Deciding the Correctness of Attacks on Authentication Protocols

Anders Moen Hagalisletto
Department of Informatics, University of Oslo
Postbox 1080 Blindern, 0316 Oslo, Norway, Email: andersmo@ifi.uio.no

***Abstract*— A new tool for automated validation of attacks on authentication protocols has been used to find several errors and ambiguities in the list of attacks described in the well known report by Clark and Jacob. In this paper the errors are presented and classified. Corrected descriptions of the incorrect attacks are given for the attacks that can be easily repaired. The underlying method for finding errors in attacks is presented, including a formal language for attack specification, a validation algorithm, and a framework for executing attacks. At the end of the paper, the connection between validation and simulation is settled: Every attack specification that can be successfully executed is valid.**

***Index Terms*— Authentication protocols, attacks, validation**

I. Introduction

The report "A Survey of Authentication Protocol Literature: Version 1.0" by Clark and Jacob [9] (in this paper denoted Clark/Jacob), has been used extensively by experts on security protocols as the main reference on authentication protocols.[1] This publication was a major achievement in security protocol design and analysis, and it was intended to be a "living document" that should be regularly updated with corrections, new protocols and attacks [9, p. 6]. Tools for analyzing protocols have used the attacks in the Clark/Jacob report as a benchmark for evaluating protocol analyzers ([6], [13], [5], [23], [4] [15], [10], [3]) or as a reference to protocol specifications and attacks ([8], [21]). It is therefore important to obtain correct knowledge about which attacks are correct. Typical claims have been:

> "So far, about 40 protocols from [9] have been analyzed on which all the previously known attacks are detected, as well as new ones." [5, p. 16]
>
> "For example, the OFMC tool finds all known attacks, and discovers a new one (on the Yahalom protocol), in a test suite of 38 protocols from the Clark/Jacob library [9] in a few seconds of CPU time for the entire suite." [4, p. 182]

A new tool for analyzing descriptions of attacks has revealed that 7 of 23 attacks in Clark/Jacob contain errors. Four of these incorrect attacks contain more than one error. Most of the errors reported here have been discovered in a fully automated way by the static validator of the protocol simulator PROSA [16]. The validation is split into several steps: First an attack description is specified in a formal language for protocols. Then the formal attack description is refined in an automated way into a description including the assumptions about actions that the agent is required to perform and assertions that the agent should possess. Finally, the validator checks whether each element in the attack has been obtained in a legal way either by past communication or by the cryptographic operations. PROSA has been used to find ambiguities in two additional attacks in Clark/Jacob, by showing exactly where the attack descriptions need to be adjusted. This paper also shows how several errors and ambiguities in the attack descriptions can be resolved by a slight modification of the attacks.

To the best of the author's knowledge, the severe errors reported in this paper have not been published before. Some of the minor errors reported here are corrected in the Security Protocol Open Repository[2] (SPORE) [14], and some attacks not listed on the webpage which might indicate that some researchers might be aware of the errors. Unfortunately, SPORE is not a systematic update of Clark/Jacob: Flawed but repairable attacks are left out, errors have migrated to SPORE from Clark/Jacob, and correct attacks have been left out. For the sake of completeness both the severe errors and the misprints are included in this paper - along with the corrected versions.

The current paper subsumes the results presented in [17], the method described in [18], but also contains a new result describing how validation relates to simulation, described in [16]. The paper contains two major parts: First we present the result of the analysis without detailed explanation of the underlying method and tool, since the results might interest a general audience. Second we present the formal language, the method and a description of the application of the tool. Part one of the paper (Section II) is organized as follows: In Section II-A, a classification of errors is presented. The incorrect attacks and their analysis are presented in Section II-B. Some

This paper is an extended version of "Errors in Attacks on Authentication Protocols, presented at ISCC 2007 [17]. The paper presents the validation algorithm used to detect the errors and the simulation framework for executing attack. A previously unpublished result connecting validation and simulation is presented at the end of the paper.

[1] At 27 November 2006, the report had 50 citations by *citeseer*.

[2] The webpage SPORE is the self-proclaimed successor of the Clark/Jacob report. Since the Internet page might be updated continuously, this paper refers to the version 27 November 2006.

attack descriptions in Clark/Jacob are incomplete although not incorrect, revisions proposed by the validator are presented in Section II-E. The results are discussed and compared with SPORE in Section II-F. In the second part of the paper the method used to find errors in attacks of authentication protocols is presented: the formal language used to specify protocols and attack descriptions (Section III-A), automated refinement of attack specification (Section IV), and the validation algorithm (Section V). Not every error can be discovered through validation, some attack descriptions must be simulated in order to discover the flaws. One example is the attack on Shamir Rivest Adelman protocol. In Section V-A, we present an operational semantics for simulations of attacks. Fortunately there is a formal relation between the two approaches, and in Section VII, we prove that validation can be embedded into simulation.

Finally in Section VIII concluding remarks are given.

II. Analyzing a library of autthentication protocols

In this section we give a careful analysis of the result of applying the PROSA tool to the most famous library on authentication protocols. This part is written without any reference to the underlying method or tool syntax, in order to make the results comprehensible to the general security expert.

A. Classification of errors

Every attack in the report by Clark/Jacob has been checked using the validator. Errors in attack descriptions can be divided into four categories:

- (a) *misprints*,
- (b) *man-in-the-middle errors*,
- (c) *incompleteness of assumptions*, and
- (d) *protocol jumps*.

The *man-in-the-middle flaw* is a common type of specification error; it means that the intruder should have intercepted and forwarded a message earlier in the protocol session. In order to make the attack description precise a protocol clause

$$(P) \quad A \longrightarrow B \quad : \quad M$$

meaning "agent A sends a message M to agent B", should be replaced by the two clauses

$$(P_1) \quad A \longrightarrow I(B) \quad : \quad M$$
$$(P_2) \quad I(A) \longrightarrow B \quad : \quad M$$

where the notation $I(B)$ means that the intruder I impersonates the agent B. Hence in (P_1) intruder I intercepts the message intended to be received by B, while in (P_2), the intruder I sends a message pretending to be A.

Incompleteness of assumptions means that there are some data in the protocol, like keys, nonces, timestamps or ciphertexts that an agent is assumed to be aware of at a given point in the attack description, but these data have not been obtained through past communication and valid decryption.

The final type of error, *protocol jump* means that an honest agent involved in the attack description does not follow the protocol that is supposed to be under attack.

Misprints and man-in-the-middle errors are easy to fix, and the attacks were revised and then validated as correct attacks. Most interpretations of the Dolev-Yao model [12] assume that the attacker controls the network, hence every honest message is intercepted by the attacker (corresponding to P_1). If an attack is discovered by an automated analysis tool for security protocols, then typically a proper subset of the interceptions is required in order for the attack to succeed. Based on the result of this inquiry we recommend that the exact interceptions should be stated explicit in any attack description. Two attacks in Clark/Jacob lacking man-in-the-middle clauses turned out to contain severe errors: it is likely that these errors could have been discovered if the attack description had been complete in the first place.

Incompleteness of assumptions and protocol jumps tend to indicate severe errors in the attacks, some of which are not easily repaired. Flaws of kind $(a-c)$ were typically found by the validator, while mistakes of kind (d) were discovered by the validator except the errors in the Shamir Rivest Adelman.

B. Notation

In the following section we present the collection of errors found by the validator. First we present five attacks that contain severe errors. Then two attacks only containing misprints are discussed. Both the protocol specifications and attack descriptions are given in a notation similar to Clark/Jacob. The messages in the protocols consists of basic entities as follows:

A, B, C, S, I, $I(A)$	agent terms
K_{AB}	symmetric key shared by A and B
K_A	A's public key
K_A^{-1}	A's private key
N_A	nonce generated by agent A
T_A	timestamp generated by agent A

There are two composition operators in the notation: concatenation denoted by "," (comma) and encryption denoted by $E(K \ : \ M)$, where K denotes a key and M a message content.

C. Attacks containing severe errors

The attacks presented in this section contain at least one severe error each: either incompleteness of assumptions (c) or protocol jumps (d). Two of the protocols, the Wide Mouthed Frog and the Denning Sacco Public Key Protocol, additionally contain misprints (a) and man-in-the-middle errors (b). Later, in Section II-F (Table I), an overview of the errors found is given.

1) The Wide Mouthed Frog: The protocol [9, p. 48], was proposed by Mike Burrows as "(...) perhaps the simplest protocol that uses shared-key cryptography and an authentication server" [7, p. 25]:

$$\begin{array}{lll} (\text{WMF}_1) & A \longrightarrow S & : A, E(K_{AS} : T_A, B, K_{AB}) \\ (\text{WMF}_2) & S \longrightarrow B & : E(K_{BS} : T_S, A, K_{AB}) \end{array}$$

An attack on the protocol was presented by Anderson and Needham in [2, p. 429], and formally described in Clark/Jacob. Two mistakes in the attack on the Wide Mouthed Frog protocol given in Clark/Jacob, were found by the validator. The attack by Clark/Jacob states:

$$\begin{array}{lll} (W.1.1) & A \longrightarrow S & : A, E(K_{AS} : T_A, B, K_{AB}) \\ (W.1.2) & S \longrightarrow B & : E(K_{BS} : T_S, A, K_{AB}) \\ (W.2.1) & I(B) \longrightarrow S & : B, E(K_{BS} : T_S, A, K_{AB}) \\ (W.2.2) & S \longrightarrow I(A) & : E(K_{AS} : T'_S, B, K_{AB}) \\ (W.3.1) & I(A) \longrightarrow S & : A, E(K_{AS} : T'_S, B, K_{AB}) \\ (W.3.2) & S \longrightarrow I(B) & : E(K_{BS} : T''_S, A, K_{AB}) \\ (W.4.1) & A \longrightarrow I(S) & : E(K_{AS} : T'_S, B, K_{AB}) \\ (W.4.2) & I(S) \longrightarrow B & : E(K_{BS} : T''_S, A, K_{AB}) \end{array}$$

The mistakes are not easy to spot at a glance. The first mistake occurs in line $(W.2.1)$: The intruder I has not obtained $E(K_{BS} : T_S, A, K_{AB})$. The reason is that I is not intercepting the second message $(W.1.2)$. The second problem occurs with the transmission $(W.4.1)$: The agent A does not have any way of deducing the already created timestamp T'_S from what has happened previously in the attack. A corrected version of the attack can be given as follows:

$$\begin{array}{lll} (W.1.1) & A \longrightarrow S & : A, E(K_{AS} : T_A, B, K_{AB}) \\ (W.1.2.a) & S \longrightarrow I(B) & : E(K_{BS} : T_S, A, K_{AB}) \\ (W.1.2.b) & I(S) \longrightarrow B & : E(K_{BS} : T_S, A, K_{AB}) \\ (W.2.1) & I(B) \longrightarrow S & : B, E(K_{BS} : T_S, A, K_{AB}) \\ (W.2.2) & S \longrightarrow I(A) & : E(K_{AS} : T'_S, B, K_{AB}) \\ (W.3.1) & I(A) \longrightarrow S & : A, E(K_{AS} : T'_S, B, K_{AB}) \\ (W.3.2) & S \longrightarrow I(B) & : E(K_{BS} : T''_S, A, K_{AB}) \\ (W.4.1.a) & A \longrightarrow I(S) & : A, E(K_{AS} : T'_A, B, K_{AB}) \\ (W.4.2) & I(S) \longrightarrow B & : E(K_{BS} : T''_S, A, K_{AB}) \end{array}$$

In this description, message $(W.1.2)$ is replaced by a man-in-the-middle interception: $(W.1.2.a)$ and $(W.1.2.b)$. The application clause $(W.4.1)$ was not a sentence that could be interpreted as belonging to a session of the Wide Mouthed Frog protocol. In $(W.4.1.a)$ the agent A starts a re-authentication of the agent B with a new timestamp T'_A, and is fooled in message $(W.4.2)$ to believe that S has replied with the appropriate timestamp T''_S. The timestamp is not fresh, B falsely believes that S has been involved in the last session.

2) Yahalom: The Yahalom protocol [9, p. 49] uses symmetric keys, in order to establish a new session key K_{AB} to be shared by agent A and B. The protocol was invented by Raphael Yahalom and presented in [7, p. 30]:

$$\begin{array}{ll} (\text{Y}_1) & A \longrightarrow B : A, N_A \\ (\text{Y}_2) & B \longrightarrow S : E(K_{BS} : A, N_A, N_B) \\ (\text{Y}_3) & S \longrightarrow A : E(K_{AS} : B, K_{AB}, N_A, N_B), \\ & \qquad\qquad E(K_{BS} : A, K_{AB}) \\ (\text{Y}_4) & A \longrightarrow B : B, E(K_{BS} : A, K_{AB}), E(K_{AB} : N_B) \end{array}$$

In the attack presented in Clark/Jacob, the attacker tries to make the respondent B believe that the concatenation of nonces N_A, N_B plays the role of the new secret session key, in other words the attack is a typical type flaw:

$$\begin{array}{ll} (Y.1) & I(A) \rightarrow B : A, N_A \\ (Y.2) & B \rightarrow I(S) : E(K_{BS} : A, N_A, N_B) \\ (Y.3) & \qquad\qquad \text{Omitted} \\ (Y.4) & I(A) \rightarrow B : B, E(K_{BS} : A, N_A, N_B), \\ & \qquad\qquad E(N_A, N_B : N_B) \end{array}$$

The validator immediately found that the intruder I does not possess $E(N_A, N_B : N_B)$ before entering $(Y.4)$. There are two reasons why the intruder I cannot build the sentence $E(N_A, N_B : N_B)$: I does not possess N_B, and can neither build the fake key N_A, N_B nor the content N_B to be encrypted. There is no obvious way to repair this attack. Note that Donovan et. al. considered the attack to be erroneous without giving an explanation or analysis of how they considered it flawed [13, p. 6].

