Using Predicate-based Model Checker for Verifying E-Commerce Protocols

Tarek M. El-Sakka[1] and M. Zaki[2]
(Corresponding author: Tarek M. El-Sakka)

Central Laboratory for Agricultural Expert Systems (CLAES), Ministry of Agriculture, Giza, Egypt[1]
Systems & Computers Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt[2]
(Email: telsaka@sharjah.ac.ae)

(Received Dec. 06, 2009; revised and accepted Mar. 30 & May 31, 2010)

Abstract

Over the past decade, researchers have demonstrated that the technique of model checking can be extremely effective when applied to security or e-commerce protocols. Model checking is the process of determining whether a formal model of the analyzed system satisfies a correctness property specified as a temporal logic formula. Model checking result is either a claim that the property is true or else a counterexample showing that the property is false. E-Commerce protocols are techniques used to secure E-Commerce transactions. E-Commerce protocols have to own one or more from the security properties like safety, aliveness, authentication, and integrity. Unfortunately, the conventional model checking does not have the definition of these security properties, which are essential for the E-Commerce protocols. In addition, scalability is a desirable property of a protocol, which indicates its ability to handle growing amounts of work in a graceful manner, or to be readily enlarged. In this paper, we extend the conventional NuSMV model checker by adding new predicate layers to enhance its ability for verifying properties of E-Commerce protocols. The new predicates are scalable that are used to check gradient properties of different E-Commerce protocols. The new model combines features for model checking with predicate facilities. The new model can analysis and verify E-Commerce protocols easily and effectively. Therefore, we use to analyze some E-Commerce protocols to verify its security properties.

Keywords: E-Commerce protocols, model checking, predicate abstraction, scalability

1 Introduction

With continuing growth of E-Commerce on the Internet, the issues of trust and fairness are becoming increasingly important [16]. Therefore, E-Commerce protocols have to address standard requirements such as confidentiality of information, integrity of data, cardholder account authentication, and merchant authentication. These requirements and corresponding functions are not specific to E-Commerce protocols. These requirements are the security requirements for E-Commerce protocols.

Consequently, various methods are proposed to address the security risks arising in E-Commerce. Secure Socket Layer (SSL) or the SSL-based protocol Transport Layer Security (TLS) are usually used in preference to Secure Electronic Transaction (SET) for Internet E-Commerce transaction security. Recently, the three-domain (3D) architecture has been introduced to try meet both security and implementation requirements. Two main examples of 3D schemes have been proposed, namely 3-Domain Secure, which builds on the SSL/TLS protocol, and 3D SET, which is a 3D version of SET [12]. To ensure fairness on the E-transactions, there are protocols for fairness like the NetBill and Fair-Exchange protocols. In addition, many crypto-protocols use cryptographic techniques included to support authentication of E-Commerce like Kerberos and Needham-Schroeder (NS) protocols.

Software testing is one of the oldest forms of verification. A 2002 study commissioned by the National Institute of Standards and Technology (NIST) found that software errors cost the United States economy about $59.5 billion annually [13]. Traditionally formal methods and software testing have been seen as rivals [11]. Formal methods are a combination of a mathematical or logical model of a system and its requirements, together with an effective procedure for determining whether a proof that a system satisfies its requirements is correct [17]. There are several different formal methods for analyzing the protocol security, detecting protocol failures, and designing secure protocols [17].

Over the past decade, researchers have demonstrated that the technique of model checking can be extremely effective when applied to security or e-commerce protocols [14]. Model checking is the process of exploring a finite state space to determine whether a property holds [2]. Model checking result is either a claim that the property is true or else a counterexample showing that the property is false. An important advance in model checking was the introduction of symbolic representations of state spaces, which allowed direct exploration of the state space to be replaced by the manipulation of data structures representing the transition relation of the state space [2]. The major problem of model checking is that the state spaces arising from practical problems are often huge, generally making exhaustive exploration infeasible [2].

One of the most successful symbolic model checkers is the branching time model checker SMV (Symbolic Model Verifier), which has been developed at school of computer science of Carnegie Mellon University (CMU) [16]. NuSMV originated from the reengineering, reimplementation and extension of SMV [7]. The conventional NuSMV model checker is used to verify NetBill, Fair Exchange, and Needham-Schroeder protocols.

E-Commerce protocols have to own one or more of the security properties like safety, authentication, integrity, freshness. Unfortunately, the available NuSMV tool does not have the definition of several important security properties, which are essential for the E-Commerce protocols. In addition, the definition of the built-in fairness and effectiveness predicate does not support complex expressions that are required for most protocols. The conventional NuSMV tool uses the branching time temporal logic CTL (Computation Tree Logic) that based on a branching notion of time. However, the temporal logic does not support fixed time representation like timestamps that are required to model nonces, which are used to ensure freshness of protocol messages. Furthermore, the available NuSMV model checker has large state-space size when representing big systems, which leads to the complexity of the model reading, understanding, and maintaining.

NuSMV had distributed with an Open Source license, which allows anyone interested to use freely and to participate in its development. Therefore, NuSMV model checker has the ability to add new source code to improve its old functionalities. An extension is added on the top of the conventional NuSMV model checker to improve the pervious discussed limitations. A predicate is a function that will always evaluate to true or false [8]. New predicates are added on the top of conventional NuSMV model checker to enhance its capability for the verification and analysis of E-Commerce and security protocols.

Predicates are proposed to represent both Finite State Machine (FSM) and its specifications (properties) for the E-Commerce protocols. A scalable model of predicates that used to represent the security properties of the E-Commerce protocols is developed. The new extension adds new facilities to the NuSMV model checker, which are scalability, reusability, and simplicity of predicate abstraction. In addition, the proposed extension has a translator that interfaces between the proposed predicates and the NuSMV tool to transform the predicate form into the SMV language form.

The next section presents the related works. Section 3 presents the conventional model checker. Section 4 presents extension of the NuSMV model checker. Section 5 presents the scalable model of E-Commerce protocols. Section 6 describes how the conventional NuSMV tool used to analyze E-Commerce protocols. Finally, Section 7 describes how to analyze E-Commerce protocols using the extended NuSMV tool.

2 Related Work

Predicate abstraction is a popular form of over-approximation and forms the basis of a number of automated abstraction tools [22]. Constructing an exact predicate abstraction for a given set of predicates requires an exponential number of validity checks to determine the abstract transition relation. Work in predicate abstraction has thus focused on constructing a sufficient conservative upper bound and investigating ways to make validity checks efficient, using satisfiability and binary decision diagram (BDD)-based techniques [20]. Researchers have also combined predicate abstraction model with checkers. Some researchers have extended model checking to overcome its limitations and/or provide new functionalities to model checker.

