Fundamenta Informaticae 93 (2009) 185–203
DOI 10.3233/FI-2009-96
IOS Press

Simulation of Security Protocols based on Scenarios of Attacks*

Gizela Jakubowska, Piotr Dembiński
Institute of Computer Science, PAS, J.K. Ordona 21, 01-237 Warszawa, Poland

Wojciech Penczek
Institute of Computer Science, PAS, J.K. Ordona 21, 01-237 Warszawa, Poland and
Institute of Informatics, Podlasie Academy, Sienkiewicza 51, 08-110 Siedlce, Poland,

Maciej Szreter
Institute of Computer Science, PAS, J.K. Ordona 21, 01-237 Warszawa, Poland
{*gizela,penczek,piotrd,mszreter*}*@ipipan.waw.pl*

Abstract. In this paper we offer a methodology allowing for simulation of security protocols, implemented in the higher-level language Estelle, using scenarios designed for external attacks. To this aim we apply a translation of specifications of security protocols from Common Syntax to Estelle and an encoding of schemes of attacks into Estelle scenarios. We show that such an intelligent simulation may efficiently serve for validating security protocols.

1. Introduction

There are many papers addressing simulation of protocols in order to validate their correctness [4, 10, 14], but only a few of them discuss simulation of security protocols [18, 11]. The point is that validation of security protocols differs a lot from validation of network protocols. The first domain is based on searching for security flaws (which can be exploited by an intruder), while in the second one deadlocks or other not predictable situations are usually expected to be discovered. Unfortunately, simulation of security protocols is even more complex than of network protocols. In general the security problem is undecidable [8], which is one of the major challenges for automatic validation. The problem becomes NP-complete when the number of sessions and the number of nonces is bounded. In practise, this complexity results in an exponential explosion in the number of runs of a protocol, which need to be explored.

Address for correspondence: Institute of Computer Science, PAS, J.K. Ordona 21, 01-237 Warszawa, Poland

*The authors acknowledge partial support from the Ministry of Science and Higher Education under grant number N516 038 31/3853 and TINFO: 55/E-82/BWSN-0122/2008.

In this paper we define four schemes of (authentication and secrecy) attacks upon security protocols that cover almost all of the attacks from [1, 7]. These schemes include: Man-in-the-Middle (*MITM*, for short), Replay, Reflection, and Interleaving. For each of the above schemes of attacks we designed an algorithm, which for given parameters of the protocol, returns a scenario of an attack of this type. The scenarios allow for reducing dramatically (more than exponentially) the number of runs to be explored while looking for the above types of attacks by means of simulation or verification. Next, we continue the idea of validation of security protocols using the simulator/debugger of Estelle, Estelle Development Toolkit (EDT) [3], presented in [9]. However, this time concrete simulation scenarios are considered according to the schemes generated for different types of attacks. To this aim we implement a translation from scenarios of attacks in Common Syntax [7] to specifications in Estelle [5]. This task requires to model, in Estelle, the honest participants as well as the intruder who has a choice in selecting messages to be sent according to the scenario translated. As a case study we present results of simulation for a few standard protocols. Moreover, to show an important practical application of our method, we find the attack [16] upon a newly designed protocol [6] using simulation in Estelle.

Related Work. The paper [11] is closest to our work, where a modelling of authentication protocols with state machines is presented to validate some of their security properties through simulation. Moreover, simulation of cryptographic protocols was considered in [18], where a general formula of Agent for simulation is proposed together with the dynamical environment based on the Java technology.

Outline. The rest of the paper is organised as follows. Section 2 introduces security protocols in Common Syntax (CS). Schemes of attacks are discussed in Section 3. A translation from scenarios in CL to Estelle and experimental results are given in Section 4. The paper ends with some conclusions.

2. Security Protocols in Common Syntax

In this section we introduce basic syntax for writing security protocols. We describe the protocols using a standard notation, called Common Syntax (CS) [7], developed for cryptographic protocols [1]. Usually, protocols involve two, three or four roles, which we denote with capital letters A, B for the principals, and with S or S' for the servers. Let a protocol $\mathcal{Q}$ be represented by a finite sequence of instructions:

1. $X_1 \longrightarrow Y_1 : M_1$
...
n. $X_n \longrightarrow Y_n : M_n$

where $X_i, Y_i \in \{A, B, S, S'\}$, $X_i \neq Y_i$, $Y_i = X_{i+1}$, for $1 \leq i \leq n$, and M_i is called a *message* variable[1].

The informal meaning of the instruction $A \longrightarrow B :\ M$ is that a principal of role A sends a message, which is a value of the variable M, to a principal of role B. M is composed of variables ranging over identifiers representing principals ($\mathcal{PV}$), keys[2] ($\mathcal{KV}$), nonces ($\mathcal{NV}$), and possibly timestamps ($\mathcal{TV}$), and their lifetimes ($\mathcal{LV}$). Formally, the message (variables) are generated by the following grammar:
$Message ::= Component \times Component^*$
$Component ::= Cipher \mid Atom$
$Cipher ::= \{Component^*\}_K$

[1]We will frequently refer to variables via their names like *message, principal, nonce, key, timestamp* if this does not lead to confusion.
[2]Keys can be asymmetric (public and secret) or symmetric.

$Atom ::= P \mid N \mid K \mid T \mid L$,
where $P \in \mathcal{PV}$, $N \in \mathcal{NV}$, $K \in \mathcal{KV}$, $T \in \mathcal{TV}$, and $L \in \mathcal{LV}$.

For $X, Y \in \{A, B, S, S'\}$, K_{XY} is a *symmetric key* variable of X and Y, PK_X is a *public key* variable of X, SK_X is a *secret key* variable of X,N_X is a *nonce* variable of X, T_X is a *timestamp* variable of X, and L_X is a *lifetime* variable of X. A message M can be encrypted with a key K, denoted by $\{M\}_K$[3]. For example $\{T_X, N_X\}_{K_{XY}}$ is a message variable containing a timestamp variable T_X and a nonce variable N_X encrypted with the a key variable K_{XY}.

Keys can shared between an agent and a server (e.g., K_{AS}, K_{BS}), between two agents (e.g., K_{AB}), or owned by one agent only (e.g., K_{AA}, K_{BB}). The *nonces* represent random non-predictable numbers declared to be used only once by a concrete agent. The *timestamps* are unique identifiers whose values are provided by the (local) clock of its issuing entity and determine the time when a given message is generated. A value related to a timestamp is a *lifetime* defining how long since the timestamp creation it is acceptable to use each component of the messages it relates to. A concrete *message* consists of components which are built of atomic cryptographic primitives that are elements of the following finite sets of identifiers: $\mathcal{P} = \{s, s', a, b, \iota\}$ - *principals*, also called agents or participants, $\mathcal{SK} = \{k_{as}, k_{bs}, k_{ab}, k_{bb} \ldots\}$ *symmetric keys*, $\mathcal{AK} = \{pk_a, pk_b, sk_a, sk_b, \ldots\}$ - *asymmetric keys*, let $\mathcal{K} = \mathcal{SK} \cup \mathcal{AK}$ be the set of all the keys, $\mathcal{N} = \{n_a, n_b, n'_a, n'_b, \ldots\}$ - *nonces*, $\mathcal{T} = \{t_a, t_b, t_s, \ldots\}$ - *timestamps*, $\mathcal{L} = \{l, l', \ldots\}$ - *lifetimes*, $\mathcal{R} = \{r_a, r_b, \ldots\}$ - *random numbers*.

