Knowledge-Based Automated Repair of Authentication Protocols

Borzoo Bonakdarpour[1], Reza Hajisheykhi[2], and Sandeep S. Kulkarni[2]

[1] School of Computer Science
University of Waterloo, Canada
borzoo@cs.uwaterloo.ca

[2] Department of Computer Science and Engineering
Michigan State University, USA
{hajishey,sandeep}@cse.msu.edu

Abstract. In this paper, we introduce a technique for repairing bugs in authentication protocols automatically. Although such bugs can be identified through sophisticated testing or verification methods, the state of the art falls short in fixing bugs in security protocols in an automated fashion. Our method takes as input a protocol and a logical property that the protocol does not satisfy and generates as output another protocol that satisfies the property. We require that the generated protocol must refine the original protocol in cases where the bug is not observed; i.e., repairing a protocol should not change the existing healthy behavior of the protocol. We use *epistemic logic* to specify and reason about authentication properties in protocols. We demonstrate the application of our method in repairing the 3-step Needham-Schroeder's protocol. To our knowledge, this is the first application of epistemic logic in automated repair of security protocols.

1 Introduction

Automated *model repair* aims at eliminating the human factor in fixing bugs. More specifically, model repair begins with a model M and properties Σ and Π, such that M satisfies Σ but does not satisfy Π (e.g., identified by model checking). The goal is to repair M automatically and obtain a model M', such that M' satisfies both Σ and Π. In other words, model repair *adds* property Π to the original model while preserving the existing property Σ.

In this paper, we focus on developing an automated technique that deals with repairing authentication protocols. The problem of model repair in the context of security protocols creates new challenges that are not present when repair is performed to add safety, liveness or fault-tolerance properties. Specifically, the problem of adding other properties can be expressed in terms of states reached by the program, e.g., safety can be expressed in terms of states (respectively, transitions or computation prefixes) that should not be reached. On the contrary, a security property requires analysis of the knowledge of different agents in different states. Moreover, this knowledge depends upon *inference rules* (e.g., if an agent knows a message $k(m)$ and it knows the key k, then it knows message m). Even when one finds that a security violation has occurred based on the knowledge of agents, fixing the protocol creates new challenges that are not present in adding normal safety and liveness requirements. Specifically, if the state where a security property is violated is reached due to an action of the adversary, it is not possible to remove the corresponding adversary action. Moreover, even if that state was reached due to a regular agent action, the way that the action can be changed depends upon (1) the type of keys that can be used, (2) assumptions about initial distribution of keys, (3) inference rules that identify the roles of keys, and so on.

Based on this discussion, repairing a security protocol involves three steps: The first step involves identifying the state where the security violation occurs. The second step involves identifying the step that could be altered to potentially eliminate the security violation. This step is essential since all steps (e.g., actions taken by adversary) are not fixable. This step also involves identifying the *corresponding adversary-free* states and identifying the *knowledge-difference* between states reached in the presence of the adversary and states reached in the absence of the adversary. Finally, the third step involves utilizing this knowledge-difference to repair the protocol. This step depends upon the types of changes one can do including the use of new nonces, existing or new keys, types of messages that may be permitted, etc.

Our contribution in this paper is twofold. We introduce a novel *epistemic* [10] algorithm that repairs a given authentication protocol, so that it satisfies the authentication requirement in the presence of a newly identified threat. Moreover, the algorithm preserves the behavior of the protocol in the absence and presence of already known attacks (i.e., the repair algorithm does not damage the existing sound features of the protocol). Our approach for repairing security protocol is as follows. We assume that the repeated application of inference rules is terminating, as without this assumption, even the verification problem could be undecidable. This can be achieved by bounding the structure of messages used in the protocol (e.g., number of fields, depth of encryption, etc) and requiring all legitimate participants to reject messages that violate this structure. Under this assumption, our approach is sound and complete for the first step; i.e., if the security property is violated, then it would be detected. For the second step, our approach is sound and (intentionally) incomplete. Specifically, we identify potential steps where the security protocol can be repaired. However, to identify the *knowledge-difference*, for the sake of efficiency, we only focus on atomic knowledge propositions. This step can be made sound and complete at the increased computational cost. Finally, the third step is a sound and incomplete heuristic, as the choices made in the repairing the protocol (e.g., whether new nonces can be used, what types of keys can be used, etc.) depend upon external factors such as efficiency, user-preference that cannot be modeled during repair. However, our algorithm still preserves soundness during this step, by ensuring that the soundness only depends upon the inference rules rather than the heuristics used in this step. We also demonstrate the application of our method in repairing the bug the 3-step Needham-Schroeder public-key protocol.

Organization. In Section 2, we present the preliminary concepts on epistemic logic. Section 3 describes our high-level computation model. The formal statement of knowledge-based repair problem is presented in Section 4. Section 5 describes our repair algorithm, while Section 6 presents the application of the algorithm to repair the Needham-Schroeder's protocol. Related work is discussed in Section 7. We conclude in Section 8.

2 Preliminaries [10]

2.1 The Notion of Knowledge

Let Φ be a nonempty finite set of *atomic propositions*, typically labeled p, p', q, q', ... Also, let $1, 2, \ldots, n$ be the names of a nonempty finite set of *agents*. We define the syntax and semantics of our epistemic language as follows.

Definition 1. *Epistemic formulas are defined inductively as follows:*

$$\varphi ::= \mathit{true} \;\mid\; p \;\mid\; \neg\varphi \;\mid\; \varphi_1 \wedge \varphi_2 \;\mid\; K_i\varphi$$

where $p \in \Phi$, $i \in \{1, \ldots, n\}$, and K_i is the modal operator read as 'agent i knows*'.* □

We formalize the semantics of our epistemic language in terms of *Kripke structures*. A Kripke structure M for n agents over atomic propositions Φ is a tuple $(S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$, where S is a nonempty set of *states*, π is an *interpretation* which associates with each state in S a truth assignment to the atomic propositions in Φ (i.e., $\pi(s) : \Phi \to \{\mathit{true}, \mathit{false}\}$ for each state $s \in S$), and $\mathcal{K}_i$ is a binary equivalence relation on S, that is, a set of pairs of elements of S. Intuitively, we think of $\mathcal{K}_i$ as a *possibility* relation; i.e., it defines what states agent i considers possible at any given state. For example, Figure 1 [10] shows a Kripke structure $M = (S, \pi, \mathcal{K}_1, \mathcal{K}_2)$ over $\Phi = \{p\}$ with states $S = \{s, t, u\}$. Proposition p holds in states s and u and it does not hold in state t. We now define the notion of $(M, s) \models \varphi$, which is read as '(M, s) *satisfies* φ'.