Chu and Brockmeyer present a new approach for online predicate detection in distributed systems [6]. This approach divides an on-going distributed computation into a series of partial computations and encodes them into NuSMV models once they are observed. NuSMV is invoked to check these models and reports the satisfiability of the predicate. This approach is efficient and scalable since predicate detection is based on fast symbolic model checking and is performed while the computation is executing. Experiment results show that it is suitable for online monitoring and scales well.

Another work, made by Ray and Sumners [20], presents a procedure for proving invariants of computing systems that uses a combination of theorem proving and model checking. Their procedure automates invariant proofs while imposing no restriction on the expressiveness of the language used to define systems and their properties. The procedure includes lightweight theorem proving to generate a predicate abstraction, which they then explore through model checking.

Bryant and Rajamani [4] present automatic techniques for formally verifying hardware and software by creating Boolean abstractions of the underlying unbounded system state variables. They use two verification tolls that support predicate abstraction, in which the state of the system is characterized by a set of Boolean predicates, describing properties and relations among the state variables, yielding a Boolean abstraction of the underlying system.

In addition, Visser and his colleagues [22] investigate the use of abstraction techniques to reduce the state-space of a real-time operating system kernel written in C^{++}. They show how informal abstraction arguments could be formalized and improved upon within the framework of predicate abstraction, a technique based on abstract interpretation. They introduce some extensions to predicate abstraction that all allow it to be used within the class-instance framework of object-oriented languages. They then demonstrate how these extensions were integrated into an abstraction tool that performs automated predicate abstraction of Java programs.

Dams and Namjoshi [8] propose a framework, based on predicate abstraction and model checking, for shape analysis of programs. Shape analysis is used to collect information about program stores. Rather than use a specialized abstract interpretation based on shape graphs, they instantiate a generic and automated abstraction procedure with shape predicates from a correctness property. The correctness property is then checked on the abstraction with a model-checking tool. To enable this process, they calculate weakest preconditions for common shape properties, and present heuristics for accelerating convergence.

Eiter et al. [9] present novel complexity results on answer set checking for non-ground programs under two methods for representing answer sets and a variety of syntactic restrictions. They consider set-based and bitmap-based representations, which are popular in implementations of Answer Set Programming systems. Based on these results, they also derive new complexity results for the canonical reasoning tasks over answer sets, under the assumption that predicate are bounded by some constant. Their results imply that in such a setting more efficient implementations than those currently available may be feasible.

Zheng et al. [24] propose a new requirement capture and analysis method, which constitutes a smooth, small-step, incremental and iterative development process cut in by FORMS, clewed and cored by SUBJECT, PREDICATE and OBJECT logic. When applying this method they can overcome the incoherence and bounce in the course of requirement analysis and form requirement stage to design stage as well.

3 Model-based Checking

Many security problems in protocol analysis can be formulated in terms of the properties of a discrete system. Therefore, it has long realized that formal methods can be useful for the analysis of the security of cryptographic protocols. There are several different techniques of formal methods, which have applied to analyze cryptographic protocols:

1) Language based methods;

2) Algebra based methods;

3) Logic based methods;

4) Automata based methods;

5) Model based methods [10].

Model-based methods are software tools that incorporate a protocol state-transition model [10]. While the abstract model includes the usual state variable for the intruder's set of known items, the search algorithms expressed recursively use a state representation with no explicit mention of the known set [3]. These systems attempt to locate protocol security flaws by an exhaustive search of the states space. In addition, it has an equation-solving facility for terms using encryption and other operators used in authentication protocols. It models protocol participants as communicating state machines with an intruder who can intercept participants' messages to each other and destroy, modify, or pass messages through without modification.

Model checking is the process of exploring a finite state space for determining whether a property holds [5]. Model checking result is either a claim that the property/specification is true or else a counterexample showing that the property is false [16]. A temporal logic defines specifications/properties. Generally, the temporal logic that has used is either CTL or Linear Temporal Logic (LTL). One of the most successful symbolic model checkers is the branching time model checker SMV, which was a BDD-based model checker, developed at school of computer science of CMU [16]. SMV is a tool for symbolic model checking of finite state systems against specifications written in a temporal logic. SMV supports both deterministic and nondeterministic models, and provides for modular system descriptions. SMV contains Boolean, scalar, and fixed array data types.

NuSMV is a symbolic model checker originated from the reengineering, reimplementation and extension of SMV [7]. The additional features contained in NuSMV include a textual interaction shell and graphical interface, extended model partitioning techniques, and facilities for LTL model checking [16]. NuSMV allows for the analysis of specifications expressed in CTL, LTL, and Property Specification Language (PSL) [7]. NuSMV had distributed with an Open Source license, which allows anyone interested to use freely and to participate in its development [7].

Figure 1 illustrates NuSMV system taking as input a FSM and a property φ expressed in temporal logic and outputs true if the finite state system satisfies φ or false otherwise [23]. If the outcome is false, NuSMV outputs a counterexample, which allows users to understand why the finite state system does not satisfy the property. NuSMV model checker simulates all possible behaviors of the system model need to analyze, in order to verify whether a requirement (property) is satisfied.

In most systems like payment systems, protocols used to improve security and protect against frauds. The NuSMV model checker is used to verify the E-Commerce protocols, like NetBill, Fair Exchange, and Needham-Schroeder protocols. E-Commerce protocols have to own one or more of the security properties like safety, aliveness, authentication, integrity. Unfortunately, the available NuSMV tool does not have the definition of these

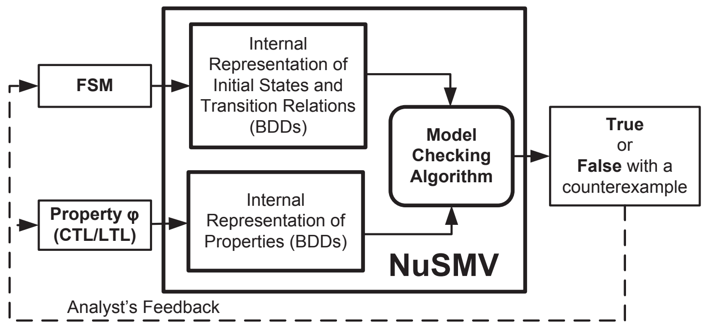
FSM
Property φ (CTL/LTL)
Internal Representation of Initial States and Transition Relations (BDDs)
Internal Representation of Properties (BDDs)
Model Checking Algorithm
NuSMV
True or False with a counterexample
Analyst's Feedback

Figure 1: Conventional NuSMV model checking software

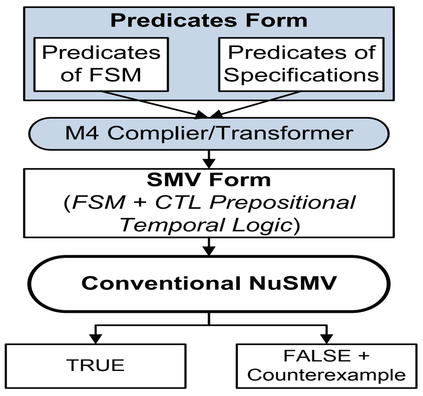

Figure 2: Layout of the verification work flow on the extended NuSMV model checker

security properties, which are important for E-Commerce protocols. In addition, the NuSMV tool does not support using complex expressions for the built-in fairness and effectiveness predicates.

