An intruder model with Message Inspection for model checking 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 security protocols is based on an intruder model that represents the eavesdropping or interception of the exchanged messages, while at the same time performs attack actions against the ongoing protocol session(s). Any attempt to enumerate all messages that can be deduced by the intruder and the possible actions in all protocol steps results in an enormous branching of the model's state space. In current work, we introduce a new intruder model that can be exploited for state space reduction, optionally in combination with known techniques, such as partial order and symmetry reduction. The proposed intruder modeling approach called Message Inspection (*MI*) is based on enhancing the intruder's knowledge with metadata for the exchanged messages. In a preliminary simulation run, the intruder tags the analyzed messages with protocol-specific values for a set of predefined parameters. This metadata is used to identify possible attack actions, for which it is a priori known that they cannot cause a security violation. The MI algorithm selects attack actions that can be discarded, from an open-ended base of primitive attack actions. Thus, model checking focuses only on attack actions that may disclose a security violation. The most interesting consequence is a non negligible state-space pruning, but at the same time our approach also allows customizing the behavior of the intruder model, in order e.g. to make it appropriate for model checking problems that involve liveness. We provide experimental results obtained with the SPIN model checker, for the Needham Schroeder security protocol.

KEYWORDS: security protocols, verification, model checking, intrusion attacks

1. Introduction

Analyses of existing cryptographic protocols have shown that even when cryptographic primitives are considered perfectly secure (e.g. perfect encryption by key-based cryptographic schemes, infeasible inversion of hash functions, nonce values that cannot be predicted) the protocol itself may have flaws, which can be exploited by an intruder. In the

related bibliography [1, 2] there are examples of protocols that were published with errors, which remained undiscovered for many years. Thus, formal ways of reasoning [3] for whether a given protocol meets its security goals is an absolute necessity.

Model checking is a widespread fully automatic formal analysis that has been successful in discovering flaws in protocols considered to be correct. However, ongoing research has not stopped to look for new ways to tackle the problem of state space explosion, which still prevents analyses of complex protocols and protocol configurations (e.g. higher bounds in the number of ongoing protocol sessions).

In general-purpose model checking [4], state space explosion comes from the asynchronous composition of the modeled concurrent processes and the inherent symmetry redundancy of models in many different problem domains. In security model checking, one additional factor that makes the problem harder is the complexity of the assumed intruder behavior.

Model checking security guarantees such as secrecy and authentication is based on the hardest possible assumptions for the dominance of the intruder over the communication between the protocol participants. These assumptions represent the general Dolev-Yao intruder model [5]: the intruder can intercept any message transmitted on a public communication channel and can also replace it with a message constructed from his initial knowledge and parts of the messages sent by the participants in the same or in other protocol sessions (intruder's knowledge base). The new messages are created by applying one or more out of four (4) basic operations: *encryption*, *decryption*, *concatenation* and *projection*. Also, a typical Dolev-Yao intruder model includes additional assumptions, such as the un-breakability of the encryption used and the possibility the intruder to prevent an original message from reaching its destination.

With the mentioned assumptions, any attempt to enumerate all possible attacks in all protocol steps results in an enormous branching of the state space. In the general case, for a given set of eavesdropped messages, the Dolev-Yao operations may be combined recursively, thus producing infinitely many possible fake messages. In explicit state model checking, analysts bound the size of fake messages, in order to set their models finite. However, memory space becomes crucial, due to the need to store information for each state, including the local states of all protocol participants and the accumulated knowledge of the intruder, for the protocol execution. An additional problem is that under the described assumptions the involved protocol parties interact asynchronously through the same communication channel. The interleaving and concurrency among them may easily result into state space explosion. Analysts observed that the size of the state space increases exponentially with the number of protocol sessions.

Let us consider a protocol execution with two parties A and B acting as initiators of two separate protocol sessions. The state where initiator A has started the protocol and B is idle is symmetric to the state where A is idle and B has started the protocol. Symmetry reductions partition the state space into various equivalence classes, which are exploited by taking into account only one state from each partition. Symmetry reductions for security protocol verification have been first implemented in Brutus [6]. In another work [7], the same authors address the complications of applying partial order reduction, due to tracking the accumulated knowledge of the intruder. Partial order reduction avoids creating states that cannot be affected by interleaving the execution of the model's processes. Results from

model checking experiments with partial order reduction pruned the state space by a factor of 10 to 1877, depending on the examined protocol and the numbers of initiators and responders. Model checking experiments with symmetry reductions that were applied together with partial order reduction resulted in reductions of the state spaces by a factor of up to 58, with a more significant effect in experiments with four to five initiators and four to five responders.

Techniques that delimit the branching of the state space, due to the intruder's fake messages – without excluding possible attacks – have been implemented in specialized security model checkers [8]. Some techniques [9] exploit certain properties that have been identified as characteristics of security protocols, but there is also one recent work [10] that proposes a "divide-and-conquer" approach for reducing the amount of memory needed. A broad family of state space reduction techniques adopts a symbolic representation of the state space, in order to avoid to explicitly enumerating all possible messages that the intruder can generate. In general, most techniques can be exploited, only if the analyst will adopt the model checking tool that implements the respective technique. We provide a detailed review of related work in Section 5. However, we believe that any new proposal for state space reduction still contributes into improving the efficiency and the feasible size of model checking tasks.

In current article, we introduce the Message Inspection (MI) intruder model, which is essentially a Dolev-Yao style man-in-the-middle intruder based on the idea of improving his knowledge with protocol-specific metadata that provide information for the exchanged messages. In a preliminary simulation run, the intruder tags the eavesdropped messages with specific metadata parameters enabling him to validate all possible attack actions. The MI algorithm then decides based on this enhanced knowledge, which of the attacks will certainly fail and the simulation run terminates with a report of the attack actions that can be discarded.

This approach guides the pruning of the model's state-space, since the intruder avoids performing attacks for which it is a priori known that they cannot uncover a protocol flaw. The described two-stage procedure does not limit the overall model checking effectiveness, because the overall analysis can still capture security violations that are encoded as safety guarantees (secrecy and authentication) and at the same time allows customizing the intruder model for capturing security violations that involve liveness (e.g. non-repudiation).

Section 2 introduces basic terminology and describes in detail the general Dolev - Yao intruder model. Section 3 presents the Message Inspection intruder model. The model structure is formally defined and subsequently we introduce the MI algorithm that decides, which attack actions will be performed against the analyzed protocol. In Section 4, we provide experimental results for a MI intruder model in the SPIN model checker [11], when compared with a generic Dolev-Yao intruder model applied upon the Needham Schroeder security protocol (NSPK) [12]. Section 5 reviews related work on intruder modeling and state space reduction techniques, in order to point out the differences from the proposed intruder model and eventually to discuss its strengths and its weaknesses. Finally, conclusions and future work prospects are discussed in Section 6.

2. Basic terminology and the general Dolev - Yao intruder model

Current section introduces basic terminology for protocol specification and provides a brief presentation of the attack actions of the general Dolev - Yao intruder model.

An *atomic message* is any member of one of the following sets:

- *Keys*: This set includes all the keys used for encryption, 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*: The set including all names of the *honest protocol participants*.
- *Nonces*: This is a set with members representing randomly generated numbers being used as *timestamps*: upon receipt of information that includes a *nonce* the receiver knows that the communicated information has been sent after the time instant where the nonce was generated.
- *Data*: The members of this set represent the plaintext strings exchanged between the protocol's participants.

The intruder is represented by I, with $I \notin Agents$. Also, we define the binary relation,

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

In the case of public key cryptography, the cardinality of the set $is\_key\_of\,(k)$ for some $k \in Keys$ is 1. However, for symmetric cryptography the cardinality of this set is 2, since the same key is shared between the communicating protocol participants.

The set *Msgs* of all *exchanged messages* is defined inductively over the *disjoint union*, represented by *AMsgs*, of the mentioned set of atomic messages with the set $\{I\}$:

$$AMsgs = Keys \cup Agents \cup \{I\} \cup Nonces \cup Data \qquad (1)$$

with $Set_i \cap Set_j = \varnothing$ for any two Set_i, Set_j of the unified sets. More precisely:

- If $\alpha \in AMsgs$ then $\alpha \in Msgs$.
- 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 $\{msg\}_k \in Msgs$, where $\{\bullet\}_k$ represents encryption with key k.

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 identified by *noSes*, such that $1 \le noSes \le \#Ses_{\mathtt{ag}}$. In a protocol session, ag either plays the role of the *initiator* or the *responder*. We denote by $sent_n^{\mathtt{ag}_{noSes}}$ the finite-length concatenation sequence of messages sent by $\mathtt{ag} \in Agents$ in the course of session *noSes*:

$$sent_n^{\mathtt{ag}_{noSes}} = (sent_{n-1}^{\mathtt{ag}_{noSes}} \cdot msg_n) \qquad (2)$$

with the first term equal to the *null sequence*, i.e. $sent_0^{\mathtt{ag}_{noSes}} = ()$. The sequence $sent_n^{\mathtt{ag}_{noSes}}$ represents participant's ag *history* for session *noSes*, after having sent msg_n. From now on this will be denoted by $\mathtt{ag}_{history}^{noSes}$.

We write as $rcvd_n^{\mathrm{ag}_{noSes}}$ the finite-length concatenation sequence of messages received by 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:

$$\mathrm{ag}_{knowledge} = \bigcup_{noSes=1}^{\#Ses_{\mathrm{ag}}} \{rcvd_{\max(i)}^{\mathrm{ag}_{noSes}}\} \cup \mathrm{ag}_{in\_knowledge} \quad (3)$$

where $\mathrm{ag}_{in\_knowledge}$ represents the *initial knowledge base* of ag (keys, agent identities and so on) and $i > 0$ represents the terms of the received message concatenation sequences.

A protocol session for a honest participant $\mathrm{ag} \in Agents$ is defined as a 6-tuple

$$\langle \mathrm{ag}, noSes, c\text{-}\mathrm{ag}, \mathrm{ag}_{knowledge}, \mathrm{ag}_{history}^{noSes}, P \rangle \quad (4)$$

where $1 \le noSes \le \#Ses_{\mathrm{ag}}$, $c\text{-}\mathrm{ag} \in Agents \cup \{I\}$ is the receiver of the message(s) sent by ag in session *noSes* and *P* is a *process description*, i.e. a sequence of steps to be performed in the role of the initiator or the responder. The process steps are given as pairs of:

- an *action name*, some member of the set *Act* = {"send", "receive"} representing the dispatch or receipt of a message to/from another participant,
- a *message pattern* that contains one or more message variables (the set *MsgsP* of message patterns is defined in the same way as the set *Msgs* of exchanged messages, with the additional inductive rule: if *var* is a message variable, then $var \in MsgsP$).

