Synthesis of Attack Actions Using Model Checking for the Verification of Security Protocols

Stylianos Basagiannis *Panagiotis Katsaros* *Andrew Pombortsis*

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
tel.: +30-2310-998532, fax: +30-2310-998419
{basags, katsaros, apombo}@csd.auth.gr

Abstract

Model checking cryptographic protocols have evolved to a valuable method for discovering counterintuitive security flaws, which make possible for a hostile agent to subvert the goals of the protocol. Published works and existing security analysis tools are usually based on general intruder models that embody at least some aspects of the seminal work of Dolev-Yao, in an attempt to detect failures of secrecy. In this work, we propose an alternative intruder model, which is based on a thorough analysis of how potential attacks might proceed. We introduce an intruder model that provides an open-ended base for the integration of multiple basic attack tactics. Those attack tactics have the possibility to be combined, in a way to compose complex attack actions that require a number of procedural steps from the intruder's side, such as a Denial of Service attack. In our model checking approach, protocol correctness is checked by appropriate user-supplied assertions or reachability of invalid end states. The analyst can express security properties of specific attack actions that are not restricted to safety violations captured by a generic model checker. The described intruder model methodology was implemented within the SPIN model checker for verifying two security protocols, Micromint and PayWord.

KEYWORDS: Cryptographic protocols, intruder modeling, model checking

1. Introduction

The idea of model checking security protocols is based on the model of a relatively small system running the protocol of interest together with a general intruder model that interacts with the protocol. Analysts today tend to use specific intruder procedures when modeling their protocols, in order to exhaustively search for potential security flaws. This kind of security flaws are found by the use of an appropriate state exploration tool for discovering if the system can enter an insecure state, that is, whether there is an attack upon the protocol. However the developed system often produces a large state space, making the effort of the analyst for analyzing it more difficult. A usual formal development technique is to initially create two agents (often called Alice and Bob) emulating in an exact way the protocols' operational steps for establishing a successful connection.

The basic assumptions are summarized as follows: (i) *The encryption method used is unbreakable*, (ii) *The intruder can prevent any message from reaching its destination* and (iii) *The intruder can create messages of his own*. As a consequence of the foresaid assumptions, model-checking analyses treat any message sent by an honest user as a message sent to the intruder and any message received by an honest user as a message sent by the intruder. This setting refers to a system that becomes a machine, which is used by the intruder to generate words (messages). The intruder's behavior is defined as a message deducibility rule base governing composition and decomposition of messages, encryption and decryption with known keys, as well as memorization and use of eavesdropped information. In Section 2 we provide an overview of the most influential model checking approaches. All of them use the general Dolev and Yao intruder model [1], but the intruder's goal is restricted in finding out a message that is meant to be secret or in generating messages that impersonate some protocol participant. Failures of secrecy or authentication reveal a previously unknown attack on the analyzed protocol.

Sections 3 and 4 introduce a philosophically different approach in designing the intruder model. We also adopt the assumptions of the Dolev - Yao intruder model, but instead of specifying its behavior with a set of rules governing deducibility of messages, we attempt to combine multiple attack tactics based on a careful analysis of how they proceed. Attack tactics are formalized and are then combined into a single Dolev - Yao intruder model within the SPIN model-checking environment [3] [5] [13].

We extend our previous work [3] where we first implemented the intruder model by reorganizing the attack tactics into its structure. Furthermore, we altered the intruder model in order to combine those tactics and –depending on the protocol's session- to form specific attack actions. Using the formal description of those tactics we combine them to implement complex *attack actions*, depending on the specific protocol properties that want to be validated. Five different attack tactics have been implemented so far in our intruder model, namely (1) *Message Interception* (2) *Message Integrity Violation* (3) *Deflections* (4) *Reflections* and (5) *Straight Replays*. Combining the available attack tactics we have formed the following attack actions, (a) *Type Flaws* (b) *Impersonation* (c) *Parallel Sessions* and (d) *Denial of Service*.

Although we cannot claim that our approach covers all possible attacks, we do not exclude known protocol attack states that are not reflected as failures of secrecy or authentication [23] [24] [28]. The developed Dolev - Yao intruder model constitutes a supplemental model-checking mean, used as an open-ended base for implementing more specialized attack tactics, without affecting the overall model checker's capability of capturing generic safety violation states. This enables revealing attacks, which cannot be detected by existing security model checkers, such as attacks that subvert non-repudiation [15], fairness, accountability, abuse-freeness [14] or other e-commerce security guarantees [9] [10].

An interesting aspect is the comparatively smaller state spaces enabling analyses that are not restricted to small systems running the protocol of interest. As the intruder constitutes an open-ended attack base, the analyst can select only the attack tactics that according to his discretion can lead to a protocol flaw. This allows the application of the proposed intruder model to larger and more complex systems and, thus, opening new potentialities in revealing for example multi-protocol attacks [17] on cryptographic protocols that are executed in the same environment. Using the same intruder's formal description we have successfully described and implemented in [4] an intruder model for performing a Denial of Service attack to the Host Identity Protocol (HIP) [6]. While in [3] we have also used the SPIN model checker, in [4] we modeled the specified intruder (and its actions) with PRISM, using probabilistic model checking primitives. Such an example shows the usability of the proposed formal intruder methodology since the presented description is quite flexible in being adopted independently from the chosen formal analysis approach. Simulation and verification results for two micropayment schemes MicroMint [18] and PayWord [18] are presented in Section 5. Finally section 6 concludes with a discussion of the impact of our approach and future work.

2. Related work

Model checking of security protocols has been recently combined with the development of sophisticated intruder models, aiming at discovering secrecy and authentication failures. In existing security model-checkers the intruder's behavior is defined as a message deducibility rule base governing use of eavesdropped information, with the aim to find out a message that is meant to be secret or to generate messages that impersonate some protocol participant(s).

One of the first systems that used the Dolev - Yao intruder model and the secrecy failure approach was the Interrogator tool [19]. Given a final state in which the intruder knows some word, which should be secret, the Interrogator tries all possible ways of constructing a path by which that state can be reached. If it finds a path, then it has identified a security flaw. Finite state analysis of cryptographic protocols has been developed in a range of published works, which implement the secrecy or authentication failure approach within specialized security analysis tools like BRUTUS [20] or within general purpose model checkers like Murφ [21] and FDR (Failures Divergence Refinement) [22].

However, security guarantees cannot always be expressed as absence of secrecy or authentication failure. A typical case is the well-known family of replay attacks, where the intruder aims to playback previously recorded messages in an attempt to sabotage an ongoing protocol session: in [2] the authors show that failures of information exchange timeliness that enable message replays do not always manifest themselves as secrecy failures. Hence, replay attacks were analyzed [16] with special-purpose modal logics, like the BAN logic (named after its inventors called Burrows, Abadi and Needham [11]). However, [8] has shown that BAN logic is flawed. Another complication is that recent studies [14] concluded in that authentication is a protocol dependent notion and there is not a unique definition of authentication that all secure protocols satisfy.

The most detailed description of a Dolev - Yao intruder model is given for the so-called "Lazy Spy" [31]. The "Lazy Spy" was initially expressed in the traces model of Communicating Sequential Processes (CSP)/FDR and was later integrated into Casper [26], a front-end for semi-automated CSP description of security protocols. Casper works based on a custom-made set of rules governing deducibility of messages through encryption and uses a lazy exploration strategy, which examines the subset of intruder states reachable by the protocol rules.

NRL Protocol Analyzer [27] is another well-known tool with a similar Dolev - Yao intruder model. As in the case of Interrogator, the analyst specifies an insecure state and the tool attempts to construct a path to that state from the initial state. An attractive feature is that this tool allows for an unlimited number of protocol rounds in a simple path.In [29] the authors provide a thorough review of the most important state space analysis contributions until 1999. A more recent contribution for the model checking of secrecy and authentication is the so-called “lazy intruder” [31] for the on-the-fly model checker of the AVISPA security toolset [32]. The “lazy intruder” avoids an explicit enumeration of the possible messages the intruder model can generate, by storing and manipulating constraints about what must be generated. The resulted symbolic representation is evaluated in a demand-driven way and this approach reduces the search tree without excluding any attacks.

In [7] the authors present a process algebraic intruder model for verifying a class of liveness properties of security protocols. For these types of properties, the proposed intruder is proved to be equivalent to a Dolev-Yao intruder that does not delay indefinitely the delivery of messages. The intruder model is restricted by a much simpler fairness constraint adding the feature of not, indefinitely, delay the delivery of messages. In this way the intruder is able to find an attack (counterexample) for a liveness property.