The conventional NuSMV tool uses the branching time temporal logic CTL that based on a branching notion of time. However, the temporal logic does not support fixed time representation like timestamps that is required to model nonce, which used to ensure freshness of protocol messages. Furthermore, the available NuSMV model checker has large state-space size when representing big systems, which leads to the complexity of the model reading, understanding, and maintaining.

4 Predicate-based Extension to NuSMV Model Checker

The NuSMV has simplified software architecture because of its different components and functionality have been isolated and separated into modules, and it provides interfaces between modules. This should reduce the effort needed to modify and extend NuSMV. Because it is open source and has a very clean architecture, it was very easy to implement any algorithms on top of it. NuSMV model checker has the ability to add new functionalities to improve its old functionalities. The NuSMV is extended to improve the discussed limitations above by developing two layers on the top of the tool. The first layer contains new predicates that represent both FSM and specifications of the analyzed protocols. The second layer is the transformation layer that transforms the protocols written in predicates form into the SMV language form. Fixed predicates are added to represent the security properties to be used in the analysis and verification of the E-Commerce protocols.

Figure 2 illustrates the two proposed layers that extending the NuSMV tool. These layers has been developed using a macro (preprocessor) language on the top of the SMV language. The macro language is the GNU M4 language that is an implementation of the traditional UNIX macro processor. GNU M4 was originally written by Rene' Seindal, with subsequent changes by Franc, Ois Pinard and other volunteers on the Internet [24]. The M4 preprocessor language is used to build predicates syntax for representing both the FSMs and its specifications (properties) of the modelled protocols.

In addition, in the second layer, the M4 macro language is used to build evaluations and transformations of the new predicates form into SMV language form.

The new predicates at the first layer are divided into two types of predicates that are used to represent protocols in the proposed extended model checker. The first type of predicates used to represent the FSMs for the verified protocols. While, the second type of predicates represent the properties of protocols, like safety, aliveness, freshness, authentication, integrity, confidentiality, and separation. The predicates of protocol's properties are categorized as two categories. The first category contains predicates that represent the essential properties of protocols like safety and freshness. While, the second category contains predicates to represent the advanced protocol's properties.

The predicates describe each FSM as a list *States, Events,* and *Transitions.* A *State* is a predicate that evaluates to true when the State is the current state. An *Event* is a predicate represents the activity done when the predicate evaluates to true. A *Transition* is a list of next states provides the new current state when a transition made after some predicates evaluate to true. Each FSM process is defined within a SMV Module using two proposed predicates. The first predicate defines variables using the following form:

var(VariableName, *Data type*)

Where Data type can be Boolean, integer range (i.e. From..To), or atomic set of values enclosed in {}. Each module contains variable declarations, using the previous form, to determine its state space. Each variable has an initial state, which defined with the following predicate:

initialize(VariableName, *Initial value*)

Where Initial value can be value of any type related Data type used in the variable declaration. Therefore, the transition relation of the FSM can be described using states and events (states and events are predicates) as described on the following form:

State & Event → new State

Besides predicates that represent FSM, the extended NuSMV contains predicates to represent specifications of FSM. These specifications (properties) are Safety, Aliveness, Freshness, Authentication, Integrity, Confidentiality, and Separation. The Safety property means that nothing bad ever happens, e.g. never two processes in critical section at the same time. In other words, only one process is in its critical section at anytime (or no deadlock). The Safety property has the form $AG\neg(C_1 \wedge C_2)$, which means for all states, any two processes P_1 and P_2 must not have the current state $State_1$ at the same time. Safety is built with the $Safety$ predicate that has the following form:

Safety(P_1.CurrentState($State_1$), P_2.CurrentState($State_1$))

CurrentState is another predicate to check the current state of variable. The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Safety predicate. It defines the commented macros “Safety” and “CurrentState” for translating the predicate form to the SMV form.

define('Safety', 'SPEC AG !('$1' & '$2')')
define('CurrentState', 'ifelse($♯, 1, CurrentState1($@), CurrentState2($@))') define('CurrentState1', 'state = $1')
define('CurrentState2', '$1 = $2')

When calling the safety predicate through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$\rightarrow Safety(P1.CurrentState(State1) \\ \& \; P2.CurrentState(State1))$$

$$\boxed{\begin{array}{l} SPEC\ AG!(P1.CurrentState1(state1) \\ \qquad \& \; P2.CurrentState1(state1)) \\ SPEC\ AG!(P1.state = state1 \\ \qquad \& \; P2.state = state1) \end{array}}$$

The Aliveness property means something desirable will eventually happen, e.g. whenever a process takes control, it will always return it. In other words, whenever any process requests to enter its critical section, it will eventually be permitted to do so. The Aliveness property has the form $AG(T_1 \rightarrow AFC_1)$, which means for all states of process P_i, if P_i starts transition in the state $State_1$, it eventually ends transition with the state $State_2$. Aliveness is built using the Aliveness predicate that has the following form:

$$Aliveness(P_1.CurrentState(State_1), \\ P_1.CurrentState(State_2))$$

The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Aliveness predicate. It defines a commented macro 'Aliveness' for translating the predicate form to the SMV form:

$$define(`Aliveness', `SPEC\ AG(`\$1' \\ \rightarrow AF`\$2')')$$

When calling the Aliveness predicate through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$\rightarrow Aliveness(P1.CurrentState(State1), \\ P1.CurrentState(State2))$$

$$\boxed{\begin{array}{l} SPEC\ AG(P_1.CurrentState_1(state1) \\ \qquad \rightarrow AFP_1.CurrentState_1(state_2)) \\ SPEC\ AG(P_1.state = state_1 \\ \qquad \rightarrow AFP_1.state = state_2) \end{array}}$$