The assumptions mentioned in section 1 for the general Dolev - Yao intruder imply that in a given time instant the acquired *intruder's knowledge* for the ongoing protocol execution is given as:

$$I_{knowledge} = \bigcup_{\mathrm{ag} \in Agents} \bigcup_{noSes=1}^{\#Ses_{\mathrm{ag}}} \{sent_{\max(i)}^{\mathrm{ag}_{noSes}}\} \cup I_{in\_knowledge} \quad (5)$$

for all $\mathrm{ag} \in Agents$ and $i \ge 1$ representing the terms of the eavesdropped message concatenation sequences. By $I_{in\_knowledge}$ we denote the initial intruder's knowledge base. For the most powerful intruder *I*, we consider that

$$I_{in\_knowledge} = Agents \cup \{k \in is\_key\_of^{-1}(I)\} \quad (6)$$

i.e. the intruder knows the names of all honest participants and the key(s) that he uses, if participating in legitimate protocol sessions either as initiator or responder. If this is not the examined protocol execution case, then $is\_key\_of^{-1}(I) = \varnothing$.

A protocol model is defined as the asynchronous composition of the models for each protocol session, including the intruder model, whose behavior depends on the possible *attack actions*. In the general Dolev-Yao intruder model [5], attack actions involve the sending of fake messages that are inferred by applying the *deduction rules* shown in equations (7) to (10), for all messages in $I_{knowledge}$. In all cases, the premises of the rule are specified above the line, whereas the deduced messages are given below the line.

- *message concatenation*:
$$\frac{msg_x \in I_{knowledge},\ msg_y \in I_{knowledge}}{msg_x \cdot msg_y,\ msg_y \cdot msg_x \in I_{knowledge}} \quad (7)$$

- *message projection*:
$$\frac{msg_x \cdot msg_y \in I_{knowledge}}{msg_x, msg_y \in I_{knowledge}} \quad (8)$$

- *message encryption*:
$$\{\bullet\}_k \in I_{knowledge}, k \in Keys \Rightarrow \frac{msg_x \in I_{knowledge}, k \in I_{knowledge}}{\{msg_x\}_k \in I_{knowledge}} \vee \frac{msg_x \in I_{knowledge}, msg_y \in I_{knowledge}}{\{msg_x\}_{msg_y}, \{msg_y\}_{msg_x} \in I_{knowledge}} \quad (9)$$

- *message decryption*:
$$\frac{\{msg_x\}_k \in I_{knowledge}, \frac{k \in I_{knowledge}}{k^{-1} \in I_{knowledge}}}{msg_x \in I_{knowledge}} \quad (10)$$

Existing model checking approaches use the aforementioned Dolev-Yao deduction rules based on an abstract representation of the messages that the intruder manipulates. Under the assumptions (i) that the encryption method used is un-breakable and (ii) that it is possible to prevent an original message from reaching its destination, the intruder model performs non-deterministically selected attack actions that are executed within a single thread of control. Each possible execution of the model corresponds to a finite alternating sequence of global states and "send" or "receive" actions:

$$\tau = s_0\, \alpha_1\, s_1\, \alpha_2 \ldots s_n, \text{ for some } n \in \aleph \quad (11)$$

such that $s_{j-1} \xrightarrow{a_j} s_j$ for $0 < j \leq n$ and for the transition relation $\rightarrow$ defined in the cartesian product,

$$\rightarrow \subseteq S \times PS \times Act \times Msgs \times S \quad (12)$$

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

3. The MI intruder model

This section introduces the MI intruder model and describes its use throughout the preliminary simulation run and the model checking phase.

The scope of the MI intruder model includes all security properties that may be encoded as violations of safety (secrecy and authentication), but it is also possible to customize the behavior of the intruder model – according to [13] – for model checking properties that involve liveness (e.g. non repudiation or fairness). More precisely, in SPIN [14], violation of protocol safety may be detected as reachability of invalid end states or alternatively as violation of monitor assertions. Properties that involve liveness are expressed in various ways, including the use of Linear Temporal Logic (LTL). In its current form, the MI intruder model utilizes message metadata comparisons (encryption characteristics, message timestamps and message sizes) for detecting attack actions that may be removed, without excluding any attacks that the analyst needs to check. Attack actions that can be removed are encoded into an open-ended base of primitive attacks (message replays, integrity violations, parallel session attacks and type-flaw attacks) that have been formalized in [15]. Since messages are compared one by one, our model and the used open-ended base of

attack actions can be applied on analysis problems, where protocol participants exchange messages on a unicast communication basis. Extension of the MI intruder model for model checking multicast security protocols would be based on an updated set of primitive attack actions and on appropriate message metadata comparisons that will possibly take into account additional metadata values.

The model can be considered as an optimization approach, which is based on a symbolic representation that avoids explicit enumeration of the messages that the intruder can generate from $I_{knowledge}$. Instead of using the Dolev-Yao deduction rules for inferring all possible fake messages in each protocol step, the MI intruder model records the eavesdropped messages in a preliminary simulation run and at the same time creates discrete metadata values for each recorded message. In this way, the intruder model manipulates only the metadata that were initially created and not the messages themselves. MI rules that will be introduced later determine which attack actions are appropriate and must be included in the optimized intruder model for the model checking phase and which are not. We know, for example, that an encryption scheme attack cannot succeed, if the intruder does not possess the right key in his knowledge. Consequently, encrypted messages can be treated differently from plain text or partially encrypted messages and an obvious optimization is to remove from the general Dolev - Yao model all attack actions, for which it is a priori known that they cannot succeed. Thus, the analyst can use MI, in order to prune the states found to be irrelevant according to the used MI rules.

For the case shown in Figure 1, the MI intruder model acts as a man-in-the-middle attacker that dominates the communication between honest agents *A* and *B*, by eavesdropping the exchanged messages. Each message is evaluated by message characterization mechanisms called *metadata functions*, in order to create appropriate metadata values that enhance the intruder's knowledge for this specific message. The intruder model then consults the embedded base of attack actions, in order to decide which of them can be deactivated by the analyst, without excluding any attacks that may reveal a protocol security flaw. The analyst then proceeds to the model checking phase (2nd protocol execution) with the altered intruder model.

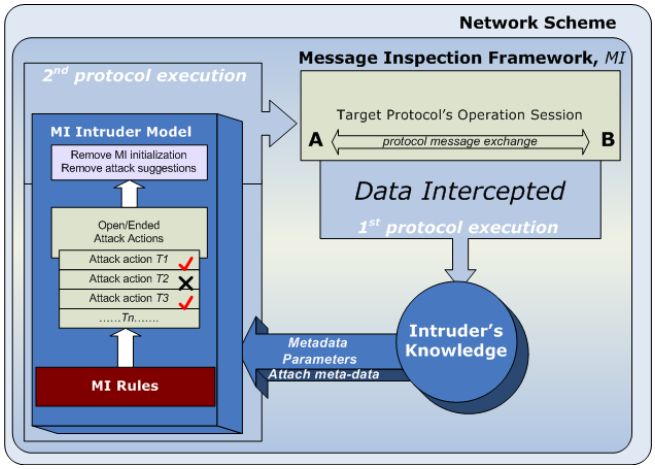


Figure 1: The Message Inspection intruder model

In the general case, the intruder model can instantiate at his discretion new protocol session(s) with the protocol participants. The intruder then reuses previously recorded messages – during MI – in order to validate against the protocol, all possible attack actions enabled in his module. An important issue is that the model does not directly attempt integrity violations to the intercepted messages through encryption, decryption, concatenation and projection operations, as implied by the general Dolev-Yao rules. Instead of this, the model exploits the stored metadata values, in order to violate message integrity only when the message contents can be read. Thus, the model restricts the inherent combinatorial complexity, when having to generate all possible concatenations of messages in $I_{knowledge}$ that can be analyzed.

3.1 Message metadata

Let us consider a protocol $\overline{\Pr}$ between participating agents *A, B, . . . , Z* ϵ `Agents` and let *z* representing the number of protocol steps. We simulate $\overline{\Pr}$ for a bounded number of protocol sessions say *n*. We use the messages of the following table, in order to derive metadata for the intruder's knowledge $I_{knowledge}$

$$\begin{array}{l} sessions \rightarrow \\ \downarrow protocol\ steps \end{array} \begin{array}{c} \begin{array}{cccc} 1^{st} & 2^{nd} & . & n^{th} \end{array} \\ \begin{array}{c} 1^{st} \\ 2^{nd} \\ . \\ z^{th} \end{array} \begin{bmatrix} msg_{1,1} & . & . & msg_{1,n} \\ msg_{2,1} & . & . & . \\ . & . & . & . \\ msg_{z,1} & . & . & msg_{z,n} \end{bmatrix} \end{array} \qquad (13)$$

$$\bigcup_{\mathrm{ag} \in Agents} \overbrace{\bigcup_{noSes=1}^{\#Ses_{\mathrm{ag}}} \{sent_{\max(i)}^{\mathrm{ag}\ noSes}\}}$$

with $msg_{a,b}$ representing a message sent at the a^{th} step of session *b* by some `ag` ϵ `Agents`.

The intruder model stores metadata for each message shown in (13). The stored metadata values for some message sent at the a^{th} step of session b, are derived by a parametric metadata function *p(a,b)* that is defined as follows.

Definition 1. *p(a,b) is a* ***Kth parametric metadata function*** *with K sub-functions,*

$$p(a,b) = \begin{cases} p(msg_{a,b})^1 \\ p(msg_{a,b})^2 \\ \quad . \quad , \ K \geq 1 \\ \quad . \\ p(msg_{a,b})^K \end{cases} \qquad (14)$$

where the value of $p(msg_{a,b})^{mtd}$, 1 ≤ mtd ≤ K depends on the metadata attribute mtd being expressed (e.g. Encryption, Size etc.) for the specific message $msg_{a,b} \in$Msgs that is sent at the a^{th} step of session b.

Based on the implemented MI function, the value of $p(msg_{a,b})^{mtd}$ may represent e.g. the size of the message or whether the message is readable (plain text) or not. The following definitions instantiate MI for the specific metadata cases of the MI intruder model used in the model checking of the NSPK.

Definition 1.1. *The sub-function $p(msg_{a,b})^{Encryption}$ of p(a,b) represents the* ***readability*** *of the intercepted message. The image of $p(msg_{a,b})^{Encryption}$ is the set E={0, 1, 2}, where each value denotes one distinct case of encryption form: 0 is used for no encryption, 1 for partial encryption and 2 for a fully encrypted message.*

$$p(a,b) = p(msg_{a,b})^{Encryption} = \begin{cases} 0 & when\ msg_{a,b} = msg_u \\ 1 & when\ msg_{a,b} = msg_y \cdot \{msg_u\}_k \cdot msg_z \\ 2 & when\ msg_{a,b} = \{msg_u\}_k \end{cases}, \forall (a,b) \in [1..z] \times [1..n] \quad (15)$$

for some $msg_u \in Msgs$, $k \in Keys$ and $msg_y \cdot msg_z \neq (\ )$, i.e. at least one of the concatenated messages is not null.

