SIMULATION BASED VALIDATION OF AUTHENTICATION PROTOCOLS

Krishnan G. Indiradevi, V. S. Suku Nair
Department of Computer Science and Engineering Southern Methodist University Dallas, TX, USA (nair@engr.smu.edu)

Authentication protocols *help to establish trust about the identities of communicating entities. Along with* authorization *and* data confidentiality, *authentication forms a critical component of most non-trivial security frameworks. Over past several years, an alarming number of seemingly secure authentication protocols have been shown to be flawed. By exploiting such flaws, malicious entities can potentially take on identities of trusted entities.*

Attacks on authentication protocols are often too subtle to uncover by simple means, hence considerable research has gone into techniques for analyzing and verifying them. Though the problem is perhaps best studied using formal methods, techniques in that category are generally rather complex and specialized.

This paper proposes a different approach - using simulation as a means of validation. Though unable to conclusively prove security, simulation can be very effective in uncovering hidden flaws. This could be particularly useful for large systems where it may be nearly impractical to apply formal methods. A framework is presented to model authentication protocols with state machines and to validate some of their security properties through simulation.

Keywords: *Authentication Protocols, Distributed Systems, Protocol Validation, and Simulation*

1. Introduction

Protecting sensitive communications from eavesdropping has always been a challenge. Once considered the worry of only governments and militaries, it has come into sharp public focus in recent years as a result of the explosive growth in deployment and use of data networks such as the Internet. With massive amounts of highly sensitive and valuable data sent across these networks, ensuring data integrity and confidentiality now concerns everyone. The availability of cheaper yet increasingly more powerful computing resources has only exacerbated the problem, be- cause, in the wrong hands they can turn into potential weapons. Clearly, securing information during its generation, manipulation, dissemination, and use has become a daunting task.

Communication networks employ various security ser- vices to protect the confidentiality and integrity of data. Authentication aims to provide mutual trust regarding the identities of communicating entities. Principals authenticated thus are given controlled resource access rights via access control mechanisms. Data confidentiality protects information from unauthorized disclosure, while data integrity protects against malicious tampering. Finally, non- repudiation prevents denial of origination or receipt of in- formation - thus enforcing a level of responsibility on the involved parties. The work presented here focuses on validating authentication protocols against their design goals. In recent years, several seemingly secure authentication protocols have been shown to have vulnerabilities. Considering the importance of ensuring foolproof authentication, this has led to extensive research into systematic analysis and verification of authentication systems. This paper presents a framework to model, analyze, and validate authentication systems. The remainder of the paper is organized as follows.

Section 2 brief discusses authentication in networks. The requirements, goals, basic principles, and types of authentication protocols are brief explained, along with common attacks against them. Section 3 examines the need for formal techniques to verify and validate authentication systems, and outlines two predominant approaches using logic based and state based models. Section 4, the focus of this paper, presents the modeling of authentication protocols with algorithmic state machines, and studies the possibility of using simulation as a means to establish a high degree of assurance regarding their security properties. The approach is illustrated through the case study of a classic protocol - the Needham- Schroeder Public Key authentication Protocol (Needham and Schroeder, 1978). The paper concludes by outlining our plans for future work.

2. Authentication

Information networks perform peer entity authentication to establish trust on identities of principals. The aim is to prevent malicious entities from masquerading as others. Most present day networks do authentication through proof by knowledge (ex: via passwords, shared secrets, etc.). Since this knowledge verification during the authentication phase is critical in ensuring the security of subsequent inter- actions, it is quite often protected cryptographically. The crypto system used is generally assumed to be strong, implying that, without proper keys it is impossible to decrypt the messages exchanged. Therefore, key guessing attempts are considered ineffective.

One of the most common ways for mutual authentication is to demonstrate verifiable secret knowledge. In shared key cryptography, pairs of communicating entities share secret keys that are verified during authentication, and used to optionally establish additional secrets for securing further communications. A radically different approach is taken in public key cryptography, where each principal has a unique pair of keys - a public key made known to everyone, and a secret private key. Most such systems have the property that, correct decryption by a public key implies original encryption with the corresponding private key - thus, a principal may demonstrate knowledge of its private key to prove authenticity.

2.1. Authentication Protocols

An *authentication protocol* is a sequence of messages designed specifically to establish the authenticity of communicating principals. If the principals are really who they claim to be, then, at the conclusion of the protocol, they will be convinced of each other's knowledge of certain secrets (and hence identity), and optionally in possession of one or more shared secrets

[Burrows et al., 1989]. A good authentication protocol establishes not only that each principal believes in a secret shared with the other, but also that each believes in the other's belief of the shared secret.

Depending on the crypto system used, authentication protocols are sometimes classified [Clark and Jacob, 1996] as *shared key systems without trusted third party*, *shared key systems with trusted third party*, *public key systems*, etc. Further, depending on the direction of authentication, they may be divided into *one-way (unilateral)* or *two-way (mutual)* protocols.

The following notation is used in the paper:

Principals in a protocol are represented as *A, B etc. Z or I denotes an intruder; an attempt by Z to impersonate A is indicated as Z(A).*

Key pairs for encryption/decryption are denoted by K, K^{-1}. Encryption of message m with key K is represented as $(m)_K$ or$\{m\}_k$. In a shared key system, K_{ab} denotes a key shared by principals A and B. In a public key system, the public and private keys of principal A are denoted respectively by K_a and K_a^{-1}.

Protocols generally use nonces (one-time random numbers) or time-stamps to ensure message freshness N_a and T_a indicate a nonce and timestamp generated by A.

Sequence of messages in a protocol run are denoted as m1, m2, m3, etc. or as 1, 2, 3, etc.

The mechanisms and reasoning of authentication are illustrated below with the Needham-Schroeder public key authentication protocol; the protocol is later used to illustrate the modeling and validation technique presented.

There is also a desire for CASE tools integration through expansible environments, driven by the demand for ever-faster development of software systems. Integrated CASE environments can help software engineers to deliver software systems on time. However, to meet these demands, an integrated CASE environment must be based on a flexible framework that provides a cost-effective tool integration mechanism, encourages portable tools, facilitates the exchange of development information, and adapts to future methodologies. In such an environment, software engineers can coherently mix and match the most suitable tools that support selected methodologies. They can then plug some tools into the environment and begin working with them.

Design is normally an iterative process, and the ability to easily navigate between different tools and notations is important to permit the designer to view concurrently different facets of software development. The ability for a designer to navigate around is also vital, as reusability is something that a CASE environment must promote. A designer must be able to browse through already-captured parts of previous designs to try to see whether any components from prior work can be reused.

2.2. Needham-Shroeder Public-Key Protocol

Based on *public key cryptography*, the Needham-Schroeder protocol [Needham and Schroeder, 1978] is one of the most widely studied and analyzed authentication protocols. In this protocol, communicating principals establish mutual trust by virtue of the properties of public and private keys. Though a trusted *certification authority (CA)* is also involved to distributes valid public keys of principals, we may safely ignore its presence by assuming that the communicating entities already posses the public keys of each other. This simplified version of the protocol is represented below:

$$
\begin{array}{lllll}
(m1) & A & \longrightarrow & B & : \{N_a, A\}_{K_b} \\
(m2) & B & \longrightarrow & A & : \{N_a, N_b\}_{K_a} \\
(m3) & A & \longrightarrow & B & : \{N_b\}_{K_b}
\end{array}
$$

Here, A and B want to verify each other's identity. First, A sends a nonce N_a along with its name to B in message m1 encrypted under B's public key; N_a can be read from m1 only by B (using its private key). B then returns N_a to A along with another nonce N_b (for mutual authentication) in message m2, encrypting it with A's public key. After decrypting m2, A verifies whether N_a is the same as originally sent. If it is, A concludes that it is indeed B on the other side as only B could have decrypted m1. A completes the protocol by returning N_b in message m3 to B, who performs a similar validation and concludes that it is indeed talking to A. Besides achieving mutual authentication, this transaction also establishes two new secrets(N_a and N_b) between A and B which may be used optionally to secure further communications. It is to be noted however, that establishing new secrets in this manner is not generally considered a primary goal of authentication.

2.3. Attacks on Authentication Protocols

Authentication gets tricky when intruders are present. It is to be generally assumed that such an entity may itself be legitimate (thus allowed to take part in authentication), and has the ability to *observe, insert, modify or delete* data in the network. However, it will not be capable of extracting information from an encrypted message for which it does not have the proper key (*strong cryptography assumption*).