The Freshness property means representation of the time interval within which the corresponding component considered valid. The freshness is passed in the messages in the form of timestamp for that message. Timestamps are numbers marking a specific instance of time. Therefore, freshness means whenever sender sends a message in t_1 time it must be received by receiver at t_2 time, where t_1 and t_2 are time units related to transition states of the FSM, and $t_1 < t_2$ by a specific period of time. The Freshness property has the form $AG(AFmsg_i(t_1) < msg_i(t_2))$, which means if the process P_1 sends the message msg_1 at time t_1 then eventually the process P_2 receives the message msg_1 at time t_2, where $t_1 < t_2$. This due to that $(t_1 + Period = t_2)$, where *Period* is a fixed time assumed as a twice state transition at ideal cases. Each time unit is mapped with one state transition. Freshness is built with the *Freshness* predicate that has the following form:

Freshness$(P_1.msg_1Sendtime), P_2.msg_1RecevTime$

The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Freshness predicate. It defines a commented macro “Freshness” for translating the predicate form to the SMV form:

$$define(`Freshness', \\ `SPECAG(AF((`\$1' + Period) = `\$2'))'),$$

where *Period* is a fixed period assumed to represent the occupied time between sending and receiving message msg1 at ideal cases. When calling the Freshness predicate through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$Freshness(P1.msg1Sendtime, P2.msg1RecevTime)$$

$$\boxed{\begin{aligned}&\text{SPEC AG (AF } ((P_1.msg_1Sendtime + Period)\\&\qquad = P_2.msg_1RecevTime))\end{aligned}}$$

The Authentication property means the protocol parties prove to each other their identities. In other words, when an entity finishes its part of the protocol, the other entity must have taken part in the protocol execution. Therefore, if an entity X ends (starts) n protocol executions with Y as responder (initiator), then in the past the entity Y must have started (ended) at least n protocol executions with X as initiator (responder). The property asserts that Y must prove its identity to X. In the case of mutual authentication protocol, there exists a similar property for representing X that must prove its identity to Y. A violation of this property means that an entity ends a session that the other entity has not started, i.e., the dishonest entity must have impersonated an entity. The Authentication property has the form $AG(X.begin(init, Y) >= Y.end(resp, X))$, which means if process P_1 begins (ends) protocol executions with process P_2 as initiator (responder) then process P_2 ends (begins) protocol executions with process P_1 as responder (initiator). The counters $BeginInitWithP_2$, $EndRespWithP_1$, $BeginInitWithP_1$, and $EndRespWithP_2$ are used to count protocol executions by processes P_1 and P_2. Authentication is built with the Authentication predicate that has the following form to authenticate both P_1 and P_2:

$$Authentication(P_1.BeginInitWithP_2, P_2.EndRespWithP_1)$$
$$Authentication(P_2.BeginInitWithP_1, P_1.EndRespWithP_2))$$

The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Authentication predicate. It defines a commented macro "Authentication" for translating the predicate form to the SMV form:

$$define(`Authentication', `SPECAG('\$1' >= `\$2')').$$

When calling the Authentication predicates (for P_1 and P_2) through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$Authentication(P_1.BeginInitWithP_2,$$
$$P_2.EndRespWithP_1)$$

$$\boxed{\text{SPEC } AG(P_1.BeginInitWithP_2 \geq P_2.EndRespWithP_1))}$$

$$Authentication(P_2.BeginInitWithP_1,$$
$$P_1.EndRespWithP_2)$$

$$\boxed{SPECAG(P_2.BeginInitWithP_1 \geq P1.EndRespWithP_2))}$$

The Integrity property means the assurance that unauthorized entity has not altered a term, which means if an entity X receives n times a term from Y, then in the past the entity Y must have expressly sent at least n times this term to X. The Integrity property has the form $AG(X.send(Y, term) \geq Y.recv(X, term))$, which means for process $P_1(P_2)$ sends (receives) message msg_1 to (from) $P_2(P_1)$, the counter $SendMsg1_ToP_2(RecvMsg_1FromP_1)$ is used to count times a message msg_1 sent (received) to (from) $P_2(P_1)$. Integrity is built with the Integrity predicate that has the following form:

$$Integrity(P_1.SendMsg_1ToP_2, P_2.RecvMsg_1FromP_1)$$

The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Integrity predicate. It defines a commented macro 'Integrity' for translating the predicate form to the SMV form:

$$define(`Integrity', `SPECAG(`\$1' = `\$2')').$$

When calling the Authentication predicate through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$Integrity(P_1.SendMsg_1ToP_2, P_2.RecvMsg_1FromP_1)$$

$$\boxed{SPECAG(P_1.SendMsg_1ToP_2 = P_2.RecvMsg_1FromP_1))}$$

The Confidentiality property means that unauthorized entities must not know a term. In other words, it is required that the dishonest entity never receives or deduces this term. Therefore, entity Y cannot obtain a term from an entity X more times than this term has been expressly sent to it. The confidentiality property has the form $AG(X.send(I, term) \geq I.recv(X, term))$, which means when the dishonest entity I take part to a certain protocol execution, nobody sends confidential terms to it, thus $X.send(I, term)$ **is always equal to zero**. Consequently, in this case the specification requires that **I.recv(X, term) must be zero.** For each process P_i ($i = 1$ to n) the counter $SendMsg_jToIntruder$ counts times a message msg_j ($j = 1$ to m) sent to Intruder. At the other hand, the counter $RecvMsg_jFromP_i$ counts times a message msg_j received from P_i. Confidentiality is built with the Confidentiality predicate that has the following form:

$$Confidentiality(P_i.SendMsg_jToIntruder,$$

$$I.RecvMsg_jFromP_i).$$

The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Confidentiality predicate. It defines a commented macro 'Confidentiality' for translating the predicate form to the SMV form:

$$define(\text{`}Confidentiality', \text{`}SPECAG((\text{'}\$1' \geq \text{`}\$2') \\ \to AF((\text{`}\$1' = 0)\&(\text{`}\$2' = 0)))').$$

When calling the Confidentiality predicate through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$\to Confidentiality(P_1.SendMsg_1ToIntruder, \\ I.RecvMsg_1FromP_1)$$

$$\boxed{\begin{array}{l} SPECAG((P_1.SendMsg_1ToIntruder \\ \quad = I.RecvMsg_1FromP_1) \to AF \\ ((P_1.SendMsg_1ToIntruder = 0) \\ \quad \&(I.RecvMsg_1FromP_1 = 0))) \end{array}}$$

The Separation property means the assurance that identity for actual entity does not been used in any session of the protocol execution. The Separation specification has the form $AG(X.send(Y, message)! = Y.recv(X, secret))$, which means if an entity X sends a term to Y, then the entity Y must have not expressly know the secret included in term that sent from X. Therefore, for a process P_1 who sends message msg_1 to P_2, the actual secret term does not be included in the msg_1. This only can happen if the system separates the actual secret (like PAN) with another dummy secret. The separation property is built with the Separation predicate that has the following form:

$$\textbf{Separation}(P_1.Msg_1, P_1.Msg_1.Secret).$$

The following fragment gives the M4 macro definitions that could form part of the extended NuSMV library for the transformation of Separation predicate. It defines a commented macro 'Separation' for translating the predicate form to the SMV form:

$$define(\text{`}Separation', \text{`}SPECAG(!(\text{'}\$1')in(\text{'}\$2'))).$$