3. Intruder Model Notation

We adopt the pessimistic assumption that the intruder has absolute control over the used communication network, as well as the basic Dolev - Yao assumptions mentioned in section 1 regarding his abilities. More precisely, the intruder *eavesdrops* or *intercepts* messages and analyzes them if he possesses the keys required for decryption. Also, the intruder *can generate messages* from his knowledge and can send them to any protocol participant.

The new messages are created from already known messages by applying one or more of four (4) basic operations: encryption, decryption, concatenation and projection. Any attempt to enumerate all meaningful messages that the intruder can send will inevitably lead to an enormous branching of the resulting state space. The model checking approaches of section 2 attempt to preserve the generality of the intruder model while applying specialized techniques to overcome the foresaid problem. However, they are only applicable to a small system running the protocol of interest. If no attack is found, there is still an open possibility for an attack upon some larger system (a principle known as the absence of model-checking completeness [33]). We aim in a less general but complementary approach for the generation of new messages based on an open-ended base of predefined attack tactics.

The structure and the number of all possible fake messages are restricted by the patterns and the number of initial messages of the available attack tactics. The intruder model can be thought as two concurrent processes, where the first aims to eavesdrop/intercept exchanged messages and the second performs a non-deterministically selected attack action against the ongoing protocol session(s) (Figure 1).

Upon reception of a fake message, by some victim, the performed attack step succeeds and the subsequent execution trace is possible to reach an invalid end state or a correctness assertion violation.

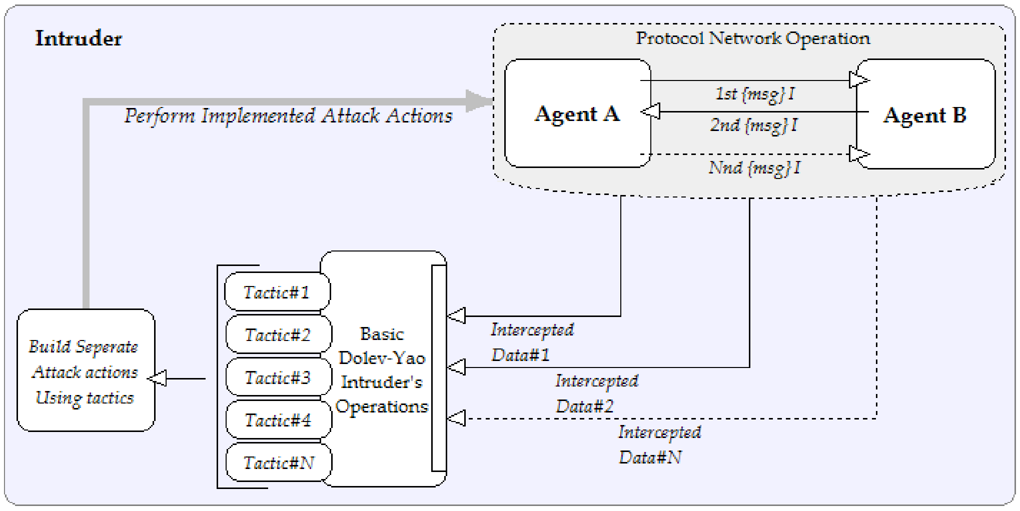


Figure 1. The intruder process

If the "victim" does not accept the sent fake message, falls into a fail-stop state, where he does not continue with the ongoing protocol execution. Protocol correctness, whether it is expressed as reachability of an invalid end state or an assertion check is thus not restricted to secrecy or authentication guarantees. An atomic message may come from one of the sets:

- *Keys*, with members that represent the keys used to encrypt messages, such that every key $k \in Keys$ has an inverse $k^{-1} \in Keys$. For symmetric cryptography the decryption key is the same as the encryption key, i.e. $k = k^{-1}$.
- *Agents*, with members that represent the names of the *honest protocol participants*.
- *Nonces*, which is an infinite set of randomly generated numbers. Members of *Nonces* are used as timestamps that is, any message containing one of them can be assumed having been generated after the nonce itself was generated.
- *Data*, with members that represent the plaintext strings exchanged between the protocol's participants. From the intruder's side, data can be generated (without any meaning). This kind of data named *bg_data (bogus data)* will be used by the intruder for corrupting previously intercepted messages that are in most cases encrypted.

We denote by I the *intruder* ($I \notin Agents$). Also, we define the binary relation:

$is\_key\_of = \{(k, id): k \in Keys, id \in Agents \cup \{I\}, \text{“}key\ k\ is\ used\ by\ the\ participant\ id\text{”}\}$

such that $|is\_key\_of\ (k)| = 1$ in the case of public key cryptography or $|is\_key\_of\ (k)| = 2$ in the case of symmetric cryptography. The set *Msgs* of *exchanged messages* is defined inductively over the *disjoint union*

$$AMsgs = Keys \cup Agents \cup \{I\} \cup Nonces \cup Data$$

that represents the set of atomic messages ($Set_i \cap Set_j = \varnothing$ for any two Set_i, Set_j of the unified sets). More precisely:

- If $\alpha \in AMsgs$ then $\alpha \in Msgs$.

Table 1. Glossary of notation

$\{msg\}_k$	message *msg* is encrypted with key *k*	$\mathtt{ag}_{\text{knowledge}}$	participant's $\mathtt{ag}$ knowledge for the ongoing protocol execution in a given time instant
$msg_x{\cdot}msg_y$	concatenation of messages msg_x and msg_y	**send** (s, r, msg)	the action whereby *s* sends *msg* to *r*
is_key_of (*k*)	returns the owner(s) of key *k*	**receive**(r, s, msg)	the action whereby *r* receives *msg* from *s*
$\#Ses_{\mathtt{ag}}$	the maximum number of sessions allowed for $\mathtt{ag}$ to participate either as initiator or responder	$\langle \mathtt{ag}, j, \mathtt{ag}_{knowledge}, \mathtt{ag}^{j}_{history}, P\rangle$	the tuple representing protocol session *j* of $\mathtt{ag}$ in a given time instant; *P* is a process description given as a sequence of actions to be performed
$sent_n^{\mathrm{ag}_{noSes}}$	the finite-length concatenation sequence of messages sent by $\mathtt{ag}$ in the course of session *noSes*	$I_{knowledge}$	intruder's knowledge for the ongoing protocol execution
$rcvd_n^{\mathrm{ag}_{noSes}}$	the finite-length concatenation sequence of messages received by $\mathtt{ag}$ in the course of session *noses*	$s_{i-1} \xrightarrow{a_i} s_i$	transition from global state s_{i-1} to the global state s_i as a result of action α_i
$\mathtt{ag}^{noSes}_{history}$	participant's $\mathtt{ag}$ history for the protocol session *noSes* in a given time instant	*exists*(*str*, *msg*)	boolean predicate indicating if the string *str* appears in message $msg \in Msgs$

- If $msg_x \in Msgs$ and $msg_y \in Msgs$ then $msg_x \cdot msg_y \in Msgs$, where $\cdot$ represents message concatenation.
- If $msg \in Msgs$ and $k \in Keys$ then $\{msg\}_k \in Msgs$.

Each $\mathtt{ag} \in Agents$ may attempt to execute the protocol for a bounded number of times say $\#Ses_{\mathtt{ag}}$ and each such attempt is a separate *protocol session noSes*, such that $1 \leq noSes \leq \#Ses_{\mathtt{ag}}$. In a protocol session, $\mathtt{ag}$ plays either the role of the *initiator* or the *responder*. We denote by $sent_n^{\mathrm{ag}_{noSes}}$ the finite-length concatenation sequence of messages sent by $\mathtt{ag} \in Agents$ in the course of session *noSes*:

$$sent_n^{\mathrm{ag}_{noSes}} = (sent_{n-1}^{\mathrm{ag}_{noSes}} \cdot msg_n)$$

with the first term equal to the *null sequence* that is, $sent_0^{\mathrm{ag}_{noSes}} = ()$. The sequence $sent_n^{\mathrm{ag}_{noSes}}$ represents participant's $\mathtt{ag}$ *history* for session *noSes*, after having sent msg_n. We denote by $rcvd_n^{\mathrm{ag}_{noSes}}$ the finite-length concatenation sequence of messages received by $\mathtt{ag}$ in the course of session *noSes*. In a given time instant the acquired *participant's knowledge* for the ongoing protocol execution is given as