The involvement of such abstract properties as *belief* and *trust* often tend to make authentication subtle and non-intuitive. A malicious entity may be able to defeat authentication by exploiting protocol vulnerabilities such as in message structure, sequencing, and timing constraints. The potentially numerous possibilities must be considered carefully when modeling and analyzing authentication systems.

An attack may help the intruder to either learn a current secret, or to trick one or more legitimate participants into using a false `secret' (an old compromised secret, for example). In either case, subsequent communication between the original participants will be insecure.

A variety of attacks on authentication systems have been reported in literature [Bird et al., 1993]. In a *replay attack* , intruder tries to *replay* old messages (from earlier authenticated sessions) as current. In an *oracle session attack*, the intruder engages in simultaneous sessions to two or more principals and manipulates one into doing the cryptographic operations needed to talk to the other. *Parallel session attack* occurs when an intruder engages in two simultaneous authentication sessions to the same principal and uses one session to generate messages required in the other.

Note that other common attacks such as *key guessing attack*, *known plain-text attack*, and *chosen cipher-text attack* result from weaknesses in the underlying crypto system and are not directly tied to authentication protocols.

2.3.1. An attack on Needham-Schroeder public key protocol

Attacks on authentication protocols are almost always too subtle and non-intuitive to be detected through simple methods. The now well-known parallel session attack [Lowe, 1995] on Needham-Schroeder public key protocol as outlined below illustrates this:

$$
\begin{array}{lllll}
(m1) & A & \longrightarrow Z & : & \{N_a, A\}_{K_z} \\
(m1') & Z(A) & \longrightarrow B & : & \{N_a, A\}_{K_b} \\
(m2') & B & \longrightarrow Z(A) & : & \{N_a, N_b\}_{K_a} \\
(m2) & Z & \longrightarrow A & : & \{N_a, N_b\}_{K_a} \\
(m3) & A & \longrightarrow Z & : & \{N_b\}_{K_z} \\
(m3') & Z(A) & \longrightarrow B & : & \{N_b\}_{K_b}
\end{array}
$$

Here the intruder *Z* waits until a principal *A* initiates an authentication session to it (recall that *Z* could be a legitimate user). Then, disguising as *A*, *Z* starts a simultaneous session with *B*. By clever manipulation of messages, *Z* succeeds in falsely taking on the identity of *A* (from the perspective of *B*).

The scenario involves two runs of the protocol - one between *A* and *Z* (*m*1, *m*2, *m*3), and the other between *Z* (posing as *A*) and *B* (*m*1', *m*2', *m*3'). *m*1 is the initial session start message sent by *A* to *Z*. *Z* simply forwards it to *B* (as *m*1') after re-encrypting it with *B*'s public key. Noticing *A*'s name as the message initiator, *B* believes it is talking to *A* and responds with *m*2' to `*A*' (ie. *Z*) - *m*2' carries *B*'s nonce N_b, and is encrypted under *A*'s key. Unable to decrypt *m*2', *Z* just forwards it to *A* as *m*2. Not knowing that *m*2 was actually generated by *B*, *A* accepts it and sends the final message *m*3 to *Z*. *Z* extracts N_b from this message and sends it over to *B* in *m*3'. After verifying N_b, *B* concludes that it is `*A*' at the other end. At this point, even without any key compromise, *Z* has succeeded in masquerading as *A*

The seriousness of this is evident when considering that it was discovered only 17 years after the original protocol was published and widely studied. Clearly, more rigorous techniques are required for verifying cryptographic protocols. The problem has several characteristics that

make the application of *formal verification techniques* suitable. The following section provides a brief overview of some of the formal techniques explored in this domain.

3. Verification and Validation of Authentication Protocols

Most current approaches to analyzing and verifying cryptographic protocols follow *logic based* [Meadows, 1995] or *state based* [Millen, 1995] paradigms.

3.1. Logic-based verification approaches

Pioneered by Burrows, Abadi and Needham [Burrows et al., 1990], logic based frameworks use specialized logics to derive beliefs in a system. Repeated application of *inference rules* on carefully chosen *initial assumptions* and *message meanings* is used to verify security properties such as goodness of keys. Though intuitive and comparatively easy to understand and apply, its high level of abstraction makes it likely to miss certain types of subtle properties while modeling.

The BAN Logic (Burrows, Abadi and Needham) [Burrows et al., 1989] has been used to verify correctness as well as to find previously unknown flaws in many authentication protocols. However, because of limitations in underlying assumptions and lack of sufficiently complex reasoning mechanisms, it has not been very successful in handling certain classes of properties [Meadows, 1995]. Yet, its relative simplicity makes it more popular than several other more powerful logic systems.

Other notable approaches using logic include an extension of BAN Logic by Gong, Needham and Yahalom (GNY Logic) [Gong et al., 1990], and the attempt to unify various logics by Syverson and Oorschot(SVO) [Syverson and van Oorschot, 1994].

3.2. State-based verification approaches

This class of approach pioneered by Dolev and Yao [Dolev and Yao, 1983], uses state-transition models to express algebraic properties. By viewing a model as an *algebraic system* manipulated by an intruder, system state changes are studied in terms of knowledge and data values to capture relatively subtle properties at a fairly low degree of abstraction. However, compared to logic based approaches, modeling and use are harder and require considerable expertise.

In the initial proposal by Dolev and Yao [Dolev and Yao, 1983], a protocol is modeled as a *language system* with a finite set of distinct *symbols* that are used to compose arbitrary *words* which when concatenated produce *messages*. *Reduction rules* (or, *term rewriting rules*) in the language allow transformation between valid word sequences. An intruder's aim is to find words (such as encryption keys and session keys) by subjecting sequences to rewriting rules.

Proving security of the system thus maps to a *word problem* in a *term-rewriting system* manipulated by the intruder [Meadows, 1995].

Several variants of state based modeling have been proposed. Approaches using *reverse search* assume a compromised (insecure) state and search backwards to determine if it is reachable from a valid initial state; the *Interrogator* [Millen, 1995] and *NRL Protocol Analyzer* [Meadows, 1994] follow this model. In approaches based on *CSP formalism* , a system is modeled using Communicating Sequential Processes and then analyzed with CSP methods to prove invariants (secure properties) [Schneider, 1996]. Another popular approach, *state enumeration* , involves searching all reachable states and checking security properties in each; though search space is theoretically infinite, in most practical protocols it is usually possible to confine the search to smaller spaces [Mitchell et al., 1997].

Our work is based on *state based modeling* using finite state machines (FSM).

4. Validation Using State Machines

Finite state machine (FSM) representations provide a framework for conveniently modeling and analyzing communication protocols [Brand and Zafiropulo, 1983]. The basic idea is to model a given protocol as a collection of interacting state machines. Most traditional methods use explicit state enumeration to analyze such systems; though theoretically attractive, this leads to potential state explosion problems in practice.

When modeling a protocol with state machines, state transitions are generally associated with message transmissions and receptions. As transmission by one component will be *coupled* with a reception in another, a system in its entirety maybe thought of as a synchronous machine (this view abstracts out the fact that components operate asynchronously with respect to each other at a finer level). Thus, it is possible to apply state-verification techniques used for synchronous sequential machines to model and verify protocols as well. Protocol design errors such as *state deadlock* , *unspecified reception* , and *non-executable interaction* have been analyzed using FSM models [Brand and Zafiropulo, 1983].

Though intuitive and clear, FSM models often have limitations that tend to restrict their use to fairly simple systems. The foremost is *state explosion* - number of states and transitions tend to grow exponentially as a system grows. Another is *limited expressive power* when compared to other formalisms such as general programming languages. Newer techniques and tools have alleviated these problems to a large extent. This, combined with the convenience and clarity of representation, and the amenability to well understood analysis, makes FSM modeling still an attractive and powerful choice. An FSM based approach presented in [Indiradevi and Nair, 1998] for verifying authentication protocols using *algorithmic state machines* and *property verification* technique is outlined in a later section.

4.1. Algorithmic state machines

Algorithmic state machines (ASM) were proposed as an extension to increase the expressive power of FSMs [Nelson et al., 1995]. They combine FSMs' descriptive power (timing, state relations) with the clarity and expressiveness of flowcharts. Despite its striking similarity with conventional flow charts, an *ASM* diagram has very different semantic interpretation. Unlike a flowchart that merely describes procedural steps and decision paths of an algorithm, an ASM also codifies the sequence of transitions among states, *inputs* triggering transitions, and resulting *outputs* associated with states or transitions. It helps in separately describing and analyzing the *data* and *control* portions of a given system.