MI enables the intruder model to act as a decision-making machine that groups attack actions into three different operational procedures corresponding to the symbolic values 0 for no encryption, 1 for partial encryption and 2 for full encryption. In this way, the MI intruder model implements the additional capability to select attack actions, for which according to known security principles – encoded as MI rules – it is a priori known that they will not succeed. For example, an encryption scheme attack will not uncover a protocol flaw, if the intruder does not possess the right key in $I_{knowledge}$. Instead of model checking a series of meaningless attacks, the MI algorithm informs the analyst for the possibility to correct his model by removing them. Thus the intruder model is simplified and in effect performs only the necessary attacks. Each attack action belongs to one of the broad categories of attacks, which were formalized in [15] as specific sequences of "send" and "receive" actions.

Definition 1.2. *The sub-function $p(msg_{a,b})^{Size}$ of p(a,b) represents the* ***message size in bits*** *for some intercepted message. The image of this sub-function is some set of symbolic values S={s: s ϵ ℵ and s>0} with natural numbers representing valuations of the size of messages for the modeled protocol.*

$$p(a,b) = p(msg_{a,b})^{Size} = \begin{cases} size(msg_{a,b}) & \text{for some mapping } size \subseteq Msgs \times S \\ 0 & \text{if } msg_{a,b} \text{ is never sent (null)} \end{cases}, \forall (a,b) \in [1..z] \times [1..n] \quad (16)$$

This specific sub-function enables the MI intruder model to track a message as a numeric valuation of its size, which in turn depends on the size valuations of its constituent parts. When the MI intruder detects two metadata values in different protocol sessions that correspond to messages of equal size, then according to [16] it is possible to mount a type flaw attack, irrespective of whether this attack will succeed or not. Furthermore, if the protocol is interrupted by some communication error and the intruder model stops receiving messages (timeout), then the size of the expected messages in all future steps of the same

protocol session will be zero (0). In this case, the intruder model ignores the metadata values of this particular sub-function for the undelivered messages.

The columns of the table shown in (*13*) represent numbered steps in the simulated protocol sessions. These columns are seen as monotonically increasing sequences with positive integer terms $b_m \in \aleph$ and $b_0 = 1$. The different terms can be considered as *message timestamps* that are set by the intruder model for the intercepted messages. They imply a relative message ordering that for two messages taken from the same or from interdependent protocol sessions may be used for checking whether one message precedes the other or not. These comparisons may be useful, since the applicability of some attack actions depends on the availability of intercepted message parts with timestamp values that are related in some way to the timestamp of the last intercepted message in the attacked protocol session. For example, an impersonation attack between two parallel sessions cannot – according to [16] and [17] – reuse message parts, with timestamp values greater than the timestamp value of the last intercepted message in the attacked protocol session.

If necessary, the MI intruder model can integrate additional metadata sub-functions besides those mentioned. After having defined all metadata sub-functions, we define now the *Intruder Knowledge Table [Ikt]* as follows:

Definition 2. *In a MI intruder model we define the intruder knowledge table [Ikt], which is populated with the values of a parametric metadata function p(a,b) for all intercepted messages* $msg_{a,b}$*, with* $(a,b) \in [1..z] \times [1..n]$:

$$[Ikt] = \begin{bmatrix} p(1,1) & . & . & p(1,n) \\ p(2,1) & . & . & . \\ . & . & . & . \\ p(z,1) & . & . & p(z,n) \end{bmatrix} \tag{17}$$

and

$$p(a,b) = \begin{cases} p(msg_{a,b})^{Size} = \begin{cases} size(msg_{a,b}) & \text{for a mapping } size \subseteq Msgs \times S \\ 0 & \text{if } msg_{a,b} \text{ is never sent (null)} \end{cases} \\ p(msg_{a,b})^{Encryption} = \begin{cases} 0 & when\ msg_{a,b} = msg_u \\ 1 & when\ msg_{a,b} = msg_y \cdot \{msg_u\}_k \cdot msg_z \\ 2 & when\ msg_{a,b} = \{msg_u\}_k \end{cases} \quad \forall (a,b) \in [1..z] \times [1..n] \\ additional\ sub-functions \ldots \end{cases}$$

for some set of symbolic values $S = \{s: s \in \aleph \text{ and } s>0\}$ *with natural numbers and some* $msg_u \in Msgs$, $k \in Keys$ *and* $msg_y \cdot msg_z \neq ( )$*. The properties of [Ikt] are:*

- *if* $msg_{a,b}$ *is never sent (null) then p(a,b)=0 and this means that the intruder has not intercepted any message sent in the* a^{th} *protocol step of session b*
- *if p(a,b)=0 then* $p(a+\varphi, b)=0\ \forall \varphi \in \aleph$: $a+\varphi \leq z$

The properties of the [*Ikt*] table enable manipulation of the collected metadata values, for deriving protocol-specific model checking improvements like for example a state space reduction, through simplification of the applied intruder model. For a protocol $\overline{\Pr}$ and an intercepted message in protocol step *a* of session *b* the intruder model fills in the metadata values *p(a, b)*. If $a < z$, then all table entries $p(a+\varphi, b)$ with $\varphi \in \aleph$: $a+\varphi \leq z$ keep their initial value, which is zero (0), until the intruder intercepts the respective message. If for some reason, the protocol session is stopped, then the values of $p(a+\varphi, b)$ remain zero.

Definition 3. *In order to compare two different [Ikt] table entries, say p(a, b) and p(c, d), such that $a \neq c \vee b \neq d$ (where $\vee$ denotes disjunction), we define the following operator:*

$$p(a, b) \cong p(c, d), \text{ if } \left(p(msg_{a,b})^1 = p(msg_{c,d})^1 \vee p(msg_{a,b})^2 = p(msg_{c,d})^2 \vee \ldots \vee p(msg_{a,b})^K = p(msg_{c,d})^K\right) \quad (18)$$

3.2 The MI intruder model in use

In the preliminary simulation run, the intruder's knowledge is enhanced with the metadata of the [*Ikt*] table. The intruder's knowledge then includes the following information:

$$I_{knowledge} = \bigcup_{\mathrm{ag} \in Agents} \bigcup_{noSes=1}^{\#Ses_{\mathrm{ag}}} \{sent^{\mathrm{ag}_{noSes}}_{\max(i)}\} \cup I_{in\_knowledge} \cup \{[Ikt]\} \quad (19)$$

In this first phase, the intruder model acts as a passive model entity, i.e. it does not execute "send" actions against honest participants. The performed simulation applies the MI algorithm to the updated $I_{knowledge}$ and enables the intruder model to manipulate the message sequences $\bigcup_{\mathrm{ag} \in Agents} \bigcup_{noSes=1}^{\#Ses_{\mathrm{ag}}} \{sent^{\mathrm{ag}_{noSes}}_{\max(i)}\}$ for all $\mathrm{ag} \in Agents$ based on the [*Ikt*] table. The obtained simulation output may include a list of attack actions that can be safely removed from the MI intruder model.

Figure 2 introduces the two phases of the MI algorithm. We consider agents *A*, *B* ϵ *Agents* that exchange messages according to process descriptions P_I and P_R with the actions performed in the roles of the initiator and the responder for some protocol, say $\overline{\Pr}$. The intruder model acts as a man-in-the-middle entity that captures all messages exchanged between protocol participants. For each intercepted message, the intruder model creates a structure *p*(*a*, *b*) corresponding to the [*Ikt*] table entry for the a^{th} step of session *b*, as shown in Figure 2.

The metadata values in the *p*(*a*, *b*) structures are used for comparing the intercepted protocol messages, in order to select the applicable attack actions. In the *MI initialization phase*, the intruder model records all intercepted messages from the executed protocol sessions. The number of fields in the created structures *p(a,b)* is the number of metadata sub-functions that are implemented. When the MI intruder intercepts a message, it updates the respective [*Ikt*] table entry, which is used for comparing it – by applying operator ≅ as defined in definition 3 – with other table entries. Attack actions that according to the made comparisons are useless are reported in the produced simulation output and it is then possible to remove them from the MI intruder model. The analyst also removes the MI

initialization part and proceeds to the model checking of the security guarantees of interest, with the optimized MI intruder model that generates a reduced state space.

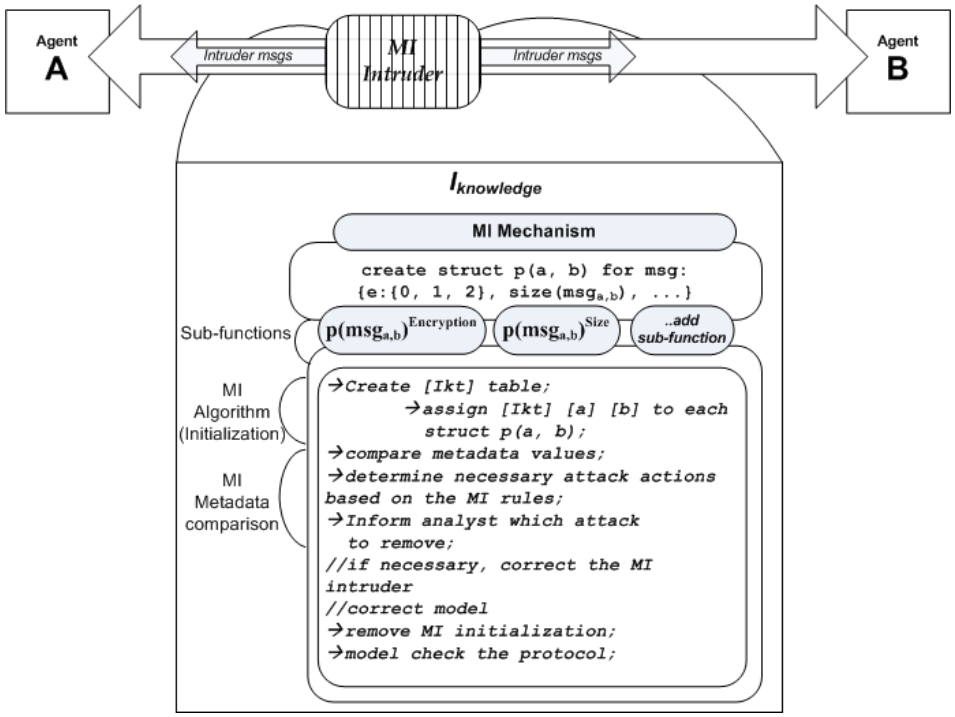


Figure 2: The MI algorithm

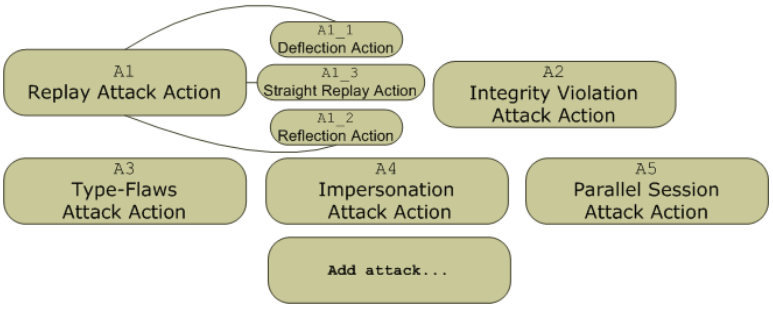


Figure 3. Attack actions for the MI intruder model

Figure 3 shows the open-ended base of attack actions that in the current implementation of the MI intruder model are checked for their feasibility. The selected attack actions appear as primitive steps in attacks reported in the related bibliography and have been proposed in published taxonomies [17, 18, 19] that formalize the observations of intruder misbehaviors, where the intruder redirects messages among protocol participants. In [15], we provided formal definitions of the selected primitive attack actions, as well as bibliographic examples, where these attack actions violate security properties of existing protocols.

