developed: Counter-Mode/CBC-MAC Protocol (CCMP) and Temporal Key Integrity Protocol (TKIP). In RSN, CCMP is mandatory. TKIP is optional and is recommended only to patch any pre-RSN equipment.

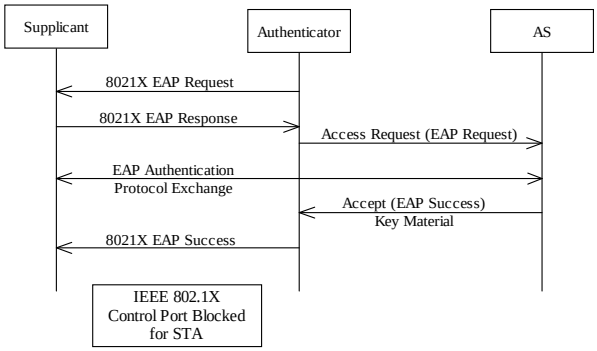


Figure 1. IEEE 802.1X EAP Authentication

During the initial security association between a station (STA) and an access point (AP), the STA selects an authorized Extended Service Set (ESS) by selecting among APs that advertise an appropriate Service Set ID (SSID). The STA then uses IEEE 802.11 Open System authentication followed by association to the chosen AP. Negotiation of security parameters takes place during association. Next, the AP's Authenticator or the STA's Supplicant initiates IEEE 802.1X authentication. The Extensible Authentication Protocol (EAP) used by IEEE 802.1X will support mutual authentication, as the STA needs assurance that the AP is a legitimate Access Point.

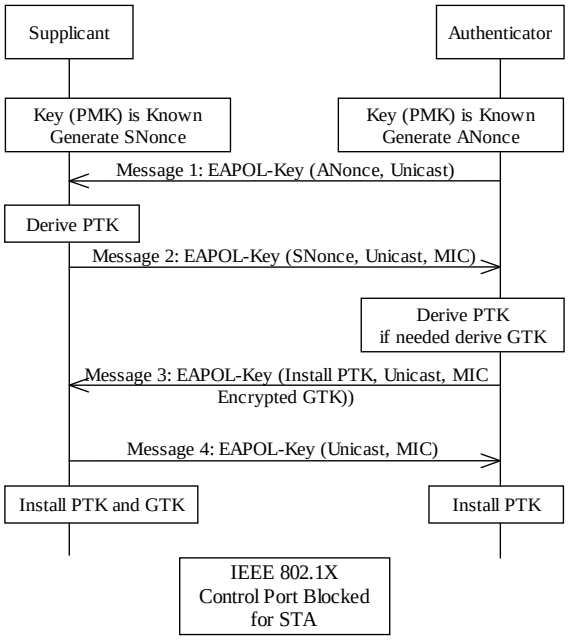


Figure 2. Establishing pairwise & group keys [2]

The last step is the key management. The authentication process creates cryptographic keys shared between the IEEE 802.1X AS and the STA. The AS transfers these keys to the AP, and the AP and STA use one key confirmation handshake, called the 4-Way Handshake, to complete security association establishment. The key confirmation handshake indicates when the link has been secured by the keys and is ready to allow normal data traffic.

In the case of roaming, an STA requesting (re)association followed by IEEE 802.1X or pre-shared key authentication, the STA repeats the same actions as for an initial contact association, but its Supplicant also deletes the PTK when it roams from the old AP. The STA's Supplicant also deletes the PTKSA when it disassociates/deauthenticates from all basic service set identifiers in the ESS. An STA already associated with the ESS can request its IEEE 802.1X Supplicant to authenticate with a new AP before associating to that new AP. The normal operation of the DS via the old AP provides

communication between the STA and the new AP.

In the next section the above-described RSN is modeled. The complete modeling, from requirements analysis to the final design models was carried out using the GSE techniques.

V.MODELING

Requirements translation is the first formal step in the GSE design process. Its purpose is to translate each natural language functional requirement, one at a time, into one or more behavior trees. This translation identifies the components (including actors and users), the states they realize (including attribute assignments), the events and decision/constraints that they are associated with, the data exchange, and the casual, logical and temporal dependencies associated with component interactions.

Following the translation and integration of the requirements, we evolve the requirements representations into design/architecture representations. The first phase of this architectural process is the "Component Architecture Transformation," and the output of that phase is a "Component Interaction Network." The primary thing that happens here is that components, which may be represented at many points in the requirements representation, are isolated out to appear only once in the solution representation. This amounts to algorithmically transforming the integrated tree of requirements into a network of components that interact (the traditional conceptual view of a system). The final stage of this process is the "Component Behavior Projection." Here, component behaviors are concentrated by separately projecting each component's behavior from the integrated requirements tree. The result of this process is a skeleton behavior tree for each component that will deliver the behavior it needs to exhibit to function as an encapsulated component in the component interaction network.

The IEEE 802.11i standard defines two classes of security framework for IEEE 802.11 WLANs: RSN and pre-RSN security frameworks. This study is mainly focused on the RSN security framework since it is expected to drive the future of distributed wireless networks. STAs in the RSN environment can make contact with the ESS in one of two ways: initial contact or Roaming. In case of roaming we are not concerned of whether the STAs are navigating inter-subnet or intra-subnet since the security policy in both cases will be the same.

Clauses 8.4.1 to 8.4.10 in the IEEE 802.11i standard describe the steps involved in the RSN security association life cycle. We have made use of these steps to develop the requirements behavior trees (RBTs) for the RSN. Each individual functional requirement is translated into one or more corresponding RBTs. Altogether, we assembled twelve functional requirements and an RBT was developed for each. As an example we have listed the fifth requirement below:

Requirement 5, Policy selection in ESS: The STA initiating an association shall insert an RSN IE into its (Re) Association Request whenever the targeted AP indicates RSN support. The initiating STA's RSN IE shall include one authentication and pairwise cipher suite from among those advertised by the targeted AP in its Beacons and Probe Responses. It shall also specify a group key cipher suite specified by the targeted AP. If the RSN capable AP receives a (Re) Association request including an RSN IE, and if it chooses to accept the association, the AP shall, to secure this association use the authentication and pairwise cipher suites specified in the RSN IE sent by the STA. The AP shall then include the selected suites Association response to the STA. Once both AP and STA agree on a common security policy they are said to be at the CONNECTING state.