ASM diagrams are built from three types of elements [Nelson et al., 1995]: *state boxes*(representing system states), *decision boxes*(determining input-based transitions), and *conditional output boxes*(associating outputs with transitions) [Nelson et al., 1995]. A state box with its associated decision and control boxes together constitute an *ASM block* which corresponds to a state in an equivalent state transition diagram. The resulting expressive power of ASM is thus comparable to that of general programming languages, for instance.

Considering their expressiveness in representing data transformation operations in authentication protocols, we chose ASMs to model protocol entities.

Having thus addressed the problem of extending FSM's expressive power, we now outline *property verification* , a powerful technique to handle the second major problem, namely, *state explosion*.

4.2. Property verification

In *property verification* [Hoskote et al., 1995] [Hoskote, 1995], a given *design* is modeled as a *deterministic* Mealy type finite state machine (possibly incompletely specified). But, unlike most approaches where properties to be verified are described in the design machine itself, the specification properties are described as separate state machines. These *specification machines (property machines)* are modeled as a type of *non-deterministic* state machines with some transitions having unspecified inputs; the partial specification enables efficient *implicit state matching*. The verification procedure computes all states in the *design machine* having input-output behavior *compatible* with that of the *start state* of the *property machine*. Presence of a *compatible state* indicates that the design exhibits the behavior expressed in the property machine, and an absence implies absence of the behavior in the design. Property verification provides an intuitive specification format and gives the power to *implicitly* search the entire state space of the design [Hoskote, 1995] in an efficient manner.

The feasibility of applying property verification to authentication protocols was demonstrated in [Indiradevi and Nair, 1998] by modeling false authentication as a property machine and verifying it against the design state machine of the Needham-Schroeder public-key authentication protocol.

Another potential approach to the problem would be to validate a given protocol through simulation. Unlike the rigorous formal verification methods mentioned so far, simulation does not guarantee correctness of the protocol; in other words, failure of a simulation to expose flaws can not be taken as assurance of their absence in the protocol. However, application of simulation to authentication protocols appears to be promising and viable. This is because most authentication protocols involve only a few entities and a small number of messages, so a well designed simulation has a fairly good chance of detecting flaws.

4.3. Validating Needham-Shroeder protocol

The Needham-Schroeder protocol and an attack on it were discussed in Section 2.1.1 and Section 2.2.1.

In the protocol, two parties A and B attempt mutual authentication through demonstration of their knowledge of private keys. In the attack scenario, a malicious (though legitimate) entity Z tries to subvert authentication by manipulating messages to fool B into believing that Z is A.

We model the protocol entities as state machines interacting through protocol messages. Because of their flexibility and expressive power, we use ASMs (Section 4.1) for modeling entities. This is valuable because, several data transform operation (such as encryption and decryption) need to be represented in the case of authentication protocols - it is neither easy nor convenient with simple state machines. The control flow parts of the machines are used for building the simulation.

For the Needham-Schroeder protocol, following entities need to be modeled:

- Initiators (I) - entities initiating authentication sessions by sending the first message in the protocol
- Responders (R) - entities that respond to messages from an initiator
- Intruder (Z) - a malicious entity (possibly legitimate, capable of engaging in authentication), assumed to have the ability to observe, delete, insert, or manipulate messages in the network; however, it can not decrypt messages for which it does not possess the correct key (strong cryptography assumption).

4.3.1. Modelling trusted entities in the system

Though any entity could function as an *initiator* or as a responder, modeling can be simplified by assuming (without loss of generality) that entities are divided into disjoint sets of initiators and responders. Initiators and responders are assumed trustworthy, who adhere to protocol rules and sequences.

Algorithmic state machines of an initiator and responder derived from the protocol description are shown in Fig.1 and Fig.3 respectively. In these figures,

m1, m2, m3: protocol messages

Tx(m):	transmission of message m
Rx(m):	reception of message m
Ki, Kr:	public keys of initiator and responder
Ki^{-1}, Kr^{-1}:	private keys
E(m):K:	encryption of m with public key K
D(m):K^{-1}:	decryption with private key K^{-1}
nonce:	generation of a fresh nonce
Ni, Nr:	nonces from initiator or responder
T, To:	time-out to detect lost messages
End:	end of an authenticated session
commit:	status of authenticated session

Basic state machines defining the control sequencing of initiator and responder (derived from the algorithmic state machines) are shown in Fig.2 and Fig.4.

4.3.2. Modelling the intruder

Modeling the *intruder* is more complex because of the variety of potential actions it can perform at each step in the protocol. It may initiate a session and follow the protocol, performing proper authentication. Or, it may initiate a session and then resort to message manipulations, out of sequence messaging, and replay of old messages. It could also observe messages between other entities in the network and possibly replay them later.

Further, the intruder is assumed to be capable of intercepting any message on the network. It may destroy the intercepted messages, forward it (modified or otherwise) to the original recipient, or forward it to someone else. It could also replace the message with an old valid message and send to the original recipient, or use the message in another protocol run. Further, being a legitimate entity itself, the intruder is capable of functioning as an initiator or responder. It may act in these roles simultaneously by engaging in multiple protocol runs, and potentially use messages from one run in the other.

Though these numerous possibilities make modeling harder, it can be done systematically (and to a degree of accuracy required for our purpose) by making some observations about the intruder's behavior. The first is that the intruder acts spontaneously only when it initiates an authentication session. This is because, except for m1, other randomly inserted messages (m2, m3) will be rejected by other entities as per protocol definition. The other observation is that, at other times the intruder's actions are reactive - triggered by a message in the network and potentially resulting in one of the actions considered above. Based on these observations, an algorithmic state machine may be derived as shown in Fig.5, Fig.6, Fig.7, and Fig.8.

4.3.3. Modelling a realistic authentication scenario

A difficult choice in modeling concerns the extent of the modeled system. The simplest possible scenario for the Needham-Schroeder public key protocol involves an initiator, a responder, and an intruder. If this system is shown to be vulnerable to false authentication, then clearly any larger system will also have the vulnerability. On the contrary, verifying that the simple system is secure does not imply that larger systems will remain secure under similar assumptions - in such cases, the system has to be expanded incrementally and verification repeated until a sufficient level of confidence has been attained or vulnerability identified.

Theoretically it may not be possible to conclude that a system is secure even after analyzing numerous cases with increasing number of entities. In practice, however, it seems the analysis can be safely limited to a convenient number of participants and protocol runs because most authentication protocols have only a few number of steps, limiting the resulting possibilities. On the other hand, the case of a failure is strong because a weakness that exists in a simple case will almost certainly be present in a larger system as well. In the case of Needham-Schroeder protocol, a weakness is found in the minimal configuration itself, allowing one to conclude that the protocol is vulnerable.

The simple configuration in our model has an *initiator A, a responder B (both trusted), and an* intruder C (able to legitimately act as initiator or responder). Thus A may initiate sessions with B or C, and C may initiate sessions with B. The state machines for A and B correspond to those shown in Fig.1 and Fig.3.

The intruder C may act as initiator (with B), or responder (with A). It may also act maliciously as a passive observer or active intruder between A and B. Actions of C are partly non-deterministic, as it may not abide by protocol sequences or rules. Fig.9 shows C's state machine describing actions of concern to this discussion.

In these figures, a message of the form mxPQ represents the xth message in the protocol run initiated from P to Q, a state of the form PYq denotes the Yth state of P in its interaction with Q, and P_commitQP denotes that P is convinced of the authenticity of Q. End of an authenticated session between P and Q is signaled by endPQ.

A transition labeled ~Rx(m)(ie. non-reception of a message) results from the loss of message m. Since we are primarily concerned with the disruptions caused by the intruder, it is implied that such transitions result when the intruder intercepts and destroys messages. Similarly, ~D(m) denotes a message that cannot be decrypted by the recipient - this could be a message meant for someone else (maliciously forwarded by the intruder) or a valid message that was tampered with (by the intruder). Transitions associated with ~N (ie. non-matching nonce) denote failure of authentication and may result from the reception of old valid messages replayed by the intruder.