3) Woo Lam Π*:* Woo Lam's Π protocol [9, p. 51] is the final of a series of one-way authentication protocols initially presented in [24]:

$$\begin{array}{lll} \text{WL}_1 & A \longrightarrow B & : A \\ \text{WL}_2 & B \longrightarrow A & : N_B \\ \text{WL}_3 & A \longrightarrow B & : E(K_{AS} : N_B) \\ \text{WL}_4 & B \longrightarrow S & : E(K_{BS} : A, E(K_{AS} : N_B)) \\ \text{WL}_5 & S \longrightarrow B & : E(K_{BS} : N_B) \end{array}$$

Clark/Jacob presents two attacks on the protocol. The final and rather obscure attack on the protocol [9, p. 53] is given by two interleaving sessions:

$$\begin{array}{lll} (L.1.1) & B \longrightarrow I & : B \\ (L.2.1) & I(A) \longrightarrow B & : A \\ (L.2.2) & B \longrightarrow I(A) & : N_B \\ (L.1.2) & I \longrightarrow B & : E(N_B : K_{IS}) \\ (L.1.3) & B \longrightarrow I & : E(E(N_B : K_{IS}) : K_{BS}) \\ (L.2.5) & I(S) \longrightarrow B & : E(N_B : K_{BS}) \end{array}$$

The attack involves five type conversions. The two main conversions regard interpreting the nonce N as a key in $(L.1.2)$ and $(L.2.5)$, and interpreting the cipher-text $E(N : K_{IS})$ as a key in $(L.1.3)$. The validator reported that agent B had no reason to believe that $E(N_B : K_{IS})$ is a key. This problem can be solved by assuming that the equation $E(\text{Key} : M) = E(M : \text{Key})$ (†) holds. Then in clause $(L.1.3)$ agent B is encrypting with key K_{BS} and sending $E(K_{BS} : E(N_B : K_{IS}))$, while the intruder I is receiving $E(E(N_B : K_{IS}) : K_{BS})$ and decrypting with the key $E(N : K_{IS})$. But then $(L.2.5)$ turns out to be a problem, since B's interaction requires two missing intermediate protocol events $(L.2.3)$ and $(L.2.4)$. One may reinterpret the attack as one single session to avoid this protocol jump:

$$
\begin{array}{lllll}
(L.1) & I(A) \longrightarrow B & : & A \\
(L.2) & B \longrightarrow I(A) & : & N_B \\
(L.3) & I(A) \longrightarrow B & : & E(N_B : K_{IS}) \\
(L.4) & B \longrightarrow I(S) & : & E(K_{BS} : E(N_B : K_{IS})) \\
(L.5) & I(S) \longrightarrow B & : & E(N_B : K_{BS})
\end{array}
$$

The clause $(L.4)$ is not part of the protocol according to WL_4, agent B is not following the protocol, by omitting the agent name A. Instead it is possible to return to the original attack by including the two missing messages:

$$
\begin{array}{lllll}
(L.2.3) & I(A) \longrightarrow B & : & M \\
(L.2.4) & B \longrightarrow I(S) & : & E(K_{BS} : A, M)
\end{array}
$$

Since Woo and Lam require that the agents can detect replays, M must be chosen different from any of the previous messages and such that B can not decrypt according to (†). One such example is $M = E(K_{IS} : E(K_{IS} : N_B))$: M is no replay, and both K_{IS} and $E(K_{IS} : N_B)$ are kept secret to agent B.

4) Denning Sacco Public Key: The protocol [9, p. 63] uses certificates to establish a secure connection between two agents A and B:

$$
\begin{array}{lllll}
(\mathrm{DS}_1) & A \longrightarrow S & : & A, B \\
(\mathrm{DS}_2) & S \longrightarrow A & : & \mathrm{C}_A, \mathrm{C}_B \\
(\mathrm{DS}_3) & A \longrightarrow B & : & \mathrm{C}_A, \mathrm{C}_B, E(K_B : E(K_A^{-1} : K_{AB}, T_A))
\end{array}
$$

The agent A uses two certificates, denoted C_A and C_B, that are distributed from a trusted server S in order to securely deliver a new session key K_{AB} to the agent B. The session key and a timestamp are signed by A's private key, in order to assure authenticity, and then encrypted with B's public key in order to provide secrecy. In the attack by Clark/Jacob the bad agent B is fooling an honest agent C to believe that B is running a session with A:

$$
(D.3) \quad B(A) \longrightarrow C : \mathrm{C}_A, \mathrm{C}_C, E(K_C : E(K_A^{-1} : K_{AB}))
$$

There is an obvious misprint, the timestamp T_A is left out in $(D.3)$. In the original attack by Abadi and Needham [1], the timestamp is included:

$$
\begin{array}{lll}
(D.3.a) & B(A) \longrightarrow C : \mathrm{C}_A, \mathrm{C}_C, & \\
 & \qquad E(K_C : E(K_A^{-1} : K_{AB}, T_A))
\end{array}
$$

The attack relies on B's capabilities to initiate sessions and intercept any previous runs of the session, as described informally in the original paper by Abadi and Needham [1]:

$$
\begin{array}{lllll}
(D.1.1) & A \longrightarrow S & : & A, B \\
(D.1.2) & S \longrightarrow A & : & \mathrm{C}_A, \mathrm{C}_B \\
(D.1.3) & A \longrightarrow B & : & \mathrm{C}_A, \mathrm{C}_B, \\
 & & & \qquad E(K_B : E(K_A^{-1} : K_{AB}, T_A)) \\
(D.2.1) & B(A) \rightarrow S & : & A, C \\
(D.2.2) & S \rightarrow B(A) & : & \mathrm{C}'_A, \mathrm{C}'_C \\
(D.2.3) & B(A) \rightarrow C & : & \mathrm{C}_A, \mathrm{C}'_C, \\
 & & & \qquad E(K_C : E(K_A^{-1} : K_{AB}, T_A))
\end{array}
$$

But even this attack description which is derived from [1] is flawed. The final clause $(D.2.3)$ is a protocol jump for the honest agent C. This can be seen by examining the certificates ([11, p. 534] or [9, p. 63]) involved in the two protocol runs:

$$
\begin{array}{ll}
\mathrm{C}_A = E(K_S^{-1} : A, K_A, T_S) & \mathrm{C}_B = E(K_S^{-1} : B, K_B, T_S) \\
\mathrm{C}'_A = E(K_S^{-1} : A, K_A, T'_S) & \mathrm{C}'_C = E(K_S^{-1} : C, K_C, T'_S)
\end{array}
$$

The agent C will not accept the certificates $\mathrm{C}_A, \mathrm{C}'_C$ as belonging to a Denning Sacco session, since the certificates received are not synchronized with respect to the timestamp:

$$
\mathrm{C}_A, \mathrm{C}'_C = E(K_S^{-1} : A, K_A, T_S), E(K_S^{-1} : C, K_C, T'_S)
$$

This is an explicit part of the protocol, hence C will abort her session after decrypting the certificates. The attack can be repaired by replacing $(D.2.3)$ with the clause $(D.2.3.b)$, in the previous description:

$$
\begin{array}{lll}
(D.2.3.b)\ B(A) \longrightarrow C & : & \mathrm{C}'_A, \mathrm{C}'_C, \\
 & & \qquad E(K_C : E(K_A^{-1} : K_{AB}, T_A))
\end{array}
$$

In this message both the certificates of A and C are synchronized on the timestamp T'_S. Agent C can not detect that B is the originator: Hence C is successfully fooled to believe that A sent her a new session K_{AB} key shared by A and C.

5) Shamir Rivest Adelman Three Pass (SRA): The Shamir Rivest Adelman protocol [9, p. 64] assumes that encryption is commutative, which means the following: $E[k_1 : E[k_2 : F]] = E[k_2 : E[k_1 : F]]$ (‡).

$$
\begin{array}{lllll}
(\mathrm{SRA}_1) & A \longrightarrow B & : & E(K_A : M) \\
(\mathrm{SRA}_2) & B \longrightarrow A & : & E(K_B : E(K_A : M)) \\
(\mathrm{SRA}_3) & A \longrightarrow B & : & E(K_B : M)
\end{array}
$$

The second attack presented in the report contains two protocol jumps. The attack involves two interleaving sessions.

$$
\begin{array}{lllll}
(R.1.1) & A \longrightarrow I(B) & : & E(K_A : M) \\
(R.2.1) & I(B) \longrightarrow A & : & E(K_A : M) \\
(R.2.2) & A \longrightarrow I(B) & : & M \\
(R.1.2) & I(B) \longrightarrow A & : & bogus \\
(R.1.3) & A \longrightarrow I(B) & : & E(K_A : bogus)
\end{array}
$$

The first mistake in this attack occurs in event $(R.2.2)$ as a reply to $(R.2.1)$. Agent A is not following the protocol, she should have been replying $E(K_A : E(K_A : M))$ according to (SRA_2). In $(R.1.2)$ the attacker I may well send the message containing $bogus$, since A is expecting a message encrypted with B's public key. But it is incorrect to claim that A is sending $E(K_A : bogus)$ in $(R.1.3)$. Since it receives bogus, in its (first) session $R.1.2$, it is not able to do any decryptions, and the session aborts. The clause $R.1.3$ is nothing that A can do as the final step in the (first) session, since it is not in accordance with A's local understanding of the protocol. It could have been the start of a third session. This is the second protocol jump in the description. This attack is not easily repaired either.

D. Two attacks containing only misprints

1) Neuman Stubblebine: The protocol is split into two parts, exchanging tickets ($\mathrm{NS}_1 - \mathrm{NS}_4$) and repeated authentication ($\mathrm{NS}_5 - \mathrm{NS}_7$):

$$
\begin{array}{lll}
(\mathrm{NS}_1) & A \longrightarrow B & : A, N_A \\
(\mathrm{NS}_2) & B \longrightarrow S & : B, E(K_{BS} : A, N_A, T_B), N_B \\
(\mathrm{NS}_3) & S \longrightarrow A & : E(K_{AS} : A, N_A, K_{AB}, T_B), \\
 & & \quad E(K_{BS} : A, K_{AB}, T_B), N_B \\
(\mathrm{NS}_4) & A \longrightarrow B & : E(K_{BS} : A, K_{AB}, T_B), E(K_{AB} : N_B) \\
(\mathrm{NS}_5) & A \longrightarrow B & : N'_A, E(K_{BS} : A, K_{AB}, T_B) \\
(\mathrm{NS}_6) & B \longrightarrow A & : N'_B, E(K_{AB} : N'_A) \\
(\mathrm{NS}_7) & A \longrightarrow B & : E(K_{AB} : N'_B),
\end{array}
$$

In the first attack on Neuman Stubblebine protocol [9, p. 57], the intruder I tries to fool B to accept the nonce N_A as a session key:

$$
\begin{array}{lll}
(N.1) & I(A) \longrightarrow B : & A, N_A \\
(N.2) & B \longrightarrow I(S) : & B, E(K_{BS} : A, N_A, T_B), N_B \\
(N.3) & & \text{Omitted} \\
(N.4) & I(A) \longrightarrow B : & E(K_{BS} : A, N_A, T_B), E(K_{AB} : N_B) \\
(N.5) & I(A) \longrightarrow B : & N'_A, E(K_{BS} : A, N_A, T_B) \\
(N.6) & B \longrightarrow I(A) : & N'_B, E(K_{AB} : N'_A) \\
(N.7) & I(A) \longrightarrow B : & E(K_{AB} : N'_B),
\end{array}
$$

The attack is not valid because of the final sentence $(N.7)$: The intruder I is not able to form $E(K_{AB} : N'_B)$, because the key K_{AB} is unknown to I. This might be a mistyped key by the authors. If we instead follow their own convention for notation and write

$$
\begin{array}{lll}
(N.6.a) & B \longrightarrow I(A) & : N'_B, E(N_A : N'_A) \\
(N.7.a) & I(A) \longrightarrow B & : E(N_A : N'_B),
\end{array}
$$

then the specification represents a correct attack.

2) Encrypted Key Exchange: In the protocol [9, p. 65], a password P is used as symmetric key to distribute a randomly generated public key K_A and a new session key R. The session key R is a secret key shared by A and B:

$$
\begin{array}{lll}
(\mathrm{E}_1) & A \longrightarrow B & : E(P : K_A) \\
(\mathrm{E}_2) & B \longrightarrow A & : E(P : E(K_A : R)) \\
(\mathrm{E}_3) & A \longrightarrow B & : E(R : N_A) \\
(\mathrm{E}_4) & B \longrightarrow A & : E(R : N_A, N_B) \\
(\mathrm{E}_5) & A \longrightarrow B & : E(R : N_B)
\end{array}
$$

The attack is given as follows [9, p. 65]:

$$
\begin{array}{lll}
(E.1.1) & A \longrightarrow I(B) & : E(P : K_A) \\
(E.2.1) & I(B) \longrightarrow A & : E(P : K_A) \\
(E.2.2) & A \longrightarrow I(B) & : E(P : E(K_A : R)) \\
(E.1.2) & I(B) \longrightarrow A & : E(P : E(K_A : R)) \\
(E.1.3) & A \longrightarrow I(B) & : E(R : N_A) \\
(E.2.3) & I(B) \longrightarrow A & : E(R : N_A) \\
(E.2.4) & A \longrightarrow I(B) & : E(R : N_A, N_B) \\
(E.1.4) & I(B) \longrightarrow A & : E(R : N_A, N_B) \\
(E.1.5) & A \longrightarrow B & : E(R : N_B) \\
(E.2.5) & I(B) \longrightarrow A & : E(R : N_B)
\end{array}
$$

The validator reports that in between the transmission $(E.1.5)$ and $(E.2.5)$, the agent B had not obtained the key R in a valid manner. The reason is that R is a shared (symmetric) key between A and B. The agent B expects to be able to know R, yet B has not created R, or received R during the session. So B is correct in claiming ownership to the key, and the validator is correct in stating that B's claim is unjustified, since R is specified to be freshly generated in the protocol run! Protocol clause $(E.1.5)$ is a simple misprint, by instead replacing it with the clause

$$
(E.1.5.a) \quad A \longrightarrow I(B) \ : \ E(R : N_B),
$$

the attack becomes valid.