Table 1. Attack actions of the MI intruder model and how they are related to the meta-data entries of the [*Ikt*] table

Attack Action	Action description
A1	*Select an intercepted message and send it to its sender (A1_2) or to its intended recipient (A1_3) or to some participant that is neither the intended recipient nor the sender (A1_1)*
A2	*Replace an intercepted message with another message or produce a fake message by concatenation with some message from* $I_{knowledge}$
A3	*Replace a (part of an) intercepted message that corresponds to some p(a,b) with a previously intercepted message (part)*
A4	*Impersonate some ag* $\in$ *Agents using a previously intercepted message that corresponds to some p(a,b) with a=1*
A5	*Initiate a new protocol session or manipulate an existing session using a previously intercepted message that corresponds to some p(a,b)*

Table 1 introduces textual descriptions of the sequences of "send" and "receive" actions for the attack actions of Figure 3, as well as how these actions are related to the metadata entries of the [*Ikt*] table.

Attack actions A1 represent the sending of an intercepted or (if combined with another attack action) a counterfeited message, either to its original sender or to its intended recipient or even to some participant that is neither the intended recipient nor the sender. The metadata values $p(a, b)$ do not influence the feasibility of this general attack action. However, we adopt the assumption that if the sent fake message does no comply with the pattern of the message expected by the "victim", then the recipient falls into a fail-stop state, i.e. he does not continue with the ongoing protocol execution. This assumption represents the expected behavior of a correct protocol implementation.

Attack action A2, when feasible, alters an intercepted message by replacing it or part of it with some message from $I_{knowledge}$. This is possible only when $p(msg_{\alpha,b})^{Encryption}$ is 0 or 1. If $p(msg_{\alpha,b})^{Encryption}=2$ and the intruder does not have in $I_{knowledge}$ the right key for decrypting the intercepted message, then the contents of the message cannot be read (un-breakability of the encryption used) and the A2 attack action is not possible.

Attack action A3 replaces a part of an intercepted message or the whole message, with another message from $I_{knowledge}$. The produced fake message can be accepted by the "victim", only if its size is the same with the size of the expected message. This can be checked by appropriate comparisons of stored metadata values for the messages in $I_{knowledge}$. Type flaws with partially altered messages are possible only when $p(msg_{\alpha,b})^{Encryption}$ is not 2, i.e. when the intercepted message is (partially) readable. Alternatively, according to [17], a type flaw attack is also possible, when in a protocol session an honest agent falls into misinterpretation of a received message, supposed to deliver specific data in some protocol step. This type flaw attack is an open possibility even when the used intercepted message is fully encrypted and the intruder does not possess in $I_{knowledge}$ the key needed to decrypt it.

Attack action A4 initiates a new protocol session be reusing a previously intercepted message that corresponds to some $p(a,b)$ with $a=1$. Finally, attack action A5 initiates a new protocol session or manipulates an existing session by reusing a previously intercepted

message. Both A4 and A5 are not based on specific requirements for the encryption form of the intercepted message.

Table 2. Rules for checking feasibility of attack actions for the MI intruder model

Metadata	Enabling conditions		Attack Actions
Readability $p(msg_{a,b})^{Encryption}$	$p(msg_{a,b})^{Encryption} = 2$		A1, A4, A5
	$p(msg_{a,b})^{Encryption} = 1$		A1, A2, A4, A5
	$p(msg_{a,b})^{Encryption} \neq 2$ *and* $\exists\, m \in Msgs$: *exists*$(m, msg_{a,b})^1$ = `true` *and* $\exists\, amsg \in AMsgs \cap I_{knowledge}$: $p(amsg)^{Size} = p(m)^{Size}$		A3
	$p(msg_{a,b})^{Encryption} = 0$		A1, A2, A4, A5
Size $s_1 = p(msg_{a,b})^{Size}$ and $s_2 = p(msg_{c,d})^{Size}$	$s_1 = s_2$ *and* $a < c$	$b = d$	A3

Table 2 introduces the MI rules for checking feasibility of attack actions in the first implementation of the MI intruder model. The enabling conditions are used in metadata comparisons like the ones described in next paragraphs, in order to determine whether an attack action is feasible or not. Attack actions that in all protocol steps are not feasible can be safely removed, thus yielding an optimized intruder model for the analyzed protocol.

The metadata sub-function $p(msg_{a,b})^{Encryption}$ plays an important role in this analysis, since its values determine whether the intercepted message $msg_{a,b}$ can or cannot be read. When $msg_{a,b}$ is fully encrypted ($p(msg_{a,b})^{Encryption}$=2), the intruder model checks in $I_{knowledge}$ if it owns the key needed to decrypt the intercepted message. If the key is found, this message is marked as non-encrypted and the metadata value $p(msg_{a,b})^{Encryption}$=0 is recorded in the corresponding field of *p(a,b)*. If it is possible to read only some part of the intercepted message $msg_{a,b}$ then $p(msg_{a,b})^{Encryption}$=1, i.e. $msg_{a,b}$ is partially encrypted. This is a sufficient condition for enabling attack actions A1, A2, A4 and A5. Moreover, the possibility to replace a part of the message, say *m*, with some atomic message *amsg* from $I_{knowledge}$ requires equal metadata values for $p(amsg)^{Size}$ and $p(m)^{Size}$. This enabling condition implements the requirement for making an agent vulnerable to misinterpret some part of the message (attack action A3), which is its size.

In most cases, $p(msg_{a,b})^{Encryption}$ will be 2, which automatically excludes the possibility of an integrity violation (attack action A2) that requires read access to some part of the intercepted message.

As we already noted, in type flaw attacks where the intercepted message is replaced as a whole, there is no special requirement for its encryption form. If the expected message has the same size with an intercepted message from a previous step of the same protocol session [17], then it is possible for the intruder to mount a type flaw attack. In the last row of Table 2 we provide the enabling conditions for this attack action. For a complete description of the attack actions mentioned in Table 2 the reader is referred to [15].

[1] Boolean predicate indicating if the string *m* appears in message $msg_{a,b}$

Let us consider a protocol with four (4) steps that runs in two sessions. For all attack actions of Table 1, the intruder model compares the metadata values of the intercepted messages. Each comparison determines if there are attack actions that according to the MI rules of Table 2 can be safely excluded in the examined protocol step. Attack actions that do not contribute in the model checking for all protocol steps are reported in the produced simulation output and it is then possible to remove them from the MI intruder model.

Examples of the comparisons made for the considered 4-step protocol are shown in Figure 4, for most of the mentioned attack actions. When *p(1,1)* ≅ *p(1,2)* and at the same time holds for these two table entries one of the conditions of Table 2 that enable attack actions A4 and/or A5, then the MI intruder model can initiate a new protocol session, in order to attempt a parallel session or an impersonation attack. When *p(2,1)* ≅ *p(2,2)* and for these two table entries hold the conditions of Table 2 that enable attack action A5, then it is possible the MI intruder to manipulate an existing parallel session for an attack, that may subvert one of the protocol's correctness properties. When *p(a,b)* ≅ *p(c,b)* and for these two table entries hold the conditions of Table 2 that enable attack actions A1, then it is possible the MI intruder to perform a deflection or a reflection message replay. Finally, when *p(a,b)* ≅ *p(c,d)* for two messages in different protocol sessions, the last intercepted message is (partially) readable and at the same time hold the conditions shown in Table 2, then it is possible the MI intruder to perform a type-flaw attack action.

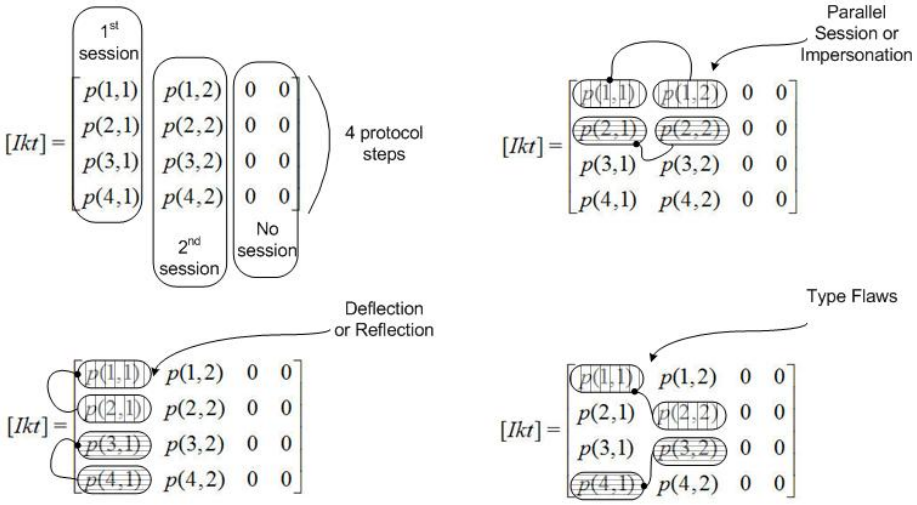


Figure 4: Metadata comparisons with the [*Ikt*] table entries for detecting the possible attack actions in each protocol step

In that case *I* triggers the attack, possibly after having altered the eavesdropped *msg* ∈ *Msgs* based on $I_{knowledge}$, thus resulting in a *msg´* ∈ *Msgs*. The subsequent action performed by *I* is either *send* (*I*, *v*, *msg´*) or *send* (*I*, *v*, $\{msg'\}_{k'}$) for a $k' \in I_{knowledge}$ such that *v* ∈ *is_key_of* (*k´*), i.e. *v* is the owner of *k´*.

This attack action 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

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

and for two sets Set_e and Set_f from the "disjoint" union *Amsgs*,

$$amsg \in Set_e \cap Set_f$$

where $i \ge 1$ represent the terms of the concatenation sequence of messages received by agent v in the course of session *noSes*.

Thus, an atomic message that was originally intended to have one type (e.g. nonce) is interpreted as having another type (e.g. key or data) meaning that the type flaw is exploited, even though this may not lead to a direct security compromise.