As can be seen in the diagram, most transitions in the intruder state machine are controlled by inputs n, d or f. These respectively denote that the intruder decided to respond normally to a message, to delete a message (preventing it from reaching the intended recipient), or to send a fake message (possibly with the intention of masquerading as another entity). States annotated with C_commitAC and C_commitCB result when authentication involving C (using its real identity) has succeeded. C may also try to masquerade as A and send fake messages such as m1A'B to B, and it is possible that state C_commitA'B is reached wherein C has falsely convinced B that it is A. If indeed this state is reached, the protocol is broken.

In the following section, we discuss how these ideas are used in a general simulation environment to validate the protocol.

4.4. Opnet simulation tool

Opnet provides a powerful discrete event driven simulation environment for modeling and simulating a wide variety of communication systems and protocols. Systematic modeling is supported by a hierarchy of *levels (layers)*. In the uppermost *network level*, communicating *nodes* and associated interconnecting *links* are defined. Structures of nodes are defined in detail in the next lower layer in the hierarchy, the *node level*, through components such as *processors*, *queues*, *transmitters*, and *receivers*. Behavior of modeled entities (nodes) is captured through *state transition models* defined at the lowest layer, the *process level*, and embedded within processors and queues. These abstractions, together with user definable message formats and powerful communication primitives for message handling provide a very versatile and flexible simulation environment to system designers.

4.5. Simulation of Needham-Shroeder protocol

A network of communicating entities that follow Needham-Schroeder public key authentication protocol is modeled using Opnet. In the studied configuration, there is one each of *initiator* and *responder*; there is also a malicious party (*intruder*). With the potential to be a legitimate entity in the system, the intruder has the functionality of a normal initiator and responder, besides its malicious behavior.

The ability of the intruder to observe any message in the network is modeled by forcing all messages to go through the intruder node as if it were a transparent *relay* between other communicating parties, a reasonable assumption in most communication networks. Being malicious, the intruder node may try to delete, modify, or manipulate messages rather than just relay.

In the model, *initiator* nodes invoke authentication sessions to randomly chosen *responders*at random intervals. It is to be noted that initiators may invoke sessions to the *intruder* node as well since the latter is assumed to be a legitimate responder. Similarly, in its role as a legitimate initiator, the intruder may invoke sessions to other responders.

Fig.10 shows this scenario implemented as an Opnet *network level* model. In the figure, *i_101* and *i_102* are initiators, *r_201* and *r_202* are responders, and *z_301* is the intruder (the

numbers in these names are just for convenience and have no other significance). Only *i_101, r_201, and z_301*were kept active in initial simulations, thus the studied system configuration was minimal.

The internal structure of nodes is defined in lower *node level* models as shown in Fig.11, Fig.12, and Fig.13. Each node has a *processor* and one or more of *transmitter/receiver* pairs. Transmitters and receivers attach to communication links connecting nodes.

The behavior of entities is defined by *processes* running in their processors. Processes are represented at the *process level* of Opnet as state transition diagrams and associated embedded code. Opnet allows executable code to be associated with states (entry/exit), transitions, or combinations of both. State transition models embedded within the different types of entities (nodes) are shown in Fig.14, Fig.15, and Fig.16. State machines of initiators and responders are straight translations of the protocol sequence and rules. However, the state machine of the intruder is considerably more complex as it embeds the behavior of initiator and responder and numerous malicious possibilities (deletion, modification and manipulation of messages).

4.5.1. Simulation setup

In the studied configuration, there are three types of possible sessions : between *i_101* and *r_201*, between *i_101* and *z_301*, and between *z_301* and *r_201*.

Session initiations are attempted by *i_101* and *z_301* at randomly chosen instants, with a mean interval of about 500 simulation seconds. A successfully authenticated session is assumed to be active for 20 simulation seconds (the values 500 and 20 were chosen to simplify simulation design, they do not affect the validity of simulation). Messages sent between *i_101* and *r_201* pass through *z_301* as explained earlier. The intruder may (in a random manner) relay, delete or manipulate the messages passing through it. If all messages of a particular session are relayed properlu, the session succeeds; if any message is deleted or modified by the intruder, the involved entities are expected to detect it according to protocol rules and abort the session if needed. However, there may be cases where manipulation by intruder produces messages that satisfy protocol requirements. Since the affected entities will not be able to identify such messages as malicious, the intruder's actions may go undetected and lead to false authentication - these are the cases of interest to us. In the simple configuration studied here, a false authentication would occur if the intruder *z_301* is able to pass off as the initiator *i_101* in a session with the responder *r_201*.

4.6. Simulation results

This paper described a framework for modeling authentication protocols with state machines and validating them through simulation. The technique was illustrated using the Needham-Schroeder public key authentication protocol as a case study. It is hoped that the presented approach may enable protocol designers to conveniently and intuitively model and validate a variety of protocols.

We plan to extend the present work by investigating its applicability to more authentication protocols and attack scenarios. The attack on Needham-Schroeder public key protocol as modeled and analyzed in this paper is among the trickier ones. Many other protocols can be attacked in simpler, but non-intuitive ways. Yet others fall to attacks that exploit subtleties in cryptographic techniques, message structure, or data formats. Such scenarios need more complex modeling - we hope that the use of algorithmic state machines to capture finer details of data manipulation and the flexibility afforded by the simulation environment will be valuable in this.

Another important issue to be explored relates to the efficiency of simulation. The present simulation assumes that the intruder's actions are fairly random. It may be possible to incorporate heuristics that increases the possibility of reaching compromised states quicker, resulting in faster convergence of simulation. However, in doing so, one must be careful not to make explicit assumptions that could give ``too much guidance" and make the simulation meaningless.

5. Conclusions and Future Directions

This paper described a framework for modeling authentication protocols with state machines and validating them through simulation. The technique was illustrated using the Needham-Schroeder public key authentication protocol as a case study. It is hoped that the presented approach may enable protocol designers to conveniently and intuitively model and validate a variety of protocols.

We plan to extend the present work by investigating its applicability to more authentication protocols and attack scenarios. The attack on Needham-Schroeder public key protocol as modeled and analyzed in this paper is among the trickier ones. Many other protocols can be attacked in simpler, but non-intuitive ways. Yet others fall to attacks that exploit subtleties in cryptographic techniques, message structure, or data formats. Such scenarios need more complex modeling - we hope that the use of algorithmic state machines to capture finer details of data manipulation and the flexibility afforded by the simulation environment will be valuable in this.

Another important issue to be explored relates to the efficiency of simulation. The present simulation assumes that the intruder's actions are fairly random. It may be possible to incorporate heuristics that increases the possibility of reaching compromised states quicker, resulting in faster convergence of simulation. However, in doing so, one must be careful not to make explicit assumptions that could give ``too much guidance" and make the simulation meaningless.

6. References

[Bird et al., 1993] Bird, R., Gopal, I., Herzberg, A., Janson, P. A., Kutten, S., Molva, R., and Young, M. (1993). Systematic design of a family of attack-resistant authentication protocols. *IEEE Journal on Selected Areas in Communications*, 11(5):679-693.

[Brand and Zaropulo, 1983] Brand, D. and Zaropulo, P. (1983). On communicating finite-state machines. *Journal of the ACM*, 30(2):323-342.

[Burrows et al., 1989] Burrows, M., Abadi, M., and Needham, R. (1989). A logic of authentication. Research Report SRC-39, DEC Systems Research Center.

[Burrows et al., 1990] Burrows, M., Abadi, M., and Needham, R. (1990). A logic of authentication. *ACM Transactions on Com- puter Systems*, 8(1):18{36.

[Clark and Jacob, 1996] Clark, J. and Jacob, J. (1996). A survey of authentication protocol literature. Survey report, University of York, UK. (http://www.cs.york.ac.uk/ jac/).

[Dolev and Yao, 1983] Dolev, D. and Yao, A. C. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, IT-29(2):198-208.

[Gong et al., 1990] Gong, L., Needham, R., and Yahalom, R. (1990). Reasoning about belief in cryptographic protocols. *In Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234{248. IEEE Computer Society Press.

[Hoskote, 1995] Hoskote, Y. V. (1995). Formal Techniques for Verification of Synchronous Sequential Circuits. PhD thesis, University of Texas, Austin.

[Hoskote et al., 1995] Hoskote, Y. V., Abraham, J. A., and Fussell, D. S. (1995). Automated verification of temporal properties specified as state machines in vhdl. *In Proceedings Fifth Great Lakes Symposium on VLSI, Bualo, NY*, pages 100 -105.

[Indiradevi and Nair, 1998] Indiradevi, K. and Nair, V. S. S. (1998). Formal verification of authentication protocols. Technical Report 98-CSE-09, Southern Methodist University, Dallas, TX, USA.