E. Concise descriptions of attacks

Two of the attacks are not directly incorrect, since the attacks lacked man-in-the-middle clauses that were, however, described informally in the text: Below, the necessary and sufficient criteria for making the attacks precise are given:

1) Andrew Secure RPC : The protocol is given in [9, p. 45] (initially presented by M. Satyanarayanan in [22, p. 256]).

$$
\begin{array}{lll}
(\mathrm{A}_1) & A \longrightarrow B & : A, E(K_{AB} : N_A) \\
(\mathrm{A}_2) & B \longrightarrow A & : E(K_{AB} : N_A + 1, N_B) \\
(\mathrm{A}_3) & A \longrightarrow B & : E(K_{AB} : N_B + 1) \\
(\mathrm{A}_4) & B \longrightarrow A & : E(K_{AB} : K'_{AB}, N'_B)
\end{array}
$$

The attack by Clark/Jacob [9, p. 24] states that the intruder intercepts the third message $(A.3)$, and then impersonates as Bob and replays message $(A.2)$ in $(A.4)$. Hence the intruder tries to fool A to believe that the nonce $N_A + 1$ is the new session key K'_{AB}.

$$
\begin{array}{lll}
(A.1) & A \longrightarrow B & : A, E(K_{AB} : N_A) \\
(A.2) & B \longrightarrow A & : E(K_{AB} : N_A + 1, N_B) \\
(A.3) & A \longrightarrow I(B) & : E(K_{AB} : N_B + 1) \\
(A.4) & I(B) \longrightarrow A & : E(K_{AB} : N_A + 1, N_B)
\end{array}
$$

Validation of the attack reveals that the second message must be intercepted and forwarded by the intruder, hence message $(A.2)$ should be replaced by:

$$
\begin{array}{lll}
(A.2.a) & B \longrightarrow I(A) & : E(K_{AB} : N_A + 1, N_B) \\
(A.2.b) & I(B) \longrightarrow A & : E(K_{AB} : N_A + 1, N_B)
\end{array}
$$

2) Neuman Stubblebine: The second attack on the protocol [9, p. 58] involves one previous session of the initiation phase, corresponding to $(N.2.1 - N.2.4)$, where the third message (NS_3) is intercepted and forwarded $(N.2.3.a)$ and $(N.2.3.b)$, hence Clark/Jacob's attack should be modified as follows:

Protocol attacks *Errors from Section II-B*	Kind of flaw	SPORE	Repairable
Wide Mouthed Frog	(b), (c/d)	not corrected	yes
Yahalom	(c)	not included	no
Woo Lam Π	(c) or (b) or (d)	not included	yes
Denning Sacco Public Key	(a), (b), (d)	not included	yes
Shamir Rivest Adelman	(d), (d)	not included	no
Neuman Stubblebine (1)	(a)	corrected	yes
Encrypted Key Exchange	(a)	not included	yes
Section II-E			
Andrew Secure RPC	(b)	not included	yes
Neuman Stubblebine (2)	(b)	not corrected	yes

TABLE I
ANALYSIS RESULT.

$$
\begin{array}{lll}
(N.2.1) & A \longrightarrow B & : A, N_A \\
(N.2.2) & B \longrightarrow S & : B, E(K_{BS} : A, N_A, T_B), N_B \\
(N.2.3.a) & S \rightarrow I(A) & : E(K_{AS} : A, N_A, K_{AB}, T_B), \\
 & & \quad E(K_{BS} : A, K_{AB}, T_B), N_B \\
(N.2.3.b) & I(S) \rightarrow A & : E(K_{AS} : A, N_A, K_{AB}, T_B), \\
 & & \quad E(K_{BS} : A, K_{AB}, T_B), N_B \\
(N.2.4) & A \longrightarrow B & : E(K_{BS} : A, K_{AB}, T_B), \\
 & & \quad E(K_{AB} : N_B) \\
(N.2.5) & I(A) \rightarrow B & : N'_A, E(K_{BS} : A, K_{AB}, T_B) \\
(N.2.6) & B \rightarrow I(A) & : N'_B, E(K_{AB} : N'_A) \\
(N.3.5) & I(A) \rightarrow B & : N'_B, E(K_{BS} : A, K_{AB}, T_B) \\
(N.3.6) & B \rightarrow I(A) & : N''_B, E(K_{AB} : N'_B) \\
(N.2.7) & I(A) \rightarrow B & : E(K_{AB} : N'_B),
\end{array}
$$

F. Discussion of the findings

The validity of 23 attacks on 15 protocols have been analyzed. Table I gives an overview of the results of the analysis: All the four error types described in Section II-E are represented. The validator reported problems with 9 attacks that can be divided into three groups: incomplete attacks, misprints, and severe flaws. Two of the attacks only contained misprints: the first attack on Neuman Stubblebine and the attack on Encrypted Key Exchange. Both attacks are easily adjusted to form real attacks. The remaining five attacks included 11 errors, six severe, three misprint and two man-in-the-middle flaws. These included two attacks that are not easily repaired, the one on Yahalom and the second on Shamir Rivest Adelman (containing two severe errors). The attack of the Wide Mouthed Frog could be redesigned in conformance with the intentions of the informal description given by Clark/Jacob, by making the implicit interception explicit, and modifying the two final clauses. The attack on the Woo Lam Π protocol was interpreted in three ways, each gives rise to errors. Fortunately the attack was possible to repair in conformance with Clark/Jacobs requirements. The attack on the Denning Sacco Public Key protocol could be made explicit and the erroneous certificate was replaced successfully. It is interesting to observe that the error occured in specifications lacking explicit descriptions of intermediate interceptions (the missing session) and where essential parts of the cryptographic primitives (the certificates) were suppressed.

It turns out that SPORE is not a systematic update of Clark/Jacob, as Table I shows. In the following each of the attacks presented in this paper is compared with the descriptions on the webpage [14]. In the SPORE description of the Wide Mouthed Frog the first error is not corrected, and the website does not mention the final two clauses where the second severe error occurs. Neither the Yahalom attack nor the two attacks on Shamir Rivest Adelman are mentioned, although the first attack on Shamir Rivest Adelman is correct. The original attack on Neuman Stubblebine in [20] and the specification in SPORE are both correctly described. The Encrypted Key Exchange protocol and Denning Sacco Public Key are not included in SPORE, nor are the attacks, even though the attacks could be easily repaired. The attack on Andrew Secure RPC occurring in Clark/Jacob is replaced in SPORE with the original attack from [7]. Finally the Neuman Stubblebine description on the webpage contains the same man-in-the-middle flaw as in Clark/Jacob. Hence there does not seem to be uniform criteria for transferring attacks and protocols from Clark/Jacob into SPORE.

G. Evaluations of the result

The rather bold claims referred in the introduction come in a strange light. Are they too good to be true? At face value it seems so. A tool that claims to find attacks that are incorrect must be inappropriate in some sense or other. What conclusions can be derived on the correctness of the existing tools? A thorough evaluate of the existing tools is outside the scope of this paper. But a general recommendation can be given: Currently it is rare that researchers from the protocol analysis community give details (that is complete descriptions) of the concrete attacks found by the most prominent tools. To avoid potential ambiguity and in order to make protocol analysis a more robust and accumulative research field, detailed lists of the attacks found by the tools should be publicly available.

III. THE METHOD

In Figure 1 we show the process of validation: Initially an attack description P is written in the security language $\mathscr{L}_P$. This description is then fed into the automated refinement algorithm which produce a raw refinement P'. Note that specification P' does not contain information about fresh entities. The raw refinement contains information about the required beliefs and extracted beliefs as well as all the encryptions an decryptions performed in the protocol specification. Fresh nonces and timestamps are inserted into the attack specification, resulting in the description P''. This final refinement is then taken as input to the actual validation algorithm. If no error is found, then it returns ok. If an ungrounded assumption is discovered, then a break point is returned (a minimal failure report), indicating the exact place in the attack where the error occurs.

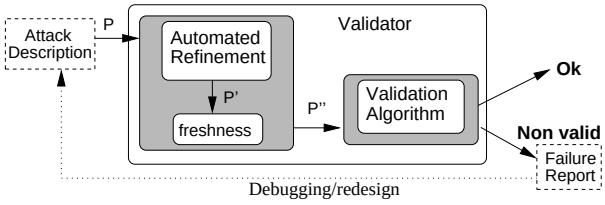

Fig. 1. Validation process.

A. A formal language for authentication protocols

The content of the messages consists of basic entities: nonces, timestamps, text strings, and agent names. In addition there are three composition operators: concatenation, hashing and encryption. A language for specifying security protocols $\mathscr{L}_P$ can be defined as follows: $\mathscr{L}_P$ consists of terms of five sorts, *agents*, *nonces*, *timestamps*, *keys*, and natural numbers. The *agent-names* are typically written "Alice", "Bob", "Server", *agent-variables* are written $x, y, x_1, x_2, \ldots$, and *agent-terms* a, b, c, ... Variables for the other sorts are labeled with the sort x^N (nonce-variable), x^T (time-variable), and x^K (key-variable) respectively, when their sorts are emphasized. Constants include *protocol names* μ, μ_1, μ_2, *encryption methods* for cryptography s (symmetric) and a (asymmetric), in addition to the indicators i (private) and u (public). There are function symbols for nonces $\mathsf{n}(t^N, a)$, time-stamps $\mathsf{stamp}(t^T, a)$, keys $\mathsf{key}(\mathsf{s}, a, b, x^K)$, $\mathsf{key}(\mathsf{a}, \mathsf{i}, a, x^K)$, $\mathsf{key}(\mathsf{a}, \mathsf{u}, a, x^K)$. Protocol names μ_1, μ_2 might be concatenated: $\mu_1\mu_2$. Ground terms for nonces t^N, time-stamps t^T, and key-terms t^K, may either denote natural numbers N or variables of the appropriate sort.

Definition 1: Let $\mathscr{L}_P$ be the least language such that:

(i) Each of the following atomic formulas are in $\mathscr{L}_P$

ε	the empty sentence
$\mathsf{Agent}(a)$	a is an agent
$\mathsf{isKey}(k)$	k is a key
$\mathsf{isNonce}(\mathsf{n}(N, a))$	$\mathsf{n}(N, a)$ is a nonce
$\mathsf{Time}(\mathsf{stamp}(N, a))$	$\mathsf{stamp}(N, a)$ is a timestamp
$\mathsf{role}(b)$	b is a protocol role

(ii) If φ, ψ, $\xi^{\mathcal{T}}$, $\xi^{\mathcal{A}}$, $\xi^{\mathcal{S}} \in \mathscr{L}_P$, then so are;

$\neg\varphi$, $\varphi \rightarrow \psi$	propositional logic
$\mathsf{Transmit}(a, b, \varphi)$	a sends φ to b
$\mathsf{Bel}_a(\varphi)$	a believes φ
$\varphi \,\mathcal{U}\, \psi$	φ holds until ψ holds
$E[k : \varphi]$	encrypt φ using key k
$D[k : \varphi]$	decrypt φ using key k
$\mathsf{Hash}[\varphi]$	hash the sentence φ
$\mathsf{Enforce}_a(\varphi)$	enforce agent a to do φ
$\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi]$	protocol operator

The connectives $\wedge$, $\vee$ and $\leftrightarrow$ are definable in terms of $\neg$ and $\rightarrow$ and $\top = \varphi \rightarrow \varphi$. The operator $\mathsf{Bel}_a(\varphi)$ reads "the agent a believes φ holds". In protocol specifications beliefs are used to explicitly state required cryptographic assumptions. The *until* operator $\varphi \,\mathcal{U}\, \psi$ means that φ holds until ψ holds. The operator *before* $\mathscr{B}$, is definable from $\mathcal{U}$ by $\varphi \,\mathscr{B}\, \psi = \neg(\neg\varphi \,\mathcal{U}\, \psi)$. Furthermore $\boxed{\mathsf{F}}\,\varphi \equiv \neg(\top \,\mathcal{U}\, \neg\varphi)$, $\diamondsuit\!\!\!\!\!\!\diamondsuit\,\varphi \equiv \neg\,\boxed{\mathsf{F}}\,\neg\varphi$ and $\boxdot\varphi \equiv (\boxed{\mathsf{F}}\,\varphi \wedge \varphi)$.

The sentence $\mathsf{Enforce}_a(\varphi)$, means: "enforce agent a to perform φ". Enforcement is the only imperative construct in the language, and is used usually with the generation of fresh entities within the protocol: We augment $\mathscr{L}_P$ with additional constructs for local construction of fresh entities: $\mathsf{newKey}(\mathsf{key}(\mathsf{s}, t_1^A, t_2^A, M))$, $\mathsf{Current}(\mathsf{stamp}(N, t^A))$, and $\mathsf{newNonce}(\mathsf{n}(t^N, t^A))$. Note that the usage of the nested operator $\mathsf{Enforce}_x(\mathsf{Bel}_x(\psi))$ shall be restrictive. The operator $\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]$ denotes: "protocol named μ with session number N, the total roles $\xi^{\mathcal{T}}$, the agent specific roles $\xi^{\mathcal{A}}$, the start-roles $\xi^{\mathcal{S}}$ and with the protocol body Φ". *Trust* can be expressed within $\mathscr{L}_P$ by: $\mathrm{Trust}(a, b, \varphi) \Leftrightarrow \mathsf{Transmit}(b, a, \varphi) \rightarrow \mathsf{Bel}_a(\varphi)$.

A subset of this language is the set of $\mathscr{P}$*-positive* sentences; that includes the atomic sentences, and composite sentences where a modal operator is the uttermost connective (thus every connective in Definition 1 part (ii), except $\neg$ and $\rightarrow$).