Fig. 3 shows the RBT for this requirement. The shaded boxes (colors used in real) in the tree denote assumed or missing requirements. In a similar fashion RBTs are created for all of the twelve requirements extracted from clause 8 of the standard.

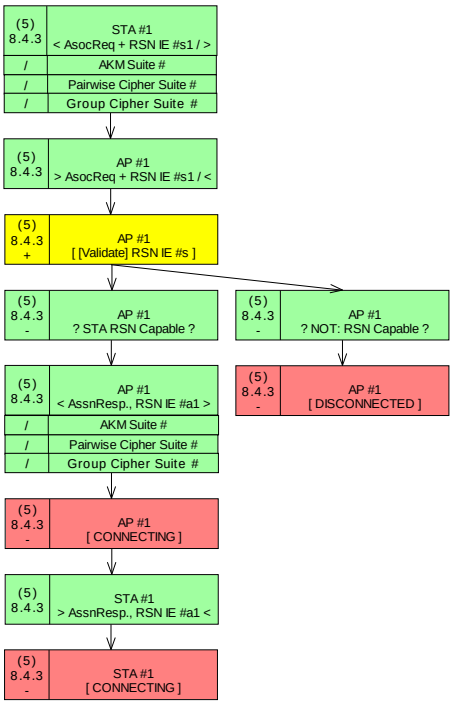
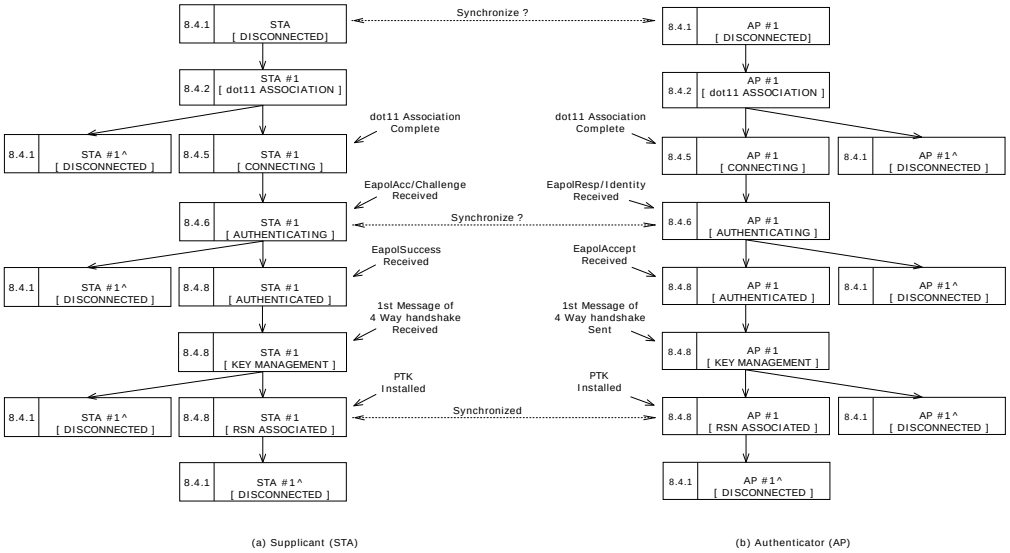


Figure 3. RBT for Requirement 5

During the development of the RBTs we encountered several incompleteness and uncertainties in requirements. We used appropriate domain expertise to resolve these ambiguities. Table 1 below lists the ambiguities and the relevant decisions taken by us.



Figue 4. Supplicant/Authenticator Component Behavior Projection

Having developed all the RBTs, we systematically and incrementally integrated the twelve RBTs to construct the design behavior tree (DBT) that satisfies all its requirements. During the integration process several integration issues were identified. These integration issues, which are mostly due to inconsistencies in pre and post conditions, were again resolved using appropriate domain expertise. A complete record of the modeling technique and the various models derived from the requirements can be found in [24].

A. Component behavior projection

Component projection models for the supplicant and the authenticator are derived by systematic inspection of the DBT. We did this by simply ignoring the component-states of all components other than the one we are currently projecting. The resulting connected behavior tree for a particular component defines the behavior of the component that we will need to implement and encapsulate in the final component-based implementation. The projected component behavior for the supplicant and the authenticator are shown in Fig. 4.

In the component projection models, both the STA and the AP are initially at the DISCONNECTED state, which means that the control port is at unauthorized state. From this state the STA begins the dot11 association by sending a ProbeReq signal. This dot11 association state of the STA is indicated as dot11 ASSOCIATION in the STA projection model. In case if the STA is unable to establish common security parameters with the AP, the STA will decline connection reverting to the DISCONNETED state. Once dot11 association is completed both the STA and the AP transfer to the CONNECTING state enabling the port filters.

Next the dot1x authentication begins. During this process both the AP and the STA first transit to AUTHENTICATING state followed by AUTHENTICATED as discussed in the earlier section. However, at any instance if the supplicant is unable to establish its identity the authenticator declines connection with the STA, thereby pushing the STA to the DISCONNECTED state.

When dot1x authentication process successfully completes, both STA and the AP share a common pairwise master key (PMK) and reach AUTHENTICATED state. This state initiates the 4-way handshake. During the 4-way handshake both the pairwise transient keys (PTK) and the groupwise transient keys (GTK) are installed on both the STA and the AP. At this point both the AP and the STA become RSN-ASSOCIATED and the controlled port reach the authorized state allowing normal data traffic. During the 4-way handshake if the STA or the AP does not disclose the correct RSN capabilities as advertised in the dot11 association process, the STA will DISCONNECT from the AP and vice-versa.