Definition 2. *Let $M = (S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$ be a Kripke structure over atomic propositions Φ, $s \in S$, and $p \in \Phi$. Semantics of our logic is defined inductively as follows:*

$$\begin{array}{lll}
(M,s) \models \mathit{true} & & \\
(M,s) \models p & \mathit{iff} & \pi(s)(p) = \mathit{true} \\
(M,s) \models \neg\varphi & \mathit{iff} & (M,s) \not\models \varphi \\
(M,s) \models \varphi \wedge \psi & \mathit{iff} & (M,s) \models \varphi \;\wedge\; (M,s) \models \psi \\
(M,s) \models K_i\varphi & \mathit{iff} & (M,t) \models \varphi \text{ for all } t \text{, such that} \\
 & & (s,t) \in \mathcal{K}_i \text{, where } 1 \leq i \leq n.
\end{array}$$

In addition, $M \models \varphi$ holds iff $(M, s) \models \varphi$ holds for every state $s \in S$. □

For example, for the Kripke structure in Figure 1, we have $(M, s) \models K_2 p$ (i.e., in state s agent 2 knows p). Also, we have $(M, s) \models \neg K_2 \neg K_1 p$.

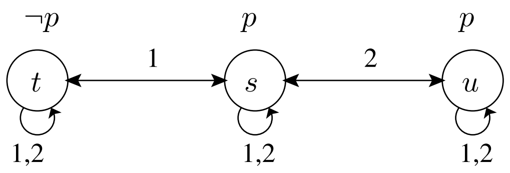

Fig. 1. A Kripke structure

2.2 Knowledge in Multi-agent Systems

In order to reason about the knowledge of agents, we leverage the notions of local state and global state of agents. Let L_i be a set of possible *local* states for agent i, for $i = 1, \ldots, n$. We take $\mathcal{G} = L_1 \times \cdots \times L_n$ to be the set of *global* states. A *run* is a function from the nonnegative integers $\mathbb{Z}_{\geq 0}$ (called *time*) to $\mathcal{G}$. Thus, a run r is a sequence of global states in $\mathcal{G}$. We refer to a pair (r, m) consisting of a run r and time m as a *point*. Notice that each $r(m)$ is of the form $(s_1, \ldots, s_n)$, where s_i, $1 \leq i \leq n$, is the local state of agent i. We say that two global states $s = (s_1, \ldots, s_n)$ and $s' = (s'_1, \ldots, s'_n)$ are *indistinguishable to agent* i, and write $s \sim_i s'$, iff i has the same state in both s and s', that is, if $s_i = s'_i$. Likewise, two points (r, m) and (r', m') are indistinguishable for agent i if $r(m) \sim_i r'(m')$ (or, equivalently, if $r_i(m) = r'_i(m')$). Clearly, $\sim_i$ is an equivalence relation on points.

Definition 3. *A* system $\mathcal{R}$ *(with global states $\mathcal{G}$) is a nonempty set of runs over a set $\mathcal{G}$ of global states.* □

We say that (r, m) is a *point in system* $\mathcal{R}$, if $r \in \mathcal{R}$. In order to connect the notion of systems to knowledge, we reason about atomic propositions in each state of the system.

Definition 4. *An* interpreted system $\mathcal{N}$ *is a pair $(\mathcal{R}, \pi)$, where $\mathcal{R}$ (with global states $\mathcal{G}$) is a system over global states $\mathcal{G}$ and π is a function from $\mathcal{G}$ to 2^{Φ}.* □

To define knowledge in interpreted systems, we associate with an interpreted system $\mathcal{N} = (\mathcal{R}, \pi)$ a Kripke structure $M_{\mathcal{N}} = (S, \pi, \mathcal{K}_1, \ldots, \mathcal{K}_n)$ as follows:

- S consists of the points in $\mathcal{N}$, and
- $\mathcal{K}_i$ is a relation in $M_{\mathcal{N}}$ defined by $\sim_i$.

Thus, we say that $(\mathcal{N}, r, m) \models \varphi$ exactly if $(M_{\mathcal{N}}, s) \models \varphi$, where $s = (r, m)$. I.e.,

$(\mathcal{N}, r, m) \models p$ (for $p \in \Phi$) iff $\pi(r, m)(p) = \mathit{true}$, and
$(\mathcal{N}, r, m) \models K_i\varphi$ iff $(\mathcal{N}, r', m') \models \varphi$ for all (r', m') such that $(r, m) \sim_i (r', m')$.

An interpreted system $\mathcal{N}$ satisfies an epistemic formula φ iff $(\mathcal{N}, r, m) \models \varphi$, for all points (r, m).

Finally, we introduce the 'always' temporal operator $\Box$. Syntactically, if φ is an epistemic formula (see Definition 1), then $\Box\varphi$ is also an epistemic formula. The semantics of the this operator is the following:

$$(\mathcal{N}, r, m) \models \Box\varphi \text{ iff } (\mathcal{N}, r, m') \models \varphi, \text{ for all } m' \geq m.$$

3 High-Level System Representation

To concisely represent a system, we use *guarded commands* (also called *actions*). Each action is of the form $L :: \; g \;\; \longrightarrow \;\; st_1; st_2; \ldots; st_k;$, , where L is a label, g is a guard, that is, a Boolean expression over a set of atomic propositions, and $st_1, st_2, \ldots, st_k$ are sequentially executed statements that prescribe how the state of agents of a system change. Given a set of actions, one can trivially obtain a system as defined in Definition 3 (i.e., a set of runs).

Since our focus in this paper is on message passing protocols, we utilize a special send(message) statement for simplicity of presentation. A *message* is of the form $S_p.R_p.S_l.R_l.msg$, where S_p is a physical sender (e.g., an IP address), R_p is a physical receiver, S_l is a logical sender (e.g., host name), R_l is a logical receiver, and msg is the message content. For example, $I_p.B_p.A.B.$"hello" means that the message "hello" is intended to be sent by agent I to agent B. However, I wants to impersonate A by choosing logical sender A.