B. Specification of protocols

Protocols are sequences of instructions in a distributed program, where the notion of ordering can be expressed by temporal logic: A protocol is a chain of events between agents. A *chain of events* is of the form $\varphi_1 \,\mathscr{B}\, \varphi_2 \wedge \varphi_2 \,\mathscr{B}\, \varphi_3 \wedge \ldots \wedge \varphi_{n-1} \,\mathscr{B}\, \varphi_n$, where each φ_i is $\mathscr{P}$*-positive*. The last event $\varphi_n = \varepsilon$, is always the empty event, marking the end of the protocol specification. Let $\Phi = \varphi \,\mathscr{B}\, [\Phi']$ denote a chain of events written by recursion, hence φ is a single event and Φ' a chain of events.

Definition 2: If Φ and Ψ are chains of events, then their *concatenation*, denoted $\Phi \frown \Psi$, is given by:

(i) $\Phi \frown \varepsilon = \Phi = \varepsilon \frown \Phi$

(ii) $(\varphi \,\mathscr{B}\, [\Phi']) \frown \Psi = \varphi \,\mathscr{B}\, [\Phi' \frown \Psi]$

We say that a *text-book protocol*, (or simply TBP) is a protocol where each single event is of the form $\mathsf{Transmit}(x_j, x_k, \varphi)$, or of the form $\mathsf{Enforce}_x(\mathsf{Bel}_x(\mathsf{newKey}(k)))$.

C. Attack protocols

The language presented so far needs few extensions in order to capture attacks: An *attack protocol* is a protocol P, such that some of the transmissions contains occurrences of *impersonation terms* $\mathtt{im}(I, A)$ either as the sender or the receiver of a transmission. The term $\mathtt{im}(I, A)$ reads "I impersonates as A". In case $\mathsf{Transmit}(\mathtt{im}(I, A), B, M)$, the real sender of the message is I, while the message itself claims that A is the originator. In case $\mathsf{Transmit}(B, \mathtt{im}(I, A), M)$, the message is intercepted by I, and A never receives the message. Protocols that do not contain any occurrence of impersonation terms are called *intended protocols*. We let TBAS denote text-book attack specifications.

IV. Automated Refinement of Attacks

An automated refinement of a text-book specification includes every local assumption about each participant in the attack. Roughly the refinement algorithm works as

follows: Given a transmission clause $A \longrightarrow B \ : \ M$, the refinement algorithm constructs a protocol as follows:

The sender's assumptions about M.	(Pre_A^M)
$A \longrightarrow B \ : \ M$	(Msg)
The receiver's information-extract. from M.	(Post_B^M)

In the following section we present the refinement algorithm that is deployed in order to perform validation. The complete algorithm with a detailed explanation is given in [18]. The chain of events described in (Pre_A^M), characterize the sender's preconditions for transmitting the message (Msg). The chain of events described in (Post_B^M), contains protocol clauses that maximize the receiver's extraction of information from M.

If a transmission involves an impersonation, then the attacker's local behavior must be reflected in the *preconditions* if the attacker fakes an honest agent, or in the *postconditions* if the attacker is an eavesdropper. In both these cases the principal agent in the refinement is the intruder, which possess beliefs or performs cryptograpic actions. Consequently we need two functions for interpreting the impersonation terms, realization of a potential impersonator and realization of a potential interceptor. We define the functions $\underline{t}$ and $\overline{t}$ as follows: If t is an agent variable or agent name, then $\underline{t} = \overline{t} = t$. If $t = \mathrm{im}(t', t'')$, then $\underline{t} = t'$ and $\overline{t} = t''$.

Definition 3: If P is s text-book protocol, then P can be refined into an assumption protocol by $\Re_{\mathrm{A}}$:

(R_0) $\Re_{\mathrm{A}}(\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi]) =$
$\quad \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Re_{\mathrm{A}}(\Phi, \varepsilon)]$

(R_1) $\Re_{\mathrm{A}}(\varepsilon, P) = P$

(R_2) $\Re_{\mathrm{A}}(\mathsf{Transmit}(t_1, t_2, F) \mathscr{B} [\Phi], P) =$
$\quad \Re_{\mathrm{A}}(\Phi, P \frown \mathrm{pre}_{\mathrm{A}}(\underline{t_1}, F, P) \frown (\mathsf{Bel}_{\underline{t_1}}(\mathsf{Agent}(\overline{t_2})) \mathscr{B}$
$\quad \mathsf{Transmit}(t_1, t_2, F) \mathscr{B} \ \mathsf{Bel}_{\underline{t_2}}(\mathsf{Agent}(\overline{t_1}))$
$\quad \mathscr{B} \ \mathsf{Enforce}_{\underline{t_2}}(\mathsf{Bel}_{\underline{t_2}}(\mathrm{Trust}(\underline{t_2}, \overline{t_1}, F))) \mathscr{B} \varepsilon)$
$\quad \frown \mathrm{post}_{\mathrm{A}}(\underline{t_2}, \overline{F}, P))$

(R_3) $\Re_{\mathrm{A}}(\mathsf{Bel}_a(F) \mathscr{B} [\Phi], P) = \Re_{\mathrm{A}}(\Phi, P \frown (\mathsf{Bel}_a(F) \mathscr{B} \varepsilon))$

(R_4) $\Re_{\mathrm{A}}(\mathsf{Enforce}_a(\mathsf{Bel}_a(\mathsf{Hash}[F])) \mathscr{B} [\Phi], P) =$
$\quad \Re_{\mathrm{A}}(\Phi, P \frown \mathrm{pre}_{\mathrm{A}}(a, F, P) \frown$
$\quad (\mathsf{Enforce}_a(\mathsf{Bel}_a(\mathsf{Hash}[F])) \mathscr{B} \varepsilon))$

(R_5) $\Re_{\mathrm{A}}(\mathsf{Enforce}_a(F) \mathscr{B} [\Phi], P) =$
$\quad \Re_{\mathrm{A}}(\Phi, P \frown (\mathsf{Enforce}_a(F) \mathscr{B} \varepsilon))$
$\quad$ if $F \neq \mathsf{Bel}_a(\mathsf{Hash}[M])$, for every M

The definition of $\Re_{\mathrm{A}}$ can be motivated as follows: In the beginning (R_0), the function is called with the empty chain. If the first argument is the empty chain ε, then $\Re_{\mathrm{A}}$ returns the accumulated chain P, and the recursion ends (R_1). In case the event is a transmission, then the assumptions required to send the message are generated as *preconditions* for the sender, and afterwards the local assumptions about the receivers extraction of information is constructed, the *postconditions* for the receiver (R_2). If the protocol contains explicit specifications of assumptions in advance, then these assumptions are included in the refined protocol (R_3 and R_5). One exception to this is hashing (R_4), if some data is hashed this data should have been obtained earlier in the protocol session.

A. Assumptions regarding transmission

We first present the assumptions, pre- and post-conditions for plain-text content hashing, and symmetric keys, and then the extension to public key cryptography is given:

1) Preconditions - constructing the subprotocol (Pre_t^M): The assumption function pre constructs every required assumption necessary for transmitting a message. First let $\mathscr{A}$ denote the following set of atomic sentences:

$\{\mathsf{Agent}(a), \mathsf{isKey}(k), \mathsf{isNonce}(n), \mathsf{Time}(r) |$ a is a agent (term), k is a key, n is a nonce, r is a timestamp $\}$.

Definition 4: The assumption function pre is defined by recursion on the syntactic complexity of the message content:

(AA) $\mathrm{pre}_{\mathrm{A}}(t, F, P) = \mathsf{Bel}_x(F) \mathscr{B} \varepsilon$, if $F \in \mathscr{A}$

(AC) $\mathrm{pre}_{\mathrm{A}}(t, F \wedge G, P) = \mathrm{pre}_{\mathrm{A}}(t, F, P) \frown \mathrm{pre}_{\mathrm{A}}(t, G, P)$

(AH_1) $\mathrm{pre}_{\mathrm{A}}(t, \mathsf{Hash}[F], P) = \mathrm{pre}_{\mathrm{A}}(t, F, P) \mathscr{B} \varepsilon$
$\quad$ if $\mathsf{Hash}\ [F] \in_{\mathrm{E}} P$

(AH_2) $\mathrm{pre}_{\mathrm{A}}(t, \mathsf{Hash}[F], P) = \mathrm{pre}_{\mathrm{A}}(t, F, P) \frown$
$\quad (\mathsf{Enforce}_t(\mathsf{Bel}_t(\mathsf{Hash}[F])) \mathscr{B} \varepsilon)$ if $\mathsf{Hash}\ [F] \notin_{\mathrm{E}} P$

(AE_1) $\mathrm{pre}_{\mathrm{A}}(x, E[\mathsf{key}(\mathsf{s}, y, z) \ : \ F], P) =$
$\quad \mathrm{pre}_{\mathrm{A}}(x, F \wedge \mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z)), P) \frown$
$\quad \mathsf{Enforce}_x(\mathsf{Bel}_x(E[\mathsf{key}(\mathsf{s}, y, z) \ : \ F])) \mathscr{B} \varepsilon$
$\quad$ if $x = y \vee x = z \vee \mathsf{Bel}_x(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \in_{\mathrm{E}} P$

(AE_2) $\mathrm{pre}_{\mathrm{A}}(x, E[\mathsf{key}(\mathsf{s}, y, z) \ : \ F], P) =$
$\quad \mathsf{Bel}_x(E[\mathsf{key}(\mathsf{s}, y, z) \ : \ F]) \mathscr{B} \varepsilon$
$\quad$ if $x \neq y \wedge x \neq z \wedge \mathsf{Bel}_x(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \notin_{\mathrm{E}} P$

2) Postconditions - constructing the subprotocol ($Post_t^M$): The postcondition function post works by extracting information from a received message, by decomposing the message using projection and decryption.

Definition 5: The extraction function post is defined by recursion on the complexity of the message content:

(PA) $\mathrm{post}_{\mathrm{A}}(t, F, P) = \mathsf{Bel}_t(F) \mathscr{B} \varepsilon$, if $F \in \mathscr{A}$

(PC) $\mathrm{post}_{\mathrm{A}}(t, F \wedge G, P) = \mathrm{post}_{\mathrm{A}}(t, F, P) \frown \mathrm{post}_{\mathrm{A}}(t, G, P)$

(PH) $\mathrm{post}_{\mathrm{A}}(t, \mathsf{Hash}[F], P) = \varepsilon$

(PE_1) $\mathrm{post}_{\mathrm{A}}(y, E[\mathsf{key}(\mathsf{s}, x, z) \ : \ F], P) =$
$\quad \mathrm{post}_{\mathrm{A}}(y, \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z)), P)$
$\quad \frown (\mathsf{Enforce}_y(\mathsf{Bel}_y(D[\mathsf{key}(\mathsf{s}, x, z) \ :$
$\quad\quad E[\mathsf{key}(\mathsf{s}, x, z) \ : \ F]])) \mathscr{B} \varepsilon) \frown \mathrm{post}_{\mathrm{A}}(y, F, P)$
$\quad$ if $y = z \vee x = z \vee \mathsf{Bel}_y(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, z))) \in_{\mathrm{E}} P$

(PE_2) $\mathrm{post}(y, E[\mathsf{key}(\mathsf{s}, x, z) \ : \ F]) =$
$\quad \mathsf{Bel}_y(E[\mathsf{key}(\mathsf{s}, x, z) \ : \ F]) \mathscr{B} \varepsilon$
$\quad$ if $y \neq x \wedge y \neq z \wedge \mathsf{Bel}_m(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, y, z))) \notin_{\mathrm{E}} P$

The previous algorithm extends easily to public key cryptography, the details can be found in [18]. Note that the extension to public key infrastructure requires that there is already an infrastructure for distributing the public keys.

B. General properties of automated refinement

A couple of high level properties of automated refinement might be proven; the exact space and time complexity of $\Re_{\mathrm{A}}$, and that the automated refinement of a protocol is a subprotocol of the original protocol.

Let P be an arbitrary nonempty protocol with message contents $M = \{F_1, \ldots, F_{lth(\mathsf{P})}\}$. The maximal message content is a sentence $F \in M$ such that for every i with $1 \leqslant i \leqslant lth(\mathsf{P})$ we have that $\deg(F) \geqslant \deg(F_i)$.

Lemma 1: Let P be TBP, with maximal message content F, then $lth(\Re_{\mathrm{A}}(\mathsf{P})) \leqslant lth(\mathsf{P}) \times (2^{\deg(F)} + 1)$.
If P is a protocol with the chain of events Ω, then we let $\#(\mathsf{P})$ denotes the space used to represent Ω.

Observation 1: If P is TBP, where the maximal message content P is F then $\Re_{\mathrm{A}}(\mathsf{P})$ is performed in time $2 \times lth(\mathsf{P}) + 2^{\deg(F)}$ and space $\#(lth(\mathsf{P}) \times (2^{\deg(F)} + 1))$. Fortunately we can apply the techniques presented in [19], in order to prove that the protocol $\Re_{\mathrm{A}}(\mathsf{P})$ really is a refinement of the protocol P.

Theorem 1: If P is a TBAS, then $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re_{\mathrm{A}}(\mathsf{P})$.

Proof: Straightforward induction on $lth(\mathsf{P})$, using similar techniques as the proof of theorem 4 in [19]. ■
Contraction of superfluous protocol sentences, denoted eq and generation of freshness actions, denoted $\star$, preserves the subprotocol relation: $\Re^{\star}_{\mathrm{A}}(\mathsf{P}) = \star(\mathrm{eq}(\mathsf{P}))$

Theorem 2: If P is TBAS, then $\mathsf{P} \sqsubseteq_{\mathrm{P}} \Re^{\star}_{\mathrm{A}}(\mathsf{P})$.