The projection models of GSE are comparable to the supplicant and authenticator PAE state machines presented in the 802.11 standards. Unlike the state machines which are created by deduction, the projection models are derived in a systematic manner from the requirements. Although the projection models do not show state transition details as in the state machines, such information can be obtained from the DBT with the requirement tracking numbers present in the BT models.

The projection models derived show the normal behavior of the STA and the AP. Accordingly, both STA and the AP have definite state changes with reversions permitted to DISCONNECTED states only. This normal behavior of our model does not permit state transitions from one intermediate state to another. Tracking this behavior in the software implementations of the STA and the AP can be a useful exercise for anomaly detection and intrusion prevention.

VI. MODEL CHECKING

First, we used a special toolset [25] developed by the ARC Center for Complex Systems to automatically translate the BT model into formal notations like *Communicating Sequential Processes* CSP [26] and SAL. The static analysis of the translated specification is possible using different analysis tools available for CSP and SAL. In this study we have translated the integrated model for security policy selection, authentication and key distribution into SAL specification.

SAL is an integrated environment of static analysis tools that includes tools for model checking and theorem proving. In the SAL environment the systems are specified using a description language for state transition. The system properties of interest are calculated in SAL based on the system expressed as a transition system in this description language. In the SAL environment a number of tools provide support for *abstraction, program analysis, theorem proving,* and *model checking*. A detailed description of the translation of BT to SAL specification is beyond the scope of this paper but only a brief overview has been provided here. In BT models concurrent systems are expressed as state transitions, hence, the translation rules for a subset of BT notation is relatively straight forward. A wider coverage of translation of BT notation is part of the ongoing research work at the ARC Center for Complex Systems [27].

The BT is represented in a single SAL transition system *module*. The components and their states are declared as *state types* in the module. The BT events that are marked with INPUT tags are translated as *input* variables. A set of special variables called *pc1*.. *pcN (*program counters) is used to keep track of concurrent state transitions in the tree. An atomic action can be either manually specified or automatically generated from a set of BT state transitions between two external (observable) events.

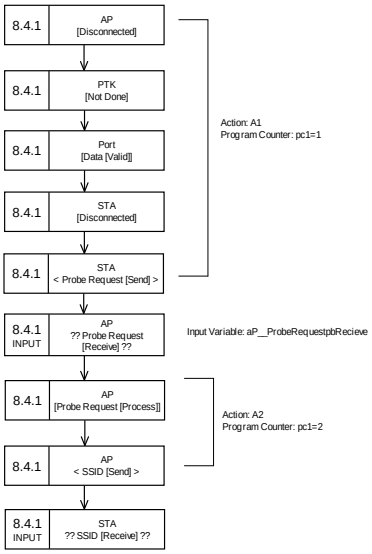


Figure 5. Translation of BT into SAL specification

In the BT segment shown in Fig. 5, the state transition from the root node *(`AP [Disconnected]`)* until the node right before the event *(`AP ??ProbeRequest[Receive] ??`)* is considered as one action which is guarded by the initial value of the program counter. The SAL translation generated by the translator for the above BT segment is shown below:

```
INITIALIZATION
pc1=1
TRANSITION
[]
A1: pc1=1
   -->aP'=apDisconnected;
      sTA'=staDisconnected;
      pTK'=notDone;
      port_Data'=pdValid;
      sTA_ProbeRequest'=pbSend;
      pc1'=2;
[]
A2: pc1=2 AND aP_ProbeRequestpbRecieve
   -->aP_ProbeRequest'=pbProcess;
      aP_SSID'=idSend;
      pc1'=3;
[]…
```

In the above SAL specification the program counter (pc1) has been initialized to 1. This program counter serves as a guard for the first action A1, which starts at the root of the tree and includes states until the node just before the first event. The new states of the state variables in the SAL code directly correspond to the first segment of the BT shown in Figure 5. The action A1 also increments the program counter pc1 to 2 to indicate the process control can now move to the next possible action. The second action, A2, is guarded by the incremented value of program counter (pc = 2) and the input variable `aP_ProbeRequestpbRecieve`. This input variable corresponds to the first event in the BT example illustrated in Figure 5. If the conditions for the second action (A2) are satisfied, then the set of state transitions occur which correspond to the BT state transitions shown after the first event in Fig. 5.