We now describe the semantics of send. Let A, B, C, and D be agents of a system with the following Boolean variables $sent_z(x.y.msg)$ and $rcvd_z(x.y.msg)$, where $x \neq y$, $z, x, y \in \{A, B, C, D\}$, and msg is the message content. Execution of statement send$(C.D.A.B.msg)$ affects the variables of agents as follows:

(1) This message is sent by physical sender C and it is sent to physical receiver D. However, it appears to have been sent from A to B. If D is not an intruder, we expect that $D = B$. Otherwise, D will discard this invalid message. However, if D is an intruder, it might accept this message since it is part of its attack routine. (2) Actual sending and receiving of a message occurs simultaneously. (3) The value of an *auxiliary* variable $rcvdfrom_D$ is set to the physical address of the sender, that is, $rcvdfrom_D = C$. This variable is only used to describe the protocol since we need an action of the form 'reply to the (physical) sender of this message'. We emphasize that this variable does not participate in state evaluation of an agent. (4) The value of variables $sent_C(A.B.msg)$ and $rcvd_D(A.B.msg)$ are set to true.

For example, consider the following actions:

$$\begin{array}{lll} L^A :: \; true & \longrightarrow & \mathsf{send}(A_p.B_p.A.B.\text{"hello"}); \\ L^B :: \; rcvd_B(A.B.\text{"hello"}) & \longrightarrow & \mathsf{send}(B_p.rcvdfrom_B.B.A.\text{"bon jour"}); \end{array}$$

Table 1 describes how the state of each agent develops in a run r.

Remark. Sending of a message send$(A_p.B_p.A.B.msg)$ sets variables $sent_A(A.B.msg)$ and $rcvd_B(A.B.msg)$ to true. It does not set $sent_B(A.B.msg)$ to true. $sent_B(A.B.msg)$ is set to true only if B concludes (based on the authentication protocol under consideration) that the message msg was indeed sent by A. Note that B will not be reading $sent_A(x.y.msg)$. However, it could conclude $K_B sent_A(A.B.msg)$ based on the inference rules.

Table 1. State and knowledge development of agents A and B

global state	local state of agent A	local state of agent B
$r(0)$	$\mathit{rcvdfrom}_A = \bot$ $\forall x : \mathit{sent}_x(\dots) = \mathit{false}$ $\forall x : \mathit{rcvd}_x(\dots) = \mathit{false}$	$\mathit{rcvdfrom}_B = \bot$ $\forall x : \mathit{sent}_x(\dots) = \mathit{false}$ $\forall x : \mathit{rcvd}_x(\dots) = \mathit{false}$
$r(1)$	$\mathit{sent}_A(A.B.$"hello"$)$	$\mathit{rcvdfrom}_B = A_p$ $\mathit{rcvd}_B(A.B.$"hello"$)$
$r(2)$	$\mathit{rcvdfrom}_A = B_p$ $\mathit{rcvd}_A(B.A.$"bon jour"$)$	$\mathit{sent}_B(B.A.$"bon jour"$)$

4 The Model Repair Problem

In this section, we formally state the repair problem in the context of authentication protocols. The intuitive description of the problem is the following. We are given an interpreted system $\mathcal{N}$ that satisfies an epistemic (authentication) property φ. However, if an *intruder* agent *intr* joins the system, the obtained system (denoted $\mathcal{N}_{+intr}$) does not satisfy φ. The system $\mathcal{N}_{+intr}$ is trivially obtained by incorporating the local states of *intr* in calculating global states of $\mathcal{N}_{+intr}$ and extending runs of $\mathcal{N}$ by the intruder's actions. Since the focus of this paper is on authentication protocols, we first define authentication in terms of epistemic formulas. Then, we discuss the problem statement for repairing a given protocol.

4.1 Authentication

Intuitively, authentication refers to the ability to conclusively decide who the sender of a given message is. This can be captured by the following epistemic formulas for any message *msg*:

$$\varphi_1 \equiv \Box K_A(\mathit{sent}_A(B.A.\mathit{msg}) \;\Rightarrow\; \mathit{sent}_B(B.A.\mathit{msg})) \tag{1}$$

$$\varphi_2 \equiv \Box K_B(\mathit{sent}_B(A.B.\mathit{msg}) \;\Rightarrow\; \mathit{sent}_A(A.B.\mathit{msg})) \tag{2}$$

4.2 Formal Problem Statement

Following the intuitive description of the problem in the beginning of this section, the *repair* problem is to obtain a system $\mathcal{N}'$, such that (1) $\mathcal{N}'$ behaves similarly to $\mathcal{N}$, and (2) $\mathcal{N}'_{+intr}$ satisfies φ, the desired authentication property such as that described by formulas φ_1 and φ_2. In order to capture the first condition, we define a *state mapping function* f from one Kripke structure to another. In particular, let $\mathcal{N}$ and $\mathcal{N}'$ be two systems (in our context, the original and repaired systems, respectively) over the set Φ of atomic propositions. Let $M_{\mathcal{N}} = (S, \pi, \mathcal{K}_1, \dots, \mathcal{K}_n)$ and $M_{\mathcal{N}'} = (S', \pi', \mathcal{K}'_1, \dots, \mathcal{K}'_n)$ be their corresponding Kripke structures, respectively. A state mapping function $f : S' \to S$ is an onto function, such that:

1. for all $s' \in S'$ and $p \in \Phi$, if $\pi(f(s'))(p) = true$, then $\pi'(s')(p) = true$
2. for all $s', r' \in S'$, if $(s', r') \in \mathcal{K}'_i$ for some i, then $(f(s'), f(r')) \in \mathcal{K}_i$.

Definition 5. *An interpreted system $\mathcal{N}'$* refines *an interpreted system $\mathcal{N}$ iff there exists a state mapping function f, such that, for each run $r' = r'(0)r'(1)\ldots$ in $\mathcal{N}'$, the run $r = f(r'(0))f(r'(1))\ldots$ belongs to $\mathcal{N}$.* □

Next, based on the above discussion, we define the problem of repairing a given protocol as follows:

Problem 1 *Given an interpreted system $\mathcal{N}$, an intruder agent intr, and an epistemic property φ, where $\mathcal{N} \models \varphi$ and $\mathcal{N}_{+intr} \not\models \varphi$, the repair problem is to obtain an interpreted system $\mathcal{N}'$ such that:*

- *$(C1)$ $\mathcal{N}'$ refines $\mathcal{N}$, and*
- *$(C2)$ $\mathcal{N}'_{+intr} \models \varphi$.*

Note that based on Constraint $C1$ and Definition 5, it follows that behaviors of $\mathcal{N}'$ in the absence of intruder correspond to behaviors in $\mathcal{N}$. Hence, if $\mathcal{N}$ does not terminate (deadlock) in some state then $\mathcal{N}'$ cannot terminate in that state either. $\mathcal{N}'$ may not have all behaviors that are included in $\mathcal{N}$; some behaviors could be removed if it is impossible to provide authentication for them in the presence of the intruder. It is straightforward to change the problem statement (and our algorithm) to require all existing behaviors be preserved by requiring the algorithm to declare failure if it is forced remove behaviors in $\mathcal{N}$.