During model checking, the MI intruder model performs all possible attack actions in all protocol steps, after having excluded - as a result of the preliminary simulation run - the attack actions whose enabling conditions are not satisfied in all protocol steps. The honest agents $\texttt{ag} \in Agents$ either accept or reject the fake messages based on the implemented protocol logic.

In essence, the metadata in the [*Ikt*] table store protocol-specific monitoring information, which is used in controlling the behavior of the MI intruder model in an effective way. Apart from the two sub-functions of the current MI intruder model, it is also possible to exploit the position of metadata entries in the [*Ikt*] table as message timestamps or to implement additional sub-functions and MI rules.

This allows further optimizations of the MI intruder behavior based on known security principles [15, 20, 21, 22, 23], as well as implementation of custom behavior that will enable model checking requirements beyond those expressed as safety guarantees (i.e. secrecy and authentication).

4. MI-based model checking

This section presents an exploratory case study for the implemented MI intruder model with the following aims:

- to provide indicative results and compare the state spaces for the general Dolev-Yao intruder model and the MI model, in a known model checking context that is frequently used in related bibliography, i.e. the verification of the Needham-Schroeder Public Key protocol (NSPK) [12];
- to provide indicative results for the state space of the two intruder models, when applying partial order reduction and other reduction or state exploration techniques that are currently available in the SPIN model checker;
- to provide indicative results for variants of the implemented MI model, where one or more attack actions have been removed.

In the introductory section, we commented on the results obtained from exploratory case studies, for other state space reduction techniques. An important observation is that the reported results cannot be generalized and therefore they cannot be compared with each other.

The achieved state space improvements intrinsically depend on the interleaving caused by the modeled protocol structure, on the encryption form of the exchanged messages and on the checked protocol configuration (the number of initiators and the number of responders used). Also, the reported results concern only the reduction techniques available in the used model checking environment. Implementation of other techniques may be not possible, if the source code of the tool is not available, or the effort required for implementing additional techniques like MI for comparison purposes may be prohibitively high. Finally, the numbers of states reported in the different studies are not comparable, since these numbers are related to the model representation implemented in the respective tool.

For these reasons, all the aforementioned studies have an exploratory character [24], i.e. they provide an opportunity to explore cause-and-effect facts. The primary aim is to derive indicative results for the effectiveness of the examined reduction techniques. Usually, this takes place in known model checking problems, i.e. verification of protocols with known security vulnerabilities that have been previously used in related studies.

4.1 The NSPK protocol

The NSPK protocol aims to establish mutual authentication between the initiator and the responder, in order to start a message exchange between them. The protocol name suggests the use of public key cryptography, for delivering authentication guarantees. The reduced version of the NSPK protocol, shown in Figure 5, includes only three (3) protocol steps, where in each step the protocol participants *A* (for the initiator) and *B* (for the responder) exchange messages with agent identities and randomly generated nonces (N_A, N_B), encrypted by the public keys *PK{A}* and *PK{B}*. The sent information can be checked by the receivers.

In addition to the agents shown in Figure 5 the developed SPIN model includes an intruder model that has absolute control over the communication between the honest protocol participants. The reported results concern with two different intruder models, i.e. the general Dolev-Yao intruder with the deduction rules specified in equations *7* - *10* and an MI intruder model. Honest agents are encoded as fail-stop processes, i.e. if the message received in a protocol step is not the expected one, then protocol execution is stopped for the receiver even though the provided security guarantees have not been violated. Potential failures of message secrecy or failures of authentication are expressed as invalid end-states in the SPIN model checking environment.

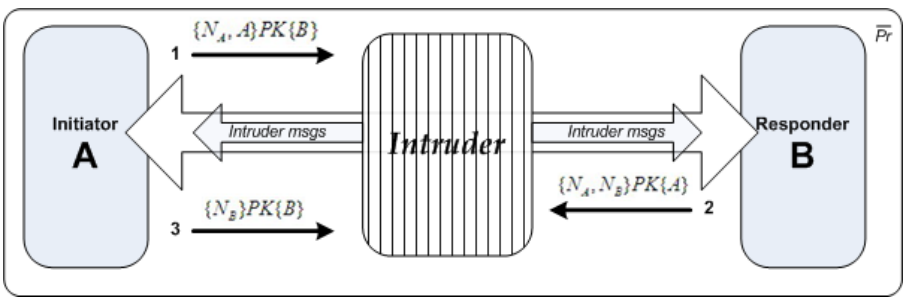


Figure 5: Reduced version of the NSPK protocol

4.2 Model checking the NSPK protocol with the MI intruder model

In a preliminary simulation run with two protocol sessions (Figure 6), the MI intruder model detected two attack actions, namely A2 and A3 that can be safely removed. More specifically, the MI intruder acts as a man-in-the-middle entity between agents A and B for the first protocol session and B and C for the second session.

Upon intercepting an NSPK message, say $msg_{a,b}$, the MI model creates appropriate metadata values for $p(msg_{a,b})^{Encryption}$ and $p(msg_{a,b})^{Size}$ that are recorded in [*Ikt*] table (Figure 6i). Since the MI intruder forwards the sent messages to the intended recipients, both protocol sessions are completed with success. We realize that all protocol messages are fully encrypted and since the decryption key is never included in $I_{knowledge}$, for all metadata values $p(msg_{a,b})^{Encryption}$ is 2. We also see the metadata values computed for $p(msg_{a,b})^{Size}$, for the message sizes shown in Figure 6, representing the assumption that there are no two size-similar atomic messages, with the first message being an agent identity and the second one coming from the set of nonces.

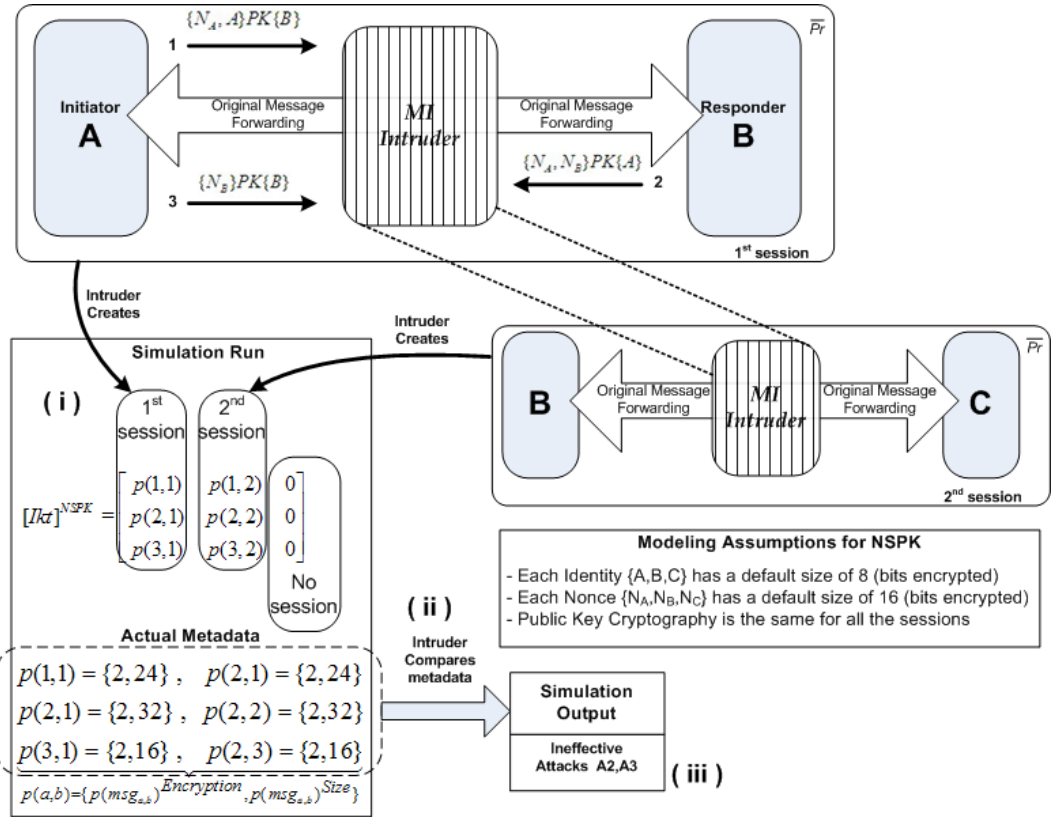


Figure 6: Preliminary MI simulation run: the intruder (i) creates the [Ikt] table, (ii) compares the metadata and (iii) proposes removal of attack actions A2 and A3

The MI intruder model then performs (Figure 6ii) the metadata comparisons discussed in section 3.2 by taking into account the MI rules of Table 2. Finally, the intruder model

outputs the decisions made (Figure 6iii). Since $p(msg_{a,b})^{Encryption} = 2$ in all protocol steps, the integrity violation attack action (A2) is excluded. Also, because $p(msg_{a,b})^{Encryption} = 2$ for all exchanged messages and at the same time there are no size-similar messages in the same protocol session, the MI-based intruder model proposes removing the type flaw attack action (A3).

Figure 7 provides the model checking result for the described NSPK model, when using the optimized MI intruder and the partial order reduction functionality of the SPIN model checker. An invalid end state is reached at depth 25 of the produced reachability graph (264 stored states and 1493 states accessed as hash conflicts) and the search is stopped after having detected the reported error (errors: 1).

A subsequent guided simulation explores the generated counterexample and creates the message sequence diagram of Figure 8. The reached invalid end state corresponds to the state, where agent B acting as responder accepts a fake NSPK message that causes him to initiate a new protocol session. The used message is created by the intruder model in the role of the initiator according to the message pattern of the first of the three messages shown in Figure 5. This invalid authentication in effect causes a successful impersonation attack against B, who perceives the intruder as an honest protocol participant.

The same security violation has been also detected with a generic Dolev-Yao intruder model, but in that case the reached depth of the detected invalid end state was 48.

```
pan: invalid end state (at depth 25)
pan: wrote pan_in.trail

(Spin Version 5.1.6 -- 9 May 2008)
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 162 byte, depth reached 24, errors: 1
     264 states, stored
     883 states, matched
    1147 transitions (= stored+matched)
       11 atomic steps
hash conflicts:     1493 (resolved)

    2.326       memory usage (Mbyte)

unreached in proctype Agent_A
        line 33, "pan_in", state 8, "-end-"
        (1 of 12 states)
unreached in proctype Agent_B
        (0 of 10 states)
unreached in proctype MI_intruder
        line 54, "pan_in", state 8, "-end-"
        (1 of 86 states)
```

Figure 7: Verification output for NSPK with the detected invalid end state at depth 25

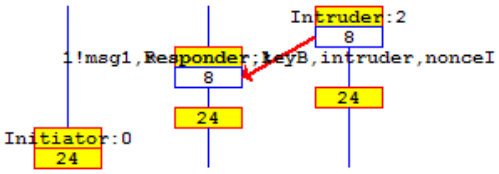


Figure 8: Guided simulation of the generated counterexample with the detected impersonation attack against NSPK

4.3 State space reduction and state exploration alternatives with the MI intruder model

To enable comparison between the general Dolev-Yao intruder model and the optimized MI intruder model for the NSPK protocol, it is necessary [24, 25] to characterize the used working modes of the SPIN model checking environment, to capture the characteristics of the compared objects and finally to highlight the unique aspects of the performed experiment. The following description provides an opportunity to assess the appropriateness of the experiment's design for understanding cause and effect relationships, which is the primary objective of the study.