In a *session* of the protocol the variables are substituted by concrete identifiers (names) of the principals and of the message elements. By a *(protocol) run* we mean a finite sequence of instructions resulting from a fixed number of possibly parallel sessions[4] of the protocol. In order to distinguish between sessions, they are numbered with natural numbers. For each session of number i in order to make clear who is sending a message to whom, we define an assignment of the principal names s, s', a, b, ι and the intruder impersonations $\iota(a), \iota(b), \iota(s), \iota(s')$ to the roles A, B, S, S', defined as the following function $p_i : \{A, B, S, S'\} \longrightarrow \mathcal{P} \cup \{\iota(a), \iota(b), \iota(s), \iota(s')\}$. We sometimes refer to this function as to $X \leftarrow y$ iff $p_i(X) = y$, provided the number of a session is understood.

A *plain session* a protocol is a session, where the roles A, B, S are substituted in the following way: $p(A) = a$, $p(B) = b$, and $p(S) = s$. By a *scenario* of an attack we mean a run composed of two or more sessions in which an assignment for each role has been defined. Typically, a scenario does not specify contents of the messages, but in some cases repetition of previously sent messages can be indicated.

3. A Taxonomy of External Attacks

We start with defining a taxonomy of attacks upon security protocols, which is based on that of [19]. We consider the four main types of attacks[5]: Man-in-the-Middle (*MITM*, for short), Replay (*REPL*), Reflection (*REFL*), and Interleaving (*INTRL*). In Table 2 (see Appendix) all the protocols classified within our taxonomy are listed. For each protocol we give reference(s) to the literature, where attack(s) for this protocol can be found. Next, we discuss each type of an attack and show an example for a selected protocol. We identify all the common features of the attacks for all the protocols susceptible to this type

[3]We depart here slightly from CS where encryption of a message M with a key K is specified as $E(K : M)$.
[4]Some steps of sessions can be omitted.
[5]Each attack is referenced by the name of the protocol, its source in the literature, and the number if there are more attack for the same protocol.

of attack. This allows for designing an algorithm, which for a given type of attack and parameters of the protocol, returns a scenario of an attack of this type. The parameters include: the number of protocol steps (n), a number of sessions ($nses$), a format of each step j of the session i (like $i.j : A \longrightarrow B : M_{i,j}$), and possibly other parameters specific for a type of attack.

3.1. REPLAY attacks

Replay attack is a breach of the security in which information is stored without authorisation and then retransmitted to trick the receiver into unauthorised operations such as false identification or authentication or a duplicate transaction. As most attacks are of the replay type, this section describes two classes which are built on typical schemas.

DoS-type REPLAY Attacks. The main intention of this type of attack is to deceive the responder of the session that the (assumed) initiator has run one or more parallel sessions with him. When the number of active sessions of the responder exceeds the limit allowed by his system, the denial of service is gained. These attacks are typically composed of two (or more) sessions one following another. The first session is fundamental and it is performed entirely, while in the next ones some initial steps which do not involve the responder are omitted. The number of the steps skipped varies from one in Denning Sacco and WFM to two in SPLICE/AS and Kerberos v.4. The first session is used to collect some important data to replay them later in next sessions. The scenario of this specific type of reply attack can be characterised as follows. First, a plain session is executed. In each new session $\iota(a)$ is assigned to A and the session starts from the first step in which the responder b is involved. The message of the same step of the former session is then replayed. If the sender of this message is s, then ι impersonates s in the whole session, otherwise only a is impersonated by ι. The sequence of steps from the second session can be repeated several times, leading to a *Denial of Service attack* (DoS).

Simple REPLAY Attacks. In the Simple Replay attacks the intruder replays any information eavesdropped during previous sessions (the first session is played honestly) rather than replaying the same message taken from the first session like in the DoS-like attacks. Moreover, all the messages are processed, so no one is omitted. In this scenario the parameter *repl* indicates the beginning step of the replaying activity of the intruder. Each message preceding this step is forwarded in every session declared, while the messages following this step are processed in a special way. A message is intercepted by the intruder impersonating the receiver and resent (with a contents changed) by the intruder impersonating the sender to the original receiver. If the step $j \geq repl$ of the protocol is of the form: $X \longrightarrow Y$: M_j, then this step of session i in the Simple Replay scenario is denoted as follows:
(i.j) $x \longrightarrow \iota(y)$: M_j, and (i.j') $\iota(x) \longrightarrow y$: M'_j,
where M'_j is a modified (by the intruder) version of the message M_j.
Note that if the algorithm generates a step in which the intruder plays the role of both the sender and the receiver, then this step is omitted in the scenario. In this scenario, the intruder's actions are based only on replacing original messages with new ones by applying a reply action. Therefore, the intruder does not impersonate any of the participants. For the first session, we have the following assignment: $A \leftarrow a$, $B \leftarrow b$, and $S \leftarrow s$, while for the remaining sessions two possible assignments are specified as: $A \leftarrow a$, $B \leftarrow b$, and $S \leftarrow s$, or $A \leftarrow \iota$, $B \leftarrow b$, and $S \leftarrow s$.

The entry [*] will be explained in Section 3.4.

Algorithm 1: Algorithm for *DoS-REPL* attack scenarios

```
parameters   : $n$ - a number of protocol steps
             : $nses$ - a number of sessions
assignments  : for $2 \leq k \leq nses$
             : $p_1(A) := a, p_1(B) := b, p_1(S) := s$
             : $p_k(A) := \iota(a), p_k(B) := b, p_k(S) := s$
variables    : $repl$ - a step where the replay starts
$i \leftarrow 1, j \leftarrow 1$
repeat
    $(i.j.)\ p_i(X) \rightarrow p_i(Y) : M_j$
    $j$++
until $j \leq n$
$i$++, $j \leftarrow 1$
while $Y \neq B$ in $j.\ X \longrightarrow Y$: $M_j$ do $j$++      %% $(i.j.)$ omitted $M_j$
if $j.\ S \longrightarrow B$: $M_j$ then $p_i(S) := \iota(s)$
$repl \leftarrow j$
repeat
    $(i.j.)\ p_i(X) \longrightarrow b : M_{1.j}$
    $j$++
    repeat
        $(i.j.)\ p_i(X) \longrightarrow p_i(Y)$: $M_j$
        $j$++
    until $j \leq n$
    $i$++, $j \leftarrow repl$
until $i \leq nses$
```

Example 3.1. Consider the protocol Andrew Secure RPC. After applying the Algorithm 2 for the parameters: $n = 4$, $nses = 2$, $repl = 4$, and $z = a$, the following scenario is returned.