5 A Knowledge-Based Repair Algorithm

5.1 Auxiliary Agent

We introduce an auxiliary agent, GA for each agent A. Agent GA can view the global communication and update the knowledge accordingly. To illustrate the use of GA, consider the example, where A receives a plaintext message "hello" from B. Based on the discussion in Section 3, the proposition $rcvd_A(B.A.\text{"hello"})$ is true. However, since A is not sure about whether B really sent it, $sent_A(B.A.\text{"hello"})$ is still false. On the contrary, $sent_{GA}(B.A.\text{"hello"})$ is true. Except for this difference, agents A and GA are identical. Note that agent GA is auxiliary and cannot be realized. It is only for analyzing the protocol to evaluate how it can be repaired.

To illustrate the role of agent GA, consider the following scenarios where a protocol action, say ac, is executed: (1) In the first scenario, ac is executed in an intruder-free scenario in a state s_0 and the resulting state is s_1, and (2) In another scenario, the action is executed in the presence of an intruder in state s_2

Algorithm 1. Epistemic_Repair

Input: An interpreted system $\mathcal{N}$, intruder agent $intr$, and epistemic formula $\Box\varphi$.
Output: An interpreted system $\mathcal{N}'$.

```
1: $R :=$ ReachableStates($\mathcal{N}$)
2: $T :=$ ReachableStates($\mathcal{N}_{+intr}$)
3: while ($T \wedge \neg\varphi \neq false$) do
4:     Let $\langle s_1, s_2, ..., s_k\rangle$ be a prefix of a run of $\mathcal{N}_{+intr}$, where $s_k \in T \wedge \neg\varphi$
5:     for all $j = k$ to 1 do
6:         if ($j = 1$) then
7:             declare failure to repair $\mathcal{N}$
8:         end if
9:         Let $ac$ be the high-level action responsible for execution of $(s_{j-1}, s_j)$
10:        if $ac$ is an intruder action then
11:            continue
12:        end if
13:        $X = \{s_0 \mid (s_0 \in R) \wedge (s_0, s_1)$ corresponds to the high level action $ac\}$
14:        if $(\forall s \in X : \exists Q : (\mathcal{N}, s \models K_{GA}Q \wedge \mathcal{N}_{+intr}, s_{j-1} \models \neg K_{GA}Q))$ then
15:            $fix(\mathcal{N}, intr, R, T, s_{j-1}, s_j, A, ac)$
16:        end if
17:    end for
18:    $R :=$ ReachableStates($\mathcal{N}$)
19:    $T :=$ ReachableStates($\mathcal{N}_{+intr}$)
20: end while
21: return $\mathcal{N}$
```

and the resulting state is s_3, and this eventually leads to a state where security requirement is violated.

To prevent this security violation, without violating $C1$, we want to prevent execution of Action ac in s_2 without preventing its execution in state s_0. If states s_0 and s_2 are distinguishable to agent A, this can be achieved trivially. If s_0 and s_2 are indistinguishable, then the auxiliary agent GA can assist in modifying the protocol, so that s_0 and s_2 are distinguishable.

5.2 Algorithm Description

Step 1: Locating the Authentication Violation The repair algorithm Epistemic_Repair (see Algorithm 1) first computes the set R of states reached in the absence of the intruder $intr$ and T, states reached in the presence of $intr$ (Lines 1 and 2). We assume that the security requirement is of the form $\Box\varphi$. Hence, if $T \wedge \neg\varphi$ is satisfiable, then some run of the protocol violates the security requirement in the presence of the intruder. Hence, the algorithm iterates and repairs until $T \wedge \neg\varphi$ becomes *false*. If $T \wedge \neg\varphi$ is *true*, then the algorithm finds a state, say s_k, in $T \wedge \neg\varphi$ and identifies how that state can be reached in a run of the protocol (Line 4).

Step 2: Identifying *repairable* Location The algorithm traverses this path backward to identify a location where the protocol could be repaired. In this backward traversal, let (s_{j-1}, s_j) be the current transition being considered. If this transition is caused by an intruder action, then it cannot be stopped (Lines 11) and the algorithm considers the previous transition (s_{j-2}, s_{j-1}). If (s_{j-1}, s_j) is not a transition of the intruder, then the algorithm evaluates the knowledge difference between s_{j-1} and corresponding states reached in the absence of the