The formal definition [26] of the performed experiment is given in Table 3. We analyze the behavior of the two intruder models (independent variable), for the purpose of exploring cause and effect facts, with respect to (a) the size of the generated reachability graphs (dependent variable) and (b) the implied memory usage (dependent variable), in the context of the NSPK protocol model and the MI intruder model that were implemented in SPIN.

Table 3. Formal definition of experiment

Analyze	the general Dolev - Yao intruder model vs. the MI intruder model
For the purpose of	exploring cause and effect relationships
With respect to	the size of the generated reachability graphs (number of states) and the implied memory usage
From the point of view of	the model checking practitioner
In the context of	the NSPK protocol model and the MI intruder model that were implemented in SPIN

The research questions and the associated metrics used in our study are:

i. Question 1: *Does the intruder model have a significant impact in the size of the model's state space?*
   Metrics used: We compare the state spaces (numbers of states and implied memory usage) produced with the two intruder models in the following cases:
   - First, we compare the size of the complete reachability graphs, i.e. model checking is not stopped when having detected the invalid end state. In our opinion, this is a more representative view of the improvement possibilities opened by an optimized MI intruder model, since the security flaw in general may be discovered at an arbitrary depth of the state space.
   - We also compare the size of the partial state spaces up to the depth of the detected invalid end state (refer to the sample results shown in Figure 7). Obviously, the size reduction achieved by the optimized MI intruder model in this case is orders of

magnitude less than the size reduction achieved in the aforementioned more general case.

ii. Question 2: *To what extend does the MI intruder model influence the size of the state space for different combinations of attack actions?*
Metrics used: We compare the size of the complete reachability graphs for 14 different versions of the MI intruder model.

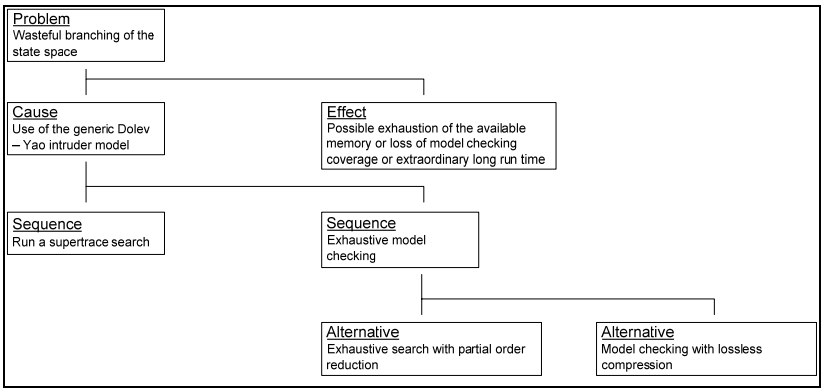
Table 4. State space reduction and state exploration techniques in SPIN

SPIN working modes	Description
Exhaustive search with partial order reduction (P.O.R.)	By default, SPIN [14] constructs both the state space and the transition relation on-the-fly by applying a Depth-First Search over the model's state space. Partial order reduction avoids creating states that cannot be affected by interleaving the execution of the model's processes. It is based on the dependencies that may occur between certain process statements, trying to discover the statements that are independent.
Supertrace search	In SPIN, to enable fast lookup of states they are stored in a hash table. Supertrace search or bit-state hashing is an option that decreases memory use by consuming only a small number of bits per state. When normal state storage is impossible, due to the limited memory, supertrace search provides an attractive alternative that increases significantly the capacity to store states. On the other hand, there is a low probability of loss of coverage, since when a hash collision happens the algorithm incorrectly infers that the state was already visited and therefore, this state is skipped. However, this cannot lead to false error reports.
Hash-compact search	In the hash-compact method, a hash function is utilized, in order to compress the state descriptor irrespective of the number of bits used to 64 bits. The resulting 64-bit values are then stored in a lookup table and in this way SPIN simulates a hash table with memory size far beyond what would be available otherwise.
State space compression	The so-called collapse compression works in the following way: SPIN identifies and stores the state configurations of each model process and instead of storing a complete state descriptor in the global state vector it uses the sequence of identifiers for the involved processes. It is a lossless compression technique that guarantees exhaustive coverage.
Breadth-First Search	This state exploration option enables on-the-fly model checking with Breadth-First Search over the generated reachability graph.
Statement merging	Statement merging is a special case of partial order reduction. This method suppresses redundant interleavings of process statements whenever possible, but it does not perform optimizations when non-interleaved sequences of statements can be merged into a single step.

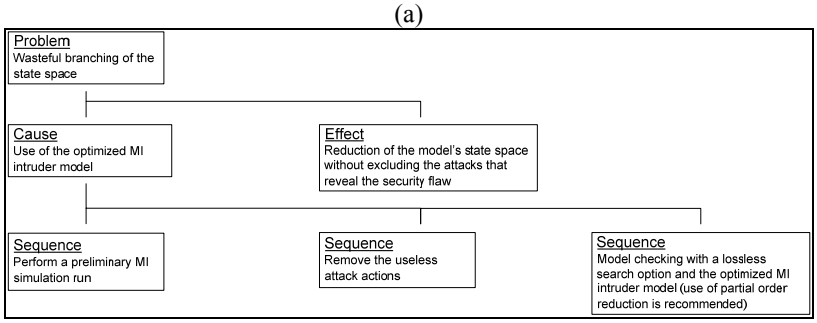
The default working mode of the SPIN model checker includes the partial order reduction functionality. However, in order to understand the cause and effect relationships associated with the use of the compared intruder models, we also report the size of the state spaces produced in the working modes of Table 4. Supertrace search tends to find errors quickly if they exist, but it is not the most productive option for demonstrating error-freedom. It is recommended as an attractive choice, when we first attempt to verify a model and the state space size is completely unknown. Hash-compact search exhibits superior accuracy, when tuned for a known state space. The main advantage of the breadth-first search option - which is effective only for safety properties (secrecy and authentication) - is that it finds the shortest path to an error state, while the depth-first search often finds a longer path. The (collapse) compression option reduces the memory requirements for an exhaustive search at the cost of increasing the run-time requirements.

Figure 9a shows a typical sequence of model checking tasks involved, when the size of the model's state space is completely unknown. It also introduces the explored cause and effect facts associated with the problem of the wasteful branching of the state space and the

use of a general Dolev-Yao intruder model. The results exposed in next paragraphs affirm the possible exhaustion of the available memory, even in the case of the NSPK protocol model that includes only three protocol steps, which were executed in two parallel sessions. Also, the obtained results confirmed the cause and effect relationship of Figure 9b, i.e. they revealed a significant reduction of the model's state space when using the optimized MI intruder model.



(a)



(b)

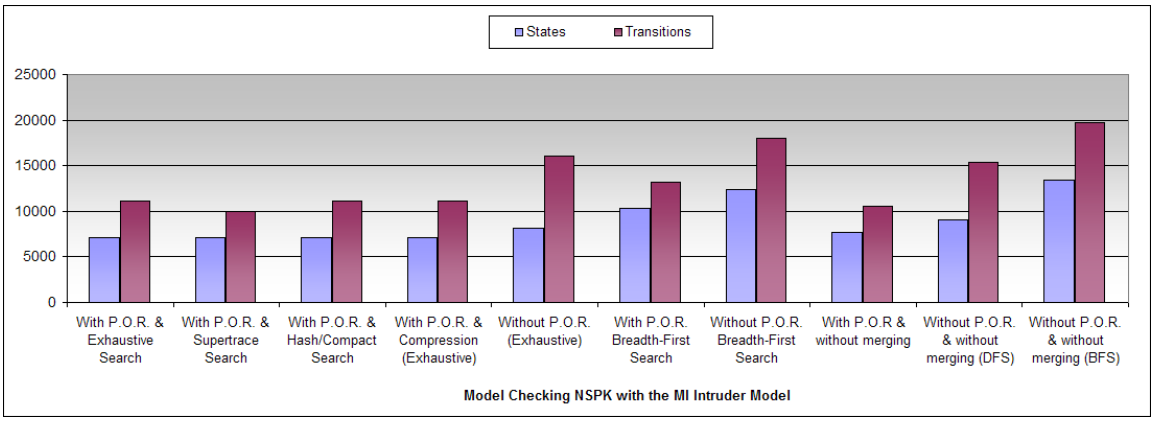
Figure 9: Cause and effect facts for (a) the general Dolev – Yao and (b) the MI intruder models

More precisely, Figure 10 provides the results obtained for the complete reachability graphs of the NSPK protocol model with (a) the optimized MI intruder model and (b) the general Dolev-Yao intruder respectively. We report the number of unique states stored by SPIN in a hash table, in order to enable fast lookup. SPIN outputs this number in all model checking reports together with the sum of stored and matched states that in fact represents the accessed transitions. The number of stored states provides an estimate of the size of the state space that also includes additional states, which are accessed as hash collisions.

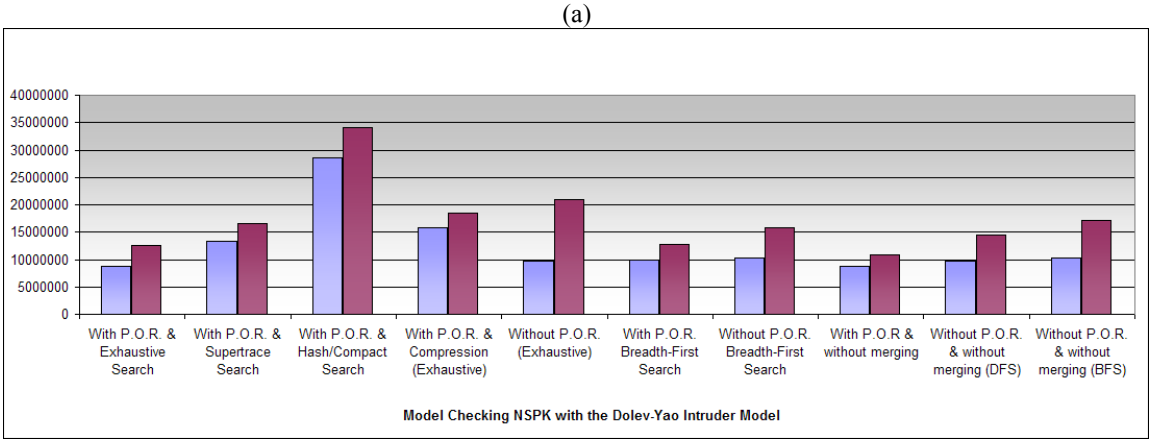
Memory usage (not shown in Figure 10) for the optimized MI intruder varies between 2.9 and 33 MB, apart from the compression alternatives that when used with partial order reduction expands to about 260 MB. This is not surprising, since according to the theory [27], in some cases compression techniques can result in memory expansion. If a compression technique performs well in a given context is typically discovered by experimentation. On the other hand, the complete reachability graph for the NSPK model with the general Dolev - Yao intruder consumed the available memory space, which was

set to 1 GB, apart from the compression alternatives, where the memory usage is limited to about 363 MB.