Proof: Follows by similar techniques as the proof of theorem 6 in [19]. ■

V. Automated validation of attacks

It is possible to use the resulting refined protocol for further analysis. It turns out that automated refinement is the "working horse" of validation. There is a close interplay between the automated validation algorithm and the beliefs obtained through automated refinement. From the presentation of the automated refinement algorithm in Section IV, it is clear that that the algorithm does apply to text-book specifications that are not cryptographically correct. The only sanity check that $\Re_{\mathrm{A}}$ performs, is whether appropriate keys have been obtained prior to encryptions and decryptions. We say that a belief statement is *grounded* if it is obtained through legal cryptographic operations or communication. From the definition of the automated refinement algorithm in Section IV, we observe that some beliefs in certain protocols might not be grounded. Ungrounded beliefs might be atomic facts as well as encrypted sentences. Candidates of ungrounded beliefs - involving encrypted messages using symmetric keys - can be produced from Definition 4, part AE_2, and Definition 5, part PE_2. Similarly, encryptions using asymmetric keys that might potentially give rise to ungrounded beliefs may come from the equations AE_4, AE_5, AE_6, PE_4, and PE_5.

The validation algorithm decides whether every belief in a refined protocol is grounded. In other words, the validation algorithm makes the informal reasoning of the protocol designer's cryptographical understanding entirely explicit.

We call the algorithm *static validation*, and write $\mathbb{v}(\mathsf{P})$ for the result of validating the text-book protocol P. In practice validation is carried out by first performing the automated refinement, then removing duplicated assumptions (denoted eq), and finally supplementing the protocol with freshness actions (denoted $\star$). In other words, given a text-book protocol P, validation amounts to apply $\mathbb{v}(\star(\mathrm{eq}(\Re(\mathsf{P}))))$. There are three cases: a given protocol sentence is grounded, hence we proceed traversing the rest, or the validation reached the end, or finally a belief statement is found that is not grounded. The first case corresponds to a pattern:

$$\mathbb{v}(\varphi \mathscr{B} \Phi)\Psi = \mathbb{v}(\Phi, \Psi \frown (\varphi \mathscr{B} \varepsilon)) \text{ if condition } C \text{ holds.}$$

If $\varphi = \mathsf{Bel}_a(F)$ then the condition C formalize the sentence "F has been obtained correctly by the agent a by past communication and the principles of cryptography". Two additional concepts are needed: A conjunction φ is *included* in another conjunction ψ, denoted $\varphi \sqsubseteq \psi$ is defined by aggregation. The notion of φ is an *element in a chain* Ψ, written $\varphi \in_{\mathrm{E}} \Psi$, is defined by obvious recursion on the length of chains Ψ.

Definition 6: We say that φ can be *embedded (cryptographically) valid inside* Ψ for an agent a, denoted $\mathbb{e}(a, F, \Psi)$, iff

$$\begin{array}{l}
\mathbb{e}(a, \varphi_1, \mathsf{Bel}_b(\varphi_2) \mathscr{B} \Psi) = \mathbb{e}(a, \varphi_1, \Psi) \\
\mathbb{e}(a, \varphi_1, \mathsf{Enforce}_a(\mathsf{Bel}_a(D[k_1 : E[k_2 : \psi]])) \mathscr{B} \Psi) = \top \\
\qquad \text{if } \varphi_1 \sqsubseteq \psi \\
\mathbb{e}(a, \varphi_1, \mathsf{Enforce}_a(\varphi_2) \mathscr{B} \Psi) = \mathbb{e}(a, \varphi_1, \Psi) \\
\quad \text{if } \varphi_2 = \mathsf{Bel}_a(D[k_1 : E[k_2 : \psi]]) \text{ implies } \varphi_1 \not\sqsubseteq \psi \\
\mathbb{e}(a, \varphi_1, \mathsf{Transmit}(b, a, \varphi_2) \mathscr{B} \Psi) = \top \qquad \text{if } \varphi_1 \sqsubseteq \varphi_2 \\
\mathbb{e}(a, \varphi_1, \mathsf{Transmit}(b, \mathtt{im}(a, c), \varphi_2) \mathscr{B} \Psi) = \top \text{ if } \varphi_1 \sqsubseteq \varphi_2 \\
\mathbb{e}(a, \varphi_1, \mathsf{Transmit}(b, c, \varphi_2) \mathscr{B} \Psi) = \mathbb{e}(a, \varphi_1, \Psi) \text{ if } \varphi_1 \not\sqsubseteq \varphi_2 \\
\mathbb{e}(a, \varphi_1, \varepsilon) = \bot
\end{array}$$

Let $\star(\mathsf{isNonce}(n)) = \mathsf{newNonce}(n)$ and $\star(\mathsf{Time}(t)) = \mathsf{Current}(t)$. Also let $\blacklozenge(t)$ denote the function *decalculating* a numeric term t, e.g. $\blacklozenge(x+1) = x$ and $\blacklozenge(x-1) = x$.

Definition 7: The validation algorithm $\mathbb{v}$ is given by:

$$\begin{array}{ll}
(i) & \mathbb{v}(\mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi]) = \mathbb{v}(\Phi, \varepsilon) \\
(ii) & \mathbb{v}(\varepsilon, \Psi) = \mathsf{Text}(\text{"No error found"}) \\
(iii) & \mathbb{v}(F \mathscr{B} \Phi, \Psi) = \mathbb{v}(\Phi, \Psi \frown (F \mathscr{B} \varepsilon)) \\
 & \quad \text{if } F = \mathsf{Bel}_a(\mathsf{Agent}(b)) \text{ or } F = \mathsf{Enforce}_a(\mathsf{Bel}_a(\varphi)) \\
 & \qquad \text{or } F = \mathsf{Transmit}(a, b, \varphi) \\
(iv) & \mathbb{v}(\mathsf{Bel}_a(F) \mathscr{B} \Phi, \Psi) = \mathbb{v}(\Phi, \Psi \frown (\mathsf{Bel}_a(F) \mathscr{B} \varepsilon)) \\
 & \quad \text{if } (F = \mathsf{isNonce}(\mathsf{n}(x^N, b)) \text{ or} \\
 & \qquad \text{or } F = \mathsf{Time}(\mathsf{stamp}(x^T, b)) \text{ or } F = \mathsf{Text}(\mathrm{S}, b)) \\
 & \quad \text{and } (\mathsf{Enforce}_a(\mathsf{Bel}_a(\star(F))) \in_{\mathrm{E}} \Psi \text{ or } \mathbb{e}(a, F, \Psi)) \\
(v) & \mathbb{v}(\mathsf{Bel}_a(\mathsf{isNonce}(\mathsf{n}(t^N, b))) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown (\mathsf{Bel}_a(\mathsf{isNonce}(\mathsf{n}(t^N, b))) \mathscr{B} \varepsilon)) \\
 & \quad \text{if } t^N \text{ is a numeric term, and} \\
 & \quad (\mathsf{Enforce}_a(\mathsf{Bel}_a(\mathsf{newNonce}(\mathsf{n}(\blacklozenge(t^N), b)))) \in_{\mathrm{E}} \Psi \\
 & \qquad \text{or } \mathbb{e}(a, \mathsf{isNonce}(\mathsf{n}(t^N, b)), \Psi) \\
 & \qquad \text{or } \mathbb{e}(a, \mathsf{isNonce}(\mathsf{n}(\blacklozenge(t^N), b)), \Psi)) \\
(vi) & \mathbb{v}(\mathsf{Bel}_a(E[k : F]) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown (E[k : F] \mathscr{B} \varepsilon)) \text{ if } \mathbb{e}(a, E[k : F], \Psi) \\
(vii) & \mathbb{v}(\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, N))) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown \mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, N))) \mathscr{B} \varepsilon) \\
 & \quad \text{if } a = x \vee a = y \\
(viii) & \mathbb{v}(\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, w^K))) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown \mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, w^K))) \mathscr{B} \varepsilon) \\
 & \quad \text{if } (a = x \vee a = y) \wedge \\
 & \quad (\mathsf{Enforce}_a(\mathsf{Bel}_a(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, x, y, w^K)))) \in_{\mathrm{E}} \Psi \\
 & \qquad \text{or } \mathbb{e}(a, \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, w^K)), \Psi)) \\
(ix) & \mathbb{v}(\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, t^K))) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown \mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, t^K))) \mathscr{B} \varepsilon) \\
 & \quad \text{if } a \neq x \text{ and } a \neq y \text{ and} \\
 & \quad (\mathsf{Enforce}_a(\mathsf{Bel}_a(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, x, y, w^K)))) \in_{\mathrm{E}} \Psi \\
 & \qquad \text{or } \mathbb{e}(a, \mathsf{isKey}(\mathsf{key}(\mathsf{s}, x, y, w^K)), \Psi)) \\
(x) & \mathbb{v}(\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{i}, y))) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown (\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{i}, y))) \mathscr{B} \varepsilon) \\
 & \quad \text{if } a = x \text{ or } \mathbb{e}(a, \mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{i}, y)), \Psi) \\
(xi) & \mathbb{v}(\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, y))) \mathscr{B} \Phi, \Psi) = \\
 & \qquad \mathbb{v}(\Phi, \Psi \frown (\mathsf{Bel}_a(\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, y))) \mathscr{B} \varepsilon) \\
(xii) & \text{otherwise } \mathbb{v}(\mathsf{Bel}_a(\varphi) \mathscr{B} \Phi, \Psi) = \neg \mathsf{Bel}_a(\varphi) \wedge \Psi
\end{array}$$

The algorithm is motivated as follows: In (ii) the original protocol to be validated is empty, which means that there are no more sentences to check, hence the protocol is correct. The main recursion involves checking the cryptographic integrity of beliefs. Hence beliefs concerning agent names, enforcements and transmissions are not validated (iii).

Nonces, timestamps, and message text (iv) might either be created by the principal agent, or have been obtained as plain-text or encrypted by transmissions from other agents. Nonces might involve numeric terms, the most common nonce term is $t^N = x + 1$. In that case the nonce might have been received as a numeric term, or as a subterm of a numeric term. In this case, the term is decalculated, $\blacklozenge(t^N)$, in order to justify if the origin of the nonce is valid. An encrypted message (vi) should have been obtained legally, received by transmission from other agents.

Symmetric keys might either be one of the agent's own keys $(vii, viii)$ or keys belonging to another agent (ix). In the former case the key is either possessed by the agent (vii), or created in the current protocol session by the principal agent or another agent, typically a server providing keys $(viii)$. Asymmetric keys is handled as follows: In case the key is a public key, every agent is supposed to be able to know it (xi). In case the agent possesses a private key, it might be its own private key, or the agent has obtained the key intentionally or as part of an attack (x).

If none of the above conditions apply then the algorithm terminates in (xii) with an exception, reporting the negation of the ungrounded assumption: $\neg\, \mathsf{Bel}_a\,(\varphi)$, and where this anomaly occurred Φ (representing the initial segment of the protocol).

A. Simulating attacks on authentication protocols

The errors in the Shamir Rivest Adelman attack were not discovered by the tool. The reason is that the validator can not discover every possible protocol jump in every attack. A recent result by the author [16] shows that the errors in the Shamir Rivest Adelman attack described in this paper can be detected by simulation. The model contained two honest agents Alice and Bob that possessed the Shamir Rivest Adelman protocol. An attacker Malice was configured to control the network, and executed the Clark/Jacob attack with Alice and Bob. Simulations showed that the attack failed to succeed in exactly the same state described in the paper, and reachability analysis of the model showed that the attack failed in any simulation. In section VI-I, we give a detailed exposition of the simulation of the attack.

VI. Operational semantics for protocols

In this section we shall first describe an operational semantics for executing standard protocols and then see how the semantics should be modified in order to include attacks. Agents communicate with other agents over a network. A message in the network consists of a message

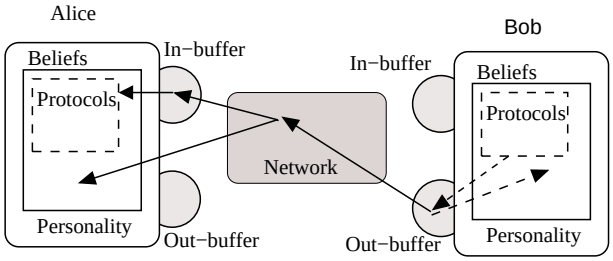

Fig. 2. Uncompromized message flow.

content m, the sender's name and the receivers name. If a and b are agent names, and m is the message content we write $\mathsf{msg}\ \ m\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b$. The entities *agents* and *messages* are the only inhabitants in the model. We introduce the *parallel operator* $\|$, and use the notation $o_1 \,\|\, o_2 \,\|\, \ldots \,\|\, o_n$, to express that the entities $o_1, o_2, \ldots, o_n$ coexists concurrently. A rewrite rule $t \longrightarrow t'$ can be interpreted as a local transition rule allowing an instance of the term t to evolve into the corresponding instance of the pattern t'. Each rewrite rule describes how a part of a configuration can evolve in one transition step. A *configuration* is a snapshot of a dynamic system evolving. The parallel operator is an abelian monoid over the set of configuration with an identity element (the empty configuration). The agent is a structure containing four slots:

$$\prec \underbrace{\mathsf{id}}_{\text{agent name}} \mid \underbrace{\mathsf{bel},}_{\text{set of sentences}} \underbrace{\mathsf{in},\ \mathsf{out}}_{\text{buffers}} \succ$$

where id denotes the *identity* or *name* of the agent, while bel denotes its current *set of beliefs*. The variable denoted in represents the *inbuffer* - messages waiting for processing in protocol while out denotes the *outbuffer* - messages intended for transmission.