Algorithm 2. fix_Function

```
Input: Interpreted system $\mathcal{N}$, intruder $intr$, set of states $R$ and $T$, states $s$, state $s'$, agent $A$, and
    action $ac$, where $ac$ is responsible for executing $(s, s')$.
Output: An interpreted system $\mathcal{N}$.
1: $X = \{s_0 \in R \mid \exists s_1 \in R : (s_0, s_1)$ is a transition of $ac\}$
2: if $X = \emptyset$ then
3:     Remove Action $ac$ from $\mathcal{N}$
4: end if
5: if $(\forall s_k \in X : \exists Q_k : (\mathcal{N}, s_k \models K_A Q_k \ \wedge \ \mathcal{N}_{+intr}, s \models \neg K_A Q_k))$ then
6:     change Action $ac$ in $\mathcal{N}$ to "if $(\bigvee Q_k)$ then $ac$"
7: end if
8: For some $B$, $m$, let $rcvd_A(B, A, \{m\}_{PK_A})$ be included in the guard of action $ac$.
9: $r = \exists B : rcvd_A(B, A, \{m\}_{PK_A})$
10: $t = \exists B : sent_B(B, A, \{m\}_{PK_A})$
11: if $(\forall s_0 \in X : (\mathcal{N}, s_0 \models K_A r \wedge \mathcal{N}_{+intr}, s \models K_A r) \wedge (\mathcal{N}, s_0 \models K_{GA} t \wedge \mathcal{N}_{+intr}, s \models \neg K_{GA} t))$
    then
12:     Replace sending of $\{m\}_{PK_A}$ in $\mathcal{N}$ by: $\{m, senderID_m\}_{PK_A}$
13:     Change action $ac$ in $\mathcal{N}$ to: "If $(rcvd_A(B, A, \{m, B\}_{PK_A}))$ then $ac$"
14: end if
15: For some $B$, $m$, let $rcvd_A(B, A, \{m\})$ be included in the guard of action $ac$.
16: $r = \exists B : rcvd_A(B, A, \{m\})$
17: $t = \exists B : sent_B(B, A, \{m\})$
18: if $(\forall s_0 \in X : (\mathcal{N}, s_0 \models K_A r \wedge \mathcal{N}_{+intr}, s \models K_A r) \wedge (\mathcal{N}, s_0 \models K_{GA} t \wedge \mathcal{N}_{+intr}, s \models \neg K_{GA} t))$
    then
19:     Replace sending of $\{m\}$ in $\mathcal{N}$ by: $\{\{m\}_{PK^{-1}_{sender_m}}\}_{PK_A}$
20:     Change action $ac$ in $\mathcal{N}$ to: "$rcvd_A(B, A, \{\{m\}_{PK_B^{-1}}\}_{PK_A})$"
21: end if
22: For some $B$, $m$, let $rcvd_A(B, A, \{m\})$ be included in the guard of action $ac$.
23: $r = \exists B : rcvd_A(B, A, \{m\})$
24: $t = \exists B : sent_B(B, A, \{m\})$
25: if $(\forall s_0 \in X : (\mathcal{N}, s_0 \models K_A r \wedge \mathcal{N}_{+intr}, s \models K_A r) \wedge$
        $\mathcal{N}, s_0 \models K_{GA} t \wedge (\mathcal{N}_{+intr}, s \models \neg K_{GA} t) \wedge (\mathcal{N}, s_0 \models K_{GA} shkey(key)))$ then
26:     Replace sending of $\{m\}$ in $\mathcal{N}$ by: $\{\{m\}_{key}\}$
27:     Change action $ac$ in $\mathcal{N}$ to: "If $rcvd_A(B, A, \{\{m\}_{key})$ then $ac$"
28: end if
29: For some $m$, $D$, $key_1$, let $rcvd_A(D, A, \{m\}_{key})$ be included in the guard of action $ac$.
30: $r_1 = \exists D : rcvd_A(D, A, \{m_1\}_{key_1})$
31: $r_2 = \exists D : rcvd_A(D, A, \{m_2\}_{key_2})$
32: $r_3 = \exists E : sent_E(E, A, \{m_1\}_{key_1}) \wedge sent_E(E, A, \{m_2\}_{key_2})$
33: if $(\forall s_0 \in X : (\mathcal{N}, s_0 \models K_A r_1) \wedge (\mathcal{N}_{+intr}, s \models K_A r_1) \wedge$
        $(\mathcal{N}, s_0 \models K_A r_2) \wedge (\mathcal{N}_{+intr}, s \models K_A r_2) \wedge (\mathcal{N}, s_0 \models K_{GA} r_3) \wedge (\mathcal{N}_{+intr}, s \models \neg K_{GA} r_3) \wedge$
        $(\mathcal{N}, s_0 \models K_{GA} shkey(key_1)) \wedge (\mathcal{N}, s_0 \models K_{GA} shkey(key_2)))$ then
34:     Replace sending of $\{m_2\}_{key_2}$ in $\mathcal{N}$ by: $\{m_2, key_1\}_{key_2}$
35:     Change action $ac$ in $\mathcal{N}$ to:
    "If $(\exists m_1, D : rcvd_A(D, A, \{m_1\}_{key_1} \wedge rcvd_A(D, A, \{m, key_1\}_{key_2})$ then $ac$"
36: end if
```

intruder as follows. It first identifies possible states s_0, where the same action is being executed (Line 13). Then, it identifies whether there exists a predicate Q, such that $K_{GA}Q$ is true in state s_0, but it is false in state s_{j-1}. For efficiency of implementation (without affecting soundness), in implementation of our case studies, we only consider atomic propositions as choices for Q. If such a predicate Q is found (Line 14), then Step 3 is invoked by calling Algorithm 2.

Step 3: Repairing the Bug Step 3 is based on heuristics to repair the given protocol so that the knowledge Q identified in Step 2 can be utilized to repair the protocol.

Removal of useless actions. Line 1 of Algorithm 2 computes the set of corresponding states, say X, reached in the absence of the intruder. If X is equal to the empty set (Lines 2-4), then action ac is never executed in the absence of the intruder and, hence, can be safely removed.

Repairing an improper implementation. If X is nonempty, but there exists a predicate Q, such that $K_A Q$ is true in all states in X and $K_A Q$ is false in state s, then we change action ac to 'if (Q), then ac' (Lines 5-7). Note that in this scenario, agent A already possesses some knowledge that would enable it to prevent violation of the security property.

Imparting knowledge of sender via public/private keys. Lines 8-14 cover an instance, where the knowledge of GA can be imparted to agent A. Here, predicate r denotes that A has received some message m encrypted by its public key. Predicate t denotes that the sender of this message (a non-intruder agent) is aware of sending this message. Furthermore, $K_A r$ is true in all states in X as well as in state s. And, $K_{GA} t$ is true in all states of X although not in state s. Here GA has the knowledge that the message received in s has not been sent by the agent who claims to have sent it. However, agent A is not aware of this. Now, the knowledge of agent GA can be imparted to A if we replace the action of sending of message m, so that the message is of the form $\{m, senderID_m\}_{PK_A}$. Moreover, A can use this knowledge if ac is changed, so that the logical sender of the message is the same as the one that is included in the message.

Likewise, the remaining actions allow GA to impart its knowledge based on public/shared keys as well as knowledge about correlation between senders of different messages.

As discussed in the Introduction, the function *fix* can include more rules. We have specified general rules that should be applied in a rich class of scenarios. An interesting observation in this case is that the correctness of the repaired protocol does not rely on the details of *fix* function. The correctness of the protocol only relies on axioms (such as those discussed in Section 6) used to update the knowledge.

Finally, after Algorithm 2 changes $\mathcal{N}$, we reevaluate R and T to ensure states in X are not reachable. Now, if there exists a state in $T \wedge \neg\varphi$, the algorithm resolves by using Algorithm 2. This process is repeated until $T \wedge \neg\varphi$ is *false*.

Theorem 1. *Algorithm* Epistemic_Repair *is sound, and the complexity of Algorithm* Epistemic_Repair *is* $O(|\mathcal{G}_{\mathcal{N}}| + O(dif + fix))$.