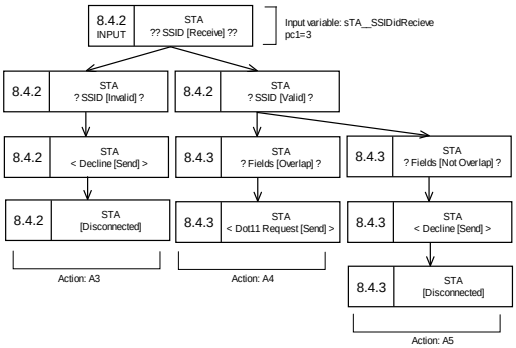


Figure 6. Translation of BT with multiple paths

In the next example we show how BT with multiple paths

as shown in Fig. 6 is translated into SAL code. In this example, the set of transitions in action A3 is guarded by input variable *sTA_SSIDidRecieve* and *pc1 = 3* along with the condition *sTA_SSID=idInvalid*. If these conditions are true then the set of state transitions indicated in action A3 will be performed. The other two actions (A4 and A5) are also possible if and when the event happens. Both of these actions are guarded by a set of conditions i.e. *pc1=3, sTA_SSIDidRecieve, sTA_SSID= idValid, sTA_Fields=Overlap* and *pc1=3, sTA_SSIDidRecieve, sTA_SSID=idValid, sTA_Fields= notOverlap*, respectively. It may be noted that the program counter has been set to 3 before the process flow can continue down any of the paths (A3, A4 or A5) in the BT. This indicates that, at this point in the BT the process flow control can go to any branch that fulfills the conditions imposed on it. The SAL code for the example is shown below:

```
…[]
A3: pc1=3 AND sTA_SSIDidRecieve AND
sTA_SSID=idInvalid
        -->sTA_SSID'=idRecieve;
           sTA_Decline'=dSend;
           sTA'=staDisconnected;
           pc1'=6;
[]
A4: pc1=3 AND sTA_SSIDidRecieve AND
sTA_SSID=idvalid AND sTA_Fields=overlap
        -->sTA_SSID'=idRecieve;
           sTA_dot11Request'=d11Send;
           pc1'=7;
[]
A5: pc1=3 AND sTA_SSIDidRecieve AND
sTA_SSID=idInvalid AND sTA_Fields=notOverlap
        -->sTA_SSID'=idRecieve;
           sTA_Decline'=dSend;
           sTA'=staDisconnected;
           pc1'=8;
[]…
```

Once the system has been specified in the SAL environment language, a number of analyses can be performed [28] on the system specification. The SAL environment contains a symbolic model checker called sal-smc. It allows users to specify properties in linear temporal logic (LTL) and computation tree logic (CTL). However, the current version of SAL does not provide counterexamples for CTL properties. When users provide an invalid property in LTL, a counter example is produced. LTL formulas state properties about each linear path induced by a module (transition system). Typical LTL operators are:

- G (p) (read “always p”), stating *p* is always true
- F (p) (read “eventually p”), stating that *p* will be eventually true
- U (p, q) (read “p until q”), stating that *p* holds until a state is reached where *q* holds
- X (p) (read “next p”), stating that *p* is true in the next state

For example, the formula G (p => F (q)) states that whenever *p* holds, *q* will eventually hold. The formula G (F (p)) states that p holds infinitely often. In the following section we describe how these properties are used to model check the IEEE 802.11i security protocol, using the sal-smc.

VII. ANALYSIS

Having generated the SAL code for the IEEE 802.11i security protocol we developed a number of theorems to formally verify the model. Firstly, we verified all the assumptions made by us during the modeling process to resolve ambiguities and/or inconsistencies. Table 1 shows all such issues and the corresponding LTL theorems used to verify. The GSE modeling technique compels all ambiguities and inconsistencies to be resolved during the requirements translation and integration. The component behavior projection models shown in Fig. 4 demonstrate the normal behavior of our security model. Accordingly, both STA and AP are expected to transit states simultaneously as defined by the following LTL formulas.

```
G ((sTA = staConnecting) => (aP = apConnecting))
G ((sTA = staAuthenticating) => (aP = apAuthenticating))
G ((sTA = staAuthenticated) => (aP = apAuthenticated))
G ((aP = rsnAssociated) => (sTA = rsnAssociated))
```

All of the above four LTL theorems were proved confirming the normal behavior as per our component behavior projection. The normal behavior of the STA and AP was further verified with the following LTLs.

```
U ((sTA = staConnecting), ((sTA = staDisconnected)
   OR (sTA = staAuthenticating))
U ((sTA = staAuthenticating), ((sTA = staDisconnected))
   OR (sTA = staAuthenticated))
U ((sTA = staAuthenticated), (sTA = staDisconnected))
   OR (sTA = rsnAssociated))
```

The above LTL theorems were also proved confirming our aim that the participating components are disconnected from every intermediate state if they do not transit to the next state. We have made these state transitions mandatory in our models to protect our system from possible security threats which can arise from exploiting the unsynchronized behavior of the STA and AP.

IEEE Clause	Req. No.	Defect	Description	LTL Formulas
8.4.1	1	Missing	Initial State of an AP assumed DISCONNECTED	INITIALIZATION
	1	Uncertain	Post-condition if an AP does not advertise a valid SSID not clear	G ((sTA_SSID = idInvalid) => F (sTA = staDisconnected))
8.4.2	3	Missing	Initial state of a STA assumed DISCONNECTED	INITIALIZATION
	3	Missing	Post-condition if STA not RSN capable assumed DISCONNECTED	G ((sTA_RSN = rsnIncapable) => F (sTA = staDisconnected))
	3	Missing	Post-condition of cipher suites mismatch assumed DISCONNECTED in ESS	G ((sTA_Fields = notOverlap) => F (sTA = staDisconnected))
	3	Missing	STA intermediate state assumed CONNECTING (ref. dot1x)	Please see examples given in the text
	4	Uncertain	STA is able to decapsulate a message but sees invalid SSID	G ((sTA_SSID = idInvalid) => F (sTA = staDisconnected))
8.4.3	5	Missing	Post-condition of RSN-IE mismatch assumed DISCONNECTED in ESS	G ((sTA_Fields = notOverlap) => F (sTA = staDisconnected))
	5	Missing	Post-condition of dot11 association failure assumed DISCONNECTED	G ((sTA_dot11 = dot11Reject) => F (sTA = staDisconnected))
8.4.5	7	Missing	Post-condition if dot1x Auth not supported assumed DISCONNECTED	G ((sTA_dot1x = notSupport) => F (sTA = staDisconnected))
8.4.6	8	Assumed	STA at ACQUIRED state when EapolReq/Identity received (ref. dot1x)	G ((sTA_Eapol = reqidReceive) => F (sTA = staAcquired))
	8	Assumed	AP at AUTHENTICATING state once EapolResp/Identity received (ref. dot1x)	G ((aP_Eapol = residReceive) => F (aP = apAuthenticating))
	8	Assumed	STA at AUTHENTICATING state once EapolAcc/Challenge received (ref. dot1x)	G ((sTA_Eapol = accchReceive) => F (sTA = staAuthenticating))
	8	Missing	Post-condition when AS rejects STA identity assumed DISCONNECTED	G ((AS_AccChallenge = stachReject) => (sTA = Disconnected))
	8	Assumed	STA at AUTHENTICATED state once EapolSuccess received (ref. dot1x)	G ((sTA_Eapol = succReceive) => F (sTA = staAuthenticated))
8.4.8	10	Missing	Pre-condition for 4-way key exchange assumed AUTHENTICATED	Please see examples given in the text
	10	Missing	Post-condition after the GwK's are installed assumed RSNA	Please see examples given in the text