Scenario	*Attack*
1.1. $a \longrightarrow b$:	$M_1 = a, \{n_a\}_{k_{ab}}$
1.2. $b \longrightarrow a$:	$M_2 = \{n_a + 1, n_b\}_{k_{ab}}$
1.3. $a \longrightarrow b$:	$M_3 = \{n_b + 1\}_{k_{ab}}$
1.4. $b \longrightarrow a$:	$M_4 = \{k'_{ab}, n'_b\}_{k_{ab}}$
2.1. $a \longrightarrow b$:	$M_1 = a, \{m_a\}_{k_{ab}}$
2.2. $b \longrightarrow a$:	$M_2 = \{m_a + 1, m_b\}_{k_{ab}}$
2.3. $a \longrightarrow b$:	$M_3 = \{m_b + 1\}_{k_{ab}}$
2.4. $b \longrightarrow \iota(a)$:	$M_4 = \{k''_{ab}, m'_b\}_{k_{ab}}$
2.4.' $i(b) \longrightarrow a$:	$M'_4 = \{k'_{ab}, n'_b\}_{k_{ab}}$

3.2. INTERLEAVING Attacks

The aim of this attack is to perform a fraud of information from the responder of a session (or the server), using the standard challenge/response exchanges. This way, the intruder may keep some credentials up to date. This attack is mostly upon protocols with a *repeated authentication part* (*RAP*, for short), where some temporal cryptographic constructs from the *main part* of the protocol are used (e.g., the ticket together with a session key or other credentials generated by the server). Typically, this attack gives the intruder an updated (refreshed) ticket with the same session key. For protocols with *RAP*, at the

Algorithm 2: Algorithm for *Simple-REPL* attack scenarios

```
parameters  : $n$ - a number of protocol steps
            : $nses$ - a number of sessions
            : $repl$ - a step, where the replay starts
            : $z \in \{a, \iota\}$ - a parameter of assignments
assignments : for $2 \leq k \leq nses$
            : $p_1(A) := a, p_1(B) := b, p_1(S) := s$
            : $p_k(A) := z, p_k(B) := b, p_k(S) := s$
$i \leftarrow 1, j \leftarrow 1$
repeat
    $(i.j.)\ p_i(X) \rightarrow p_i(Y) : M_j$
    $j$++
until $j \leq n$
$i$++, $j \leftarrow 1$
repeat
    while $j < repl$ do
        $(i.j.)\ p_i(X) \rightarrow p_i(Y) : M_j$
        $j$++
    repeat
        $(i.j.)\ p_i(X) \rightarrow \iota(p_i(Y)) : M_j$
        $(i.j.')\ \iota(p_i(X)) \rightarrow p_i(Y) : M'_j$
        $j$++ [*]
    until $j \leq n$
    $j$++
until $i \leq nses$
```

beginning of the scenario, the whole main part of the protocol is executed once with no active action of the intruder. Then, the repeated authentication part is processed starting from the step *rstep* and the first session. If the protocol with no *RAP* is considered, then the first step of the protocol is taken as *rstep*.

The scenario is guided in such a manner that each two steps (challenge/response) of a session are followed by the same two steps of the next session until the last session is reached. When one such a pass through the sessions is finished, the algorithm takes the next two messages and this process is repeated until the last message is reached. If *RAP* contains an odd number of steps, then the last pass includes the last message only. Note that if the algorithm generates a step in which the intruder plays the role of both the sender and the receiver, then this step is omitted in the scenario.

In this scenario, in *RAP* of the odd sessions (except for the first one), the intruder impersonates a initiating the session with b, while in the even sessions, the intruder is impersonating either a or b. So, we have the following assignment in each odd session: $A \leftarrow \iota(a)$, $B \leftarrow b$, and $S \leftarrow s$ or $S \leftarrow \iota(s)$, and in each even one: $A \leftarrow \iota(a)$ and $B \leftarrow b$ and $S \leftarrow s$, or $A \leftarrow \iota(b)$ and $B \leftarrow a$ and $S \leftarrow s$, or $A \leftarrow a$ and $B \leftarrow \iota(b)$ and $S \leftarrow s$, or $A \leftarrow b$ and $B \leftarrow \iota(a)$ and $S \leftarrow s$.

Example 3.2. Consider the protocol KSL:

1. $A \longrightarrow B : M_1 = N_A, A$
2. $B \longrightarrow S : M_2 = N_A, A, N_B, B$
3. $S \longrightarrow B : M_3 = \{N_B, A, K_{AB}\}_{K_{BS}}, \{N_A, B, K_{AB}\}_{K_{AS}}$

Algorithm 3: Algorithm for *INTRL* attack scenarios

```
parameters   : $n$ - a number of protocol steps
             : $nses$ - a number of sessions
             : $rstep$ - the first step of $RAP$
             : $x \in \{a, b\}$ - a parameter of assignments, where $\overline{a} = b,\ \overline{b} = a$
             : $t \in \{1, 2, 3\}$ - a case of an assignment
assignments  : dla $k \in \{2i - 1 \mid 1 \leq i \leq n\}$
t=1          : $p_k(A) := \iota(a)$, $p_k(S) := \iota(s)$, $p_k(B) := b$
             : $p_{k+1}(A) := \iota(x)$, $p_{k+1}(S) := s$, $p_{k+1}(B) := \overline{x}$
t=2          : $p_k(A) := \iota(a)$, $p_k(S) := s$, $p_k(B) := b$
             : $p_{k+1}(A) := x$, $p_{k+1}(S) := s$, $p_{k+1}(B) := \iota(\overline{x})$
t=3          : $p_k(A) := \iota(a)$, $p_k(S) := s$, $p_k(B) := b$
             : $p_{k+1}(A) := \iota(x)$, $p_{k+1}(S) := s$, $p_{k+1}(B) := \overline{x}$
variables    : $actstep$, $k$ - auxiliary variables

$i \leftarrow 1$, $j \leftarrow 1$,$k \leftarrow 1$, $actstep \leftarrow 1$
if $rstep > 1$ then
    repeat
        $(i.j.)\ x \longrightarrow y : M_j$
        $j$++
    until $j \leq rstep$
$actstep \leftarrow rstep$
repeat
    repeat
        $j \leftarrow actstep$, $k \leftarrow (j - rstep + 1)$
        repeat
            $(i.j.)\ p_i(X) \longrightarrow p_i(Y) : M_j$
            $j$++, $k$++ [*]
        until not ($k$ mod 2) $\&\&\ j < n$
        $i$++
    until $i \leq nses$
    $actstep \leftarrow (actstep + 2)$, $i \leftarrow 1$
until $actstep \leq n$
```

4. $B \longrightarrow A$: M_4 = $\{N_A, B, K_{AB}\}_{K_{AS}}$, $\{T_B, A, K_{AB}\}_{K_{BB}}$, N_C, $\{N_A\}_{K_{AB}}$
5. $A \longrightarrow B$: M_5 = $\{N_A\}_{K_{AB}}$
repeated authentication part:
6. $A \longrightarrow B$: M_6 = M_A, $\{T_B, A, K_{AB}\}_{K_{BB}}$
7. $B \longrightarrow A$: M_7 = M_B, $\{M_A\}_{K_{AB}}$
8. $A \longrightarrow B$: M_8 = $\{M_B\}_{K_{AB}}$

After applying the Algorithm 3 for the parameters: $n = 8$, $nses = 3$, $rstep = 6$, $x = a$, and $t = 1$, the following scenario is returned.

repeated authentication part:

Scenario	*Attack*
1.6. $\iota(a) \longrightarrow b$:	M_6 = m_i, $\{t_b, a, k_{ab}\}_{k_{bb}}$
1.7. $b \longrightarrow \iota(a)$:	M_7 = m_b, $\{m_i\}_{k_{ab}}$
2.6. $\iota(a) \longrightarrow b$:	M_6 = m_b, $\{t_b, a, k_{ab}\}_{k_{bb}}$

2.7. $b \longrightarrow \iota(a):$ $M_7 = m'_b,\{m_b\}_{k_{ab}}$
3.6. $\iota(a) \longrightarrow b:$ $M_6 = m'_b,\{t_b,a,k_{ab}\}_{k_{bb}}$
3.7. $b \longrightarrow \iota(a):$ $M_7 = m''_b,\{m_b\}_{k_{ab}}$
1.8. $\iota(a) \longrightarrow b:$ $M_8 = \{m_b\}_{k_{ab}}$
2.8. $\iota(a) \longrightarrow b:$ $M_8 = \{m'_b\}_{k_{ab}}$
3.8. $\iota(a) \longrightarrow b:$ $M_8 = \{m''_b\}_{k_{ab}}$

3.3. REFLECTION and MITM Attacks

The next two algorithms for scenarios of reflection and MITM attacks are based on the same pattern of shuffling the steps of the sessions involved. Typically, the known attacks of these types are composed of two sessions only, but we have extended the scenarios to an arbitrary even number of sessions.

Reflection Attacks. The reflection attack is based on using the same messages (or sometimes just parts of them) by reflection in both the sessions of each pair, which is composed of an odd session j and the even session $j+1$. So, the intruder impersonating the responder and reflecting all the messages back, makes the initiator thinking, that he is playing two symmetrical session with the same principal, while he is exchanging messages only with the intruder. Another reason for using reflection is to force the initiator of the first session to accept some non-fresh credentials in the second session.

The idea of this scenario is to begin a sequence of instructions from all the sessions of the same step by an action of the honest principal a. This way the intruder may intercept a source for the reflection action. The schema is based on the following general rule: the instructions of each sequence of odd steps are following one the other in the increasing order of the sessions, while the instructions of each sequence of even steps are put in the decreasing order. So, for example the order of instructions for 4 sessions of 3 steps looks as follows: $(1.1)(2.1)(3.1)(4.1)(4.2)(3.2)(2.2)(1.2)(1.3)(2.3)(3.3)$ $(4.3)(4.4)(3.4)(2.4)(1.4)$

We have the following assignment in all odd sessions: $A \leftarrow a$, $B \leftarrow \iota(b)$ and $S \leftarrow s$, and in the even ones: $A \leftarrow \iota(b)$, $B \leftarrow a$, and $S \leftarrow \iota(s)$. Alternatively in the odd sessions: $A \leftarrow a$, $B \leftarrow \iota$ and $S \leftarrow s$ and in the even ones: $A \leftarrow \iota$, $B \leftarrow a$ and $S \leftarrow \iota(s)$.

Notice that the intruder cannot impersonate the server in the odd sessions. Therefore, the intruder is either impersonating b if he does not possess appropriate server keys to compose a fake message, or the intruder is playing the role of himself and then he may use keys being shared with the server.

Example 3.3. Consider the protocol Shamir Rivest Adelman Three Pass [7]. After applying the above algorithm for the parameters: $n = 3$, $nses = 2$, and $z = \iota(b)$, the following scenario is returned:

Scenario	*Attack*
1.6. $a \longrightarrow \iota(b):$	$M_1 = \{msg\}_{k_{aa}}$
2.1. $i(b) \longrightarrow a:$	$M_1 = \{msg\}_{k_{aa}}$
2.2. $a \longrightarrow \iota(b):$	$M_2 = \{msg\}_{k_{aa}}$
1.2. $i(b) \longrightarrow a:$	M_2 = *text*
1.3. $a \longrightarrow \iota(b):$	$M_3 = \{text\}_{k_{aa}}$
2.3. $i(b) \longrightarrow a:$	$M_3 = \{text\}_{k_{aa}}$

Algorithm 4: Algorithm for *REFL* attack scenarios

```
parameters : $n$ - a number of protocol steps
           : $nses$ - a number of sessions
           : $z \in \{\iota(b), \iota\}$ - a parameter of assignments
assignments: dla $k \in \{2i-1 \mid 1 \leq i \leq n\}$
           : $p_k(A) := a, p_k(B) := z, p_k(S) := s$
           : $p_{k+1}(A) := z, p_{k+1}(B) := a, p_{k+1}(S) := \iota(s)$
variables  : $start, stop, direction$ - auxiliary variables
$i \leftarrow 1, j \leftarrow 1, direction \leftarrow 1, stop \leftarrow nses$
repeat
    repeat
        $(i.j.)\ p_i(X) \longrightarrow p_i(Y)$
        $i \leftarrow (i + direction)$
    until $i \leq stop$
    [*]
    $direction \leftarrow (-direction)$        %% swap the values of start and stop
    $start \leftrightarrow stop$
    $j$++, $i \leftarrow start$
until $j \leq n$
```

The assignments are a significant part of this scenario, where the initiator of any odd session plays the role of the responder in the corresponding even session. As we have already mentioned the same scenario, but with different substitutions, applies to *MITM* attack and influences the message flow.

MAN-in-the-MIDLLE Attacks. The goal of the intruder in *MITM* is to run independent sessions with the honest principals and replay messages between them. This makes the principals to believe that they are talking directly to each other when in fact the entire conversation is under control of the intruder.

In the *MITM* scenario for each pair of the defined sessions the honest initiator of the odd session is different than the honest responder of the corresponding even session. The assignment for this scenario presented in Algorithm 5 reflects this assumption. There are three possible assignments for a pair of sessions. The first one is defined by: $A \leftarrow a$, $B \leftarrow \iota(b)$ in the odd sessions and $A \leftarrow \iota(a)$, $B \leftarrow b$ in the corresponding even sessions, alternatively $A \leftarrow a$, $B \leftarrow \iota$ in the odd sessions and $A \leftarrow \iota(a)$, $B \leftarrow b$ in the even sessions. In the last case: $A \leftarrow a$, $B \leftarrow \iota(b)$ are assigned in the odd sessions and in the even ones: $A \leftarrow \iota$, $B \leftarrow b$. For all the above assignments $S \leftarrow s$.