6 Case Study: The Needham-Schroeder Protocol

In this section, we present a case study, the well-known *Needham-Schroeder* (NS) public-key authentication protocol [14]. The protocol assumes reliable communication channels and aims to establish mutual authentication between two agents, say A and B, in a system using public-key cryptography [20]. Each agent A in the system possesses a public key PK_A, that other agents can obtain from a key server. (For simplicity, we assume that each agent initially knows the public key of all other agents.) Each agent also owns a private key ${PK_A}^{-1}$ which is the inverse of PK_A.

6.1 The Original 3-Step Protocol

Step 1 The first action of the protocol $\mathcal{R}_{ns}$ is due to agent A:

$$\boxed{\mathcal{R}_{ns}^{A_1} :: \ \mathit{fresh}_A(N_a) \ \longrightarrow \ \mathit{fresh}_A(N_a) := \mathit{false}; \ \mathsf{send}(A_p.B_p.A.B.\{N_a.A\}_{PK_B})}$$

where N_a is a random number generated by agent A (called a *nonce*). As we present the protocol actions, we also explain how the run and knowledge of each agent develops. The nonce N_a is modeled by including a proposition $\mathit{fresh}_A(N_a)$ as an atomic proposition in Definition 1. The proposition $\mathit{fresh}_A(N_a)$ holds in the initial local state of agent A and it does not hold in the initial local state of agent B. (If multiple nonces are required for A then this would be achieved by having propositions such as $\mathit{fresh}_A(N_{a_1}), \mathit{fresh}_A(N_{a_2})$, etc.) We introduce the propositions $\mathit{has}_A(\mathit{msg})$ that is true when agent A *has* message msg. In action $\mathcal{R}_{ns}^{A_1}$, agent A sends a message to agent B (encrypted by the public key of B) containing the fresh nonce and the logical name of the sender agent; i.e., $\{N_a, A\}$.

We use a set of inference rules as *axioms*, such that they can be applied only a finite number of times to present the derivation of propositions automatically. These axioms are as follows:

$$\frac{\mathit{sent}_A(A.B.\mathit{msg})}{\mathit{has}_A(\mathit{msg})} \quad (3) \qquad \frac{\mathit{rcvd}_A(B.A.\{\mathit{msg}\}_{PK_A})}{\mathit{has}_A(\{\mathit{msg}\})} \quad (4)$$

$$\frac{\mathit{has}_A(\{m1.m2\})}{\mathit{has}_A(m1) \ \wedge \ \mathit{has}_A(m2)} \quad (5) \qquad \frac{\mathit{rcvd}_A(B.A.\mathit{msg})}{\exists C : \mathit{sent}_C(B.A.\mathit{msg})} \quad (6)$$

In the initial local state of A, propositions $\mathit{fresh}_A(N_a)$ and $\mathit{has}_A(N_a)$ hold. It is straightforward to observe that after execution of action $\mathcal{R}_{ns}^{A_1}$, the following formulas hold:

$K_A \mathit{sent}_A(A.B.\{N_a.A\})$	(semantics of send)
$K_B \mathit{rcvd}_B(A.B.\{N_a.A\})$	(semantics of send)
$K_A \mathit{has}_A(N_a)$	(Axiom 3)
$K_B \mathit{has}_B(N_a)$	(Semantics of send and Axioms 4, 5)
$K_A K_B \mathit{has}_B(N_a)$	(Semantics of send and Axioms 4, 5)

Step 2 The next action of the protocol $\mathcal{R}_{ns}$ is due to agent B:

$$\boxed{\begin{array}{l}\mathcal{R}_{ns}^{B_1} :: \ \mathit{rcvd}_B(A.B.\{N_a.A\}_{PK_B}) \wedge \mathit{fresh}_B(N_b) \\ \qquad\qquad \longrightarrow \ \mathit{fresh}_B(N_b) := \mathit{false}; \mathsf{send}(B_p.\mathit{rcvdfrom}_B.B.A.\{N_a.N_b\}_{PK_A})\end{array}}$$

The guard of action $\mathcal{R}_{ns}^{B_1}$ evaluates to true if agent B has received the message from A and acquires a fresh nonce. Similar to agent A, an atomic proposition

$fresh_B(N_b)$ holds in the initial state of agent B. In this case, agent B sends a message encrypted by the public key of A to agent A containing the nonce it has received from A and its own fresh nonce. Using the axioms described above, we can show that executing this action agent A can authenticate B, i.e., property φ_1 in Equation 1 of Section 4 holds.

Step 3 The last action of the protocol $\mathcal{R}_{ns}$ is due to agent A:

$$\boxed{\begin{array}{l}\mathcal{R}_{ns}^{A_2} :: \ rcvd_A(B.A.\{N_a.N_b\}_{PK_A}) \wedge has_A(N_a) \\ \qquad\qquad\qquad\qquad\qquad \longrightarrow \quad \mathsf{send}(A_p.rcvdfrom_A.A.B.\{N_b\}_{PK_B})\end{array}}$$

By executing this action, B authenticates A; i.e., φ_2 holds.

6.2 The Intruder

The intruder I is based on Dolev-Yao model attacks the system by *impersonating* agent A or B and *replaying* messages. In an impersonation action, an intruder sends a message to some agent, say B, that appears to arrive from agent A. Clearly, there are infinitely many messages the intruder could send to B. However, the (good) agents in this protocol accept a certain format of messages. Hence, any message that is not of that format will be discarded by the agent. One attempt to impersonate A is to send a message to B that conforms to the *structure* of the $\mathcal{R}_{ns}^{A_1}$.

$$\boxed{\mathcal{R}_{ns}^{I_1} :: \ has_I(N_a) \quad \longrightarrow \quad \mathsf{send}(I_p.B_p.A.B.\{N_a.A\}_{PK_B})}$$

Note that the above attack considers the situation where the adversary has learnt N_a. It does not consider the attack when I uses a random number since it would be discarded. However, if $has_I(N_a)$ becomes true based on messages I has received or by combining different message fragments, decoding encrypted messages etc then I would be allowed to attack using the above action. Thus, this modeling permits us to model a general attacker such as that in Dolev-Yao model without considering the infinitely many actions that it could take.