[Lowe, 1995] Lowe, G. (1995). An attack on the needham-schroeder public key authentication protocol. *Information Processing Letters*, pages 131 -136.

[Meadows, 1994] Meadows, C. A. (1994). The nrl protocol analyzer: An overview. *In Proceedings of the 2nd Conference on the Practical Applications of Prolog. Association for Logic Programming.*

[Meadows, 1995] Meadows, C. A. (1995). Formal verification of cryptographic protocols: *A survey. In Advances in Cryptology - Asi- acrypt '94, LNCS 917*, pages 133-150. Springer-Verlag.

[Mil3, 1997] Mil3 (1997). *Opnet Documentation*. MIL 3, Inc., Washington DC, USA. (http://www.mil3.com).

[Millen, 1995] Millen, J. K. (1995). The interrogator model. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 251-260. IEEE Computer Society Press.

[Mitchell et al., 1997] Mitchell, J. C., Mitchell, M., and Stern, U. (1997). Automated analysis of cryptographic protocols using murphi. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 141-151. IEEE Computer Society Press.

[Needham and Schroeder, 1978] Needham, R. M. and Schroeder, M. D. (1978). Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993-999.

[Nelson et al., 1995] Nelson, V. P., Nagle, H. T., Carroll, B. D., and Irwin, J. D. (1995). *Digital Logic Circuit Analysis & Design*. Prentice Hall, Englewood Cliffs, New Jersey.

[Schneider, 1996] Schneider, S. (1996). Security properties and csp. *In Proceedings of the IEEE Symposium on Security and Privacy*, pages 174-187. IEEE Computer Society Press.

[Syverson and van Oorschot, 1994] Syverson, P. F. and van Oorschot, P. C. (1994). On unifying some cryptographic protocol logics. *In Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pages 14-28. IEEE Computer Society Press.

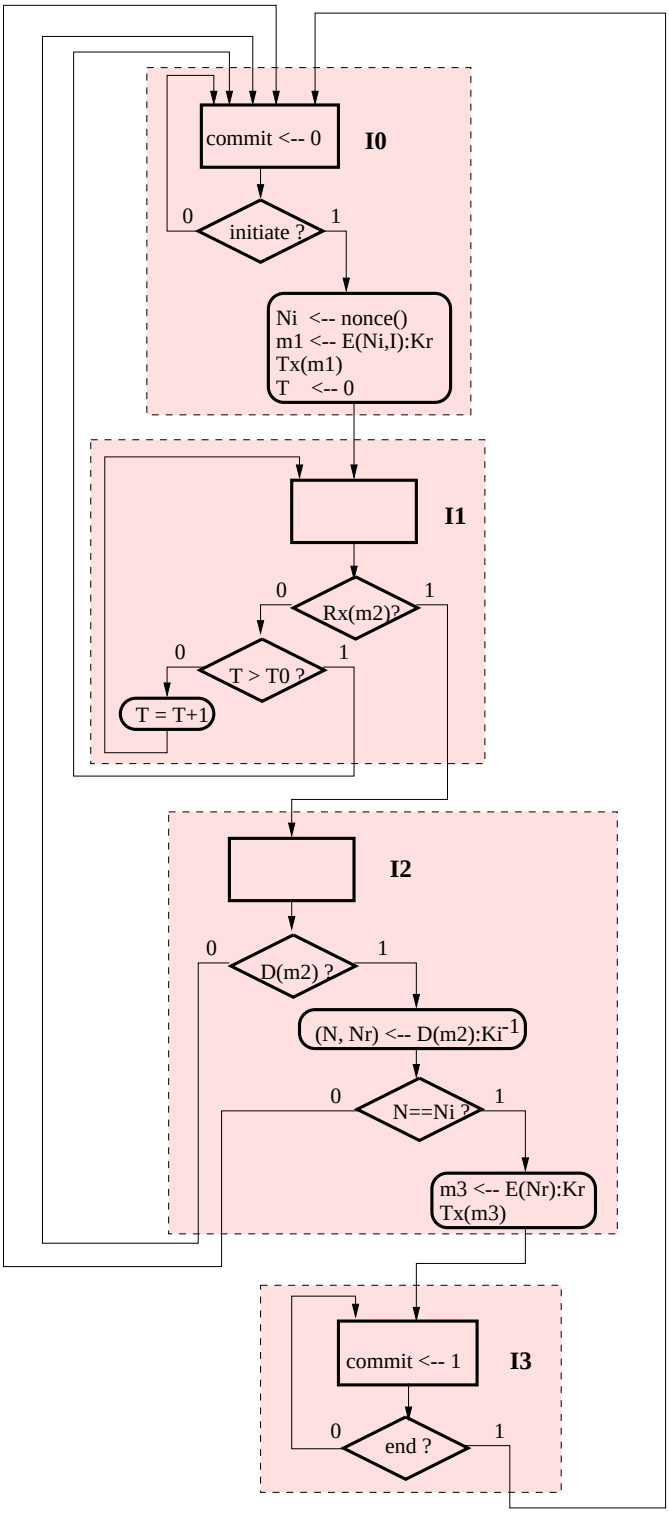

Fig.1. Algorithmic state machine for initiator.

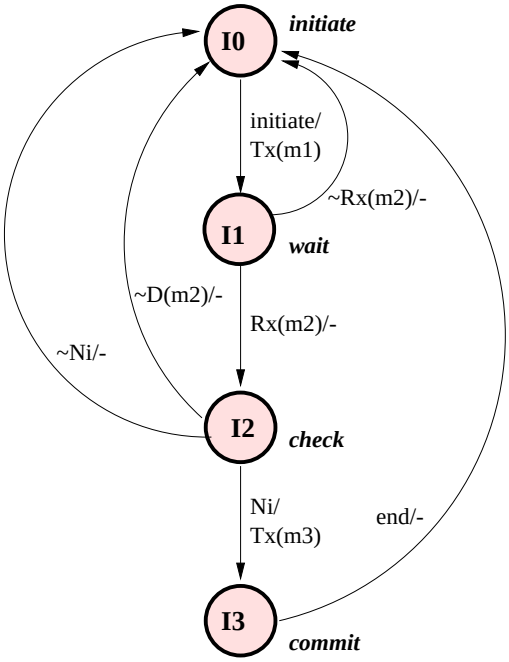

Fig. 2. State machine for initiator.