Algorithm 5: Parameters for algorithm for *MITM* attack scenario

```
parameters  : $n$ - a number of protocol steps
            : $nses$ - a number of sessions
            : $z \in \{\iota(b),\ \iota\}$ and $w \in \{\iota(a),\ \iota\}$ and $w \neq z$ parameters of assignments
assignments : dla $k \in \{2i-1 \mid 1 \leq i \leq n\}$
            : $p_k(A) := a, p_k(B) := z, p_k(S) := s$
            : $p_{k+1}(A) := w, p_{k+1}(B) := b, p_{k+1}(S) := s$
```

Example 3.4. Consider Needham Schroeder Public Key protocol. After applying the Algorithm 5 for the parameters: $n = 3$, $nses = 2$, $z = \iota$, and $w = \iota(a)$, the following scenario is returned:

Scenario	*Attack*
1.6. $a \longrightarrow \iota$	: $M_1 = \{n_a, a\}_{pk_\iota}$
2.1. $\iota(a) \longrightarrow b$	: $M_1 = \{n_a, a\}_{pk_b}$
2.2. $b \longrightarrow \iota(a)$	: $M_2 = \{n_a, n_b\}_{pk_a}$
1.2. $\iota \longrightarrow a$	: $M_2 = \{n_a, n_b\}_{pk_a}$
1.3. $a \longrightarrow \iota$	: $M_3 = \{n_b\}_{pk_\iota}$
2.3. $\iota(a) \longrightarrow b$	: $M_3 = \{n_b\}_{pk_b}$

3.4. Combining Attacks

In this section we consider scenarios of attacks, which are composed of scenarios of attacks of our taxonomy. We start with introducing one more scenario, called Server Schema, which does not generate attacks itself, but it is useful for designing scenarios of combined attacks, where the server appears as a part of the protocol. Server Schema is forcing the server to return a valid message. In such a message exchange, the challenge is sent by the intruder impersonating an honest principal while the response is intercepted and stopped by the intruder. Server Schema is defined as follows:

Algorithm 6: Algorithm for the Server Schema (*S-Schema*)

```
parameters   : $nses$ - a number of sessions
             : $w \in \{\iota(a), \iota(b)\}$ and $z \in \{\iota(a), \iota(b)\}$ - parameters of assignments
assignments  : for $1 \leq k \leq nses$
             : $p_k(A) := w, p_k(B) := z, p_k(S) := s$
variables    : $i, j$ - a current number of session and step
$i \leftarrow 1$
repeat
    $(i.j.)$   $p_i(X) \longrightarrow s$: $M_j$   if $j$. $X \longrightarrow S$: $M_j$
    $(i.j+1.)$ $s \longrightarrow p_i(Y)$: $M_{j+1}$ if $j+1$. $S \longrightarrow Y$: $M_{j+1}$
until $i \leq nses$
```

The algorithms returning scenarios which can be combined with Server Schema contain [*] in their codes. Combining consists in replacing [*] with the following instruction:

$$\textbf{if } Y = S \textbf{ in } j.\ X \longrightarrow Y:\ M_j \textbf{ then } \textit{S-Schema}(nses, j, w, z)$$

where *S-Schema*$(nses, j, w, z)$ calls the algorithm for Server Schema for $nses, j, w$ and z.
In order to distinguish sessions of different scenarios, the superscript of the number of an instruction means the scenario number. Hence, 1.3^2 stands for the 3rd step of the 1st session of the 2nd scenario.

Sequential Composition of Scenarios. The first method of combining scenarios consists in taking their sequential composition. Below, we illustrate it at the example of the protocol CCITT X.509 (3), where a new scenario is obtained by a sequential composition of a plain session (where the intruder is passive) with MITM attack scenario. This scenario generates the attack upon CCITT X.509 (3) [15].

Example 3.5. Consider the protocol CCITT X.509 (3). The first scenario is a plain session. The second scenario is obtained after applying the algorithm for MITM attack scenario for the parameters $nses = 2$, $z = \iota$, $w = \iota(a)$. The sequential composition of the above scenarios returns the following scenario:

Scenario	*Attack*
1.1.^{1} $a \longrightarrow b$	: $M^1_{1,1} = a, \{t_a, n_a, b, x_a, \{y_a\}_{pk_b}\}_{sk_a}$
1.2.^{1} $b \longrightarrow a$	: $M^1_{1,2} = b, \{t_b, n_b, a, n_a, x_b, \{y_b\}_{pk_a}\}_{sk_b}$
1.3.^{1} $a \longrightarrow b$	: $M^1_{1,3} = a, \{n_b\}_{sk_a}$
1.1.^{2} $a \longrightarrow \iota$	: $M^2_{1,1} = a, \{t'_a, n'_a, b, x'_a, \{y'_a\}_{pk_\iota}\}_{sk_a}$
2.1.^{2} $\iota(a) \longrightarrow b$	: $M^2_{2,1} = a, \{t_a, n_a, b, x_a, \{y_a\}_{pk_b}\}_{sk_a}$
2.2.^{2} $b \longrightarrow \iota(a)$	: $M^2_{2,2} = b, \{t'_b, n'_b, a, n'_a, x'_b, \{y'_b\}_{pk_a}\}_{sk_b}$
1.2.^{2} $\iota \longrightarrow a$	: $M^2_{1,2} = \iota, \{t_\iota, n'_b, a, n'_a, x_\iota, \{y_\iota\}_{pk_a}\}_{sk_\iota}$
1.3.^{2} $a \longrightarrow \iota$	: $M^2_{1,3} = a, \{n'_b\}_{sk_a}$
2.3.^{2} $\iota(a) \longrightarrow b$	: $M^2_{2,3} = a, \{n'_b\}_{sk_a}$

The above scenario generates the attack, which is equivalent to the attack of [15], where only the instruction (1.1^2) is executed two steps later, i.e., between the instructions (2.2^2) and (1.2^2). Notice, however, that this instruction does not share any of the components of the preceding instructions (2.1^2) and (2.2^2), so executing this step before the above two steps (2.1^2) and (2.2^2) has no side effects and does not change the attack.

Interleaved Composition of Scenarios. The second method of combining scenarios consists in taking their interleaved composition. Below, we illustrate it at the example of BAN simplified version of Yahalom protocol [1], where a new scenario is obtained by an interleaved composition of the interleaving attack scenario with the Server schema. The algorithm is initialised with the number of exchanges with the server and an assignment for each such session.

Example 3.6. Consider BAN simplified Yahalom protocol.

1. $A \longrightarrow B$:	M_1 = A,N_A
2. $B \longrightarrow S$:	M_2 = B,N_B,$\{A, N_A\}_{K_{BS}}$
3. $S \longrightarrow A$:	M_3 = N_B,$\{B, K_{AB}, N_A\}_{K_{AS}}$,$\{A, K_{AB}, N_B\}_{K_{BS}}$
4. $A \longrightarrow B$:	M_4 = $\{A, K_{AB}, N_B\}_{K_{BS}}$,$\{N_B\}_{K_{AB}}$

After applying the algorithm for interleaving attack scenario for the parameters $nses = 2$, $rstep = 1$, $t = 2$, $x = b$ and the Server Schema (called from [*]) for the parameters $nses = 1$, j, $w = \iota(a)$, and $z = \iota(b)$ the following scenario is returned:

Scenario	*Attack*
1.1.^{1} $a \longrightarrow \iota(b)$	: $M^1_{1,1}$ = a,n_a
1.2.^{1} $\iota(b) \longrightarrow \iota(s)$	: $M^1_{1,2}$ *(* omitted *)*
2.1.^{1} $\iota(b) \longrightarrow a$	: $M^1_{2,1}$ = b,n_a
2.2.^{1} $a \longrightarrow \iota(s)$	: $M^1_{2,2}$ = a,n'_a,$\{b, n_a\}_{k_{as}}$
1.2.^{2} $\iota(a) \longrightarrow s$	: $M^2_{1,2}$ = a,n_a,$\{b, n_a\}_{k_{as}}$
1.3.^{2} $s \longrightarrow \iota(b)$	: $M^2_{1,3}$ = n_a,$\{a, k_{ab}, n_a\}_{k_{bs}}$,$\{b, k_{ab}, n_a\}_{k_{as}}$
1.3.^{1} $\iota(s) \longrightarrow a$	: $M^1_{1,3}$ = n_ι, $\{b, k_{ab}, n_a\}_{k_{as}}$,$\{a, k_{ab}, n_a\}_{k_{bs}}$,
1.4.^{1} $a \longrightarrow \iota(b)$	: $M^1_{1,4}$ = $\{a, k_{ab}, n_a\}_{k_{bs}}$,$\{n_\iota\}_{k_{ab}}$

The above scenario generates the attack, which is equivalent to the attack of [15] as only the *omitted* step (1.2^1) takes place a few steps earlier.