Likewise, Agent I can impersonate user B by sending a message that conforms to the one expected by that agent.

$$\boxed{\mathcal{R}_{ns}^{I_2} :: \ has_I(N_a) \wedge has_I(N_b) \quad \longrightarrow \quad \mathsf{send}(I_p.A_p.B.A.\{N_a.N_b\}_{PK_A})}$$

$$\boxed{\mathcal{R}_{ns}^{I_3} :: \ has_I(N_b) \quad \longrightarrow \quad \mathsf{send}(I_p.B_p.A.B.\{N_b\}_{PK_B})}$$

Observe that in the above message, the physical sender of the message is the intruder. However, the logical sender is A (for $\mathcal{R}_{ns}^{I_1}$ and $\mathcal{R}_{ns}^{I_3}$) or B (for $\mathcal{R}_{ns}^{I_2}$)

In a replay action, the intruder replays a message it had received earlier. Specifically, if the intruder receives a message from B, then it re-sends it to A, so that it appears to have been sent by I. Thus, the action where the intruder replays a message sent by B is as follows: (The action where intruder replays a message sent by A is similar).

$\mathcal{R}_{ns}^{I_4} :: \ rcvd_I(B.A.m) \quad \longrightarrow \quad \mathsf{send}(I_p.A_p.I.A.m)$

$\mathcal{R}_{ns}^{I_5} :: rcvd_I(A.I.\{N_b\}_{PK_I}) \quad \longrightarrow \quad \mathsf{send}(I_p.B_p.A.B.\{N_b\}_{PK_B})$

This action can be *derived* from previous actions of the intruder. It is included here only to simplify the presentation.

We also note that the intruder actions modeled thus can be easily extended to other attacks, such as eavesdropping (modeled by revising the protocol, so that a copy of each message (respectively, selected messages) is sent to the intruder), packet drop (modeled by having each message routed through the intruder who can choose to drop it or forward it), and so on.

6.3 Application of the Repair Algorithm

Based on the description of Algorithm 1, we first identify a state in $T \wedge \neg\varphi_2$ and analyze the run (Line 4). One (prefix of a) run that reaches a state in $T \wedge \neg\varphi_2$ is as shown below:

1. Action $\mathcal{R}_{ns}^{A_1}$	$(A \rightarrow I)$:	$\mathsf{send}(A.I.\{N_a.A\}_{PK_I})$
2. Impersonation Action $\mathcal{R}_{ns}^{I_1}$	$(I \rightarrow B)$:	$\mathsf{send}(I_p.B_p.A.B.\{N_a.A\}_{PK_B})$
3. Action $\mathcal{R}_{ns}^{B_1}$	$(B \rightarrow I)$:	$\mathsf{send}(B_p.I_p.B.A.\{N_a.N_b\}_{PK_A})$
4. Relay Action $\mathcal{R}_{ns}^{I_4}$	$(I \rightarrow A)$:	$\mathsf{send}(I_p.A_p.I.A.\{N_a.N_b\}_{PK_A})$
5. Action $\mathcal{R}_{ns}^{A_2}$	$(A \rightarrow I)$:	$\mathsf{send}(A_p.I_p.A.I.\{N_b\}_{PK_I})$
6. Impersonation Action $\mathcal{R}_{ns}^{I_5}$	$(I \rightarrow B)$:	$\mathsf{send}(I_p.B_p.A.B.\{N_b\}_{PK_B})$

The algorithm begins with step 6 of this run and consider earlier states (Line 5). Observe that step 6 in the above scenario is an intruder action. Hence, according to Line 11, we consider the previous step of the run, where A sends a message to I in action $\mathcal{R}_{ns}^{A_2}$. In the repair algorithm, we need to either remove or restrict this action. Now, we can observe that the guard of the corresponding action satisfies the constraint on Lines 8-14. Hence, the original protocol is revised so that in Step 3, the ID of the sender, namely B is included in the message. Moreover, the action in A is modified to expect this ID to be present. Thus, the revised actions are as follows:

$\mathcal{R}_{ns}^{\prime B_1} :: \ rcvd_B(A.B.\{N_a.A\}_{PK_B}) \wedge fresh_B(N_b)$
$\quad \longrightarrow \ fresh_B(N_b) := false; \mathsf{send}(B_p.rcvdfrom_B.B.A.\{N_a.N_b.B\}_{PK_A})$

$\mathcal{R}_{ns}^{\prime A_2} :: \ rcvd_A(B.A.\{N_a.N_b.B\}_{PK_A}) \wedge has_A(N_a)$
$\quad \longrightarrow \ \mathsf{send}(A_p.rcvdfrom_A.A.B.\{N_b\}_{PK_B})$

One can verify that this repaired protocol satisfies constraints $C1$ and $C2$ of Problem 1.

7 Related Work

Automated model repair is a relatively new area of research. To the best of our knowledge, this paper is the first work on applying model repair in the context of epistemic logic and, in particular, security protocols. Model repair with respect to CTL properties was first considered in [4]. Model repair for CTL using abstraction techniques has been studied in [8]. The theory of model repair for memoryless LTL properties was considered in [12] in a game-theoretic fashion; i.e., a repaired model is obtained by synthesizing a winning strategy for a 2-player game. In [3], the authors explore the model repair for a fragment of LTL (the UNITY language [6]). Most results in [3] focus on complexity analysis of model repair for different variations of UNITY properties. Model repair in other contexts includes the work in [2] for probabilistic systems and in [22] for Boolean programs.

Synthesizing security protocols from BAN logic [5] specifications has been studied in [21]. Unlike our work that repairs an existing protocol, the techniques in [15, 16, 21, 23] synthesize a protocol from scratch and, hence, cannot reuse the previous efforts made in designing an existing protocol. The approaches proposed in [7, 13] address controller synthesis for enforcing security properties. In particular, the technique in [7] studies synthesis of fair non-repudiation protocols for digital signatures and the work in [13] concerns enforcing security objectives expressed in LTL. None of these methods are knowledge-based, which is the focus of this paper.

In the context of repairing security protocols, Pimentel et al. have proposed applying formulation of protocol patch methods to repair the security protocols automatically [17, 18]. In order to guide the location of the fault in a protocol, they use Abadi and Needham's principles [1] for the prudent engineering practice for cryptographic protocols. However, by its nature, this work applies to protocols where principles from [1] are not followed. By contrast, our approach follows a more general approach of using epistemic logic about knowledge to repair the given protocol. Since authentication protocols essentially rely on 'who knows what and when', we expect this method is especially valuable for repairing security protocols.