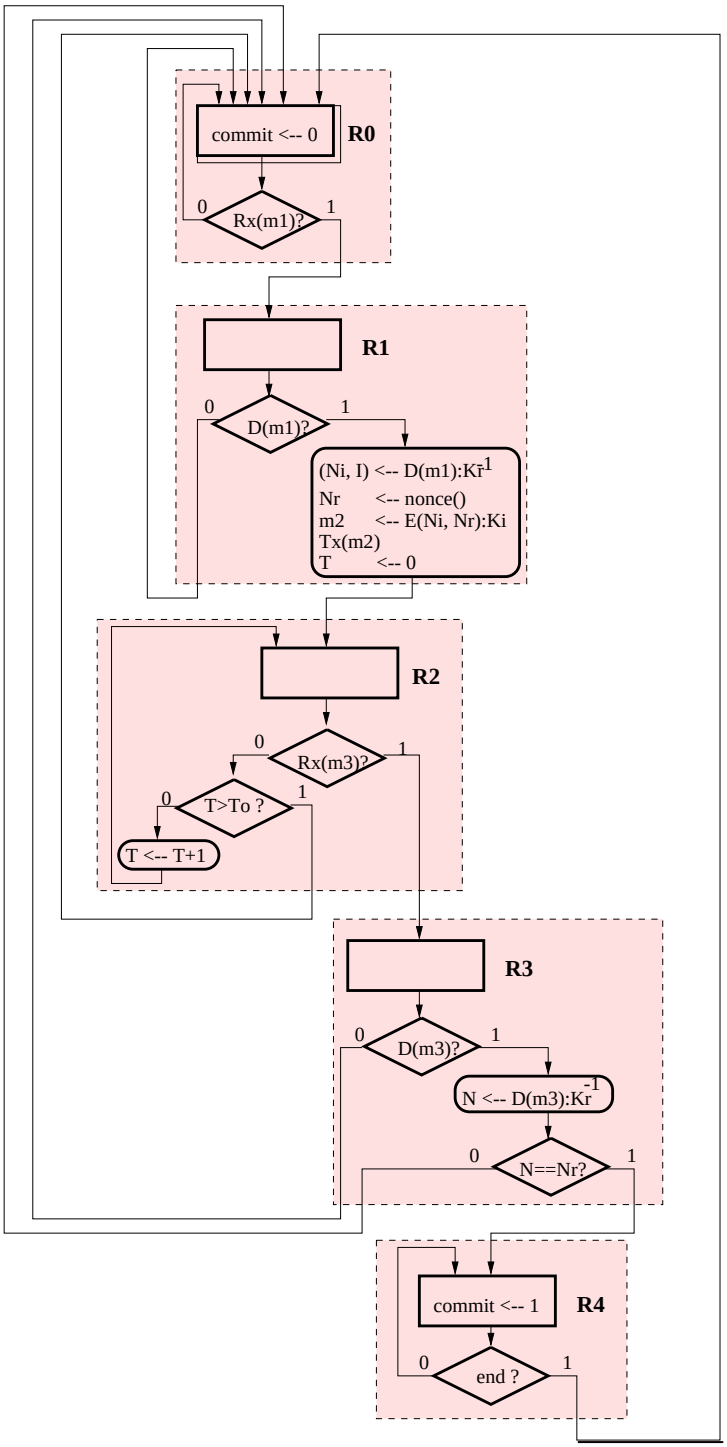

Fig.3. Algorithmic state machine for responder.

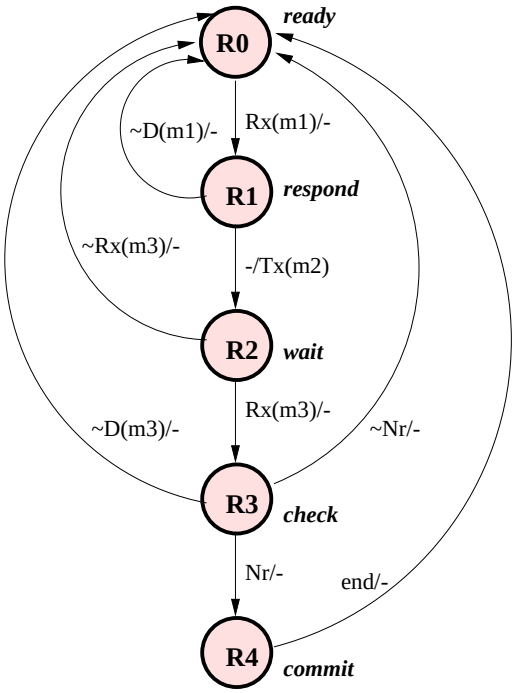

Fig.4. State machine for responder.

Transactions of the SDPS

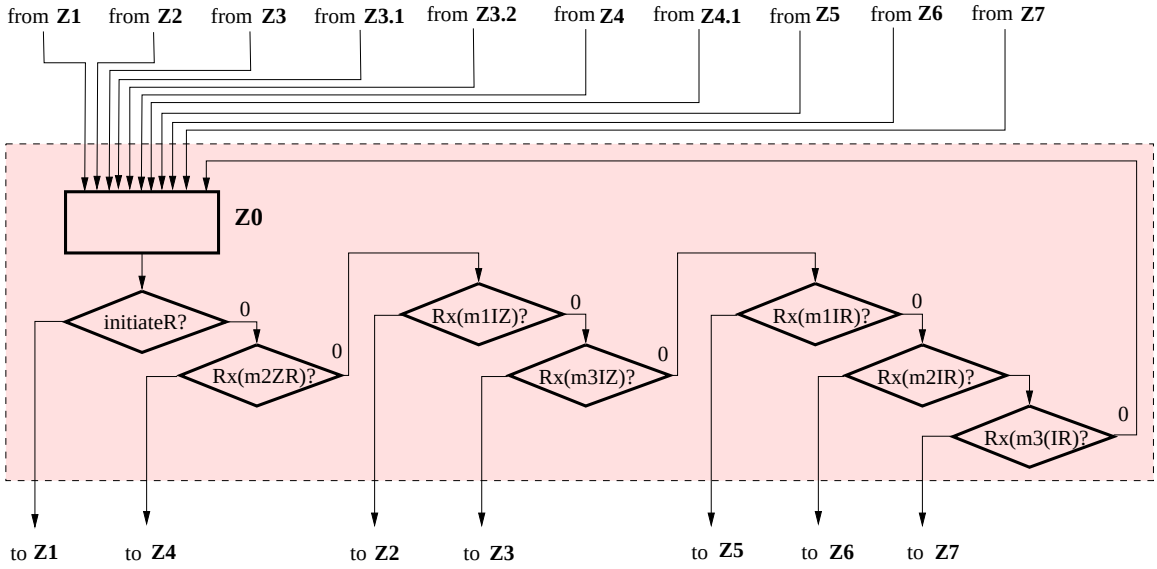

Fig.5. Part 1 of Algorithmic state machine for intruder.

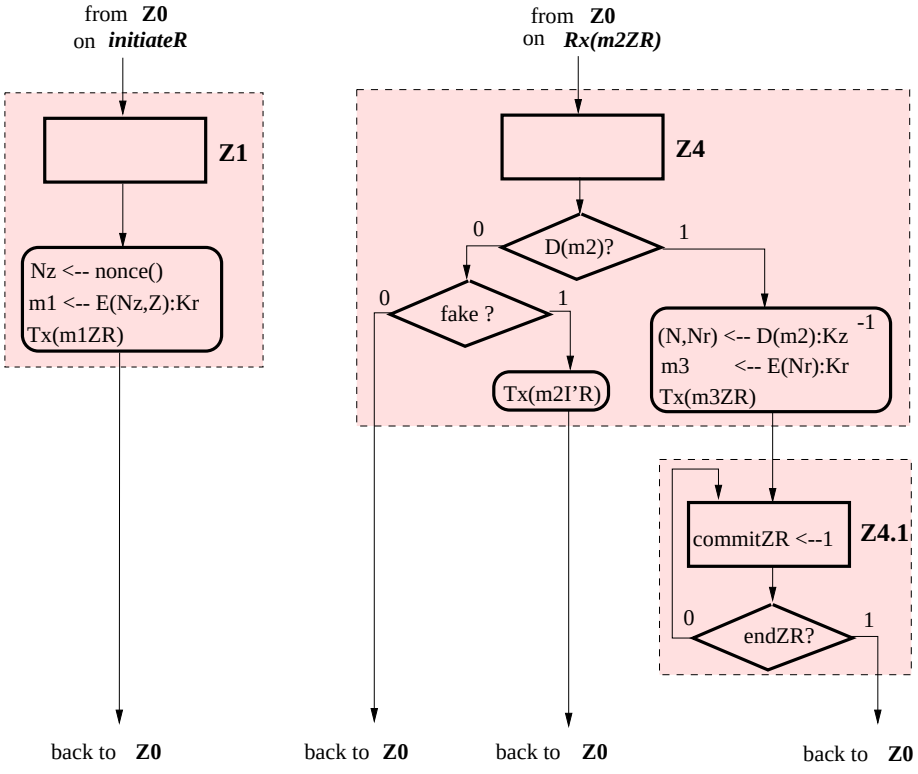

 Fig.6. Part 2 of Algorithmic state machine for intruder.