When calling the Separation predicate through the extended NuSMV, it will process this code and should generate the corresponding SMV form:

$$\to Separation(P_1.Msg_1, P_1.Msg_1.Secret)$$

$$\boxed{SPEC\ AG\ !(P_1.Msg_1 \text{ in } P_1.Msg_1.Secret)}$$

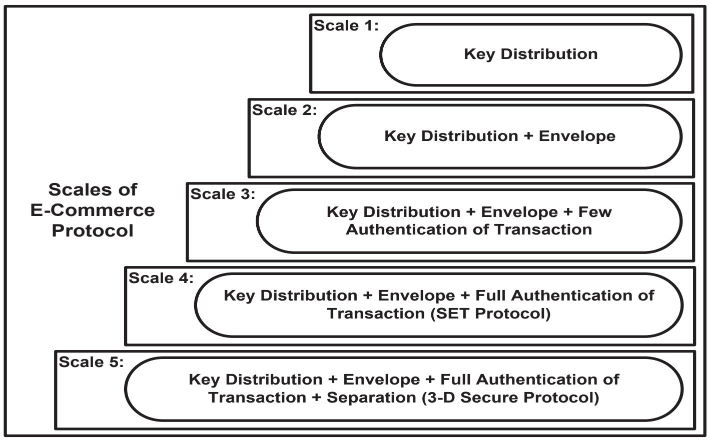
Scale 1:
Key Distribution
Scale 2:
Key Distribution + Envelope
Scales of E-Commerce Protocol
Scale 3:
Key Distribution + Envelope + Few Authentication of Transaction
Scale 4:
Key Distribution + Envelope + Full Authentication of Transaction (SET Protocol)
Scale 5:
Key Distribution + Envelope + Full Authentication of Transaction + Separation (3-D Secure Protocol)

Figure 3: Proposed model for scalable e-commerce protocol

The predicates of properties in the proposed M4 library are implemented using the general definition of the security specifications. These predicates represent the most famous security properties of the E-Commerce protocols. The second layer evaluates predicates then transforms predicate forms into the SMV language forms. The second layer is the M4 transformation library that contains the M4 macro definitions for transforming predicates of both FSM and its specifications. This library is open source macro file that everyone can modify and extend at any time.

5 Proposed Scalable Protocol

E-Commerce protocols have to address standard requirements such as confidentiality of information, integrity of data, participant authentication, and transaction authentication. With E-Commerce continuing to grow, E-Commerce protocols need to be highly scalable. The question is, not only if the E-Commerce protocol system can scale, but also if it can scale efficiently. E-Commerce participants need to select cryptography techniques carefully to avoid having more costly solutions in order to scale adequately. A model for applying scalability to E-Commerce protocol is presented, as shown in Figure 3. This model is a multi-layer model that provides different scales of E-Commerce protocol. These scales are implemented using E-commerce protocol definitions with its security properties. Each scale of E-Commerce protocol presents a different situation for using E-Commerce protocol, which provides different version of E-Commerce protocol. These protocol versions/situations define all security requirements of E-Commerce protocol on different layers.

Figure 3 illustrates the different scales of E-Commerce protocol. The first scale of E-Commerce protocol defines E-Commerce protocol as a key distribution protocol. The second scale adds to the previous scale the envelope features to the E-Commerce protocol. The third scale adds to the second scale some authentication to E-Commerce protocol (party authentication). The fourth scale adds to the third scale full authentication of the transaction by providing more digital signatures and envelopes (like SET protocol).

Finally, the fifth scale adds the separation property to the fourth scale (like 3-D Secure protocol).

These different scales of E-Commerce protocol provide different levels of E-Commerce protocol implementations. These different implementations have different security requirements. Security requirements vary from scale to the other according to the required cryptography features in each scale associated with different situation. Scales of E-Commerce protocol advanced from low to high security. These gradients of security requirements are used in different situations to provide scalability of E-Commerce protocol. Each scale provides a version of E-Commerce protocol to present different E-Commerce protocols from simple (Key Distribution protocol) to complicated protocol (3-D Secure protocol). This will provide in real environment different versions with different security properties, which leads to customization of the cryptographic development.

The different security properties of these scales are implemented using the proposed predicates. The proposed model provides a set of predicates that are categorized as many categories to present these scales. These categories of predicates are mapped to security properties of E-Commerce protocols. One or more of these predicate categories can be used to implement the desired E-Commerce protocol. The proposed extension of model checker can test all of these predicate categories to verify the modelled E-Commerce protocol.

Predicates are used to implement the proposed scales of E-Commerce protocols in the extended NuSMV. In addition, the extended NuSMV verifies properties of those scales, which are authentication, integrity, confidentiality, and separation. The extended NuSMV verifies these properties and provides its truth or falsity. Finally, the extended NuSMV proves these scales of E-Commerce protocol against its expected properties.

5.1 First Scale of E-Commerce Protocol

The first scale of E-Commerce protocol is provided as a key distribution protocol. Key distribution features are authentication and secrecy of key. This scale appears if both protocol parties need to exchange their keys only to authenticate themselves. Therefore, this scale of E-Commerce protocol can work as a separate edition of E-Commerce protocol to exchange keys between any two parties. Therefore, the first scale of E-Commerce protocol ensures authentication and integrity of information (included key as part of signed messages) sent between E-Commerce participants. However, this scale does not provide the other security properties, like authentication and secrecy of information, whereas the secrecy of key provided. This due to simplicity of the E-Commerce scale that is assumed in this situation. To provide other security requirement another scale of E-Commerce protocol must be built.

5.2 Second Scale of E-Commerce Protocol

This scale applies "Digital Envelope" encryption on the messages of the E-Commerce protocol. Therefore, second scale provides secrecy of secret information or secret keys using. The digital envelope ensures confidentiality that is an important security feature. The second scale of E-Commerce protocol adds to the first scale (key distribution scale) an extra envelope security feature. Therefore, this scale provides the security properties: secrecy, integrity, and authentication of keys. To provide other security features, another scale of E-Commerce protocol have to be built.