3.5. Completeness of our Algorithms

We show completeness of our algorithms with respect to the following set of protocols $PROT$={*Andrew Secure RPC, BAN Andrew Secure RPC, Neumann Stubblebine, Denning Saco, SPLICE/AS SPLICE/AS CJ modified Hwang Chen, WMF (2), Kerberos v.4, EKE, Shamir Rivest Adleman Three Pass, Woo Lam Mutual Authentication (2), KSL (2), Needham Schroeder PK, TMN, SHARE, LPD: Low-Powered Devices, IKEv2: Internet Key Exchange, ISO1-PK-1U, ISO-CCF-1U, PBK fixed version*}

Let $Algorithm_X$ be the algorithm for X attack scenarios, where $X \in${*DoS-REPL, Simple-REPL, REFL, INTRL, MITM*}.

Theorem 3.1. For each attack of type X on a protocol $y \in PROT$, $Algorithm_X(y)$ returns a scenario of this attack.

Proof. See Appendix.

There are two groups of protocols/attacks that have not been classified within our taxonomy. The first group consists of the following external attacks: Neumann Stubblebine ([1].3), KSL([1].1), SPLICE/AS ([1].3). In these attacks the intruder is forcing the receiver to answer a message sent by him, but exploiting only a part of the protocol, possibly to get some updated data to use them later in another session Other attacks that are outside of the taxonomy include: KSL ([1].3), Kao Chow, Denning-Sacco ([1].1). The first one breaks the rule of not changing the order of performing steps of the protocol, while the rest assumes that the encrypted key can be guessed.

It is important to mention that since this paper is about external attacks, as only for these we can get a reduction in the number of runs to be considered, all the internal (including type flaw) attacks are outside of our taxonomy.

3.6. Analytical Results

Notice that our algorithms return scenarios of attacks without specifying contents of the messages sent by the intruder, but can specify repetition of messages sent earlier. Clearly, the messages sent by honest participants are fully defined by each step of the protocol and the former messages. This still leaves some room for a verifier to try to find messages for the intruder which fit to the given scenario. However, exploration guided by our scenarios can dramatically reduce the number of runs examined. Below, we estimate the ratio of reduction under the assumption that exploration is executed for TWO honest participants, the intruder, and possibly a server, who are running k parallel sessions of length n each. To simplify the estimation we do not take into account the number of messages which can be sent by the intruder. Let's start with computing the number of the runs $R(k,n)$, which are generated by all the interleavings of the above k sessions. We assume that for each session an assignment of the roles is given.

$$R(k,n) = \binom{kn}{n}\binom{(k-1)n}{n}\cdots\binom{2n}{n} >> (k!)^n$$

It follows from the above that the number of runs is growing so fast that even for protocols of length 5 exploring runs generated by 5 sessions is a very difficult task. Notice that our algorithms generate the number of scenarios (runs) of complexity $O(k \times n)$, which gives a spectacular reduction. For attacks that are interleaved compositions of two pure 2-session attacks of the taxonomy, the number of runs is reduced from $R(4,n)$ to $R(2,2n)$, i.e., from $(4!)^n = 2^{3n}3^n$ to 2^{2n}, which gives an exponential reduction.

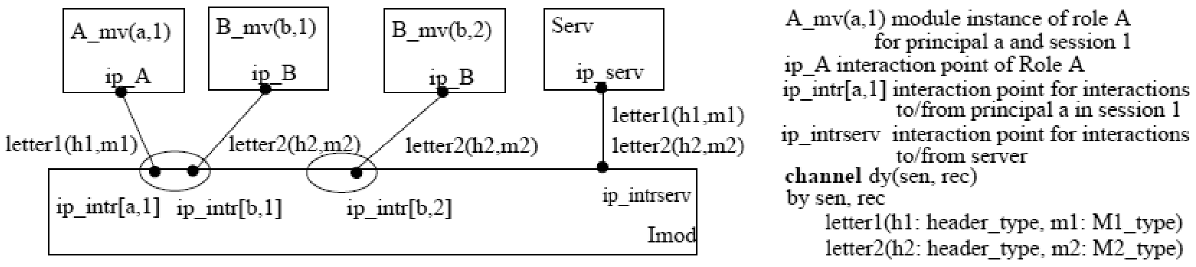
A_mv(a,1)
ip_A
B_mv(b,1)
ip_B
B_mv(b,2)
ip_B
Serv
ip_serv
letter1(h1,m1)
letter2(h2,m2)
letter2(h2,m2)
letter1(h1,m1)
letter2(h2,m2)
ip_intr[a,1]
ip_intr[b,1]
ip_intr[b,2]
ip_intrserv
Imod
A_mv(a,1) module instance of role A for principal a and session 1
ip_A interaction point of Role A
ip_intr[a,1] interaction point for interactions to/from principal a in session 1
ip_intrserv interaction point for interactions to/from server
channel dy(sen, rec)
by sen, rec
letter1(h1: header_type, m1: M1_type)
letter2(h2: header_type, m2: M2_type)

Figure 1. The communication model for WMF and channel `dy`

4. Security Protocols and Scenarios in Estelle

An Estelle specification of a protocol consists of components, called modules, or actually, *module instances*, as there may be more then one copy of each module defined. Each module has a number of interaction points, associated with a channel(s). A channel defines classes of interactions (messages) which are sent through associated interaction points. To specify which modules are able to exchange interactions, communication links (*connect*) between modules interaction points are then specified. An interaction received by a module instance at its interaction point is appended to an unbounded FIFO queue, associated with this interaction point. The *communication* in the model is *asynchronous* in that a module (process) may always send a message or, in other words, operations of sending and receiving messages are *not synchronised*. The *internal dynamic behaviour* of the module is characterised in terms of non-deterministic extended state transition model by defining a set of states, a subset of initial states (specified by an initialisation part of the module definition), and a set of transitions enclosed in a transition part of the module. A *transition* is defined by a guard and an (atomic) action to be performed when it is executed (fired). A *guard* is a boolean condition over the space of the module instance internal states and the contents of its communication queues storing incoming messages. It specifies when the transition is enabled. In each of the computation steps, the model performs one of the transitions whose guards evaluate to true.