8 Conclusion

Vulnerabilities of security protocols can be thought of in two categories: (1) where existing assumptions are found to be false, e.g., due to cryptanalytic attacks, and (2) where a new attack that violates the security property is discovered. Examples of former include keys that are not large enough, ability of an intruder to guess nonces (e.g., the attack in an early implementation of SSL by Netscape [11]). For these vulnerabilities, one must utilize *prevention* mechanisms, e.g., with use or larger keys or new algorithms to generate nonces. Examples of the latter include cases where unanticipated behaviors (e.g., imposed by an intruder) can break the soundness of a protocol. For instance, the Needham-Schroeder protocol breaks when one of the agents decides to misbehave. For such vulnerabilities, we advocate a formal *repair* approach.

In this paper, we presented a knowledge-based sound algorithm for repairing authentication protocols. Our algorithm compares this knowledge with the knowledge of an (auxiliary) agent that can obtain additional information based on the *actual* partial run that reached the current state. Subsequently, we identify how the knowledge of the auxiliary agent can be mapped to a real agent in the protocol. Our repair algorithm preserves the existing properties of the protocol by ensuring that the repaired protocol refines the initial one in the absence of the intruder. We illustrated the application of our algorithm on the Needham-Schroeder public-key authentication protocol [14]. We have implemented our algorithm and found that it was possible to repair this protocol in a reasonable time. We argue that our repair algorithm can be utilized for some other security properties as well, e.g., in privacy. Suppose a bit b is to be kept private from an adversary I. In this case, this requirement can be expressed as $\Box(\neg K_I(b = 0) \wedge \neg K_I(b = 1))$.

Our approach is generic for repair of authentication protocols, where the vulnerability lies in the protocol (as opposed to violation of assumption of the strength of encryption). Authentication deals with a requirement that if agent A accepts a message m to be from agent B, then the message is indeed sent by agent B. This is exactly the kind of specification we have used in our case study. Sometimes, the identity of agent B is not precisely known to A; instead it requires that two messages are sent by the same agent. This is also easily possible with our approach. We have not considered the issue of whether a received message is fresh or not. However, this issue can be modeled easily. For example, if A wants to be sure that the message from B is fresh, it can be encoded by a requirement of the form 'B knows something that A knows to be fresh'. Once again, this requirement is identical to the properties considered in this paper. Also, our approach is generic enough to model several threats. Our example considered attacks such as replay and impersonation.

There are several future extensions of this work. One extension is based on developing repair algorithms that utilize the notion of *distributed knowledge* [10]. Another extension is for repairing security protocols for problems such as information flow, where a more general notion of *hyperproperties* [9] is required.

Acknowledgements. This work has been supported in part by Canada's NSERC Discovery Grant 418396-2012, NSERC Strategic Grant 430575-2012, and U.S.A. NSF grants CNS-1329807 and CNS-1318678.

References

1. Abadi, M., Needham, R.: Prudent engineering practice for cryptographic protocols. IEEE Transactions on Software Engineering 22(1), 6–15 (1996)
2. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 326–340. Springer, Heidelberg (2011)
3. Bonakdarpour, B., Ebnenasir, A., Kulkarni, S.S.: Complexity results in revising UNITY programs. ACM Transactions on Autonomous and Adaptive Systems (TAAS) 4(1), 1–28 (2009)

4. Buccafurri, F., Eiter, T., Gottlob, G., Leone, N.: Enhancing model checking in verification by AI techniques. Elsevier Journal on Artificial Intelligence 112, 57–104 (1999)
5. Burrows, M., Abadi, M., Needham, R.M.: A logic of authentication. Proceedings of the Royal Society of London 426(1), 233–271 (1989)
6. Chandy, K.M., Misra, J.: Parallel program design: a foundation. Addison-Wesley Longman Publishing Co., Inc., Boston (1988)
7. Chatterjee, K., Raman, V.: Synthesizing protocols for digital contract signing. In: Kuncak, V., Rybalchenko, A. (eds.) VMCAI 2012. LNCS, vol. 7148, pp. 152–168. Springer, Heidelberg (2012)
8. Chatzieleftheriou, G., Bonakdarpour, B., Smolka, S.A., Katsaros, P.: Abstract model repair. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 341–355. Springer, Heidelberg (2012)
9. Clarkson, M.R., Schneider, F.B.: Hyperproperties. Journal of Computer Security 18(6), 1157–1210 (2010)
10. Fagin, R., Halpern, J., Moses, Y., Vardi, M.: Reasoning About Knowledge. The MIT Press (1995)
11. Goldberg, I., Wagner, D.: Randomness and the netscape browser, `http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html`
12. Jobstmann, B., Griesmayer, A., Bloem, R.: Program repair as a game. In: Etessami, K., Rajamani, S.K. (eds.) CAV 2005. LNCS, vol. 3576, pp. 226–238. Springer, Heidelberg (2005)
13. Martinelli, F., Matteucci, I.: A framework for automatic generation of security controller. Software Testing, Verification and Reliability 22(8), 563–582 (2012)
14. Needham, R.M., Schroeder, M.D.: Using encryption for authentication in large networks of computers. Communications of ACM 21(12), 993–999 (1978)
15. Perrig, A., Song, D.X.: Looking for diamonds in the desert - extending automatic protocol generation to three-party authentication and key agreement protocols. In: CSFW, pp. 64–76. IEEE Computer Society (2000)
16. Perrig, A., Song, D.X.: A first step towards the automatic generation of security protocols. In: NDSS. The Internet Society (2000)
17. Pimentel, J.C.L., Monroy, R., Hutter, D.: A method for patching interleaving-replay attacks in faulty security protocols. In: Electronic Notes in Theoretical Computer Science (ENTCS), pp. 117–130 (2007)
18. Lopez P., J.C., Monroy, R., Hutter, D.: On the automated correction of security protocols susceptible to a replay attack. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 594–609. Springer, Heidelberg (2007)
19. RFC 5746, `http://tools.ietf.org/html/rfc5746`
20. Rivest, R.L., Shamir, A., Adleman, L.M.: A method for obtaining digital signatures and public-key cryptosystems. Communications of ACM 21(2), 120–126 (1978)
21. Saidi, H.: Toward automatic synthesis of security protocols. AAAI archives (2002)
22. Samanta, R., Deshmukh, J.V., Emerson, E.A.: Automatic generation of local repairs for boolean programs. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 1–10 (2008)
23. Song, D., Perrig, A., Phan, D.: AGVI - automatic generation, verification, and implementation of security protocols. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 241–245. Springer, Heidelberg (2001)