$$\mathtt{ag}_{\text{knowledge}} = \bigcup_{\mathrm{ag}_j} \{rcvd_{\max(i)}^{ag_j}\} \cup \mathtt{ag}_{in\_knowledge},$$

for all $1 \leq j \leq \#Ses_{\text{ag}}$, where $\texttt{ag}_{in\_knowledge}$ represents the *initial knowledge base* of ag (keys, agent identities and so on) and $i > 0$ represent the terms of the received message concatenation sequences. A protocol session for a honest participant $\texttt{ag} \in Agents$ is defined formally as a 5-tuple$\langle \texttt{ag}, j, \texttt{ag}_{knowledge}, \texttt{ag}^{j}_{history}, P\rangle$, where $1 \leq j \leq \#Ses_{\text{ag}}$ and P is a *process description* given as a sequence of *actions* to be performed. We consider the actions **send** and **receive** for sending and receiving messages to/from other protocol's participants. The assumptions mentioned in section 1 for the general Dolev - Yao intruder model imply that in a given time instant the acquired *intruder's knowledge* for the ongoing protocol execution is given as

$$I_{knowledge} = \bigcup_{\text{ag}_j} \{sent^{ag_j}_{\max(i)}\} \cup I_{in\_knowledge},$$

for all $1 \leq j \leq \#Ses_{\text{ag}}$, $\texttt{ag} \in Agents \cup \{I\}$, where $I_{in\_knowledge}$ represents the initial intruder's knowledge base and $i \geq 1$ represent the terms of the eavesdropped message concatenation sequences.

The *protocol model* is given as the asynchronous composition of the models for each protocol session, including the intruder model, whose behavior depends on the defined *attack tactics*. Attack tactics are non-deterministically selected and are then executed within a single thread of control. Each possible *execution* of the model corresponds to a finite alternating sequence of *global states* and actions:

$$\tau = s_0\ \alpha_1\ s_1\ \alpha_2 \ldots s_n, \text{ for some } n \in N$$

such that $s_{i-1} \xrightarrow{a_i} s_i$ for $0 < i \leq n$ and for the transition relation $\rightarrow$ defined as :

$$\rightarrow \subseteq S \times PS \times A \times Msgs \times S$$

where S is the set of global states, PS is the set of protocol sessions and A is the set of action names.

An important technique that we have introduced to our intruder model is its ability of combining tactics in order to perform more complex attack actions. In this case we define predefined sequences of actions for the intruder model including his basic attack tactics operations. The outcome will be an integrated model that according to its tactics (in SPIN/PROMELA will be procedural execution steps), targets more complex attack scenarios such as a Denial of Service attack for one of the legitimate agents of the protocol.

4 Intruder's Attack Tactics and Actions

We formalized and subsequently implemented a series of basic attack tactics. First, we present the elementary tactics that are also used in forming more complex attacks. Later on we use those tactics to form more complex attack actions for our intruder model. The implemented attack tactics are the ones that are most often reported in related bibliography.

4.1 Message INterCePTion attack tactic (INCPT)

Message interception takes place after the occurrence of some action **send**(ag, v, msg)[1], for some ag, $v \in Agents$ and some $msg \in Msgs$, if there is no **receive**(v, u, msg)[2] with $u \in$

[1] The action whereby ag sends msg to v

[2] The action whereby v receives msg from u

{ag, I} in the suffix execution trace. When intercepting an encrypted message $\{msg\}_k$ there is no **receive** (v, u, $\{msg\}_k$) action in the suffix execution trace. It is obvious that the specified attack tactic is pre-enabled as the intruder has the absolute power over the communication network, between the protocols' participants.

4.2 Replay attack tactics

Replay attacks take place when the intruder redirects eavesdropped or altered messages within one or more interleaved protocol session(s). We adopt the replay attack classification of [25] and we formalize the following replay attack tactics (Figure 2).

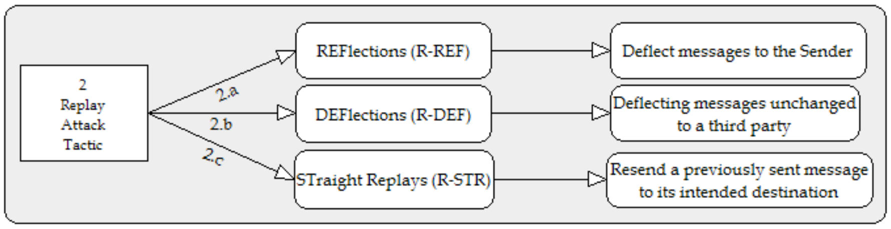

2 Replay Attack Tactic
2.a
2.b
2.c
REFlections (R-REF)
DEFlections (R-DEF)
STraight Replays (R-STR)
Deflect messages to the Sender
Deflecting messages unchanged to a third party
Resend a previously sent message to its intended destination

Figure 2. Replay attack tactics

REFlections (R-REF):

In a *reflection attack* the intruder resends an altered version of a previously sent message back to its sender. *Run-internal reflections* are performed within the same protocol session. *Interleaving reflections* use contemporaneous protocol sessions and *classic reflections* use messages obtained from already finished protocol sessions.

The R-REF attack takes place anytime after the occurrence of some action **send** (v, ag, msg), with msg representing any non-encrypted $msg \in Msgs$ or after the occurrence of some action **send**(v, ag, $\{msg\}_k$) such that $I \notin is\_key\_of\,(k) \wedge k^{-1} \in I_{knowledge}$. The foresaid actions result in a global state where either

$$exists(msg,\ sent^{v_j}_{\max(i)})^3 = \texttt{true} \text{ or respectively } exists(\{msg\}_k,\ sent^{v_j}_{\max(i)}) = \texttt{true}$$

for some $1 \le j \le \#Ses_v$, with $i \ge 1$ representing the terms of an eavesdropped message concatenation sequence. In the performed reflection attack the intruder alters msg based on $I_{knowledge}$ and uses the altered $msg' \in Msgs$ in an action **send**(I, v, msg') or respectively **send**(I, v, $\{msg'\}_{k'}$) for some $k' \in I_{knowledge}$ such that $v \in is\_key\_of\,(k')$.

The R-REF attack succeeds only when v performs the action **receive**(v, I, msg') or respectively the action **receive** (v, I, $\{msg'\}_{k'}$) with the following potential outcomes:

- run-internal reflection

$$exists(msg',\ rcvd^{v_j}_{\max(i)}) = \texttt{true} \text{ or } exists(\{msg'\}_{k'},\ rcvd^{v_j}_{\max(i)}) = \texttt{true}$$

- classic or interleaving reflection

$$\exists\, j' \neq j\text{: } exists(msg',\ rcvd^{v_{j'}}_{\max(i)}) = \texttt{true} \text{ or } exists(\{msg'\}_{k'},\ rcvd^{v_{j'}}_{\max(i)}) = \texttt{true}$$

DEFlections (R-DEF):

In a *deflection attack* the intruder redirects a possibly altered sent message to some participant that is neither the message's recipient nor the sender. *Run-internal deflections*

[3] Boolean predicate $exists(msg, str)$ is true if the message $msg \in Msgs$ appears in string str

are performed within the same protocol session. *Interleaving deflections* use contemporaneous protocol sessions and *classic reflections* use messages obtained from already finished protocol sessions.

STraight Replays (R-STR):
In a *straight replay attack* the intruder resends a previously sent message to its intended destination. Depending on whether this attack is performed within the same session or contemporaneous or non-interleaved sessions, straight replays are also characterized either as run-internal, interleaving or classic replays.

4.3 INTegrity Violation attack tactic (INTV)
For an integrity violation attack, a received *msg* or $\{msg\}_k$ derived from a **receive**(v, u, msg) or a **receive**(v, u, $\{msg\}_k$) must be replaced with a *msg'* or a $\{msg\}_k{}' = \{msg\}_k \cdot bg\_data$. The new composed message can be used later for performing a number of possible attack actions.

4.4 Type flaw attack action (TFLAWS)
A *type flaw attack* arises when the recipient of a message accepts that message as valid, but imposes a different interpretation on the bit sequence than the protocol participant who created it. Type flaw attacks follow the action sequences of the replay attack tactics and may be optionally combined with a *message interception (INCPT)*, in order to prevent reception of intercepted message by its recipient such as to perform a type flaw based message replay.