4.1. Overview of the Cryptographic Protocol Model

The following general translation from Common Syntax (CS) to Estelle is proposed. Both the roles (initiator and responder) are implemented as Role modules `A_ROLE` and `B_ROLE`. A module for the server is implemented additionally. Each Role is parametrised with a principal identifier, the number of a session, and a set of initial parameters which are actually the initial knowledge of this principal. Module instances of the roles communicate by exchanging interactions through interaction points defined inside the modules and linked by the `channel` dy. The interactions are defined as *letters* of the protocol. The channel dy declares a bidirectional transfer of all types of messages of the protocol modelled, together with their headers. Fig. 1 presents the communication model for WMF[6].

A letter is described as an ordered pair composed of a message and a header. Each header contains an information about the sender and the receiver of the message as well as the number of a session of

[6]1. $A \longrightarrow S$: $M_1 = A, \{T_A, B, K_{AB}\}_{K_{AS}}$, 2. $S \longrightarrow B$: $M_2 = \{T_S, A, K_{AB}\}_{K_{BS}}$.

the protocol. By a *component's type* in our specification we understand a sequence of the types of its atoms encrypted with a key, or just one primitive cryptographic type. In order to easily define letters, new record types for components are introduced. This way n types of messages are defined as records of components, where n is the number of the steps of a protocol. The grammar used for composing components is presented in Section 2.

Assuming that the environment is not reliable, the intruder is placed in the network and transfers the letters between the principals, so that he can control the flow and the contents of the letters. To fulfil all the above assumptions, Principal and Server modules are *connected* through Intruder module. The idea of the communication is presented in Fig. 1.

First, a Principal p sends a letter of the session i to another Principal through a communication link established between him and the Intruder[7]. The letter is received by the Intruder at its interaction point (`ip_intr[`p`,i]`). Then, the Intruder is expected to pass this (or modified) letter through the communication link established between him and Principal declared as the receiver of the letter. Finally the letter is appended to a queue associated with the receiver interaction point (`ip_A` or `ip_B`, depending on the role played).

Intruder's Knowledge and Scenarios. The Intruder keeps all the intercepted components, according to their types, in his knowledge (IK) represented as set of arrays of records. Each array contains one of the defined types of components, and for each entry some information about its origin is written, which is the number of a session, the number of a message, and the position inside the message. This allows the Intruder for an easy access to the desired type of a component in order to compose any type of a message consistent with the next step of the scenario.

We do not implement actions of the intruder since they are provided through the translation of the algorithm to Estelle specification of the scenario. On the other hand, a procedure making a copy of the message received and decomposing it according to the component types is implemented. The detailed description of the functions for modifying IK is available in [12]. The set of states of the Intruder is composed of the states, generated by the scenario and of these added when the templates of composing the messages are included.

4.2. Experimental Results

In our experiments we are focused on checking authentication and secrecy properties exploiting the correspondence property[8], which is defined as follows: *If a principal x has finished N sessions with a principal y in a protocol run, then the principal x must have started at least N sessions with the principal y.* When the above relation is symmetric we capture the *mutual entity authentication*.

In order to check whether a protocol is vulnerable to an attack tested, the continuous simulation is performed. The idea is to simulate these runs only, which are specified by the scenario. To this aim we have used the Estelle Simulator environment. The honest participants are directly modelled in Estelle along with the intruder and the appropriate communication channels. The attack scenarios are built into the intruder so that it controls the execution of every session. In order to detect attacks and breaches of security we use the observers checking the required properties at certain states of the protocol execution.

[7]Notice that when we say Principal we actually mean the Role instance of the Principal instance (i.e, the principal in a particular session).

[8]The way of defining secrecy in terms of correspondence is presented in [17, 13].

A random seed value is driving the non-deterministic choices during the simulation, which is performed a specified number of times. Moreover, the simulation is performed regardless on whether messages sent by the intruder are accepted or not. If a message composed by the intruder in some session is not correct, then this session is terminated while the remaining sessions are continued. Our experimental results for two protocols are shown in Table 1. In order to confirm a high potential of our approach we detect the attacks described before, i.e., DoS-REPL for WMF and MITM for NSPK (see Example 3.4) for multiple sessions (i.e., for 2,3, and 4 pairs of sessions) of these protocols.

Table 1. Experimental results

Protocol	No of sessions	Nr of simulations	Nr of attacks	Execution time
WMF	4	100	200	10 s
WMF	6	100	300	15 s
WMF	8	100	400	20 s
NSPK	4	600	567	35 s
NSPK	6	600	842	47 s
NSPK	8	600	1016	81 s

One can observe that for WMF, every execution of the scenario leads to an attack. This is quite obvious, as for DoS-REPL scenario simulated for the protocol with only two steps, where $repl = 2$, there is no possibility for the intruder to compose any new message, and only the replay of the last message is performed. For NSPK this is not the case. While the choice of protocol steps is deterministic, there can be many ways in which the intruder composes messages. Some of these choices may lead to either honest or invalid executions. The number of choices grows with each protocol step executed, when new elements are added to Intruder's knowledge. Notices that while NSPK contains three steps, the scenario for MITM allows for non-determinism in composing a message at each of its steps. This explains why we get less successful attacks for NSPK. One can expect that for longer protocols, experimental results could be a bit worse, but this is still a very promising way for finding the known types of attacks upon new protocols. It is also worth mentioning here that new protocols are becoming more complicated these days as they are composed of more than five steps and involve groups of users. The standard model checking methods seem to useless for finding attacks for such protocols. Notice for example that for NSPK, SATMC [2] gives results for 6 sessions, while the other AVISPA tools cannot deal with more than 4 sessions.

As soon as we have implemented an automated translator to Estelle specifications of security protocols, we plan to simulate all the known protocols, looking for attacks involving multiple session runs.

5. Conclusions

In this paper we considered a taxonomy of attacks on security protocols in terms of a message origin and destination in addition to a session and step number. This taxonomy has been used for designing algorithms, which for a given type of attack and parameters of the protocol return a scenario of an attack of this type. This way we can more than exponentially reduce the number of protocol runs to be generated in order to look for an attack upon a protocol in the process of validation using simulation. The experimental results of simulation in Estelle prove the efficiency of our approach.

References

[1] Security protocols open repository. http://www.lsv.ens-cachan.fr//spore, 2003.

[2] A. Armando and L. Compagna. Sat-based model-checking for security protocols analysis. *Int. J. Inf. Secur.*, 7(1):3–32, 2008.

[3] S. Budkowski. Estelle development toolset (edt). *Comput. Netw. ISDN Syst.*, 25(1):63–82, 1992.

[4] S. Budkowski, A. B. Alkhechi, M.-L. Benalycherif, P. Dembinski, M. Gardie, E. Lallet, J. P. Mouchel La Fosse, and Y. Souissi. Formal specification, validation and performance evaluation of the xpress transfer protocol. In *Proceedings of the IFIP TC6/WG6.1 International Symposium on Protocol Specification, Testing and Verification XIII*, pages 191–206, Amsterdam, The Netherlands, 1993. North-Holland Publishing Co.

[5] S. Budkowski and P. Dembinski. An introduction to estelle: A specification language for distributed systems. *Computer Networks*, 14:3–23, 1987.

[6] W. Chocianowicz, J. Pejaś, and A. Ruciński. The proposal of protocol for electronic signature creation in public environment. *Enhanced methods in computer security, biometric and artificial intelligence systems*, pages 113–124, 2005.