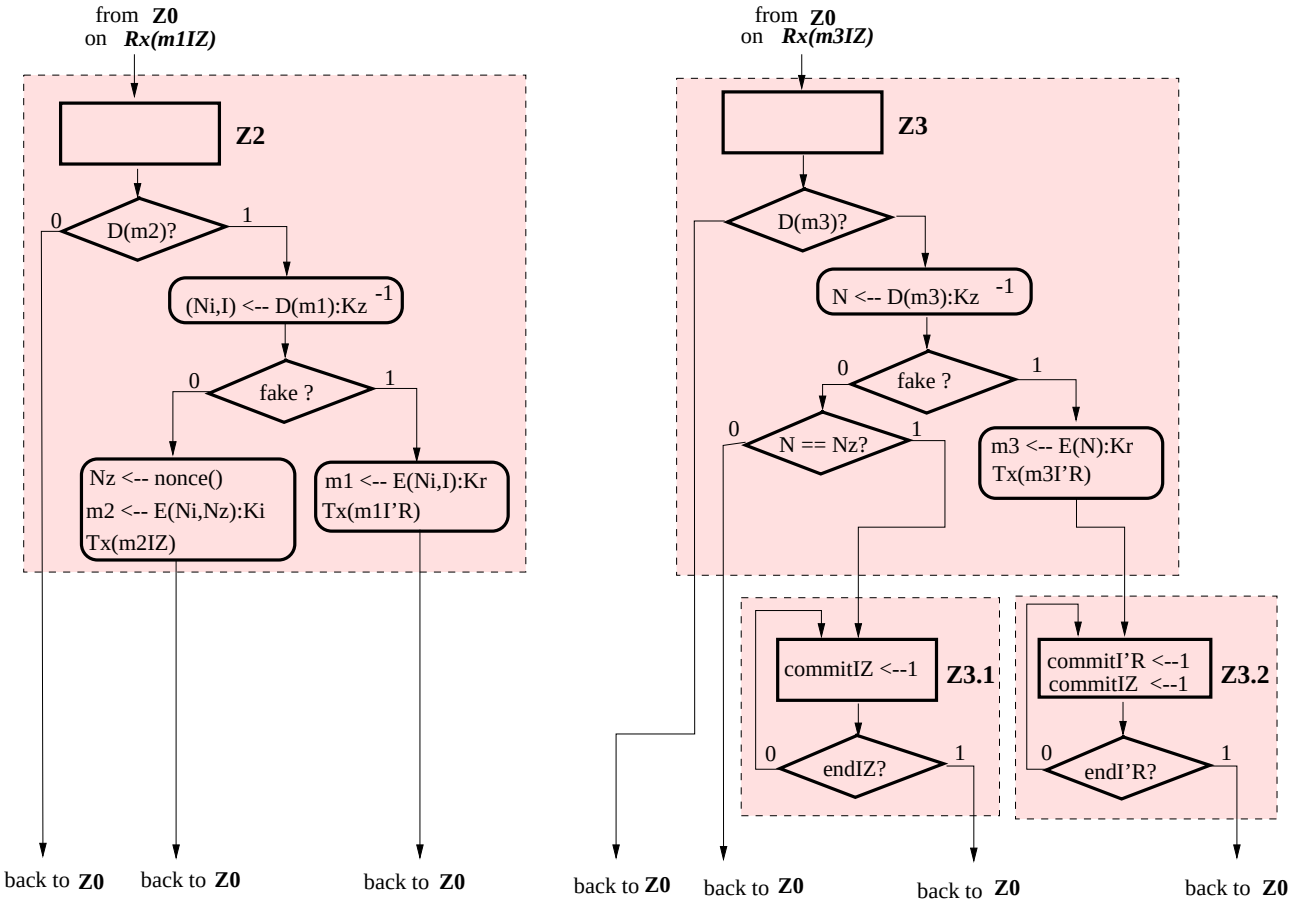

Fig.7. Part 3 of Algorithmic state machine for intruder.

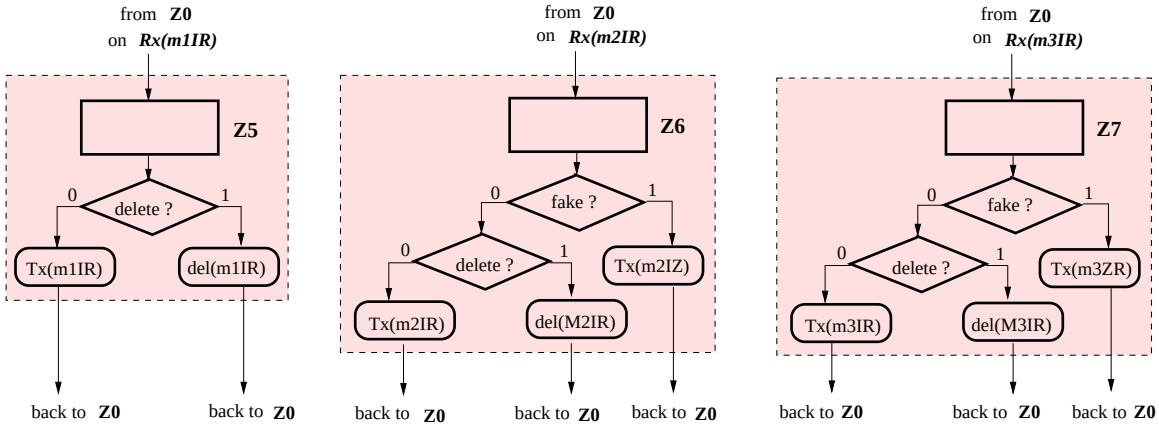

Fig.8. Part 4 of Algorithmic state machine for intruder.

Transactions of the SDPS

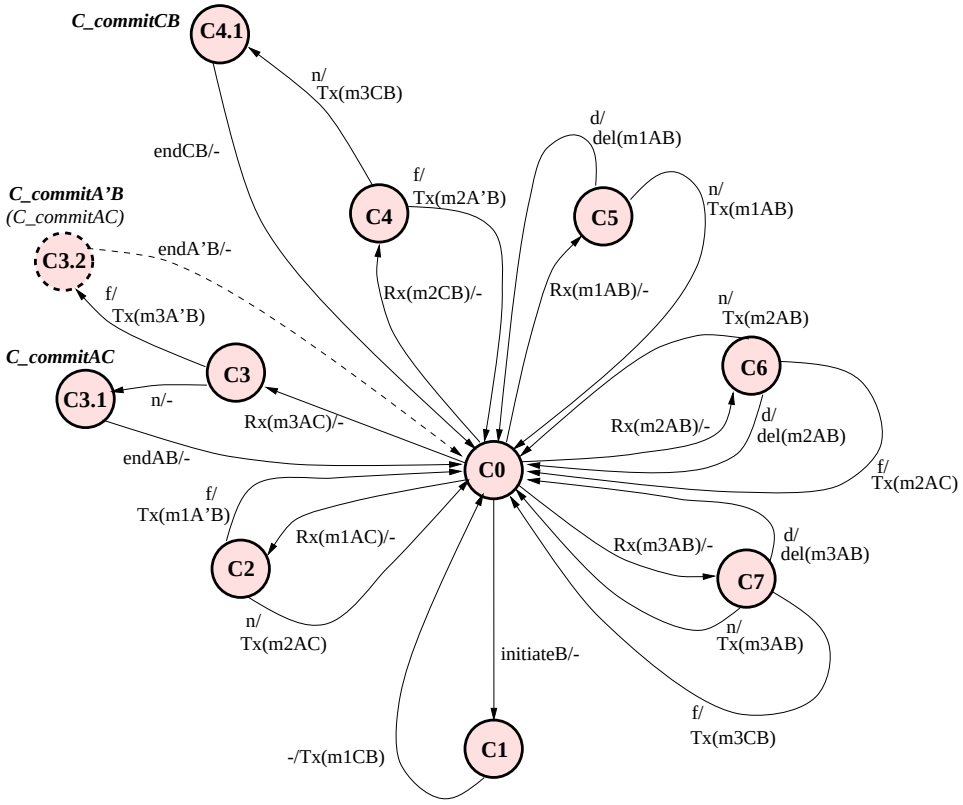

Fig.9. State machine for intruder C

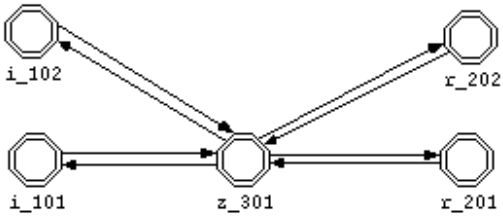

Fig.10. Authentication scenario- network view (Opnet)

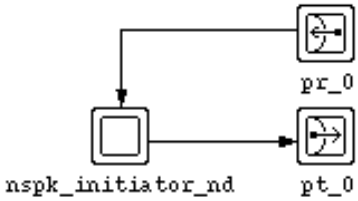

Fig.11. Initiator node (Opnet)

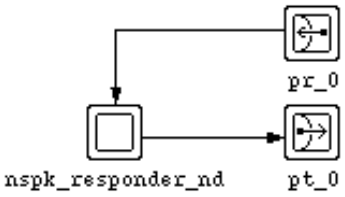

Fig.12. Responder node (Opnet)

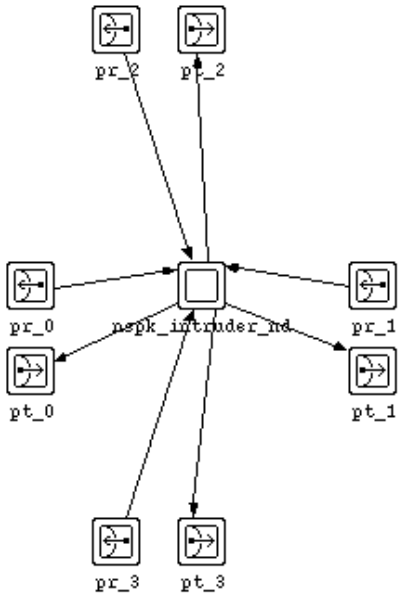

Fig.13. Intruder node (Opnet)