I triggers a type flaw attack possibly after having altered an eavesdropped $msg \in Msgs$ based on $I_{knowledge}$, thus resulting in some $msg' \in Msgs$. The subsequent action performed by *I* is either **send** (I, v, msg') or **send** (I, v, $\{msg'\}_{k'}$) for some $k' \in I_{knowledge}$ such that $v \in is\_key\_of\,(k')$. This attack tactic succeeds if in the global state after the occurrence of the action **receive** (v, I, msg') or respectively **receive** (v, I, $\{msg'\}_{k'}$) there is some atomic message *amsg*, such that

$$exists(amsg,\ rcvd^{v_j}_{\max(i)}) = \texttt{true},\ 1 \le j \le \#Ses_v$$

with $i \ge 1$ representing the terms of $rcvd^{v_j}_{n}$ and for two sets Set_i and Set_j from the "disjoint" union *Amsgs*, $amsg \in Set_i \cap Set_j$.

A useful term that we insert in this attack action is the *protocol step s(i)*. A *protocol step s(i)*would be assigned as the numerical sequence of the distinct authentication steps that are required from a protocol's session to finalize successfully (figure 3). The described insecure global state expresses the fact that it is possible for an atomic message that was originally intended to have one type (e.g. nonce) to be interpreted as having another type (e.g. key or data). However, this possibility occurs only when both types are represented as bit sequences of the same length, so that when the intruder positions an atomic message in place of a type flawed one, the recipient is fooled into accepting the used atomic message as the one expected according to the owned process description (*P*).

We note that type flaw attacks [34] may not lead to a direct security compromise, since it is possible that the plaintext bit string of the atomic message used by *I* to be unknown to him (the secrecy is still preserved).

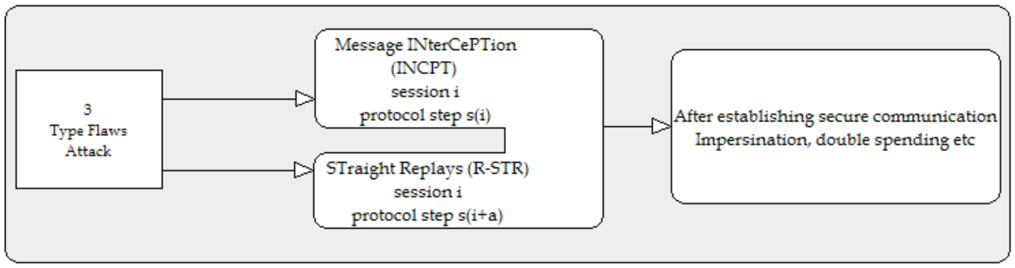


Figure 3. Type Flaws attack action

However, if for example a nonce is used as a key, this is not a good key, because the main concern in generating nonces is to be unique in a protocol session, as opposed to keys that basically have to be non-predictable. Type flaw attacks may result in failures of security properties beyond the typical secrecy and authentication properties, like for example anonymity and non-repudiation [35].

4.5 Simple IMPersonation attack action (IMP)

An *insecure state* (precondition) for the performance of a simple IMP attack is any state where *I* can read the contents of a protocol message sent by some $\texttt{ag} \in Agents$, who acts as initiator of a new protocol session:

$$\{\exists\, sent_1^{\mathrm{ag}_{noSes}} \in I_{knowledge},\ \texttt{ag} \in Agents,\ 1 \leq noSes \leq \#Ses_{ag} :$$

$$\{ sent_1^{\mathrm{ag}_{noSes}} = msg \text{ for some non-encrypted } msg \in Msgs\}$$

$$\vee \{ sent_1^{\mathrm{ag}_{noSes}} = \{msg\}_{\mathrm{k}}\text{: } is\_key\_of(k) = I \vee (is\_key\_of(k) \neq I \wedge k^{-1} \in I_{knowledge})\}\}$$

The IMP attack tactic takes place when the intruder performs the following three subsequent actions against some victim $v \in Agents$, such that $v \notin is\_key\_of(k)$ and $v \neq \texttt{ag}$:

$$\textbf{send}\ (I, v, msg'),\ \textbf{receive}\ (I, v,\ sent_1^{v_{newSes}}),\ \textbf{send}\ (I, \texttt{ag},\ sent_1^{v_{newSes}})$$

where $msg' = sent_1^{\mathrm{ag}_{noSes}}$, when the latter is a non-encrypted message or otherwise $msg' = \{msg\}_{k'}$, with $k' \in I_{knowledge}$ and $v \in is\_key\_of(k')$. Also, v_{newSes} is a unique session identifier for session *newSes*, in which victim *v acts as responder* and the boolean predicate $exists(v,\ sent_1^{v_{newSes}})$ is `false`. If the last mentioned predicate would be `true`, `ag` would realize that the responder in session $\texttt{ag}_{noSes}$ is not the one selected and would subsequently abort the corrupted protocol session.

4.6 Parallel session attack action (PARSES)

Parallel session attacks take place by subsequent interleaving replays among contemporaneous protocol sessions, in which the intruder manipulates protocol participants in multiple roles (initiator or responder), in order to subvert the protocol's goals. The intruder can under special conditions use the cryptographic protocol dialogs:

- as an *oracle* that is, to foretell the contents of otherwise perfectly encrypted messages (refer to the oracle session attack shown in [36]);
- to impersonate a protocol participant (e.g. the BAN-Yahalom attack in [12]); possibly to subvert properties beyond secrecy and authentication.

In a parallel session attack (figure 4) the execution sequence τ includes a series of action cycles that open with some action **send** (`ag`, *v*, *msg*) or **send** (`ag`, *v*, *{msg}$_k$*) and this results in:

$$exists(msg,\ sent^{\mathrm{ag}_j}_{\max(i)}) = \texttt{true} \text{ or respectively } exists(\{msg\}_k,\ sent^{\mathrm{ag}_j}_{\max(i)}) = \texttt{true}$$

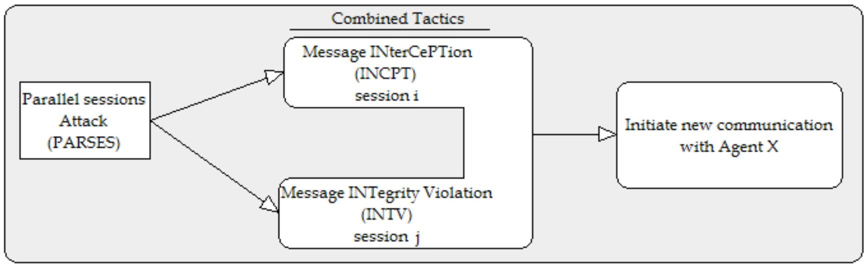

Combined Tactics
Parallel sessions Attack (PARSES)
Message INterCePTion (INCPT) session i
Message INTegrity Violation (INTV) session j
Initiate new communication with Agent X

Figure 4. Parallel sessions attack action

I either opens a new protocol session v'_{newSes} or responds to an already opened session say v'_m (with $v' \in Agents$ including `ag` and *v*), for which the last action of the process description *P* is not included in the prefix execution sequence of τ. The attack is performed possibly after having altered the eavesdropped $msg \in Msgs$ (based on $I_{knowledge}$), thus resulting in sending some $msg' \in Msgs$ by **send** (I, v', msg') or **send** (I, v', $\{msg'\}_{k'}$) for some $k' \in I_{knowledge}$ such that $v' \in is\_key\_of\ (k')$. The interleaving replay succeeds if the action cycle ends with a *receive* action by v', yielding a global state such that

$$exists(msg',\ rcvd^{v'_m}_{\max(i)}) = \texttt{true} \text{ or respectively } exists(\{msg'\}_{k'},\ rcvd^{v'_m}_{\max(i)}) = \texttt{true}$$

with max(i) = 1, if *m* represents a new protocol session (*newSes*). A number of successive interleaving replays may end up in a *fail-stop global state* or in either an invalid end state or violation of a *protocol correctness assertion*. The latter possibilities reveal a previously unknown parallel session attack.

4.7 Denial of Service attack action (DoS)

A Denial of Service attack can be considered as a set of attack tactics. The DoS attack takes place anytime after the occurrence of some action **send**(*v*, `ag`, *msg*), with *msg* representing any non-encrypted $msg \in Msgs$ or after the occurrence of some action **send**(*v*, `ag`, $\{msg\}_k$) such that $I \notin is\_key\_of\ (k) \wedge k^{-1} \in I_{knowledge}$. The foresaid actions result in a global state where either:

$$exists(msg,\ sent^{v_j}_{\max(i)})^4 = \texttt{true} \text{ or respectively } exists(\{msg\}_k,\ sent^{v_j}_{\max(i)}) = \texttt{true} \quad (1)$$

for some $1 \le j \le \#Ses_v$, with $i \ge 1$ representing the terms of an eavesdropped message concatenation sequence. In the performed DoS attack the intruder alters *msg* based on

[4] Boolean predicate *exists*(*msg*, *str*) is true if the message $msg \in Msgs$ appears in string *str*

$I_{knowledge}$ and uses the altered $msg' \in Msgs$ in an action **send** (I, v, msg') or respectively **send** $(I, v, \{msg'\}_{k'})$ for some $k' \in I_{knowledge}$ such that $v \in is\_key\_of(k')$.