A. Asynchronous communication

In case of the honest agent there are two rules for communication, the rule for sending and receiving messages. Referring to figure 2, the send rule involves updating the agents beliefs, and puts the message $\mathsf{msg}\ F\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b$ into the network. Since a is required to be honest, the transmission of the rule requires that $\mathsf{Bel}_a(F)$. Operationally this is interpreted by:

$$\begin{array}{l} \prec a \,|\, \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(a,b,F)\} \succ \\ \longrightarrow \\ \prec a \,|\, \mathsf{bel} \cup \{\mathsf{Transmit}(a,b,F)\}, \mathsf{out} \succ \ \|\ \mathsf{msg}\ \ F\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b \\ \qquad\qquad \text{if}\ \ \mathsf{Honest}\,(a) \in \mathsf{bel} \end{array}$$

The receive rule involves transferring the message from the network and memorizing the message:

$$\begin{array}{l} \prec b \,|\, \mathsf{bel}, \mathsf{in} \succ \ \|\ \mathsf{msg}\ \ F\ \ \mathsf{from}\ \ a\ \ \mathsf{to}\ \ b \longrightarrow \\ \prec b |\, \mathsf{bel} \cup \{\mathsf{Transmit}(a,b,F), \mathsf{Agent}(a)\}, \mathsf{in} \cup \{\mathsf{Transmit}(a,b,F)\} \succ \end{array}$$

B. Protocol execution - the protocol machine

The operational semantics contains some meta-concepts required for executing protocol specifications. We introduce a meta-predicate $\mathbb{E}(a, \mathsf{P})$ (for *execute*),

modeling the concept *protocol machine*. The predicate $\mathbb{E}(a, \mathsf{P})$ reads "agent a at stage P" in the protocol μ, and is an operator for executing any protocol μ in $\mathscr{L}_P$. The execution predicate consumes one protocol event at a time, starting from the top. The rule for transmitting a message in a protocol session then reads:

$$\begin{array}{l}\prec a \,|\, \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \mathsf{Transmit}(a, b, F) \mathscr{B}\, [\Phi]\,])\}, \mathsf{out} \succ \\ \longrightarrow \\ \prec a \,|\, \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi])\}, \mathsf{out} \cup \{\mathsf{Transmit}(a, b, F)\} \succ\end{array}$$

The rule for receiving messages in a protocol session reads:

$$\begin{array}{l}\prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \quad \mathsf{Transmit}(t_1, a, F) \mathscr{B}\, [\Phi]\,])\}, \mathsf{in} \cup \{\mathsf{Transmit}(t_2, a, F')\} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \qquad \underline{\mathrm{sub}}(\mathbb{S}(F', F) \cup \{\langle t_2, t_1\rangle\}, \Phi)\,])\}, \mathsf{in} \succ \\ \qquad\qquad \text{if } \mathbb{M}(F', F) \wedge \mathbb{M}(t_2, t_1)\end{array}$$

The boolean function $\mathbb{M}(F', F)$ decides if F' may match F. The function $\mathbb{S}(F', F)$ performs the matching of the terms in F' with variables in F, resulting in a set of pairs of variables and terms: $\{\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle\}$. Then if the head of the protocol body is a passive transmission, and a message in the inbuffer matches the head, the protocol proceeds by substituting the result of $\mathbb{S}(F', F) \cup \mathbb{S}(t_2, t_1)$, into the rest of the protocol body Φ. The function $\underline{\mathrm{sub}}(S, \mathsf{P})$ recursively substitute a set of matching pairs S in the protocol denoted P.

C. Session administration

For the purpose of this paper we omit multi-threading and the mechanisms for listening at sockets. The execute operator is accompanied by an await operator $\mathbb{A}(b, \mathsf{P})$, which reads "agent b awaits in the responder role for an incoming message in order to proceed executing the protocol P". The function $\mathfrak{F}(a, \mathsf{P})$ filters out agent a's relevant sub-protocol: that is, the function keeps transmissions and assumptions where a plays a principal role, and throws the rest of the protocol.

1) From listening to execution: When the agent receives a message from its environment, it matches the message with one of its running threads of protocol "sockets".

$$\begin{array}{ll}\prec b \mid \mathsf{bel} \cup \{\mathsf{protocol}[\mu, M, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]\} \cup & (1) \\ \{\mathbb{A}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}_{\mathcal{I}}, \mathsf{role}(x), \xi^{\mathcal{S}}, \mathsf{Transmit}(t, b, F) \mathscr{B}\, [\psi]\,])\}, & (2) \\ \qquad \mathsf{in} \cup \{\mathsf{Transmit}(a, b, F')\} \succ & (3) \\ \longrightarrow & \\ \prec b \mid \mathsf{bel} \cup \{\mathsf{protocol}[\mu, M+1, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]\} \cup & (4) \\ \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}_{\mathcal{I}}, \mathsf{role}(x), \xi^{\mathcal{S}}, \underline{\mathrm{sub}}(\mathbb{S}(F', F) \cup \{\langle a, t\rangle\}, \psi)\,])\} & (5) \\ \cup \{\mathbb{A}(b, \mathfrak{F}(b, \mathrm{sub}(b, x, \mathsf{protocol}[\mu, M+1, \xi^{\mathcal{T}}_{\mathcal{I}}, \mathsf{role}(x), \xi^{\mathcal{S}}, \Phi\,])))\}, \mathsf{in} \succ & (6) \\ \qquad \text{if } \mathbb{M}(F', F) \wedge \mathbb{M}(a, t) & (7)\end{array}$$

The rule combines three operations at once, *message handling*, *start of session*, and invocation of *new session thread*: The message in the inbuffer (3) is matched with the head of the protocol body (2) and (7), and removed from the inbuffer (6). An active listening (2) is replaced by a new active protocol session (6). A new thread of listening is generated (6), with the current session number, generated from the initial protocol (2).

2) Done with protocol: Every protocol ends with the empty event ε, which terminates the session:

$$\begin{array}{l}\prec a \,|\, \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \varepsilon])\} \succ \\ \quad \longrightarrow \quad \prec a \mid \mathsf{bel} \cup \{\mathrm{Done}(a, \mu, N)\} \succ\end{array}$$

$\mathrm{Done}(a, \mu, N)$ is a signal for the successful termination of the protocol session.

D. Assumptions and cryptography

Assertions about a belief F that an agent a possess is expressed by $\mathsf{Bel}_a(F)$. Hence the assertion rule is given by:

$$\begin{array}{l}\prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \mathsf{Bel}_a(F) \mathscr{B}\, \varphi\,])\} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathbb{E}(a, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ \quad \text{if } F \in \mathsf{bel}\end{array}$$

1) Cryptography: An agent may encrypt a sentence F using the key k only if it possess both F and k:

$$\begin{array}{l}\prec b \mid \mathsf{bel} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \\ \qquad\qquad \mathsf{Enforce}_b(\mathsf{Bel}_b(E[k : F])) \mathscr{B}\, \varphi\,])\} \succ \\ \longrightarrow \\ \prec b \mid \mathsf{bel} \cup \{E[k : F]\} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ \\ \qquad \text{if } \mathsf{isKey}\,(k) \in \mathsf{bel} \text{ and } F \in \mathsf{bel}\end{array}$$

Decryption is given by the following rule:

$$\begin{array}{l}\prec b \mid \mathsf{bel} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \\ \qquad\qquad \mathsf{Enforce}_b(\mathsf{Bel}_b(D[k : F])) \mathscr{B}\, \varphi\,])\} \succ \\ \longrightarrow \\ \prec b \mid \mathsf{bel} \cup \{D[k : F]\} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ \\ \text{if } F \in \mathsf{bel}\end{array}$$

Decryption of cipher-text requires that the agent a possess the appropriate keys as follows:

key possessed	permitted cryptographic action
$\mathsf{key}(\mathsf{s}, a, b)$	$D[\mathsf{key}(\mathsf{s}, a, b) : E[\mathsf{key}(\mathsf{s}, a, b) : F]] = F$
$\mathsf{key}(\mathsf{a}, \mathsf{i}, a)$	$D[\mathsf{key}(\mathsf{a}, \mathsf{i}, a) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, a) : F]] = F$
$\mathsf{key}(\mathsf{a}, \mathsf{u}, a)$	$D[\mathsf{key}(\mathsf{a}, \mathsf{u}, a) : E[\mathsf{key}(\mathsf{a}, \mathsf{i}, a) : F]] = F$

This gives a distinction between the agent's *intention* of performing a decryption of a cipher-text, and the agent's actual *capability* to decrypt.

2) Fresh generation of keys, nonces and timestamps: Freshly generated data requires that there is a *generator*, that memorize past values of nonces, timestamps and keys. The rule for constructing fresh symmetric keys reads:

$$\begin{array}{l}\prec b \mid \mathsf{bel} \cup \{\mathsf{keyGen}(\mathsf{key}(\mathsf{s}, a, b, M))\} \cup \\ \qquad \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \\ \qquad\qquad \mathsf{Enforce}_b(\mathsf{Bel}_b(\mathsf{newKey}(\mathsf{key}(\mathsf{s}, a, b, x)))) \mathscr{B}\, \varphi\,])\} \succ \\ \longrightarrow \\ \prec b \mid \mathsf{bel} \cup \{\mathsf{keyGen}(\mathsf{key}(\mathsf{s}, a, b, M+1))\} \cup \\ \qquad \{\mathsf{isKey}(\mathsf{key}(\mathsf{s}, a, b, M{+}1))\} \cup \{\mathbb{E}(b, \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \varphi\,])\} \succ\end{array}$$

The rules for constructing fresh nonces and timestamps are similar, each requires that there is a suitable generator.

E. Simulation with the Dolev-Yao attacker

In order to run attack-protocols, the operational semantics should reflect attack specifications, both interception and impersonation should be possible. Running attack-specifications requires that the message flow is modified.

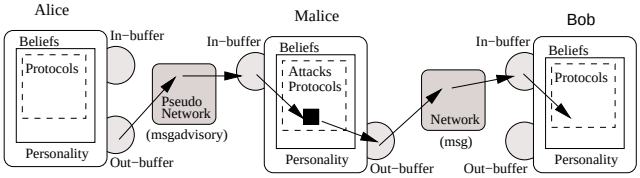

Fig. 3. Compromised communication.

F. Rules for executing attack protocols

Malicious agents can do whatever good agents can and in addition fake and intercept messages. This means that the protocol machine must be extended with the corresponding rules for malicious protocol behaviour. Suppose that μ^A is an attack protocol, and i denote an agent (to be thought of as the intruder). Then impersonating by sender is given by the rule:

$$\begin{array}{l}\prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^A, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \qquad\qquad \mathsf{Transmit}(\mathbb{im}(i,a), b, \varphi) \mathscr{B}\,[\Phi]\,])\}, \mathsf{out} \succ \\ \longrightarrow \\ \prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^A, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi])\}, \\ \qquad\qquad \mathsf{out} \cup \{\mathsf{Transmit}(\mathbb{im}(i,a), b, \varphi)\} \succ\end{array}$$

Interception of messages reads:

$$\begin{array}{l}\prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^A, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \mathsf{Transmit}(t_1, \mathbb{im}(i,a), \varphi) \mathscr{B}\,[\Phi]\,])\}, \mathsf{in} \cup \{\mathsf{Transmit}(t_2, a, \varphi')\} \succ \\ \longrightarrow \\ \prec i \mid \mathsf{bel} \cup \{\mathbb{E}(i, \mathsf{protocol}[\mu^A, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \\ \underline{\mathrm{sub}}(\mathbb{S}(\varphi', \varphi) \cup \{\langle t_2, t_1\rangle\}, \Phi)\,])\}, \mathsf{in} \succ \\ \qquad\qquad \text{if } \mathbb{M}(\varphi', \varphi) \wedge \mathbb{M}(t_2, t_1)\end{array}$$

G. Message flow in the Dolev-Yao model

A standard approach in protocol analysis, is to let the attacker *control* the entire network. In an agent-centric approach like the one advocated in this paper the intruder is placed as malicious router that inspects and manipulates every message put into the network. Hence the network is split into two, first the *advisory* network and then the standard normal network. In Figure 3, the revised message flow according to the Dolev-Yao interpretation is depicted.

Agents sending messages acting as good agents: The good agents transmits messages, by putting the message into the channel **msgadvisory**, in which a message is denoted $\underline{\mathsf{msg}}$. Note that Malice is not permitted to use the rule, since Malice is intercepting every message, and then would intercept herself. Hence the communication rule for good agents, presented previously, is replaced by the following rule:

$$\begin{array}{l}\prec a \mid \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(a, b, \varphi)\} \succ \\ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(a, b, \varphi)\}, \mathsf{out} \succ \ \| \underline{\mathsf{msg}}\ \varphi \ \mathsf{from}\ a\ \mathsf{to}\ b \\ \qquad\qquad \text{if } \mathsf{Honest}\,(a) \text{ and } \varphi \in \underline{\mathsf{bel}} \text{ and } \neg\ \mathsf{Malicious}\,(a)\end{array}$$

Malice receives the message: A message in the **msgadvisory** is received by Malice. Malice's role in the interception is captured by the impersonation construct, $\mathsf{Transmit}(a, \mathbb{im}(i,b), \varphi)$.

$$\begin{array}{l}\prec i \mid \mathsf{bel}, \mathsf{in} \succ \ \| \underline{\mathsf{msg}}\ \varphi \ \mathsf{from}\ a\ \mathsf{to}\ b \ \longrightarrow \\ \prec i \mid \mathsf{bel} \cup \{\underline{\mathsf{Transmit}}(a, \mathbb{im}(i,b), \varphi)\}, \\ \qquad\qquad \mathsf{in} \cup \{\mathsf{Transmit}(a, \mathbb{im}(i,b), \varphi)\} \succ \ \text{if } \mathsf{Malicious}\,(i)\end{array}$$

Malice sending messages as normal participant: In order to avoid that Malice intercepts his own messages, the honest transmissions by Malice is placed directly into the network:

$$\begin{array}{l}\prec i \mid \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(i, b, \varphi)\} \succ \ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(i, b, \varphi)\}, \mathsf{out} \succ \ \| \ \mathsf{msg}\ \varphi\ \mathsf{from}\ i\ \mathsf{to}\ b \\ \qquad\qquad \text{if } \mathsf{Malicious}\,(i)\end{array}$$

Malice sending messages by impersonation: The rule is straightforward, if Malice intends to send a message by impersonation, then this fact is stored in the belief-set of Malice and the message is put on the network as message transmitted from the agent a.

$$\begin{array}{l}\prec i \mid \mathsf{bel}, \mathsf{out} \cup \{\mathsf{Transmit}(\mathbb{im}(i,a), b, \varphi)\} \succ \ \longrightarrow \\ \prec a \mid \mathsf{bel} \cup \{\mathsf{Transmit}(i, b, \varphi)\}, \mathsf{out} \succ \ \| \ \mathsf{msg}\ \varphi\ \mathsf{from}\ a\ \mathsf{to}\ b \\ \qquad\qquad \text{if } \mathsf{Malicious}\,(i)\end{array}$$

Any agent (good or bad) receiving messages: The rule for receiving messages is the same for Malice as it is for any other good agent. Hence to conclude: Executing the attack protocols require few changes into the protocol machine. The communication model involve the introduction of one extra state that the messages must pass through, the Dolev-Yao attacker.