[7] J. A. Clark and J. L. Jacob. A survey of authentication protocol literature. Technical Report 1.0, 1997.

[8] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *Journal of Computer Security*, 14(1):1–43, 2006.

[9] P. Dembinski and G. Jakubowska. Modelling and validating of security protocols. In *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'07)*, pages 100–115. Warsaw University, 2007.

[10] L. Hohwiller and S. Wendling. Fieldbus network simulation using a time extended estelle formalism. In *MASCOTS '00: Proc. of the 8th Intern. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, page 92, Washington, DC, USA, 2000. IEEE Computer Society.

[11] K. G. Indiradevi and V. S. S. Nair. Simulation based validation of authentication protocols. *J. Integr. Des. Process Sci.*, 8(4):79–98, 2004.

[12] G. Jakubowska and W. Penczek. Is your security protocol on time ? In Farhad Arbab and Marjan Sirjani, editors, *Proc. of Int. Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007*, volume 4767 of *LNCS*, pages 65–80. Springer, 2007.

[13] G. Jakubowska, W. Penczek, and M. Srebrny. Verifying security protocols with timestamps via translation to timed automata. In *Proc. of the International Workshop on Concurrency, Specification and Programming (CS&P'05)*, pages 100–115. Warsaw University, 2005.

[14] C. Jard and T. Jron. Verification and distributed observation of the alternating bit protocol. In *FORTE/PSTV'98, ECASP: Special Session on Educational Case Studies in Protocols*, Paris, France, 1998.

[15] G. Lowe. A family of attacks upon authentication protocols. Technical Report 1997/5, Department of Mathematics and Computer Science, University of Leicester, 1997.

[16] M. Olszewski and L. Cyra. An integrated framework for security protocol analysis. In *ASIACCS '08: Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 77–86, New York, NY, USA, 2008. ACM.

[17] M. Panti, L. Spalazzi, and S. Tacconi. Using the NuSMV model checker to verify the kerberos protocol. In *Simulation Series*, volume 34, pages 230–236. Society for Computer Simulation, 2002.

[18] N. Su, R.N. Zobel, and F.O. Iwu. Simulation in cryptographic protocol design and analysis. In *Proc. of the 15th European Simulation Symposium (ESS'03)*, lll, page 123. v, 2003.

[19] Paul F. Syverson. A taxonomy of replay attacks. In *Proc. of the 7th IEEE Computer Security Foundations Workshop (CSFW'94)*, pages 187–191. IEEE Computer Society, 1994.

APPENDIX

Protocol	**MITM**	**REPL**	**REFL**	**INTRL**
Andrew Secure	—	S(CJ)	—	—
BAN Andrew Secure	—	—	S	—
Neumann Stubblebine	—	—	—	S(CJ)
Denning Saco	—	S	—	—
CCITT X.509(1)	S	—	—	—
CCITT X.509(3)	*L	—	—	—
SPLICE/AS	—	S	—	—
Hwang Chen SPLICE/AS CJ	—	S(CJ)	—	—
BAN Yahalom	—	S	—	*S
WMF	—	S(CJ)	—	*S(CJ)
Kerberos v.4	—	LP	—	—
EKE	—	—	CJ	—
SRA Three Pass	—	—	CJ	—
Woo Lam MA	—	—	S(CJ)	S
KSL	—	—	—	S/S
NSPK	S(CJ)	—	—	—
TMN	—	—	—	S
SHARE	A	—	—	—
LPD	A	—	—	—
IKE v.2	A	—	—	—
ISO1-PK-1U	—	CJ	—	—
ISO-CCF-1U	—	CJ	—	—
PBK Fixed	—	A	—	—

Table 2. The protocols classified with respect to 4 types of attacks

S-Spore Repository, CJ-Clark Jacobs Library, A-AVISPA library, L-[15], P-[17], (*) combined attack

Proof of Theorem 4.1

Proof of Theorem Proof. By comparing the scenarios generated by our algorithms to the corresponding attacks. A careful analysis of the attacks shows that there are only three attacks that cannot be obtained directly from the corresponding scenarios. The first such an attack is upon the protocol TMN ([1].3) and it is given as the following sequence of instructions:

1.1. $\iota(a) \longrightarrow s$: $M_1 = b,\{pk_i\}_{pk_s}$
1.2. $s \longrightarrow b$: $M_2 = a$
2.3. $b \longrightarrow s$: $M_3 = a,\{pk_b\}_{pk_s}$
1.4. $s \longrightarrow \iota(a)$: $M_4 = b,\{pk_b\}_{pk_s}$
2.1. $a \longrightarrow s$: $M_1 = b,\{pk_a\}_{pk_s}$
2.2. $s \longrightarrow \iota(b)$: $M_2 = a$
2.3. $\iota(b) \longrightarrow s$: $M_3 = a,\{pk_b\}_{pk_s}$
2.4. $s \longrightarrow \iota(a)$: $M_4 = b,\{pk_b\}_{pk_s}$

The scenario returned by the algorithm for interleaving attack scenario of TMN for the parameters $nses = 2$, $x = 2$, and $t = 2$ is the following sequence of steps: (1.1) (1.2) (2.1) (2.2) (1.3) (1.4) (2.3) (2.4), where in (i,j): i is a session number and j a step number. It is easy to check that the scenario gives the same attack but using a different run.

The two scenarios received for Woo Lam Mutual Authentication protocol use different runs than the attacks in [1]. As this protocol is composed of seven steps, we only describe the difference between the runs of our scenarios for both the attacks.

The first scenario is received after applying the algorithm for Reflection Attack Scenario for the parameters $z = \iota$, $nses = 2$, and looks as follows: (1.1) (2.1) (2.2) (1.2) (1.3) (2.3) (2.4) (1.4) (1.5) (2.5) (2.6) (1.6) (1.7) (2.7). The sequence of instructions of the attack [1].1 is the following: (1.1) (2.1) (2.2) (1.2) (1.3) (1.4) (1.5) (1.6) (1.7) (2.3) (2.4) (2.5) (2.6) (2.7). The difference is at four positions. Notice that the instruction (2.3) of the attack depends on (1.1) and (1.2) only while (2.4) is dependent on (2.3) only. So, both the instructions (2.3) and (2.4) should be executed at least after (1.2), which is satisfied in the scenario. The next two instructions (2.5) and (2.6) must be executed after the step (1.5), and the scenario fulfils this condition.

The last scenario to be discussed is for the attack presented in [1].2. After applying the algorithm for Interleaving attack scenario for the parameter $nses = 2$ and $t = 3$, the following scenario is received: (1.1) (1.2) (2.1) (2.2) (1.3) (1.4) (2.3) (2.4) (1.5) (1.6) (2.5) (2.6) (1.7) (2.7). The attack is the sequence of the following instructions: (1.1) (1.2) (1.3) (1.4) (2.1) (2.2) (2.3) (2.4) (1.5) (1.6) (1.7). The difference is at the positions of the instructions (2.1) and (2.2). Notice that the instruction (2.1) depends on the instruction (1.2) only, and (2.2) should be performed after (2.1), which is satisfied in the scenario. **End of proof.**