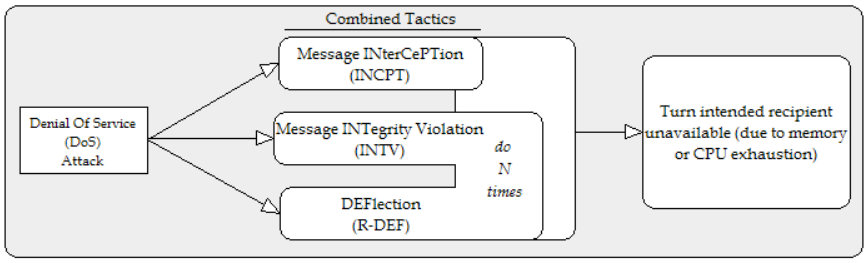

Combined Tactics
Message INterCePTion (INCPT)
Denial Of Service (DoS) Attack
Message INTegrity Violation (INTV)
do N times
DEFlection (R-DEF)
Turn intended recipient unavailable (due to memory or CPU exhaustion)

Figure 5. DoS attack action

At this point the intruder performs an integrity violation attack of the received message *msg* or $\{msg\}_k$. The specific alteration of the *msg* can be done at both encrypted and non-encrypted *msg* as we consider the intruder that concatenates some bogus data say *bg_data.* In this case we have:

$$msg' = msg \cdot bg\_data \text{ or respectively } \{msg\}_k' = \{msg\}_k \cdot bg\_data \qquad (2)$$

Having the intruder altering the received message he then performs the above action (2) *N* times representing *N* distinct requests. For achieving that each *bg_data* that the intruder concatenates to the primary intercepted message has to be distinct in order for msg to be distinct as well. As a result the intruder has to alter for each distinct message *msg'* that he creates the *bg_data* bogus data he concatenates to it. Combining all the above to have:

```
$exists(msg,\ sent^{v_j}_{\max(i)}) = \texttt{true}$
    do for N times
      $msg' = msg \cdot bg\_data(i), \quad i=0..N;$
      $bg\_data' = bg\_data(i) \cdot bg\_data(i+a), \quad 0<a<N-i$
      send$(I, v, msg')$
    end_do
```

In order for the attack to be successful, as seen in figure 5, at the recipient's *v* side, the recipient has to perform for each distinct message *msg'* the action **receive**(v, I, msg'). We should have:

$$\text{For each } msg': exists(msg', rcvd^{v_j}_{\max(i)}) = \texttt{true} \text{ or } exists(\{msg'\}_{k'}, rcvd^{v_j}_{\max(i)}) = \texttt{true}$$

The affect of the Denial of Service attack [30] on the recipient's resource can be either on the memory resources (when $N>>0$) available at its side or on the computational cost that the recipient *v* after receiving say a $\{msg\}_k' = \{msg\}_k \cdot bg\_data$, enters its procedure to decrypt all the bogus encrypted messages.

5 Experimental Results: Verification of two micropayment protocols

We assume that the protocol model is designed and implemented without any errors (proof of correctness) and before the insertion of an intruder mechanism in it. Protocol model should be in accordance with the predefined message approach that we saw above. We define four basic design and implementation steps for the intruder development methodology as follows:

- *Refine the security properties that should be verified.*
- *Select the attack tactics that could prevent the successful verification of the predefined security properties. Form the attacks with the available tactics selected (following the proposed formal description)*
- *Implement the intruder's attacks according to the protocol's specification and the modeling language (i.e. message content).*
- *Attach the intruder to the protocol model as a Man-in-the-Middle entity.*

In this way, based on the formal description of the attacks described above, the internal structure of the produced intruder model is depicted in figure 6.

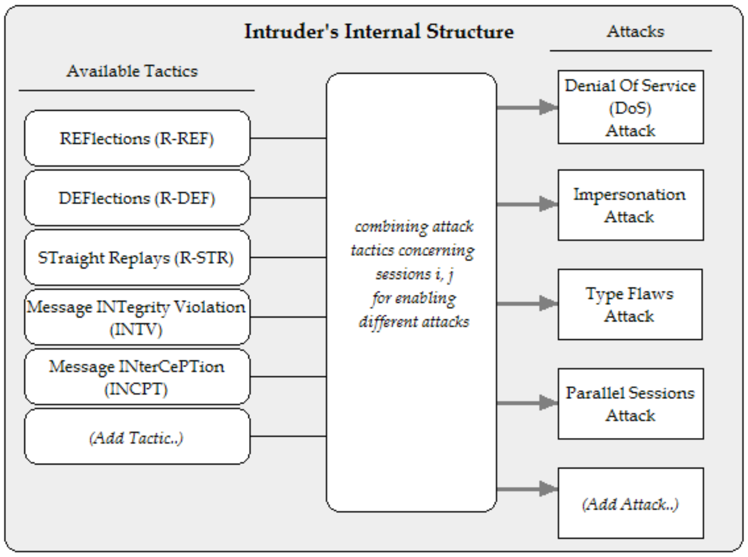

Intruder's Internal Structure
Available Tactics
Attacks
REFlections (R-REF)
DEFlections (R-DEF)
STraight Replays (R-STR)
Message INTegrity Violation (INTV)
Message INterCePTion (INCPT)
(Add Tactic..)
combining attack tactics concerning sessions i, j for enabling different attacks
Denial Of Service (DoS) Attack
Impersonation Attack
Type Flaws Attack
Parallel Sessions Attack
(Add Attack..)

Figure 6. Combining attack tactics

In the next subsections we give detail results of the two micropayment schemes that we have analyzed using our model checking approach.

5.1 The PayWord micropayment protocol

We focus on the analysis of the PayWord micro-payment protocol that was first proposed by Rivest and Shamir in [18]. PayWord is a credit based off-line protocol implemented by the use of *hash chains* that are called *chains of PayWords* (Table 2). In our work we will assume the use of the MD5 hash function [37] denoted by `w(i)`. Three participants are involved in a protocol session: the *Customer*, the *Broker* and the *Vendor*. The Customer (*C*) establishes an account with the Broker (*B*) who issues a certificate containing customer's information and *B*'s name. This certificate will authorize *C* to construct PayWord chains validating himself to some Vendor (*V*). The basic steps of PayWord micro-payments are shown in Figure 7. Upon reception of the foresaid certificate (`certC`), *C* computes the PayWord chain *w* in reverse order based on a randomly chosen term. Then, he signs the so-called commitment (*M*) of the PayWord protocol which consists of the calculated first term of the chain `(w(0))` along with the required customer information; *M* is sent to *V*.

Table 2. Glossary of the PayWord protocol notation

`IDc`	`Customer ID`	`Addrc`	`Customer address`
`IDb`	`Broker ID`	`certC`	`Customer certificate`
`IDv`	`Vendor ID`	`Exp`	`Certificate expiration time stamp`
`SKb`	`Broker’s key`	`Ic`	`Customer’s information`
`PKc`	`Customer’s public key`	`Im`	`Vendor’s information`
`SKc`	`Customer’s secret key`	`D`	`Date`

In every single payment, a chain term of type, *P*: `(w(i),i)` is sent to *V* until the last payment, *P*: `(w(I),I)`. We consider the (attacked) variable-size payment scenario, where the value of each payment varies between 1 and *n*. *V* verifies the payments *P*, by applying the hash function w to the last valid payment v times, where v is the value of the requested payment `(w(i−v))`. At the end of the day, *V* reports to *B* the last (highest-indexed) payment `(w(I),I)` - where `I=max(i)` - received from *C* within the current day, together with the owned *C*’s commitment. Figure 7 shows PayWord’s basic steps modeled with the developed intruder entity.