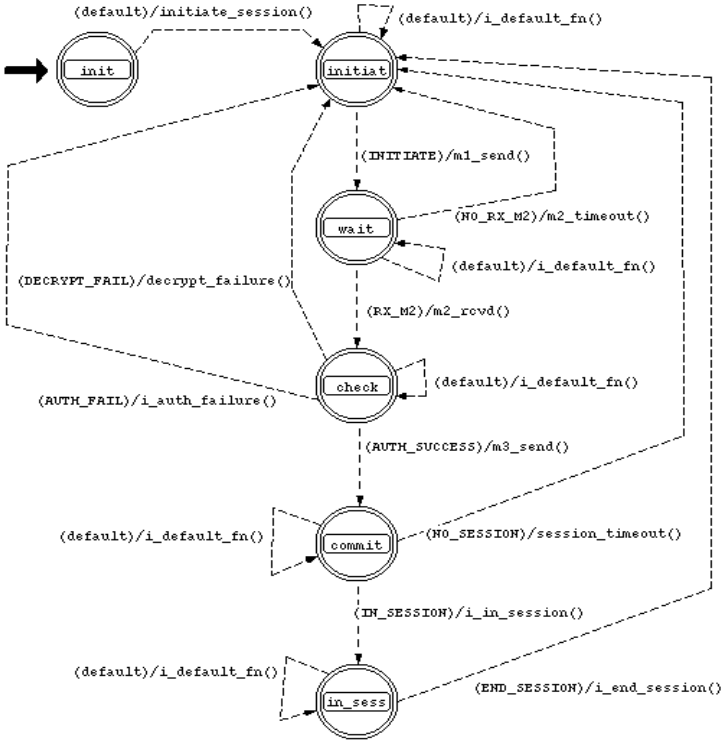

Fig.14. Initiator state diagram (Opnet)

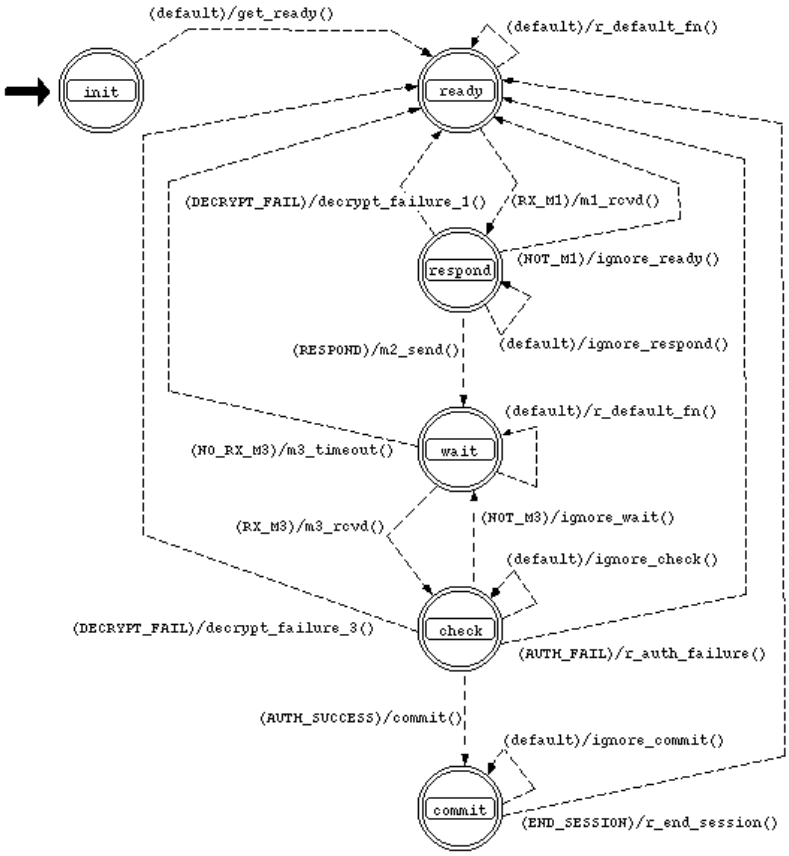

Fig.15. Responder state diagram (Opnet)

Transactions of the SDPS

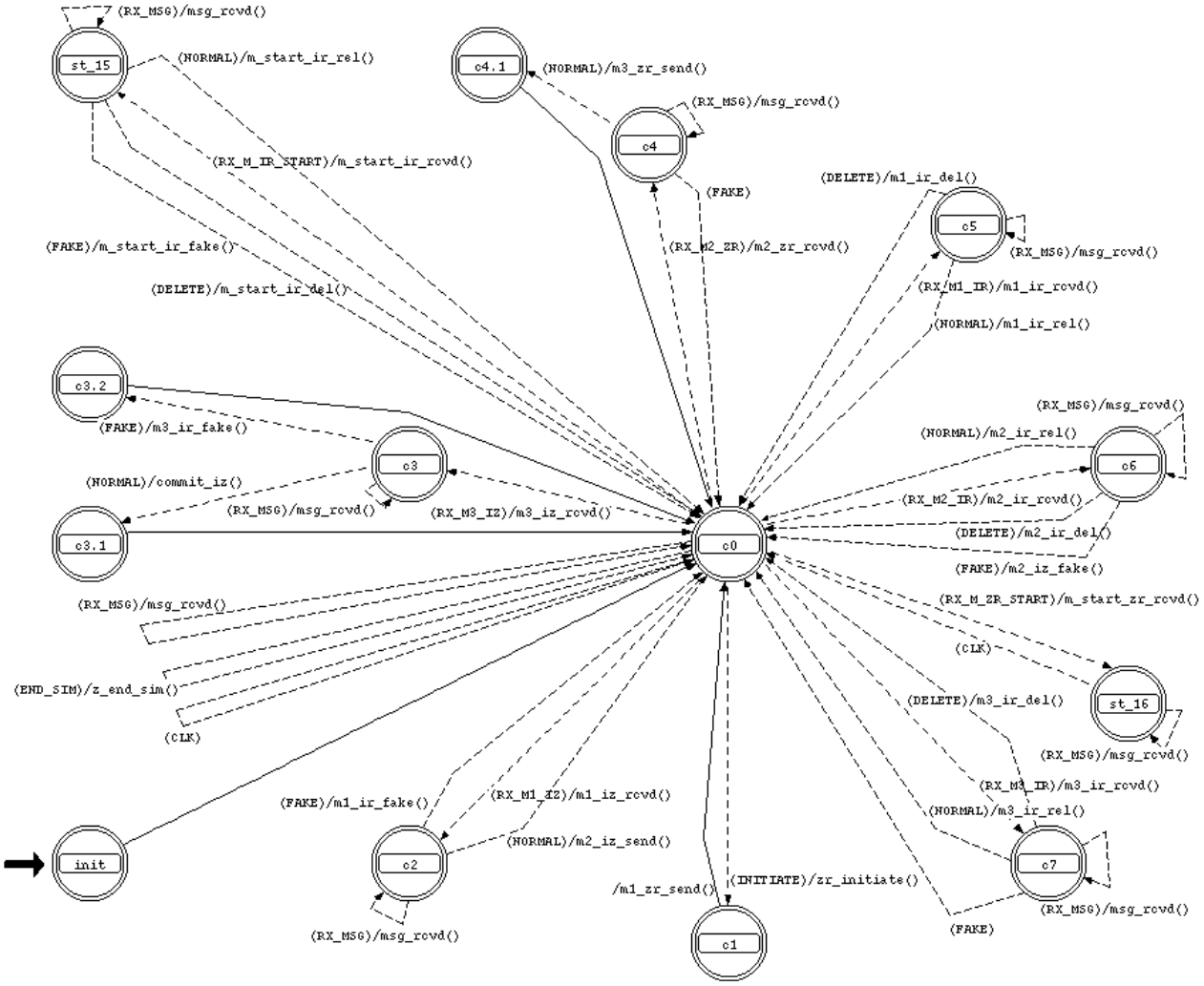

Fig.16. Intruder state diagram (Opnet)