H. Denotational semantics

Since our language is based on epistemic logic, the natural semantics to formulate a denotational semantics is Kripke models. Some aspects of the operational semantics can be embedded in a denotational semantics, suitable for expressing security properties about a given configuration. The *denotational semantics* for the language can be given by connecting the transition relation from the operational semantics. First we interpret the single transition arrow; $\mathscr{C} \longrightarrow \mathscr{C}'$ iff $\mathscr{C} \longmapsto^1 \mathscr{C}'$. The arrow $\stackrel{+}{\longmapsto}$ is the transitive closure of the one step arrow $\longrightarrow^1$, that is $\mathscr{C} \stackrel{+}{\longmapsto} \mathscr{C}'$ iff $\mathscr{C} \longmapsto^1 \mathscr{C}' \vee \exists \mathscr{C}''(\mathscr{C} \longrightarrow^1 \mathscr{C}'' \wedge \mathscr{C}'' \stackrel{+}{\longmapsto} \mathscr{C}')$. A *contextual model* is a pair $\langle \mathcal{C}, \longrightarrow^* \rangle$, where $\mathcal{C}$ is a set of configurations, and $\stackrel{+}{\longmapsto} \subseteq \mathcal{C} \times \mathcal{C}$.

Definition 8: The truth of a formula $\phi \in \mathscr{L}_S$ in a configuration in a contextual model $\mathscr{M} = \langle \mathcal{C}, \stackrel{+}{\longmapsto} \rangle$, denoted $\mathscr{M} \models_{\mathscr{C}} \phi$, is defined by:

$$\begin{array}{lll}\mathscr{M} \models_{\mathscr{C}} a = b & \text{iff} & a^{\mathscr{M}} =^{\mathscr{M}} b^{\mathscr{M}} \\ \mathscr{M} \models_{\mathscr{C}} \mathsf{Agent}(a) & \text{iff} & \prec a \mid \mathsf{bel} \succ \in \mathscr{C} \\ \mathscr{M} \models_{\mathscr{C}} \mathsf{Transmit}(a, b, F) & \text{iff} & (\mathsf{msg}\ F\ \mathsf{from}\ a\ \mathsf{to}\ b) \in \mathscr{C} \\ \mathscr{M} \models_{\mathscr{C}} \mathsf{Bel}_a(\varphi) & \text{iff} & \prec a \mid \mathsf{bel} \cup \{\varphi\} \succ \in \mathscr{C} \\ \mathscr{M} \models_{\mathscr{C}} \neg\varphi & \text{iff} & \mathscr{M} \not\models_{\mathscr{C}} \varphi \\ \mathscr{M} \models_{\mathscr{C}} \varphi_1 \rightarrow \varphi_2 & \text{iff} & \mathscr{M} \models_{\mathscr{C}} \varphi_1 \implies \mathscr{M} \models_{\mathscr{C}} \varphi_2 \\ \mathscr{M} \models_{\mathscr{C}} \forall x \in \mathsf{Ag}\, \phi(x) & \text{iff} & \\ & & \forall a\ (\prec a \mid \mathsf{bel} \succ \in \mathscr{C} \implies \mathscr{M} \models_{\mathscr{C}} \phi(a)) \\ \mathscr{M} \models_{\mathscr{C}} \varphi_1 \mathcal{U} \varphi_2 & \text{iff} & \exists \mathscr{C}'(\mathscr{C} \stackrel{+}{\longmapsto} \mathscr{C}' \wedge \mathscr{M} \models_{\mathscr{C}'} \psi \\ & & \wedge \forall \mathscr{C}''(\mathscr{C} \stackrel{+}{\longmapsto} \mathscr{C}'' \wedge \mathscr{C}'' \stackrel{+}{\longmapsto} \mathscr{C}' \implies \mathscr{M} \models_{\mathscr{C}''} \varphi)) \\ \mathscr{M} \models_{\mathscr{C}} \forall X\, \varphi(X) & \text{iff} & \forall \psi \in \mathscr{L}_S\ \mathscr{M} \models_{\mathscr{C}} \varphi(\psi)\end{array}$$

A sentence φ is *valid* in a model $\mathscr{M} = \langle \mathcal{C}, \stackrel{+}{\longmapsto} \rangle$, denoted $\mathscr{M} \models \varphi$, iff for every $\mathscr{C} \in \mathcal{C}$, $\mathscr{M} \models_{\mathscr{C}} \varphi$.

$$
\begin{array}{l}
\mathsf{protocol}[\mathrm{SRA}, 0, \mathsf{role}(A) \wedge \mathsf{role}(B), \mathsf{role}(A) \wedge \mathsf{role}(B), \mathsf{role}(A), \\
\quad \mathsf{Transmit}(A, B, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (R_1) \\
\mathscr{B}\ \mathsf{Transmit}\ (B, A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, B) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]]) \qquad (R_2) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, B, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (R_3) \\
\mathscr{B}\ \varepsilon]
\end{array}
$$

Fig. 4. Shamir Rivest Adelman in $\mathscr{L}_P$.

$$
\begin{array}{l}
\mathsf{protocol}[\mathrm{SRA}\mathrm{attack2}, 0, \\
\quad \mathsf{role}(A) \wedge \mathsf{role}(B) \wedge \mathsf{role}(I), \mathsf{role}(I), \mathsf{role}(A), \\
\quad \mathsf{Transmit}(A, \mathrm{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (R.1.1) \\
\mathscr{B}\ \mathsf{Transmit}\ (\mathrm{im}(I, B), A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (R.2.1) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathrm{im}(I, B), \mathsf{Text}(\mathrm{MSG}, A)) \qquad (R.2.2) \\
\mathscr{B}\ \mathsf{Transmit}\ (\mathrm{im}(I, B), A, \mathsf{Text}(\mathrm{Bogus}, I)) \qquad (R.1.2) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathrm{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]) \qquad (R.1.3) \\
\mathscr{B}\ \varepsilon]
\end{array}
$$

Fig. 5. Attack on Shamir Rivest Adelman.

I. Shamir Rivest Adelman Three Pass

In this section we shall investigate one application of attack simulation: the validation of attacks on authentication protocols. The Shamir Rivest Adelman protocol (SRA) [9, p. 64] assumes that encryption is commutative, which means $E[k_1 : E[k_2 : F]] = E[k_2 : E[k_1 : F]]$.

$$
\begin{array}{lcl}
(\mathrm{SRA}_1) & A \longrightarrow B & : \quad E(K_A : M) \\
(\mathrm{SRA}_2) & B \longrightarrow A & : \quad E(K_B : E(K_A : M)) \\
(\mathrm{SRA}_3) & A \longrightarrow B & : \quad E(K_B : M)
\end{array}
$$

The protocol is transfered to $\mathscr{L}_P$ by The second attack in Clark/Jacob involves two interleaving sessions.

$$
\begin{array}{lcl}
(R.1.1) & A \longrightarrow I(B) & : \quad E(K_A : M) \\
(R.2.1) & I(B) \longrightarrow A & : \quad E(K_A : M) \\
(R.2.2) & A \longrightarrow I(B) & : \quad M \\
(R.1.2) & I(B) \longrightarrow A & : \quad bogus \\
(R.1.3) & A \longrightarrow I(B) & : \quad E(K_A : bogus)
\end{array}
$$

Although this attack on the SRA protocol is erroneous, it is not possible to detect the attack through static validation as was reported in [17]. Fortunately attack simulation uncovers the two flaws in the attack. The second attack presented in the report contains two protocol jumps. The attack was specified in $\mathscr{L}_P$ as described in Figure 5. The attack includes the three roles initiator A, responder B and attacker I. The automated refinement of the previous attack specification is shown in Figure 6. When validating the refined attack specification, no error is reported. After configuring a scenario involving two good agents Alice and Bob possessing the SRA protocol, and an intruder agent Malice that possesses the attack protocol, the scenario is started with Alice as the initiator of the authentication. Let this initial configuration be denoted $\mathscr{C}_{\mathrm{init}}$. The simulation stops in a configuration $\mathscr{C}_{\mathrm{end}}$ with two unresolved execution predicates $\mathbb{E}$ inside Alice (recall Section VI-B), and one unresolved execution predicate inside Malice. Malice is waiting to intercept a message to be sent by Alice $(R.2.2)$, (line 13 in Figure 6):

$$
\begin{array}{l}
\quad \mathsf{protocol}\ [\mathrm{SRA}\ \mathrm{attack2}, 0, \\
\qquad \mathsf{role}(A) \wedge \mathsf{role}(B) \wedge \mathsf{role}(I),\ \mathsf{role}(I),\ \mathsf{role}(A), \\
\mathsf{Enforce}_A(\mathsf{Bel}_A(\mathsf{newText}(\mathrm{MSG}, A))) \qquad (1) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{u}, A))) \qquad (2) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)])) \qquad (3) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{Agent}(B)) \qquad (4) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathrm{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (5) \\
\mathscr{B}\ \mathsf{Bel}_I\ (\mathsf{Agent}(A)) \qquad (6) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathrm{Trust}(I, A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]))) \qquad (7) \\
\mathscr{B}\ \mathsf{Bel}_I\ (E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (8) \\
\mathscr{B}\ \mathsf{Transmit}\ (\mathrm{im}(I, B), A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]) \qquad (9) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(\mathrm{Trust}(A, B, \\
\qquad E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]))) \qquad (10) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{isKey}(\mathsf{key}(\mathsf{a}, \mathsf{i}, A))) \qquad (11) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(\\
\qquad D[\mathsf{key}(\mathsf{a}, \mathsf{i}, A) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{MSG}, A)]])) \qquad (12) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathrm{im}(I, B), \mathsf{Text}(\mathrm{MSG}, A)) \qquad (13) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathrm{Trust}(I, A, \mathsf{Text}(\mathrm{MSG}, A)))) \qquad (14) \\
\mathscr{B}\ \mathsf{Bel}_I\ (\mathsf{Text}(\mathrm{MSG}, A)) \qquad (15) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathsf{newText}(\mathrm{Bogus}, I))) \qquad (16) \\
\mathscr{B}\ \mathsf{Transmit}\ (\mathrm{im}(I, B), A, \mathsf{Text}(\mathrm{Bogus}, I)) \qquad (17) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(\mathrm{Trust}(A, B, \mathsf{Text}(\mathrm{Bogus}, I)))) \qquad (18) \\
\mathscr{B}\ \mathsf{Bel}_A\ (\mathsf{Text}(\mathrm{Bogus}, I)) \qquad (19) \\
\mathscr{B}\ \mathsf{Enforce}_A\ (\mathsf{Bel}_A(E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)])) \qquad (20) \\
\mathscr{B}\ \mathsf{Transmit}\ (A, \mathrm{im}(I, B), E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]) \qquad (21) \\
\mathscr{B}\ \mathsf{Enforce}_I\ (\mathsf{Bel}_I(\mathrm{Trust}(I, A, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]))) \qquad (22) \\
\mathscr{B}\ \mathsf{Bel}_I\ (E[\mathsf{key}(\mathsf{a}, \mathsf{u}, A) : \mathsf{Text}(\mathrm{Bogus}, I)]) \qquad (23) \\
\mathscr{B}\ \varepsilon]
\end{array}
$$

Fig. 6. Automated refinement of the attack on Shamir Rivest Adelman.