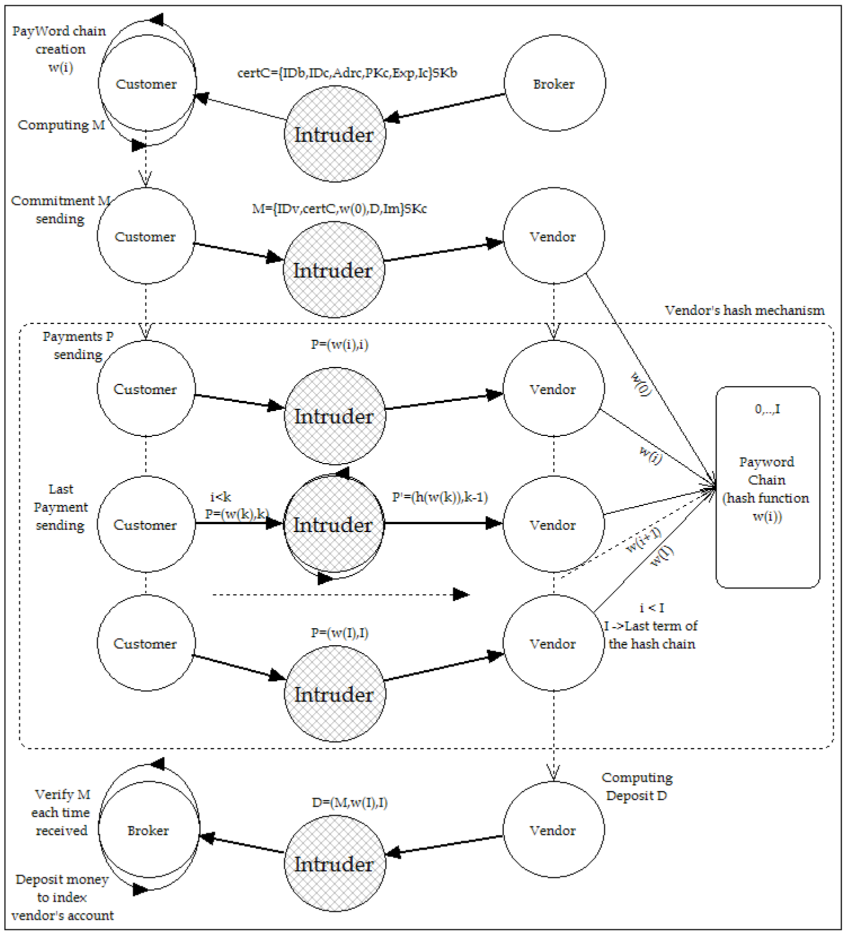
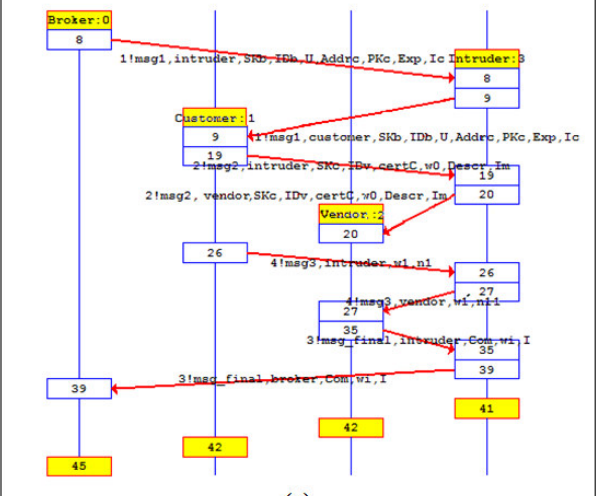
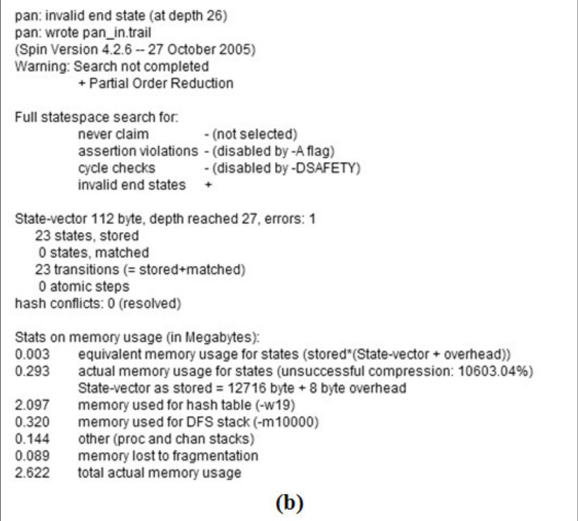


Figure 7. The modeled PayWord protocol scheme

While the use of the hash chain ensures reduced computational requirements for V, the attack found on the protocol is based on V's mechanism, when accepting an altered hashed message. Provided the intruder's ability to perform hash function calculations by MD5, the detected attack takes place when the intruder intercepts and alters a variable-size payment request. We provide simulation and verification results obtained within the SPIN model checking environment for the developed PayWord model, when combined with the described intruder model. The simulation output is shown by the automatically generated Message Sequence Chart of the SPIN model checker.


Broker:0
8
1!msg1,intruder,SKb,IDb,U,Addrc,PKc,Exp,Ic
Intruder:3
8
9
Customer:1
9
1!msg1,customer,SKb,IDb,U,Addrc,PKc,Exp,Ic
19
2!msg2,intruder,SKc,IDv,certC,w0,Descr,Im
19
20
2!msg2, vendor,SKc,IDv,certC,w0,Descr,Im
Vendor:2
20
26
4!msg3,intruder,w1,n1
26
27
4!msg3,vendor,wi',n11
27
35
3!msg_final,intruder,Com,wi,I
35
39
39
3!msg_final,broker,Com,wi,I
41
42
42
45
(a)

```
pan: invalid end state (at depth 26)
pan: wrote pan_in.trail
(Spin Version 4.2.6 -- 27 October 2005)
Warning: Search not completed
         + Partial Order Reduction

Full statespace search for:
         never claim          - (not selected)
         assertion violations - (disabled by -A flag)
         cycle checks         - (disabled by -DSAFETY)
         invalid end states   +

State-vector 112 byte, depth reached 27, errors: 1
    23 states, stored
     0 states, matched
    23 transitions (= stored+matched)
     0 atomic steps
hash conflicts: 0 (resolved)

Stats on memory usage (in Megabytes):
0.003    equivalent memory usage for states (stored*(State-vector + overhead))
0.293    actual memory usage for states (unsuccessful compression: 10603.04%)
         State-vector as stored = 12716 byte + 8 byte overhead
2.097    memory used for hash table (-w19)
0.320    memory used for DFS stack (-m10000)
0.144    other (proc and chan stacks)
0.089    memory lost to fragmentation
2.622    total actual memory usage
```
(b)

Figure 8. (a) INTV attack of a variable-size payment (*P*): *V* accepts an altered message, (b) INTV attack reported by SPIN model checker

Figure 8a shows the detected INTV attack. In state 19 *C* sends a commitment (*M*), which is not affected by the intruder and continues with the first variable-size payment attempt (*P*). In state 27 the intruder alters message `(w1,n1)` thus resulting in the fake message `(w1',n1-1)`, which is eventually accepted by *V*. Finally, *V* dispatches message *D* (deposit) and the protocol session ends with a successful INTV attack that is encoded as an invalid end state. Figure 8b shows the obtained verification output that revealed the described attack scenario. The performed state space search reports an error and generates a counterexample reflecting a feasible path to the defined invalid end state. By the use of the error trail simulation feature of SPIN we roll back the protocol execution and identify the detected flaw. Completing the verification procedure using the SPIN model checker, an error for the specific model is reported back to us, as shown in figure 8b.

5.2 The MicroMint micropayment protocol

The second micropayment scheme that we have verified was MicroMint, presented in [18]. MicroMint is designed to be small and efficient without using any public-key cryptography. Since cryptographic operations are considered to present a significant computational overhead, the proposed protocol tries to achieve authentication of his participants, using other mechanisms.

Its basic idea is that the agent called Broker *Br* produce coins and sells them to a Customer *Cm* that issue requests for them. A Customer can handle these coins to pay the Vendor *Ve* anytime he wants to purchase a service or a product. In return, each Vendor will get request and acquire its bank proposal to the Broker by redeeming these coins. In this way MicroMint can be regarded as a debit-based payment system trying to provide reasonable security guaranties at low cost without using any public key cryptographic operations.

The coins are made with so-called Hash Function Collisions algorithm. Each of coins is a bit string with certain size. These coins have a property that they have the same hash image under a special hash function. So we say these coins are Hash Collided. A coin is a bit-string whose validity can be easily checked by anyone, but which is hard to produce. Table 3 gives the notation used for all the necessary MicroMint elements used in our study.