5.3 Third Scale of E-Commerce Protocol

The third scale of E-Commerce protocol uses the extra encapsulation with signature operator. Therefore, it provides extra authentication feature for the exchanged data. It adds an authentication to the protocol transactions. Authentication is ensured through validating some required secure data that are sent between protocol participants. Beside authentication, the third scale provides digital signature, digital envelope, and key exchange (via digital certificate). Therefore, the third scale provides one more security feature to the second scale. The third scale can work as a new version of E-Commerce protocol using the extra encapsulation cryptography operator that is applied to some messages of protocol. This scale provides the key distribution features, Envelope feature, and some authentication feature to the transaction.

5.4 Fourth Scale of E-Commerce Protocol

The fourth scale uses the dual signature cryptography operator that used in the typical SET protocol. Dual signature is an important innovation introduced in SET. The purpose of the dual signature is to link two messages that intended for two different recipients. However, the two messages must be linked in a way that can be used to resolve disputes if necessary. The fourth scale uses the extra encapsulation operator (EncX) in conjunction with the dual signature, to ensure full confidentiality and authentication of the transaction. This conjunction can authenticate transaction using the integrity of information sent in the transaction. Therefore, the fourth scale of SET protocol provides one more security feature to the third scale. This provides full authentication of the transaction by which E-Commerce protocol provides non-repudiation of transactions. Therefore, this scale works like the SET protocol.

5.5 Fifth Scale of E-Commerce Protocol

The fifth scale uses the security features previously presented in the fourth scale of E-Commerce protocol (digital signature, digital envelope, extra encapsulation, and dual signature). The fifth scale adds separation property to security features of the fourth scale. Therefore, it moves the E-Commerce protocol from the SET protocol to be close to the 3-D Secure protocol.

6 Analysis of E-Commerce Protocols Using the Conventional NuSMV

E-Commerce protocols depend on cryptographic methods to ensure security properties for E-Commerce transactions. The NuSMV model checker has been very useful for the analysis of communication and cryptographic protocols. The conventional NuSMV tool is used to analyze and verify three E-Commerce protocols. These protocols are NetBill, Fair Exchange, and Needham-Schroeder protocols. Although, it successfully verifies the NetBill and Fair-Exchange protocols using the fairness property implemented in the tool, it requires to add complex expressions for the fairness property that not allowed by the available tool.

The conventional NuSMV tool fails to verify the Fair Exchange protocol and the Needham-Schroeder protocol because they need security properties. The conventional tool does not include the security properties like safety, aliveness, authentication, integrity. These properties are essential properties for most E-Commerce protocols to verify. The conventional NuSMV success to analysis and verify these protocols because they require simple specifications. Next, the Needham-Schroeder (NS) protocol is analyzed. NS has complicated structure and requires advanced specifications, as described on the following section.

The NS public key protocol has used for communication between parties that trust each other [15]. The NS Public Key Protocol serves to exchange nonces between an initiator A and a responder B through the trusted third party server S using public keys for mutual authentication, i.e. both A and B want to be assured of the identity of the other. The full NS protocol consists of seven steps, four of which are devoted to distributing public keys [19]. The full protocol is as follows:

$$\begin{aligned} A &\rightarrow S : A, B \\ S &\rightarrow A : \{K_b, B\}K_s \\ A &\rightarrow B : \{Na, A\}K_b \\ B &\rightarrow S : B, A \\ S &\rightarrow B : \{K_a, A\}K_s \\ B &\rightarrow A : \{N_a, N_b\}K_a \\ A &\rightarrow B : \{Nb\}K_b. \end{aligned}$$

A reduced version of the protocol is described by the following three steps:

$$\begin{aligned} A &\rightarrow B : \{N_a, A\}K_b \\ B &\rightarrow A : \{N_a, N_b\}K_a \\ A &\rightarrow B : \{N_b\}K_b. \end{aligned}$$

To initiate the NS protocol, initiator A generates a nonce N_a, encrypts it with his own ID under responder B's public key K_b and sends it to B. Then B knows N_a by decrypting the message, generates a nonce N_b, and encrypts N_b as well as N_a under A's public key K_a. When A receives the second message, A knows that B got the initial message by comparing the nonce N_a in these two messages. After that A sends the last message to B, in the same way, B authenticates A [18].

The NS protocol, in the conventional NuSMV tool, is modelled as illustrated in Figure 4. The client Alice (A) is the initiator and the client Bob (B) is the responder of the NS system, while the Intruder (I) is the attacker of the NS system, which have a full control of the transmission between A and B. Therefore, the input of A and B comes only from I, and the output of A and B passes to I, i.e. the principals are not interacting directly with each other's but indirectly through the intruder module. In this way, the intruder can overhear, delete, or store each message and generate new messages by using his knowledge of overheard messages.

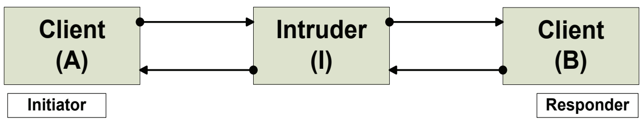

Figure 4: Principals of NS protocol with intruder

The NS protocol depends on nonce included in messages that are exchanged between protocol's initiator and responder. The nonce used in protocol messages to verify its freshness. Freshness represented by a unique value, which are an atomic number. The conventional NuSMV tool uses CTL that based on a branching notion of time. It refers to the fact that at each moment there may be several different possible futures. However, the temporal logic does not support using fixed time representation like timestamps that is required to model nonce, which used to ensure freshness of protocol messages. Therefore, the conventional NuSMV tool fails to represent freshness of the NS protocol and then it fails to analyze it.

In addition, the NS public-key protocol needs security properties to verify, which are safety, aliveness, freshness, authentication, confidentiality, and integrity. The conventional NuSMV tool does not contain definitions for these properties as built-in predicates. The FSM model for the NS public key protocol is implemented. The protocol simulation in the conventional NuSMV tool stops running after 16 states. Therefore, the conventional NuSMV successes to analyze the FSM model for the NS protocol. However, it fails the verify specifications of the NS protocol that are very important to validate correctness and security of the protocol. Specifications of the NS protocol can be built as CTL expressions that may differ from one protocol to the others. Hence, fixed predicates are required to add into the conventional NuSMV, to represent the security properties that can be used in the verification of E-Commerce protocols.

7 Analysis of E-Commerce Protocols Using the Extended NuSMV

The conventional NuSMV model checker is extend by adding two layers on the top the NuSMV. The first layer contains predicates to build FSMs models and define security properties for E-Commerce protocols. These predicates represent the most important security properties for E-Commerce protocols, like Safety, Aliveness, Freshness, Authentication, Confidentiality, Integrity, and Separation. These properties are categorized into two categories. The first category contains the correctness properties, which are essential properties for all protocols, e.g. Safety, Aliveness, and Freshness. The second category contains Authentication, Confidentiality, Integrity, and Separation properties. These properties are required for complicated protocols. The second layer contains transformer from the predicate form into the SMV language form. The extended model checker is used with the first category of predicates to verify the Kerberos protocols. Then, the second category is used to verify the 3-D SET and 3-D Secure protocols.