Table 1. Theorems Proved to Resolve Ambiguities

Once the model was verified for normal behavior, we then verified the consistency of the security association process. Clauses 8.4.2 and 8.4.3 of the standard describe the security policy selection process in the WLAN. To begin with, an AP returns the SSID in response to a probe request from the STA. An STA has to decline associating with the AP if it receives an invalid SSID. The following LTL theorem was established to test this condition. The *sTA_SSID* is a sub state in our BT model and is reached once the SSID is received from the AP.

G ((sTA_SSID = idInvalid) => F (sTA = staDisconnected))

The above LTL formula was proved to confirm that an STA eventually disconnects itself, if it receives an invalid SSID. However, if the STA receives a valid SSID it then validates the RSN capabilities advertised by the AP. If the AP is not RSN capable or does not match the capabilities of the STA, the STA disconnects itself maintaining strict RSN policy. In the following theorem we check for the validity of authentication and cipher suites.

G ((aP_Rsn = inCapable) OR (aP_Cipher = inValid)
OR (aP_Auth = inValid)) => F (sTA = staDisconnected))

The above LTL formula was proved confirming the behavior of the STA when it receives invalid RSN suites. On the other hand, when the STA advertises its capabilities the AP also performs similar validation. Once both STA and AP agree on the common security policy they both transit to CONNECTING state and continue with the authentication process.

In our BT model we introduced a component named *control__Port* to indicate the controlled port between the AP and the STA. This component was initialized to *pUnauthorized* state to indicate the unsecured state of the port. During IEEE 802.1X authentication and key distribution process this port continues to remains unauthorized. Finally, only when the PTK is installed on both the AP and the STA it transits to the authorized state - *pAuthorized* enabling normal data traffic. Having this in mind, we used the following LTL formula to examine that the control port remains unauthorized during the authentication and the 4-way key handshake process.

G ((control__Port = pUnauthorized) =>
((sTA = staConnecting)
OR (sTA = staAuthenticating)
OR (sTA = staAuthenticated)
OR (sTA = 4wayHandshake)))

This formula was proved confirming the notion that the control port remains unauthorized throughout the authentication and key distribution process. Finally, we tested the authorized state of the control port with the following LTL theorem.

G ((control__Port = pAuthorized) =>
((aP = apptkInstalled) AND (sTA = staptkInstalled)))

The above LTL theorem was proved to confirm the authorized state of the control port implying PTK is installed on both AP and STA. All of the above analysis was done to verify the normal behavior of our model during the security association process. However, one must keep in mind that during this process the participating hosts will demonstrate their normal behavior if both communicating hosts operate in a synchronized manner. If we cannot guarantee the synchronized operation of the hosts the security of the participating hosts become dubious as discussed below.

A. Security Issues

As highlighted in **Clause 8.4.1** permitting APs to advertise their SSIDs can lead to malicious associations. Furthermore, as shown in Fig. 4, during the dot11 association both supplicant and the authenticator operate independently without any form of software synchronization. Therefore, in a situation where the supplicant or the authenticator is allowed to make presumptions about the characteristics of other participating components can lead to malicious associations or Identity-Theft [29]. In case of a re-association request by a roaming STA, we first transit the STA into DISCONNECTED state before it is made to associate with the new AP. This makes the RSN more reliable so that session-hijack attacks [30] can be avoided. In our model the following LTL formula

was proved to be false endorsing our decision:

U ((sTA = rsnAssociated), ((sTA = staDisconnected)
OR (sTA = staAuthenticating))

Although this is a possible state transition in an RSN, we have deliberately DISCONNECTED the STA to make the RSN more reliable, i.e. an attacker cannot pretend as an Authenticating STA. Further, in **Clause 8.4.2**, we force the STA to DISCONNECT from the AP if it is not RSN capable. Similarly an AP DISCONNECTs itself if it sees an STA that is RSN incapable.

G ((aP_Rsn = inCapable) => F (sTA = staDisconnected))
G ((sTA_Rsn = inCapable) => F (aP = apDisconnected))

In **Clause 8.4.3** both STA and AP disassociate with each other if they don't mutually agree on a common security policy. In Fig. 8, we have illustrated a malicious association scenario making use of this condition. The BT model shows an intruder, InAP (shaded boxes), reading messages from a legitimate AP and appropriately issuing an Association Request to the legitimate AP with a wrong RSN IE (as if it is sent by the associating STA). The legitimate AP disassociates itself assuming the STA is not RSN capable. In the mean time the intruder pretends as the legitimate AP and associates with the mistaken STA.

G (inAP_dot11Request=d11Send) = >
(sTA_Rsn = inCapable) AND (aP = apDisconnected))

If this type of malicious association takes place the intruder can simply walkthrough the legitimate STA via the entire process of authentication and key distribution acquiring all the necessary security information and characteristics of the legitimate STA. This type of an association cannot be stopped by software means and requires some form of synchronization at lower layers of communication.

Let us now focus on the authentication process. As shown in Fig. 9 this process is initiated by the STA issuing the EapStart message. The STA eventually communicates with the Authentication Server (AS) via a secure channel to validate its credentials. As said in **Clause 8.4.6**, if the AS does not recognize the challenge sent by the STA, the STA has to eventually reach the disconnected state as proved by the following LTL formula.