Table 3. Glossary of the MicroMint protocol notation

Cm	Customer's ID
Ve	Vendor's ID
Br	Broker's ID
$x_1,\dots,x_k$	Generated k-way hash collisions
coins	k-tuple of $(x_1,\dots,x_k)$
Hashf	Used hash function
Req_c	Request for coins
Req_wp	Request for webpage
wp(n)	WebPages available to Vendor *Vr*
b(coins)	Set of coins (bins)
exp_date	Expiration date of produced coins

As we can see, the validity of these coins is very easy to check because they have the same hash result with a public-known hash function. On the other hand, the *Hash Function Collision* has an interesting property. It is very difficult to find the first hash collision. But once the threshold is passed, the growth rate of collision is exponential. This means a MicroMint *Broker* can make huge amount of coins at low per-unit cost with big initial investment. The more coin that broker can make and sell, the cheaper each coin will be, and the more money that broker can make if he sell coins at the same price. The general sketch of a typical MicroMint system attached with the intruder model may include the steps depicted in figure 10.

The protocol's procedure shown in figure 9 is as follows: a) Broker *Br* at first has the essential computational power to make *coins* b) Then he sells the coins to the Customer *Cm* that he has requested for them in order to start purchasing from a Vendor *Ve*. For security reasons, new coins have to be produced on a different hash function *Hashf* after a period of time *exp_date*. c) When *Cm* buys a webpage he requests for the webpage paying by a coin. *Ve* verifies the coin by checking them with validity criteria declared by the appropriate *Br*. d) After confirmation, he releases the purchased webpage to *Cm*. e) At the end of each day *Ve* returns the collected coins to *Br* for redemption. f) At the end of a period of time *exp_date*, *Br* needs to recollect all the unused *coins* from all the Customers and assign to them new ones.

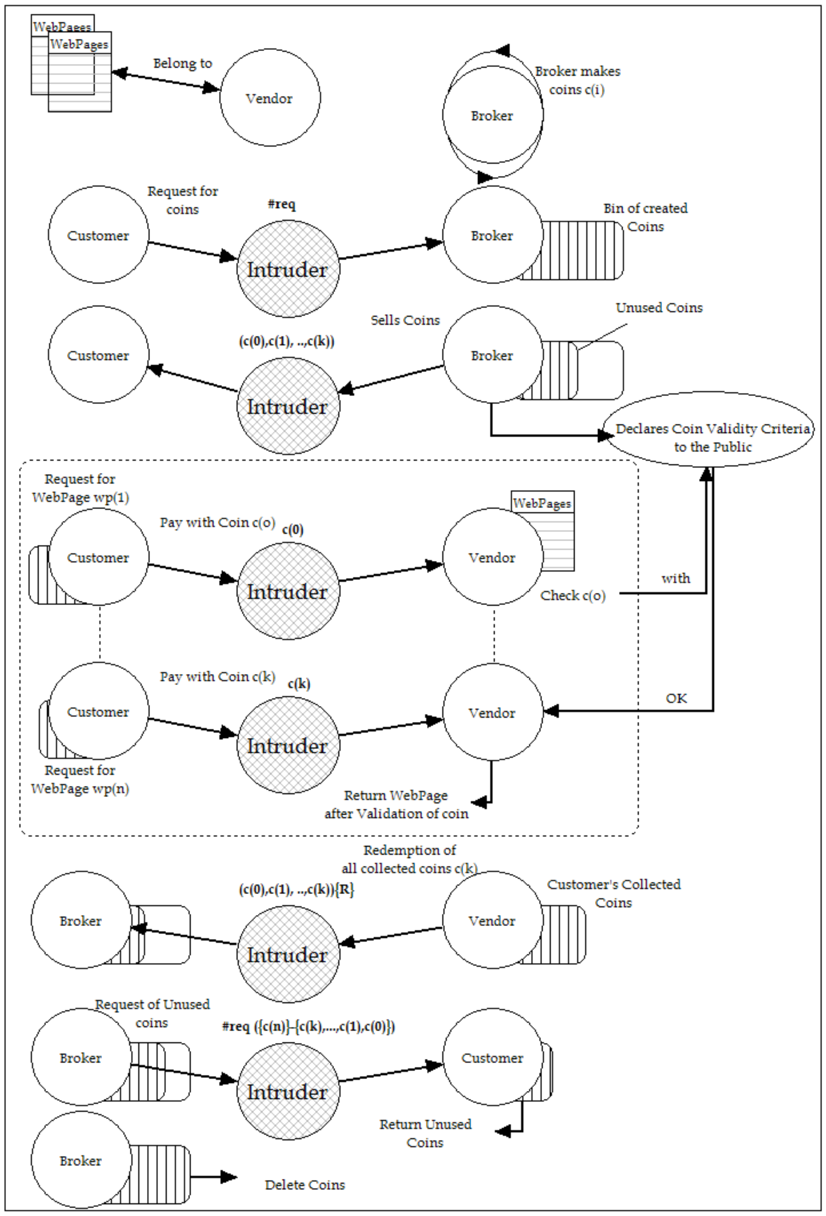
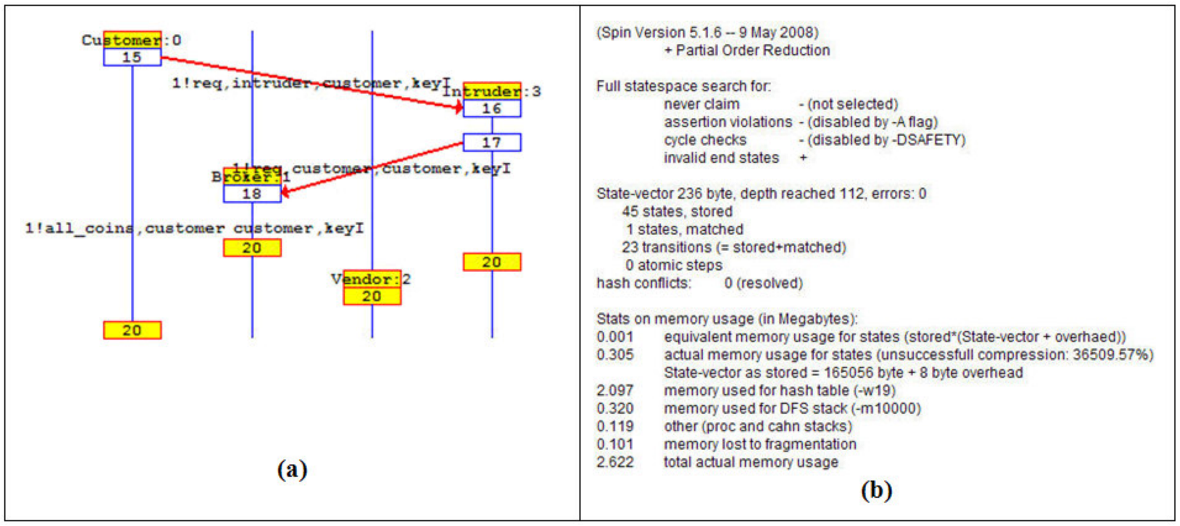


Figure 9.The modeled MicroMint protocol scheme

We have modeled MicroMint using the same structured intruder model as we did with PayWord. The intruder is composed of the attack actions, with parameterized attack tactics according to each protocol's specifications (i.e. expected message delivery channels in PROMELA). Although the effectiveness of the proposed technique in the PayWord protocol, for MicroMint the SPIN model checker returned no error during the verification of the model. Figure 10a and figure 10b presents the produced simulation and verification results of MicroMint respectively.

In a random simulation of our model the intruder procedure is the one that initiates a protocol's session with a Micromint participant trying to impersonate Alice, while the intruder has intercepted a protocol message from the customer *C*. Upon the intruder's message arrival to the legitimate participant's knowledge, the participant aborts the protocol (figure 10a, step 18). Verification results do not indicate any error found in the state space produced by the developed model, as seen in figure 10b.