The conventional NuSMV success to verify the simple cases for E-Commerce protocols but fail to success the complicated cased of E-Commerce protocols. Therefore, the NuSMV is extended with a set of predicates to analysis and verify the complicated E-Commerce protocols. The extended NuSMV success on the verification of Kerberos protocol using the first category of predicates (Safety, Aliveness, and Freshness). In addition, it success to verify the 3-D SET protocol with second category of predicates (Authentication, Confidentiality, Integrity, and Separation) and states that the 3-D SET does not contain the separation property. Finally, the extended NuSMV is used to verify the 3-D Secure protocol. Participants of the 3-D Secure protocol are implemented: Cardholder, Merchant, Acquirer Payment Gateway, Issuer Access Control, Visa Directory, and Intruder. Intruder has a full control of the transmission among protocol parties: Cardholder, Merchant, and Issuer Access Control.

The extended NuSMV model checker simulates the 3-D secure protocol execution that stopped after running 57 states. These states simulate the ideal running process of the FSM that represents the 3-D secure protocol. The implemented security properties for the FSM of the 3-D secure protocol are safety, Aliveness, authentication, integrity, confidentiality, and separation. The extended NuSMV verifies those properties and inform that they are valid for the whole simulation running of the 3-D secure protocol (57 states).

The predicates Safety, Aliveness, Authentication, Confidentiality, Integrity, and Separation are represented using the general predicates represented above. The extended NuSMV evaluates these predicates and informs that it has the value TRUE, as shown in Figure 5. This should verify the security properties for the 3-D secure protocol. Specially, the Separation property is verified - the special property of the 3-D secure protocol. Therefore, the extended NuSMV success to analysis and verify the complicated E-Commerce protocols.

8 Conclusions

Our conclusion is the presenting of the extended model checker, which overcomes limitations of the conventional NuSMV model checking when verifying E-Commerce protocols. The extended NuSMV uses the predicates to represent FSMs and its properties. In addition, we provide a scalable model to present security properties of E-Commerce protocols. This new model provides multi versions of E-Commerce protocols suitable to be used in different situations. This scalable model overcomes the complexity implementation problem of E-Commerce protocols because scales have different structures from simple to complicated cryptographic operations. In addition, it overcomes the delay and slow of using complicated cryptographic operations in the E-Commerce protocol that only used on the last scales. Our scalable model with the predicate abstraction model provides the extended NuSMV to reuse these new predicates in the analysis and verification of other protocols/systems. We add a transformation layer to the conventional model checker to transform from the predicate form to the SMV language form. Our transformer evaluates predicates syntax before transformation. Finally, our extended predicate model checker is used to verify security properties of E-Commerce protocol.

References

[1] *3-D Secure Introduction*, Copyright © Visa International Service Association, Sep. 26, 2002.

[2] P. J. Anderson, P. Beame, S. Burns, W. Chan, F. Modugno, D. Notkin, and J. D. Reese, "Model checking large software specifications," *Proceedings of SIGSOFT'96* , ACM, CA, USA, 1996.

[3] D. Bolignano, "An approach to the formal verification of cryptographic protocols," *Proceedings of the 3rd ACM Conference on Computer and Communications Security*, pp. 106-118, 1996.

```
NuSMV > check_ctlspec
-- specification AG !(cardholder.state = send & merchant.state = send) is true
-- specification AG !(cardholder.state = send & ACS.state = send) is true
-- specification AG !(merchant.state = send & VDir.state = send) is true
-- specification AG !(merchant.state = send & paymentgateway.state = send) is true
-- specification AG !(ACS.state = send & paymentgateway.state = send) is true
-- specification AG !(ACS.state = send & VDir.state = send) is true
-- specification AG (cardholder.state = send -> AF cardholder.state = receive) is true
-- specification AG (merchant.state = receive -> AF merchant.state = send) is true
-- specification AG (paymentgateway.state = receive -> AF paymentgateway.state = send) is true
-- specification AG (ACS.state = receive -> AF ACS.state = send) is true
-- specification AG (VDir.state = receive -> AF VDir.state = send) is true
-- specification AG cardholder.BeginInitWithMerchant >= merchant.EndRespWithCardholder is true
-- specification AG merchant.BeginInitWithCardholder >= cardholder.EndRespWithMerchant is true
-- specification AG merchant.BeginInitWithPaymentGateway >= paymentgateway.EndRespWithMerchant is true
-- specification AG paymentgateway.BeginInitWithMerchant >= merchant.EndRespWithPaymentGateway is true
-- specification AG cardholder.BeginInitWithACS >= ACS.EndRespWithCardholder is true
-- specification AG ACS.BeginInitWithCardholder >= cardholder.EndRespWithACS is true
-- specification AG paymentgateway.BeginInitWithACS >= ACS.EndRespWithPaymentGateway is true
-- specification AG ACS.BeginInitWithPaymentGateway >= paymentgateway.EndRespWithACS is true
-- specification AG merchant.BeginInitWithVDir >= VDir.EndRespWithMerchant is true
-- specification AG VDir.BeginInitWithMerchant >= merchant.EndRespWithVDir is true
-- specification AG ACS.BeginInitWithVDir >= VDir.EndRespWithACS is true
-- specification AG VDir.BeginInitWithACS >= ACS.EndRespWithVDir is true
-- specification AG (cardholder.SendCRToMerchant = merchant.RecvCRFromCardholder -> AF
(cardholder.SendCRToMerchant > 0 & merchant.RecvCRFromCardholder > 0)) is true
-- specification AG (merchant.SendCRECToCardholder = cardholder.RecvCRECFromMerchant -> AF
(merchant.SendCRECToCardholder > 0 & cardholder.RecvCRECFromMerchant > 0))  is true
-- specification AG (cardholder.SendSAToACS = ACS.RecvSAFromCardholder -> AF (cardholder.SendSAToACS > 0
& ACS.RecvSAFromCardholder > 0))  is true
-- specification AG (ACS.SendSARToCardholder = cardholder.RecvSARFromACS -> AF (ACS.SendSARToCardholder
> 0 & cardholder.RecvSARFromACS > 0))  is true
-- specification AG (merchant.SendTDToPaymentGateway = paymentgateway.RecvTDFromMerchant -> AF
(merchant.SendTDToPaymentGateway > 0 & paymentgateway.RecvTDFromMerchant > 0))  is true
-- specification AG (paymentgateway.SendPAToMerchant = merchant.RecvPAFromPaymentGateway -> AF
(paymentgateway.SendPAToMerchant > 0 & merchant.RecvPAFromPaymentGateway > 0))  is true
-- specification AG (ACS.SendPAToPaymentGateway = paymentgateway.RecvPAFromACS -> AF
(ACS.SendPAToPaymentGateway > 0 & paymentgateway.RecvPAFromACS > 0))  is true
-- specification AG (paymentgateway.SendPARToACS = ACS.RecvPARFromPaymentGateway -> AF
(paymentgateway.SendPARToACS > 0 & ACS.RecvPARFromPaymentGateway > 0))  is true
-- specification AG (ACS.SendCMToVDir = VDir.RecvCMFromACS -> AF (ACS.SendCMToVDir > 0 &
VDir.RecvCMFromACS > 0))  is true
-- specification AG (VDir.SendPIToACS = ACS.RecvPIFromVDir -> AF (VDir.SendPIToACS > 0 &
ACS.RecvPIFromVDir > 0))  is true
-- specification AG (merchant.SendPIToVDir = VDir.RecvPIFromMerchant -> AF (merchant.SendPIToVDir > 0 &
VDir.RecvPIFromMerchant > 0))  is true
-- specification AG (VDir.SendLACSToMerchant = merchant.RecvLACSFromVDir -> AF (VDir.SendLACSToMerchant
> 0 & merchant.RecvLACSFromVDir > 0))  is true
-- specification AG !(ACS.MsgData in pan)  is true

NuSMV >
```