The numbers of stored unique states in Figure 10 show a reduction of about 10^3 times, when applying the optimized MI intruder. In model checking with a compression working mode this gap is widened and if using the hash-compact search with partial order reduction, the model with the Dolev-Yao intruder stores about $4 \cdot 10^3$ times the number of states for the MI intruder.



(a)



(b)

Figure 10: Size of complete reachability graphs for NSPK (a) with the optimized MI intruder model and (b) with the general Dolev-Yao intruder model

In Figure 10b, the working modes with no compression exhibit less stored unique states when compared with the working modes that apply compression. This is explained by a vast number of hash collisions that are not shown in the graph. As an example, when the Dolev-Yao intruder model is applied with partial order reduction, the typical exhaustive search yields $4.3 \cdot 10^6$ hash collisions in a hash table with $8 \cdot 10^6$ stored unique states. If using the optimized MI intruder, the reported hash collisions do not exceed the 43 cases in all tested working modes.

We conclude that for a protocol model with more steps, model checking with the general Dolev-Yao model is likely to require a compression technique like supertrace

search or hash-compact search, which in fact open a possibility for coverage loss. From this perspective, the optimized MI intruder yields a reduced state space with improved possibilities to be handled by SPIN in a predictable manner. In Figure 10a we also observe that partial order reduction, when combined with Breadth-First Search, is not as effective as it is when combined with Depth-First Search.

Figure 11, provides results for the partial state spaces generated up to the depth of the detected invalid end state. The NSPK protocol model with the Dolev-Yao intruder model generated state spaces with about 2.5 times the number of stored unique states for the MI intruder. This improvement depends on the depth where the error is discovered and from this point of view the Breadth-First Search finds the shortest path to the error. While in Figure 7 (MI intruder with Depth-First Search) the error was detected at depth 25 ($1.4 \cdot 10^3$ hash collisions in a hash table with 264 stored unique states), when using Breadth-First Search the error was discovered at depth 5 (no hash collisions in a hash table with 112 stored unique states). On the other hand, when verifying NSPK with the Dolev-Yao intruder model and Depth-First Search the error is detected at depth 48 resulting in $7.3 \cdot 10^4$ hash collisions in a hash table with 698 stored unique states.

Figure 11: Size of partial state spaces for the NSPK security flaw with the optimized MI intruder model (MI) and the general Dolev-Yao intruder model (DY)

In the second part of this exploratory study, Figure 12 shows the effects on the size of the complete reachability graph, when using different versions of the MI intruder model, where each version includes different combinations of attack actions. We report stored unique states obtained with exhaustive search, where hash collisions are negligible. These results make it possible to compare relative state space reductions, when using the different versions of the MI intruder model. In effect, they provide potentially valuable data for costs and benefits towards extending the MI rules of Table 2, for more efficient use of the MI intruder model. However, as we already noted the reported results depend on the interleaving caused by the modeled protocol structure, as well as on the checked protocol configuration (the number of initiators and the number of responders used).

Although it is shown that the intruder model has a dominant effect in memory usage and the size of the state space, an obvious threat to the internal validity of our exploratory study is the possibility of confounding. In the related bibliography, this term is often used to refer to all factors covarying with the considered independent variables (intruder model used), which also influence the dependent variables (memory usage and size of the state space) but have not been included in the experiment's design. We explored many possible combinations of the two intruder models with the reduction and the state exploration

techniques provided in SPIN, but our experiment was not extended to different numbers of initiators and responders, which we think are factors that may covary with the selected intruder model. The most prominent threat to the external validity of the presented study is that the reported reductions in the state space of the NSPK protocol with the optimized MI intruder model, cannot be generalized to other protocols and protocol configurations. As noted in the introduction of the described study, there is no similar study with results that can be generalized, since reported state space improvements intrinsically depend on the protocol structure reflected by the analyzed model.

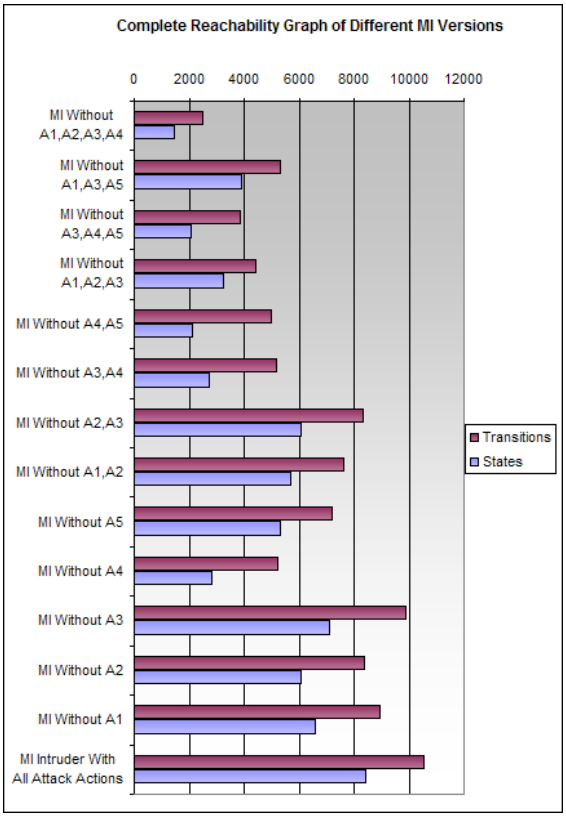


Figure 12: Complete reachability graphs for MI intruder models in NSPK model checking with different combinations of attack actions

4.4 Guidelines for the model checking practitioner

For the model checking practitioner, an effective procedure for applying the MI intruder model to large model checking problems includes the following steps:

1. A first preliminary simulation run with the MI intruder model will provide valuable feedback for the applicable optimizations in the protocol model.
2. Since the size of the state space is still unknown, there is no need to directly apply the detected optimizations in the first verification attempt. We propose this first verification to take place with an efficient search option such as supertrace search

together with partial order reduction. However, we remind that the mentioned search option opens a possibility for loss of coverage.

3. For proving the absence of security flaws in a model with very large state space, the analyst now can choose between the following two options.
    a. To apply an alternative compression method if available (like the hash/compact search or the collapse compression). This option incurs an additional cost, either in the form of increased model checking run time or in the form of repetitive trials, in order to fine tune the state space.
    b. To apply the possible MI intruder model optimizations. With the current model implementation, the analyst modifies the PROMELA code manually, in order to remove the useless attack actions.
4. The optimized MI intruder model may set the model checking problem in a size, which can be handled by the available exhaustive search options. If however this is still not achieved, then the analyst returns to step 3, where either tries to apply an accurate compression method together with additional reduction techniques such as symmetry reductions or alternatively to implement new optimizations in the current MI intruder model.

New optimizations will be based on inventing new MI rules for more efficient use of the MI intruder model or on step-by-step analyses of additional attack actions, beyond those mentioned in Table 1. However, any extension to the current MI intruder model requires PROMELA programming skills, as well as a basic understanding of the algorithmic details of the current implementation.

5. Related work

In related bibliography, there are significant research contributions concerning uses, extensions and improvements of the Dolev-Yao intruder model. Many of these works [2, 28, 29, 30] provided a basis for integrating a custom user-specified intruder model into innovative model checking techniques for the analysis of security properties.

One of the first systems that implemented the Dolev - Yao assumptions and the secrecy failure verification approach was the Interrogator tool [28]. Given a final state in which the intruder knows some message, which should be secret, the Interrogator tries all possible ways of constructing a path that reaches this particular state. If it finds such a path, then it has identified a security flaw.

Finite state analysis of cryptographic protocols can take place in specialized security model checkers, like BRUTUS [6], where security violations are encoded as failures of secrecy or authentication. Alternatively, finite state analysis is often carried out in general-purpose model checkers like Murφ [31] and the FDR (Failures Divergence Refinement) [32] model checker.

When focused on the problem of the state space explosion, a series of interesting works exploit symmetry and partial order reduction techniques [6, 7, 31, 33]. In [10], the authors propose model checking with pre-configuration, which is a divide-and-conquer method for verifying security protocols.

In [9], the authors prove the soundness of two optimizations for the intruder model. The first optimization technique lets the intruder always intercepting messages sent by the honest protocol participants, instead of making such interception optional. The second technique prevents the intruder from sending messages to honest participants in states where at least one of the honest participants is able to send a message. This can be considered as an alternative to partial order reduction techniques that exploit the relative independence of the honest participants. In the performed experiments, the first mentioned technique resulted in significant reductions in the number of reachable states (by a factor of 20) and the execution time. The second mentioned optimization technique resulted in further 43% reduction in the number of states and a 40% reduction in the execution time.

An interesting variant of the Dolev - Yao intruder model is proposed in [30], for analyzing an unbounded number of protocol sessions with either bounded or unbounded numbers of messages.

Symbolic reduction [29] exploits a symbolic state transition relation, which gives rise to a finite symbolic state space [34, 35], where symmetry redundancy is eliminated. Each symbolic state summarizes a – possibly infinite – number of concrete states that can be obtained by instantiating variables in the symbolic state specification.

"Lazy" intruder models [29, 33] aim in a demand-driven exploration of the model's state space by overlooking fake messages that do not match the patterns of the messages awaited by the protocol participants.

In [36], the authors introduce an optimized intruder model for the verification of satisfiability properties, provided the interception of all exchanged messages and the assumption that some of the abilities of the intruder have an instantaneous effect.

Athena [37] builds on a different model representation, where in contrast to the conventional trace-based modeling approaches, a set of protocol runs that differ only in the order of interleaving executions of the individual participants is represented by only one state. This is achieved due to a clever extension to the strand space model representation. There is some form of symbolic reduction functionality, but Athena also allows the development of protocol-specific or general pruning theorems. Through this semi-automated approach the analyst uses theorems, in order to prune from the state space all states proved that do not contribute to the final result.

In [13], the authors propose a specialized intruder model for verifying a class of liveness properties in security protocols. The intruder's behavior is proved that conforms to the Dolev-Yao assumptions, with the only difference that he does not delay the delivery of intercepted messages.