Customer:0
15
1!req,intruder,customer,keyI
Intruder:3
16
17
1!req,customer,customer,keyI
Broker:1
18
1!all_coins,customer customer,keyI
20
20
Vendor:2
20
20
(a)
(b)

```
(Spin Version 5.1.6 -- 9 May 2008)
          + Partial Order Reduction

Full statespace search for:
          never claim          - (not selected)
          assertion violations - (disabled by -A flag)
          cycle checks         - (disabled by -DSAFETY)
          invalid end states   +

State-vector 236 byte, depth reached 112, errors: 0
      45 states, stored
       1 states, matched
      23 transitions (= stored+matched)
       0 atomic steps
hash conflicts:      0 (resolved)

Stats on memory usage (in Megabytes):
0.001     equivalent memory usage for states (stored*(State-vector + overhaed))
0.305     actual memory usage for states (unsuccessfull compression: 36509.57%)
          State-vector as stored = 165056 byte + 8 byte overhead
2.097     memory used for hash table (-w19)
0.320     memory used for DFS stack (-m10000)
0.119     other (proc and cahn stacks)
0.101     memory lost to fragmentation
2.622     total actual memory usage
```

Figure 10. (a) Simulation Results for the Micromint Micropayment Protocol, (b) Verification Results of the Micromint

6 Conclusion

This work introduces an open-ended Dolev - Yao intruder model that combines elementary attack tactics to form more complex attack actions, in an attempt to subvert security protocol guarantees. We provided a formalized description of the most often reported attack tactics, which were implemented within the SPIN model-checking environment. We have composed a successful target-oriented intruder model, as a formal guidance manuscript giving the opportunity to correct model a protocol system while enforcing an intruder with specific attack tactics, aiming in this way, the model checker to validate the protocol against known attack actions. Although the difference of the used formal analysis in [3] and [4], we denote in this paper the scalability and the effectiveness of adopting such an intruder development approach for the verification of security protocols.

The obtained intruder model was applied successfully to two known electronic micropayment protocols, PayWord and MicroMint. Although the proposed model is bound to the absence of model checking completeness - as all published approaches - it constitutes a supplemental model-checking mean, capable to reveal violations of protocol correctness properties, beyond those checked by existing security model checkers. In this case the model checker do not lose its capability of capturing general security flaws; instead of this we force the model to reveal possible attack states that can occur in a protocol's operation provided a malicious ontology. Finally, the proposed intruder model is open to extensions aiming to integrate more specialized attack tactics or actions that may subvert e-commerce security guarantees like non-repudiation, fairness, anonymity and so on.

References

1. Dolev D., Yao A., On the security of public-key protocols, IEEE Transactions on Information Theory 2/29, pp. 198-208, 1983.
2. Woo T. Y. C., Lam S. S., A semantic model for authentication protocols, In Proc. of the IEEE Symposium on Research in Security and Privacy, 1993.

3 Basagiannis S., Katsaros P., Pombortsis A. Intrusion Attack Tactics for the model checking of e-commerce security guarantees, In Proc. of the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP), Nuremberg, Germany, LNCS 4680, Springer Verlag, 238-251, 2007.

4 Basagiannis S., Katsaros P., Pombortsis A, Alexiou N., A probabilistic attacker model for quantitative verification of DoS security threats, In Proc. of the 32nd Annual International Computer and Applications Software (COMPSAC) , Turku, Finland, 2008.

5 The Spin Model Checker: Primer and Reference Manual Addison-Wesley, ISBN 0-321-22862-6.

6 Host Identity Protocol, Internet Engineering Task Force (IETF) - Network Working Group, Internet Draft, February 2007.

7 Cederquist J.G., Dashti M.T., An intruder model for verifying liveness in security protocols, In Proc. of the fourth ACM workshop on Formal methods in security (FMSE '06), Alexandria, Virginia, USA, pp. 23-32, 2006.

8 Dan M. Nessett, A critique of the Burrows, Abadi and Needham logic (ACM SIGOPS) Operating Systems Review, v.24 n.2, pp.35-38, 1990.

9 Hamdi M., Boudriga N., Computer and network security risk management: Theory, challenges, and countermeasures, International Journal of Communication Systems, Vol. 18(8), pp.763-793, 2005.

10 Meadows C. A., Formal verification of cryptographic protocols: A survey, Advances in Cryptology International Conference on the Theory and Application of Cryptology (Asiacrypt), LNCS 917 Springer-Verlag, pp. 133-150, 1995.

11 Burrows M., Abadi M., Needham R., A logic of authentication, ACM Transaction on Computer Systems 8/1, pp. 18-36, 1990.

12 Syverson P., Cervesato I., The logic of authentication protocols, In Proc. of the 1st International School on Foundations of Security Analysis and Design (FOSAD 2000), LNCS 2171, Springer-Verlag, pp. 63-137, 2001.

13 The SPIN model checker official website, available at http://spinroot.com/ (last accessed 12/12/2008).

14 Holzmann G. J., Design and Validation of Computer Protocols, Prentice-Hall, 1991.

15 Kremer S., Markowitch O., Zhou J., An intensive survey of fair non-repudiation protocols, Computer Communications, 25/17, pp. 1606-1621, 2002.

16 Shmatikov V., Mitchell J. C., Finite-state analysis of two contract signing protocols, Theoretical Computer Science, 283, pp. 419-450, 2002.

17 Cremers C. J. F., Feasibility of multi-protocol attacks, In Proc. of the First International Conference on Availability, Reliability and Security, IEEE Computer Society Press, 2006.

18 Rivest R. L., Shamir A., Payword and Micromint: Two simple micropayment schemes, In Proc. of the Fourth International Workshop on Security Protocols, LNCS 1189 Springer-Verlag, pp. 69-87, 1996.

19 Millen J. K., Clark S. C., Freedman S. B., The Interrogator: Protocol Security Analysis, IEEE Transactions on Software Engineering Vol13/2, 1987.

20 Clarke E. M., Jha S., Marrero W., Verifying security protocols with Brutus, ACM Transactions on Software Engineering and Methodology 9/4, pp. 443-487, 2000.

21 Mitchell J. C., Mitchell M., Stern U., Automated analysis of cryptographic protocols using Murφ, In Proc. of the IEEE Symposium on Research in Security and Privacy, IEEE Computer Society, pp. 141-153, 1997.

22 Roscoe A. W., Modeling and verifying key-exchange protocols using CSP and FDR, In Proc. of the 8th IEEE Computer Security Foundations Workshop, IEEE Computer Society, pp. 98-107, 1995.

23 Obaidat M. S., A Methodology for Improving Computer Access Security, Computers & Security, Vol. 12, pp.657-662, 1993.

24 Roscoe A. W., The theory and practice of concurrency, Prentice Hall, 1997.

25 Roscoe A. W., Goldsmith, M., The perfect spy for model-checking cryptoprotocols, In Proc. of the Workshop on Design and Formal Verification of Security Protocols (DIMACS), 1997.

26 Lowe G., Casper: a compiler for the analysis of security protocols, In Proc. of the IEEE Computer Security Foundations Workshop, IEEE Computer Society, pp. 18-30, 1997.

27 Meadows C., Kemmerer R., Millen, J., Three systems for cryptographic protocol analysis, Journal of Cryptology 7/2, pp.79-130, 1994.

28 Yi Qian, Kejie Lu, Bo Rong, Tipper D., A Design of Optimal Key Management Scheme for Secure and Survivable Wireless Sensor Networks, Security and Communication Networks, Vol. 1, No. 1, pp. 75-82, Jan. 2008.

29 Gritzalis S., Spinellis D., Georgiadis, P., Security protocols over open networks and distributed systems: formal methods for their analysis, design, and verification, Computer Communications 22, pp.697-709, 1999.

30 Liao Q., Cieslak D. A., Striegel A. D., Chawla N. V., Using selective, short-term memory to improve resilience against DDoS exhaustion attacks, in Security and Communication Networks, vol. 1, no. 4, pp. 287-299, Jul/Aug 2008.

31 Basin D., Modersheim S., Vigano L., OFMC: A Symbolic Model-Checker for Security Protocols, International Journal of Information Security, 2004.

32 AVISPA: Automated validation of internet security protocols and applications, 2003, FET Open Project IST-2001-39252, http://www.avispa-project.org

33 Lowe G., Towards a completeness result for model-checking of Security Protocols, In Proc. of the 11th Computer Security Foundations Workshop. IEEE Computer Society Press, 1998.

34 Clark J., Jacob J., A survey of authentication protocol literature: version 1.0, Technical Report, University of York, 1997.

35 Heather J., Lowe G., Schneider S., How to prevent type flaw attacks on security protocols, In Proc. of the 13th IEEE Computer Security Foundations Workshop, IEEE Computer Society, pp, 255-268, 2000.

36 Carlsen U., Cryptographic protocol flaws – Know your enemy, In Proc. of the 7th IEEE Computer Security Foundations Workshop, IEEE Computer Society, pp. 192-200, 1994.

37 Rivest R. L., The MD5 Message-Digest Algorithm, Internet informational RFC 1321, 1992.