G ((aS_AccChallenge = stachReject)
=> F (sTA = staDisconnected))

This situation also implies that STA and AP will never reach a state where the controlled port becomes authorized. The following LTL formula proves that the control port never reaches the authorized state if STA does not prove its identity.

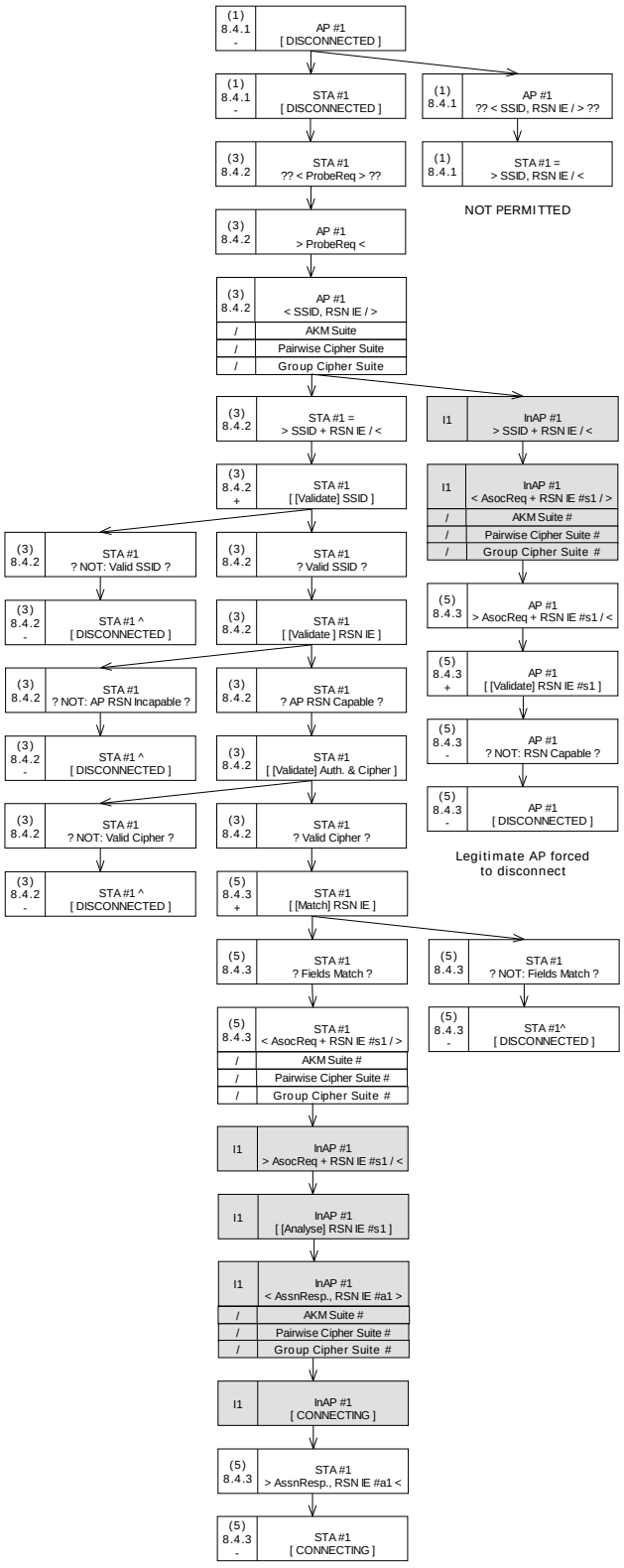


Figure 8. Malicious Association

G ((aS_AccChallenge = stachReject) =>
(control__Port = pUnauthorized))

We use this condition to demonstrate another malicious association scenario in Fig. 9. The intruder having gained vital information about a legitimate STA during the dot11 association, now wants to act as a legitimate STA, shown as InSTA (shaded boxes) in the BT model. The intruder keeps monitoring the messages exchanged during the authentication process. The intruder tracks all messages and just before the AP is ready to authenticate the STA (by sending *EapSuccess* message), it disassociates the STA by sending an *EapFailure* message as if sent from the legitimate AP. The intruder then associates with the AP receiving the *EapSuccess* message.

G ((InSTA_Eap = instaFailure) =>
F (sTA = staDisconnected))

G ((ap_Eap = apSuccess)
=> F (InsTA = instaAuthenticated))

As such, if the intruder successfully associates he will now be in possession of the primary master key (PMK), thereby enabling him to continue the association process and becoming RSN associated. Thus, the intruder has penetrated the organizational network. Although, this sought of an association is difficult with the use of digital certificates or smart card like authentication options, the chances of poorly configured users associating with a wireless AP are high. Unless, every user is educated on the consequences of security breaches and are adequately trained to protect their own environments, the attackers will find these loopholes to gain access to the organizational networks.

According to **Clause 8.4.6**, the control port remains unauthorized during and after the Authentication process. However, the STA reaches the Authenticated state once PMK is received by both AP and the STA or if the shared key is installed.

G ((sTA = staAuthenticated) =>
(((STA_PMK = stapmkInstalled)
AND (AP_PMK = appmkInstalled))
OR (PMK = PSK))

When both STA and the AP reach the Authenticated state the AP begins the 4-way key handshake by sending the first message of the handshake process.

The purposes of the 4-way handshake are:

- Confirm the existence of the PMK. The second message transfer occurs only if the PMKIDs of STA and AP match each other.

G ((sTA_Key2 = K2Send) =>
(STA_PMKID = AP_PMKID))

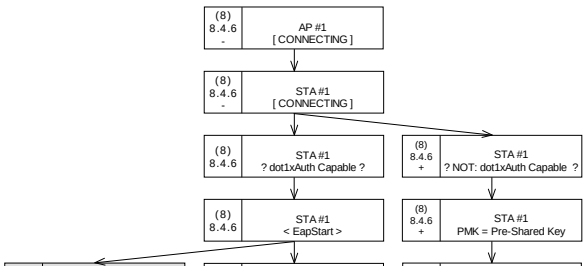


Figure 9. Malicious Association during Authentication

- Ensure that security association keys are fresh. If both STA and AP derive the transient keys they will eventually transit to RSN Associated state.