Figure 5: Properties of 3-d secure protocol in the extended NuSMV tool

[4] R. E. Bryant and S. K. Rajamani, "Verifying properties of hardware and software by predicate abstraction and model checking," *Proceedings of the International Conference on Computer Aided Design*, pp. 437-438, 2004.

[5] M. Carmen Ruiz, D. Cazorla, F. Cuartero, and J. J. Pardo, "Analysis of the SET e-commerce protocol using a true concurrency process algebra," *Proceedings of the symposium on Applied computing (SAC'06)*, pp. 879-886, Apr. 2006.

[6] C. Chu, and M. Brockmeyer, "Fast online predicate detection using symbolic model checking," *Proceedings of the Conference of Computers and Their Applications*, pp. 470-477, 2005.

[7] A. Cimatti, E. Clarke, E. Giunchiglia, et al., "NuSMV 2: An opensource tool for symbolic model checking," *Proceedings of CAV'02*, pp. 359-364, 2002.

[8] D. Dams and K. S. Namjoshi, "Shape analysis through predicate abstraction and model checking," *Proceedings of the Conference of Verification, Model Checking and Abstract Interpretation*, pp. 310-324, 2003.

[9] T. Eiter, W. Faber, M. Fink, G. Pfeifer, and S. Woltran, "Complexity of model checking and bounded predicate arities for non-ground answer set programming," *Proceedings of the Conference of Principles of Knowledge Representation and Reasoning*, pp. 377-387, 2004.

[10] S. Gritzalis, D. Spinellis, and P. Georgiadis, "Security Protocols over open networks and distributed systems: Formal methods for their analysis, design, and verification," *Computer Communications*, vol. 22, no. 8, pp. 695-707, May 1999.

[11] R. M. Hierons, et al., "Using formal specifications to support testing," *ACM Computing Surveys (CSUR)*, vol. 41, no. 2, Feb. 2009.

[12] P. Jarupunphol and C. J. Mitchell, "E-Commerce and the media influences on security risk percep-

tions," *Proceedings of the Workshop on Internet Technologies, Applications, and Social Impact*, pp. 163-174, 2002.

[13] F. Kazemian and T. Howles, "A software testing course for computer science majors," *SIGCSE Bulletin*, vol. 37, no. 4, pp. 50-53, Dec. 2005.

[14] S. Lu and S. A. Smolka, "Model checking the secure electronic transaction (SET) protocol," *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 358-364, Oct. 1999.

[15] C. Meadows, "Open issues in formal methods for cryptographic protocol analysis," *Proceedings of DISCEX 2000*, pp. 237-250, 2000.

[16] A. Miller, A. Donaldson, and M. Calder, "Symmetry in temporal logic model checking," *ACM Computing Surveys*, vol. 38, no. 3, Sep. 2006.

[17] C. Meadows, "Formal Methods for cryptographic protocol analysis: emerging issues and trends," *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 1, pp. 44-54, Jan. 2003.

[18] T. Nipkow, L. C. Paulson, and M. Wenzel, *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, Springer-Verlag, 1st edition, ISBN: 3540433767, 2002.

[19] L. Paulson, "The inductive approach to verifying cryptographic protocols," *Journal of Computer Security*, vol. 6, pp. 85-128, 1998.

[20] S. Ray and R. Sumners, "Combining theorem proving with model checking through predicate abstraction," *IEEE Design & Test of Computers Journal*, vol. 24, no. 2, pp. 132-139, Mer. 2007.

[21] *Secure Card Payments on the Internet*, European Committee for Banking Standards (ECBS), TR410 Version 1.0 V Nov. 2002.

[22] W. Visser, S. Park, and J. Penix, "Using predicate abstraction to reduce object-oriented programs for model checking," *Proceedings of ACM Conference of Formal Methods in Software Practice*, pp. 3-12, 2000.

[23] Y. Zhang, C. Wang, J. Wu, and X. Li, "Using SMV for cryptographic protocol analysis: A case study," *ACM SIGOPS Operating Systems Review*, vol. 35, no. 2, pp. 43-50, Apr. 2001.

[24] Y. Zheng, H. Wan, L. LI, and C. Deng, "A new software requirement method based on subject, predicate and object logic," *Proceedings of SPW 2005*, Beijing, China, *Springer-Verlag-Verlag*, LNCS 3840, 2005.

T. El-Sakka is a researcher in Central Laboratory for Agricultural Expert Systems (CLAES), Ministry of Agriculture, Giza, Egypt. He received the BS and MS degrees from Computers Engineering Department, Al-Azhar University, Cairo, Egypt. He was a Ph.D. student in Computer Engineering Department at Al-Azhar University and he received his Ph.D. degree on Network Security on March 2010. His research interests include traffic management, routing protocols and Network Security.

M. Zaki is the professor of software engineering, Computer and System Engineering Department, Faculty of Engineering, Al-Azhar University at Cairo. He received his B.Sc. and M.Sc. degrees in electrical engineering from Cairo University in 1968 and 1973 respectively. He received his Ph. D. degrees in Computer Engineering from Warsaw Technical University, Poland in 1977. His fields of interest include artificial intelligence, soft computing, and distributed systems.