Finally in [38], the authors extend a flexible specification framework based on the Dolev - Yao intruder model. The analysis is designed for security protocol verification based on typed multi-set rewriting with a static check called access control. The proposed static check aims in catching specification errors, such as a principal trying to use a key that he is not entitled to access.

The MI intruder model can be compared only with other approaches that optimize the branching of the state space, due to the intruder's fake messages, without excluding possible attacks. Existing intruder models [6, 9, 29, 31, 33], whether they are based on mere state space exploration or whether they combine it with natural deduction-style reasoning or "lazy" evaluation, delimit the state space branching by exploiting information about the

messages that protocol participants expect. This optimization avoids generating from the intruder's knowledge, fake messages that cannot have an effect in the protocol's execution, if every protocol participant rejects them as unexpected.

To the best of our knowledge, the MI intruder model is the first model that uses message metadata collected from a preliminary simulation run. This data enhances the intruder's knowledge with additional information regarding protocol behavior facts that in some cases cannot be observed dynamically across the explored state space paths. It is thus possible to improve the pruning of the state space by exploiting known security principles. Given the intruder's knowledge for the protocol execution, these principles allow determining in advance, whether an attack action can or cannot cause a security violation.

An apparent weakness of the described two-stage model checking procedure is that it is likely to result in a less general protocol-specific intruder model. However, we believe that this may be more useful from a generic Dolev - Yao model that is potentially difficult to yield results for a computationally hard model checking task. A worth to mention limitation is that the MI intruder model can be implemented only in model checkers that allow simulated execution of the protocol model. This is true for many general-purpose model checkers, like Murφ and SPIN. In current implementation, the analyst is expected to interfere with the intruder model, in order to remove the useless attack actions. This requires some basic knowledge of the intrinsic details of the intruder model, but this shortcoming may be bypassed in a prospective MI-based model checking environment that will automate the described approach.

The strength of the MI intruder model lies in the fact that it provides an approach for customizing the intruder behavior, in order to deliver additional model checking optimizations. For a potential extension concerning insertion of feasibility check for a new attack action the analyst will have to implement additional MI rules and metadata comparisons. Appropriate enabling conditions will be derived by natural deduction over the sequence of send and receive actions that specify the analyzed attack action. If necessary, the model may be extended by including additional metadata parameters, but this will cause modifications in the MI initialization code that stores metadata values for the intercepted messages.

Finally, besides the provided results for the combined use of the MI intruder model with partial order reduction, MI may be combined with symmetry reduction and/or other reduction techniques from those mentioned in current section. An open problem is our model's potential integration into security model checkers that employ advanced state space exploration techniques, such as "lazy" evaluation.

6. Conclusion

The MI intruder model aims to restrict the inherent combinatorial complexity of security model checking with general Dolev-Yao intruder models. This is achieved through message inspection that allows customizing the intruder behavior, by taking into account protocol specific metadata for the structure and the characteristics of the exchanged messages. The only requirement is that it can be implemented in model checking environments that

support both simulation and model checking of the analyzed security protocol. We conducted a series of experiments for exploring the improvements in the model checking of the NSPK protocol, when compared with the generic Dolev-Yao intruder model. The PROMELA code for the MI-based model checking of the NSPK protocol is available online in [39].

The MI intruder model provides an open-ended framework for integrating additional protocol-specific model checking optimizations. More innovative extensions considered as future prospects include integration into the intruder's knowledge of fine grained information for the patterns of messages expected by the protocol participants. In this case, appropriate MI rules will make it possible to further restrict the set of messages composed by the intruder. Other future perspectives are the incorporation of intruder model optimizations that exploit certain security protocol properties [9], as well as implementation of a customized intruder model for properties that involve liveness [13], like session termination or timeliness. In these cases, the intruder model has to fulfill specific fairness assumptions, which are not covered in the general case. A long-term development goal is the design of an integrated modeling environment that will provide the described functionality in a usable model checking package.

Acknowledgments

We acknowledge the anonymous referees for their helpful comments, which contributed to improving the quality of the article.

References

1. Burrows M., Abadi M. and Needham R., "A logic of authentication", ACM Transactions on Computer Systems, Vol. 8 (1), pp.18-36, 1990.
2. Lowe G., "An attack on the Needham-Schroeder public key authentication protocol", Information Processing Letters, Vol. 56 (3), pp.131-136, 1995.
3. Lopez, J., Ortega, J. J., Troya, J. M., "Protocol engineering applied to formal analysis of security systems", In Proc. of Int. Conf. of Infrastructure Security, Bristol, UK, LNCS 2437, Springer-Verlag, pp. 246-259, 2002.
4. Clarke M. E., Grumberg O. and Peled D. A., "Model Checking", MIT Press, 1999
5. Dolev D. and Yao A., "On the security of public-key protocols", IEEE Transactions on Information Theory, Vol. 2(29), pp.198-208, 1983.
6. Clarke E. M., Jha S. and Marrero W., "Verifying security protocols with Brutus", ACM Transactions on Software Engineering and Methodology Vol. 9(4), pp.443-487, 2000.
7. Clarke, E., Jha, S. and Marrero, W., "Partial order reduction for security protocol verification", In Proc. of the 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS), Berlin, Germany, 2000.
8. Gritzalis, S., Spinellis, D. and 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.
9. Shmatikov, V. and Stern, U., "Efficient finite-state analysis for large security protocols", In Proc. of the 11th Workshop on Computer Security Foundations (CSFW), pp. 106-120, 2000.

10. Kim K., Abraham J. A. and Bhadra J., "Model Checking of Security Protocols with Pre-configuration", In Proc. of the 4th International Workshop on Information Security Applications, WISA, Korea, LNCS 2908, Springer-Verlag, pp.1–15, 2003.
11. The SPIN model checker official website, available at http://spinroot.com (last accessed 13/2/2009).
12. Needham, R. M. and Schroeder, M. D., "Using Encryption for Authentication in Large Networks of Computers," Communications of the ACM 21, 1978.
13. Cederquist J., Torabi I. and Dashti M. , "An intruder model for verifying liveness in security protocols", In Proc. of the 2006 ACM Workshop on Formal Methods in Security Engineering (FMSE), ACM Press, New York, USA, pp.23-31, 2006.
14. Holzmann G. J., "The Spin Model Checker - Primer and Reference Manual", Addison-Wesley, 2003.
15. Basagiannis S., Katsaros P. and 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, Springer Verlag LNCS 4680, pp.238-251, 2007.
16. Huang D., Sinha A. and D. Medhi, "A double authentication scheme to detect impersonation attack in link state routing protocols", In Proc. of the IEEE International Conference on Communications (ICC), 2003.
17. Heather, J., Lowe G. and 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.
18. Syverson, P. and Cervesato, I., "The logic of authentication protocols", In Proc. of the 1st International School on Foundations of Security Analysis and Design (FOSAD 2000), Springer-Verlag LNCS 2171, pp. 63-137, 2001.
19. 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.
20. Zhang, C., Zhou, M. C. and Yu, M., "Ad hoc network routing and security: A review", International Journal of Communication Systems, Vol. 20 (8), pp.909-925, 2007.
21. Basagiannis S., Katsaros P., Pombortsis A. and Alexiou N., "Probabilistic model checking for the quantification of DoS security threats", Computers & Security, Vol. 28 (6), pp. 450-465, 2009
22. Grunske, L., Joyce, D., "Quantitative risk-based security prediction for component-based systems with explicitly modeled attack profiles", Journal of Systems and Software, Vol. 81, pp.1327-1345, 2008.
23. Amadio R. M. and Charatonik W., "On Name Generation and Set-Based Analysis in the Dolev-Yao Model", In Proc. of CONCUR, Springer-Verlag LNCS 2421, pp.499-514, 2002.
24. Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., El Emam, K. and Rosenberg, J. "Preliminary guidelines for empirical research in software engineering", IEEE Transactions on Software Engineering, Vol. 28, Issue 8, pp. 721 – 734, 2002.
25. Wohlin, C., Petersson, H., Aurum, A., Shull, F. and Ciolkowski, M., "Software inspection benchmarking – A qualitative and quantitative comparative opportunity", In Proc. of the 8th IEEE International Symposium on Software Metrics, IEEE Computer Society, pp. 118-127, 2002.
26. Briand, L., Bunse, C., Daly, J., and Differding, C., "An experimental comparison of the maintainability of object-oriented and structured design documents", Empirical Software Engineering, Vol. 2, Issue 3, pp. 291–312, 1997.
27. Gregoire, J.-C., "State space compression in SPIN with GETSs", In Proc. of the 2nd SPIN Workshop, pp. 90-108, 1996.

28. Millen J. K., Clark S. C. and Freedman S. B., “The Interrogator: Protocol Security Analysis”, IEEE Transactions on Software Engineering, Vol. 13(2), 1987.
29. Basin D., Modersheim S. and Vigano L., “OFMC: A Symbolic Model-Checker for Security Protocols”, International Journal of Information Security, 2004.
30. Chevalier Y., Kuesters R., Rusinowitch M., Turuani M. and Vigneron L. “Extending the Dolev-Yao Intruder for Analyzing an Unbounded Number of Sessions”, Computer Science Logic (CSL) and 8th Kurt Goedel Colloquium (8th KCG), 2003.
31. Mitchell, J. C., Mitchell, M. and Stern, U., “Automated analysis of cryptographic protocols using Murφ”, In Proc. of the IEEE Symposium on Security and Privacy, pp. 141-151, 1997.
32. 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.
33. Roscoe A. W. and Goldsmith, M., “The perfect spy for model-checking cryptoprotocols”, In Proc. of the 1997 DIMACS Workshop on Design and Formal Verification of Security Protocols, 1997.
34. Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, M. and Tacchella, A., “NuSMV 2: An OpenSource Tool for Symbolic Model Checking”, In Proc. of the International Conference on Computer-Aided Verification (CAV), Copenhagen, Denmark, 2002.
35. Panti, M., Spalazzi, L. and Tacconi., S., “Using the NUSMV model checker to verify the Kerberos Protocol”. In Proc. of the 3rd Collaborative Technologies Symposium (CTS), pp. 27-31, 2002.
36. Armando A. and Compagna L., ”An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols”, in Proc. of the Workshop on Automated Reasoning for Security Protocol Analysis ARSPA, ENTCS, pp.91-108, 2004.
37. Song D., “Athena: a new efficient automatic checker for security protocol analysis”. In P. Syverson, ed., In Proc. of the 12th IEEE Computer Security Foundations Workshop, Italy, IEEE Computer Society Press, pp.192-202, 1992.
38. Cervesato I., “The Dolev-Yao intruder is the most powerful attacker”, In Proc. of the 16th Annual Symposium on Logic in Computer Science (LICS), IEEE Computer Society Press, 2001.
39. Message Inspection Intruder Modeling Framework, online: http://mathind.csd.auth.gr/mi_work.html (last accessed 13/2/2009).