G ((sTA_PTK = staptkDerive) =>
F (sTA = staRsnAssociated))
G (((aP_PTK = apptkDerive)) =>
F (aP = apRsnAssociated))

- Synchronize the installation of temporal keys. For both STA and AP to reach RSN Associated state transient keys must be installed.

G (((aP = apRsnAssociated) AND
(sTA = staRsnAssociated)) =>
((aP_PTK = apptkInstall) AND
(sTA_PTK = staptkInstall)))

Other than the session hijack issue discussed earlier, the IEEE 802.11i security mechanism has another weak point in the key distribution process. In the earlier case we described how the intruder could gain access to the PMK. Now, having gained access to the PMK, we show in Fig. 10 how the intruder can perform a session hijack. During the second and third message exchanges both the STA and AP validate the RSN IEs. They both compare the RSN IEs advertised during the dot11 association with that of the RSN IEs transferred during the second and third messages. If the RSN IEs do not match they disassociate. Therefore, an intruder monitoring the key exchanges can deliberately disassociate an STA by issuing a third message with an invalid RSN IE as if it is sent by the AP. Having disconnected the legitimate STA, the intruder will now receive the correct third message from the AP, installs the PTK and become RSN associated. As such, the intruder is now in full control of the wireless network.

However, one could argue that the above security issues are insignificant and with the strong authentication and key distribution mechanisms proposed in IEEE 802.11i, they are annulled. Yet, the point is that intruders one way or another make use of such loopholes to gain some knowledge of the participating hosts and eventually succeed in penetrating the network. Therefore, it is our responsibility to take utmost precaution in every perspective to protect the WLAN.

The above discussion shows the importance of a complete and a consistent set of requirements together with a proven analysis technique. Issues in requirements can lead to defects in the final system. In this case such defects can lead to significant wireless attacks that could endanger an entire organization. However, merely conducting a rigorous analysis is not ultimate. It is also necessary to ensure that issues resolved during the analysis are effective and pertinent. The GSE technique together with formal verification is effective not only for requirements analysis and validation; but it also provides a systematic approach to integrate the system ensuring that all parts of the system corporate and coordinate correctly with good traceability. The formal nature of the Behavior Tree notation used within GSE requires all ambiguities, incompleteness and inconsistencies are resolved at the time of integration.

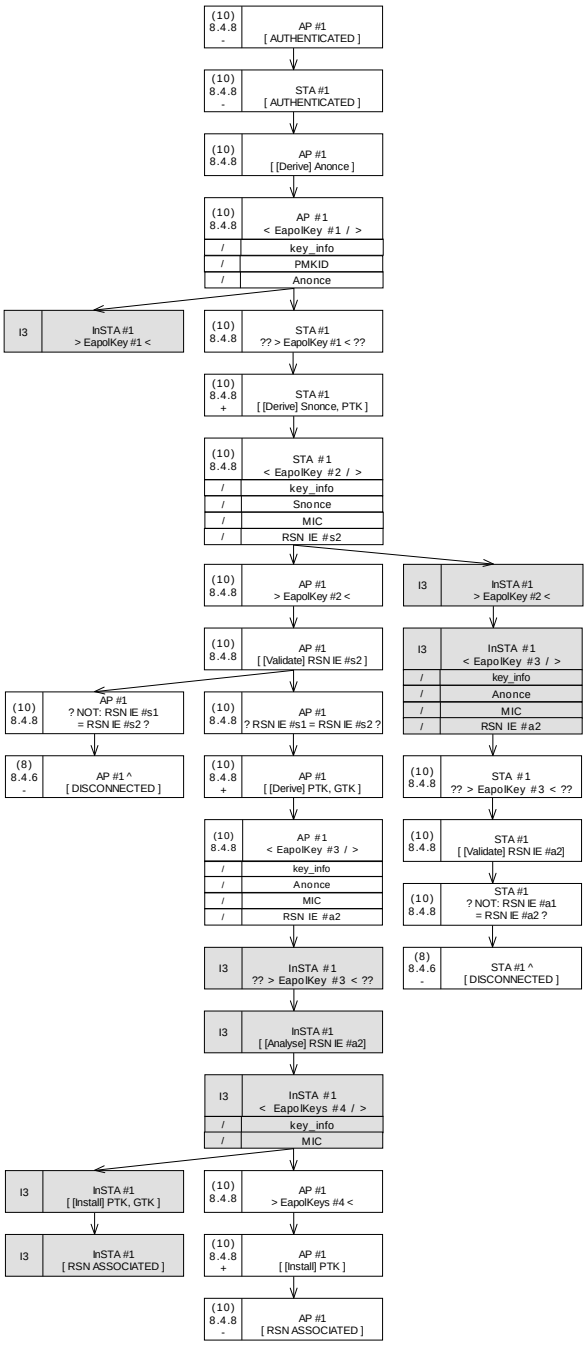


Figure 10. Session Hijack during Key Distribution

VII. CONCLUSION

From our use of the GSE methodology for modeling the

security protocol we have found it to be a promising tool to express the behavior of the system. It definitely helps in getting the system complete and consistent. The systematic approach enables designers to resolve ambiguities early with reliability. Since requirements translation and integration enables issues to be resolved early the resulting model is guaranteed complete and consistent. With our wide experience in applying this methodology to model communication protocols we are confident that this technique is capable of handling such applications and provides designers an opportunity to develop cohesive and reliable systems with good tracebility and control over complexity.

The systematic analysis performed in this study using the GSE methodology identified a number of ambiguities and defects in specifications. Many of the identified incompleteness issues and ambiguities in the standard's requirements arise from semi-tacit and tacit knowledge not being specified. This enabled us to acquire considerable domain knowledge to analyze, design and implement a complete and consistent security model. The formal verification carried out using LTL formulas has proved that GSE models developed are robust and consistent.

However, there were significant gaps in the verification process mainly due to the limitations of SAL translation. The symbolic code generated by SAL is generic and does not meet the requirements of a communication/security protocol. SAL does not provide support symbolic analysis of authentication and key distribution mechanisms. Therefore, we are now focusing on translating the BT models into protocol specific symbolic code which will enable us to verify security protocols more completely.

REFERENCES

[1] Stubblefield, A. Ioannidis, J, Rubin, A.D. A key recovery Attack on the 802.11b Wired Equivalent Privacy Protocol (WEP)", ACM Transactions on Information and System Security, Vol. 7, No. 2, May 2004, pp. 319-332.

[2] IEEE Std. 802.11i-2004, "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. Amendment 6: Medium Access Control (MAC) Security Enhancements", July 2004.

[3] Wi-Fi Alliance. "Wi-Fi Protected Access", October 2002. URL: http://www.wi-fi.org/OpenSection/pdf/Wi-Fi Protected Access Overview.pdf

[4] IEEE Std. 802.1X-2001, "Local and Metropolitan Area Networks – Port-Based Network Access Control", June 2001.

[5] Dromey, R.G. From Requirements to Design: formalizing the key steps, Proc. 1st International Conference on Software Engineering and formal methods, September 2003, pp. 2-11.

[6] Bensalem, S. Ganesh, V. Lakhnech, Y. Munoz, C. Owre, S. Rue, H. Rushby, J. Rusu. V. Saidi, H. Shanker, N. Singerman, E. Tiwari, A. "An Overview of SAL", Proc. 5th NASA Langley Formal Methods Workshop, June 2000, pp. 1-10.

[7] J. Hajek. Automatically verified data transfer protocols. In Proceedings of the 4th ICCC, pages 749 - 756, 1978.

[8] C.H. West. General technique for communications protocol validation. IBM Journal of Research and Development, 22(4), 1978.

[9] M.C. Edmund, and M.W. Jeannette, Formal methods: state of the art and future directions. ACM Computing Surveys, 28(4):626{643, 1996.

[10] McMillan K. Symbolic Model Checking. Kluwer Academic Publishers, 1993.

[11] Gerard J. Holzmann. The model checker SPIN. Software Engineering, 23(5):279 - 295, 1997.

[12] David L. Dill, Andreas J. Drexler, Alan J. Hu, and C. Han Yang. Protocol verification as a hardware design aid. In IEEE International Conference on Computer Design: VLSI in Computers and Processors, pages 522 -525, 1992.

[13] P. Godefroid. Model checking for programming languages using VeriSoft. In Proceedings of the 24th ACM Symposium on Principles of Programming Languages, 1997.

[14] David L. Detlefs, K. Rustan M. Leino, Greg Nelson, and James B. Saxe. Extended static checking, 1998.

[15] David Evans, John Guttag, James Horning, and Yang Meng Tan. LCLint: A tool for using specifications to check code. In Proceedings of the ACM SIGSOFT '94 Symposium on the Foundations of Software Engineering, pages 87 - 96, 1994.

[16] D.R. Engler, B. Chelf, A. Chou, and S. Hallem. Checking system rules using System specific, programmer written compiler extensions. In Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, October 2000.

[17] David Evans and David Larochelle. Improving Security Using Lightweight Static Analysis, In Proceedings of IEEE Software, Jan/Feb 2002.

[18] Manuvir Das, Sorin Lerner, and Mark Seigle. ESP: Path-sensitive program verification in polynomial time. In Conference on Programming Language Design and Implementation, 2002.

[19] J.W. Gray and J. McLean. Using temporal Logic to specify and verify cryptographic protocols (progress report). In proceedings of the 8th IEEE Computer Security Workshop, 1995.

[20] T.Y.C. Woo and S.S. Lam. A semantic model for authentication protocols. In proceedings of the IEEE Symposium on Research in Security and Privacy, 1993.

[21] D. Dolev and A. Yao. On the security of public key protocols. IEEE transactions on Information Theory, 29(2):198-208, March 1989.

[22] C. Medows. Applying formal methods to the analysis of a key management protocol. Journal of Computer Security. 1:5-53, 1992.

[23] Robert L. Glass, Is this a revolutionary idea, or not? Communications of the ACM, 47(11):23 – 25, 2004

[24] Sithirasenan, E. Muthukkumarasamy, V. "IEEE 802.11i WLAN Security Protocol – A Software Engineer's Model", to appear on the AusCERT 2005 Proceedings, May 2005.

[25] C. Smith, K. Winter, I. Hayes, R.G. Dromey, P. Lindsay, and D. Carrington, "An Environment for Building a System Out of Its Requirements," presented at Tools Track, 19th IEEE International Conference on Automated Software Engineering, Linz, Austria, 2004.

[26] K. Winter, "Formalising Behavior Trees with CSP," presented at International Conference on Integrated Formal Methods, IFM'04, 2004.

[27] L. Grunske, P. Lindsay, N. Yatapanage, and K. Winter, *Unpublished Results*, ARC Centre for Complex Systems, 2005.

[28] Moura, L. d., *SAL: Tutorial*, SRI International, 2004

[29] Arbaugh, W.A. Shankar, N. Wan, J. "Your 802.11 Wireless Network Has No Cloths", IEEE Wireless Communications, Dec. 2002, pp. 44-51.

[30] Mishra, A. Arbaugh, W.A. "An Initial Security Analysis of the IEEE 802.1X Standard", Critical Infrastructure Grant, National Institute of Standards, February 2002.