$$
\begin{array}{l}
\mathscr{M} \vDash_{\mathscr{C}_{\mathrm{end}}} \mathsf{Bel}_{\mathrm{Malice}}(\mathbb{E}(\mathrm{Malice}, \mathsf{protocol}[\mathrm{SRA}\ \mathrm{attack2}, 1, \\
\mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}) \wedge \mathsf{role}(\mathrm{Malice}), \mathsf{role}(I), \mathsf{role}(A), \\
\qquad \mathsf{Transmit}(\mathrm{Alice}, \mathrm{im}(\mathrm{Malice}, \mathrm{Bob}), \\
\qquad\quad \mathsf{Text}(\mathrm{MSG}, \mathrm{Alice}))\, \mathscr{B}\, \Phi'])))
\end{array}
$$

where the protocol header includes the protocol name $\mathrm{SRA}\ \mathrm{attack2}$, the session number 1, instantiated with the three agents Alice, Bob,Malice, playing the roles A, B, I respectively. Malice can only play the attacker role I, and there is one start role A. Finally Φ' denotes the rest of the instantiated attack protocol. In the configuration $\mathscr{C}_{\mathrm{end}}$, the inbuffer of Malice contains the message:

$$
\begin{array}{l}
\mathsf{msg}\ E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathrm{Alice}) : E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathrm{Alice}) : \mathsf{Text}(\mathrm{MSG}, \mathrm{Alice})]] \\
\quad \mathsf{from}\ \mathrm{Alice}\ \mathsf{to}\ \mathrm{im}(\mathrm{Malice}, \mathrm{Bob})
\end{array}
$$

which indicates that Alice has responded to the second session of the protocol, the request $(R.2.1)$, with the encryption $E(K_A : E(K_A : M))$, originally intended to be sent from Alice to Bob. Alice expected $E(K_B : M)$, yet she has no way of discovering that she was fooled and instead got the cipher text $E(K_A : M)$. Hence we have both

$$
\begin{array}{l}
\mathscr{M} \vDash_{\mathscr{C}_{\mathrm{end}}} \mathsf{Bel}_{\mathrm{Alice}}(\mathbb{E}(\mathrm{Alice}, \mathsf{protocol}[\mathrm{SRA}, 1, \ldots, \\
\mathsf{Transmit}(\mathrm{Bob}, \mathrm{Alice}, \\
\qquad E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathrm{Alice}) : \mathsf{Text}(\mathrm{MSG}, \mathrm{Alice})])\, \mathscr{B}\, \Phi''])))
\end{array}
$$

and

$$
\begin{array}{l}
\mathscr{M} \vDash_{\mathscr{C}_{\mathrm{end}}} \mathsf{Bel}_{\mathrm{Alice}}(\mathbb{E}(\mathrm{Alice}, \mathsf{protocol}[\mathrm{SRA}, 2, \ldots, \\
\mathsf{Transmit}(\mathrm{Bob}, \mathrm{Alice}, E[\mathsf{key}(\mathsf{a}, \mathsf{u}, B^{\%}) : \\
\qquad E[\mathsf{key}(\mathsf{a}, \mathsf{u}, \mathrm{Alice}) : \mathsf{Text}(\mathrm{MSG}, \mathrm{Alice})]])\, \mathscr{B}\, \Phi'''])))
\end{array}
$$

where the header is supressed and where Φ'' and Φ''' denote the respective instantiated protocol tails. In the first session $(\mathrm{SRA}, 1)$, Alice plays the responder role B, while the second session $(\mathrm{SRA}, 2)$ she plays the initiator A. Thus in the first session Alice does never receive the appropriate final message instance from Bob or by Malice impersonating Bob. In the second session Alice is waiting

for an instance of the second message (SRA_2), which never appear in Alice's inbuffer, Malice is blocked by the attack clause ($R.2.2$). Reachability analysis of the attack shows that there are no more simulations than the one previously described. In other words, reachability analysis shows that

$$\mathscr{M} \vDash_{\mathscr{C}_{\mathrm{init}}} \boxdot\neg\, \mathsf{Bel}_{\mathrm{Malice}}\,(\mathsf{Done}(\mathrm{SRAattack2}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}) \wedge \mathsf{role}(\mathrm{Malice}))),$$

hence the attack can never succeed. The fact that Alice is not able to finalize any intended session is also a question of reachability, it can justified (by the model checker in Maude) that

$$\mathscr{M} \vDash_{\mathscr{C}_{\mathrm{init}}} \boxdot\neg\, \mathsf{Bel}_{\mathrm{Alice}}\,(\mathsf{Done}(\mathrm{SRA}, \mathsf{role}(\mathrm{Alice}) \wedge \mathsf{role}(\mathrm{Bob}))).$$

To conclude, the attack contains two severe errors, and there is no obvious way to repair the specification.

VII. Connecting validation and simulation

There is a connection between static validation and simulations of attack protocols: if the simulation of an attack specification P shows that the attack can be executed, then the validation algorithm will succeed for P too.

The expression $\mathbb{v}(\Re^{\star}(\mathsf{P})) = \mathsf{Text}(\text{“No error found”})$ is a bit cumbersome to work with inside proofs, hence we shall stipulate that $\mathsf{Text}(\text{“No error found”}) = \varepsilon$.

Lemma 2: Let $\mathcal{R}_{\mathrm{A}}$ denote the set of operational rules for performing assumptions.

(i) If $\mathbb{v}(\Phi, \varepsilon) = \varepsilon$, then $\mathbb{v}(\Phi \frown (F \mathscr{B} \varepsilon), \varepsilon) = \mathbb{v}(\Phi, \varepsilon) \frown \mathbb{v}(F \mathscr{B} \varepsilon, \varepsilon)$ if $F = \mathsf{Bel}_a(\mathsf{Agent}(b))$ or $F = \mathsf{Enforce}_a(\mathsf{Bel}_a(\varphi))$ or $F = \mathsf{Transmit}(a, b, \varphi)$

(ii) Let $\mathsf{P} = \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \varphi \mathscr{B} \Phi\,]$ be an executable protocol, and $\mathbb{v}(\Psi \frown (\varphi \mathscr{B} \varepsilon), \varepsilon) = \mathsf{Text}(\text{“No error found”})$, and $\varphi \mathscr{B} \Phi \longmapsto_r^1 \Phi$ denotes one application of a rule $r \in \mathcal{R}_{\mathrm{A}}$, then $\mathbb{v}(\varphi \mathscr{B} \Phi, \Psi) = \mathbb{v}(\Phi, \Psi \frown (\varphi \mathscr{B} \varepsilon))$.

Proof: Part (i) is proven by obvious application of definition 7. Part (ii) follows by thorough of each of the rules $r \in \mathcal{R}_{\mathrm{A}}$ and definition 7. ■

Theorem 3: Let $\mathsf{P} = \mathsf{protocol}[\mu, N, \xi^{\mathcal{T}}, \xi^{\mathcal{A}}, \xi^{\mathcal{S}}, \Phi\,]$ be an intended protocol spesification, containing n distinct roles $\xi^{\mathcal{T}} = \bigwedge_{i=1}^{n} \mathsf{playRole}(a_i, x_i, \mu)$. Suppose that there exists a configuration $\mathscr{C}$ containing only the agents $a_1, \ldots a_n$ such that $\mathscr{M} \vDash_{\mathscr{C}} \bigwedge_{i=1}^{n} \mathsf{Bel}_{a_i}(\Re^{\star}(\mathsf{P}))$ and such that the agents in $\mathscr{C}$ does not possess any other protocol, and we have that $\mathscr{M} \vDash_{\mathscr{C}} \mathsf{Bel}_{\xi^{\mathcal{S}}}(\mathrm{start}(\mu, \xi^{\mathcal{T}}))$. If there exists a sequence of session numbers denoted $\vec{N} = \langle N_1, \ldots, N_n \rangle$, such that

$$\mathscr{M} \vDash_{\mathscr{C}} \diamondsuit_{\mathsf{F}} \Big(\bigwedge_{i=1}^{n} \mathsf{Bel}_{a_i}(\mathsf{Done}(\mu, \xi^{\mathcal{T}}, N_i))\Big) \text{ , then}$$

$$\mathbb{v}(\Re^{\star}(\mathsf{P})) = \mathsf{Text}(\text{“No error found”}).$$

Proof: (Sketch) Suppose that $\mathscr{C}$ is such a configuration that satisfies the premiss of the theorem, which means that the protocol P is executable. We prove that if $\mathscr{C} \overset{+}{\longmapsto} \mathscr{C}'$ such that $\mathscr{M} \vDash_{\mathscr{C}'} \bigwedge_{i=1}^{n} \mathsf{Bel}_{a_i}(\mathsf{Done}(\mu, \xi^{\mathcal{T}}, N_i))$, then we have that $\mathbb{v}(\Re^{\star}(\mathsf{P})) = \mathsf{Text}(\text{“No error found”})$. The strategy of the proof is to extend an executable protocol with piece by piece: hence the theorem is proven by induction on $lth(\mathsf{P})$. The result follows by application of lemma 2 and definition 7. ■

Theorem 4: Let P be a specification of a protocol as described in Theorem 3. Suppose furthermore that $\mathsf{P}^{\mathrm{A}} = \mathsf{protocol}[\mu^{\mathrm{A}}, N, \xi_{\mathrm{A}}^{\mathcal{T}}, \xi_{\mathrm{A}}^{\mathcal{A}}, \xi_{\mathrm{A}}^{\mathcal{S}}, \Phi\mathrm{A}\,]$ is an attack specification. Suppose that there exists a configuration $\mathscr{C}$ containing only the agents $a_1, \ldots a_n, a_{n+1}$, with the requirement that $\mathscr{M} \vDash_{\mathscr{C}} \bigwedge_{i=1}^{n} \mathsf{Bel}_{a_i}(\Re^{\star}(\mathsf{P}))$ such that the agents $a_1, \ldots a_n$ does not possess any other protocol than P. Suppose that the attacker a_{n+1} satisfies the following: $\mathscr{M} \vDash_{\mathscr{C}} \mathsf{Bel}_{a_{n+1}}(\Re^{\star}(\mathsf{P}^{\mathrm{A}}))$. If either there exists a $j \in \{1, \ldots, n\}$ such that $\mathscr{M} \vDash_{\mathscr{C}} \mathsf{Bel}_{a_j}(\mathrm{start}(\mu, \xi^{\mathcal{T}}))$ or $\mathscr{M} \vDash_{\mathscr{C}} \mathsf{Bel}_{a_{n+1}}(\mathrm{start}(\mu^{\mathrm{A}}, \xi_{\mathrm{A}}^{\mathcal{T}}))$, and there exists a sequence of session numbers $\vec{N} = \langle N_1, \ldots, N_n \rangle$, such that

$$\mathscr{M} \vDash_{\mathscr{C}} \diamondsuit_{\mathsf{F}} (\mathsf{Bel}_{a_{n+1}}(\mathsf{Done}(\mu^{\mathrm{A}}, \xi^{\mathcal{T}}, N_{n+1})) \wedge \mathsf{playRole}\,(a_{n+1}, w, \mu^{\mathrm{A}}) \wedge (\textstyle\bigwedge_{i=1}^{n} \mathsf{Bel}_{a_i}(\mathsf{Done}(\mu, \xi^{\mathcal{T}}, N_i)))),$$

then $\mathbb{v}(\Re^{\star}(\mathsf{P}^{\mathrm{A}})) = \mathsf{Text}(\text{“No error found”})$.

Proof: The proof is settled by a similar argument as the proof of Theorem 3. ■

A natural question posed by Theorem 3 and Theorem 4 is: does simulation make validation superflous, since the theorems imply that every error discovered by validation is also discovered by simulation? The answer is no, simulation always regards several agents running concurrently, and it is therefore harder to understand an unsuccessful simulation than a failure report from validation. An error discovered by validation points to the exact place in the attack description where something is wrong.

VIII. Conclusion

Several errors have been found in the most frequently cited library on authentication protocols. A close investigation revealed that errors migrate from original papers to the report by Clark/Jacob, and from the report to SPORE and papers on protocol analysis. This show that flaws in protocol attacks do occur and that they can be hard to discover by humans. Our experience indicates that attack descriptions should be described as accurately as protocols and their correctness should be analyzed as formally as protocols.

Then we presented the underlying algorithms for static validation of attack descriptions. Attack specifications were refined in an automated way and the resulting refined descriptions were then validated in order to find ungrounded beliefs. If such beliefs were found then the specifications where considered invalid, and a break point was given. Although the validation algorithm was used to find several errors in Clark/Jacob, it can not find several protocol jumps. The reason is that attack descriptions containing protocol jumps might not be discovered through

the validation algorithm. But fortunately, every protocol jump can be discovered through simulation.

Finally we have shown how a simulator for executing security protocols can be extended to execute attacks on these protocols. The attack simulator has proven to be useful in checking the correctness of attacks: the second attack on the Shamir Rivest Adelman protocol could only be detected by simulation, not by static validation [17]. A recent result relating validation and simulation was presented at the end showing that validation can be properly embedded into simulation.

ACKNOWLEDGEMENT

Thanks to Lothar Fritsch, Habtamu Abie, Peter Csaba Ölveczky, Olaf Owe, Chik How Tan, David Basin, Atle Refsdal, Thor Kristoffersen, Alfredo Pironti, Bjarte M. Østvold, Truls Fretland, Wolfgang Leister, Joakim Bjørk and an referee for comments on earlier drafts of this paper.

REFERENCES

[1] Martín Abadi and Roger Needham. Prudent Engineering Practice for Cryptographic Protocols. *IEEE Transactions on Software Engineering*, 22(1):6–15, 1996.

[2] Ross Anderson and Roger Needham. Programming Satan's Computer. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 426–441. Springer-Verlag, 1995.

[3] Michael Backes, Sebastian Mödersheim, Birgit Pfitzmann, and Luca Viganò. Symbolic and Cryptographic Analysis of the Secure WS-Reliable Messaging Scenario. In *Proceedings of FOSSACS 2006*, volume 3921 of *Lecture Notes in Computer Science*, pages 428–445. Springer-Verlag, 2006.

[4] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, June 2005.

[5] Chiara Bodei, Pierpaolo Degano, Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Techniques for Security Checking: Non-Interference vs Control Flow Analysis,. In *Proceedings of the Theory of Concurrency, Higher Order and Types Workshop*, volume 62. Electronic Notes in Computer Science, Elsevier, 2001. Udine (Italy).

[6] Stephen H. Brackin. Evaluating and Improving Protocol Analysis by Automatic Proof. In *IEEE Computer Security Foundations Workshop (CSFW)*, pages 138–152, 1998.

[7] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication. Technical report, february 1989. Report 39, Digital Systems Research Center.

[8] Carlos Caleiro. Deconstructing Alice and Bob. In P. Degano and L. Viganò, editors, *Proceeding of the Second Workshop on Automated Reasoning for Security Protocol Analysis*, pages 3–22. Electronic Notes in Computer Science, 2005.

[9] John Clark and Jeremy Jacob. A Survey of Authentication Protocol Literature, 1997. Version 1.0, Unpublished Report, University of York, `http://cs.york.ac.uk/~jac/papers/drareview.ps.gz`.

[10] Ricardo Corin, Sandro Etalle, and Ari Saptawijaya. A logic for constraint-based security protocol analysis. In *Security and Privacy*, pages 155–168. IEEE Computer Society Press, 2006.

[11] Dorothy E. Denning and Giovanni M. Sacco. Timestamps in Key Distribution Protocols. *Comm. of the ACM*, 24(8):533–536, 1981.

[12] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–207, 1983.

[13] Ben Donovan, Paul Norris, and Gavin Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*. IEEE Computer Society Press, 1999.

[14] Laboratoire Spécification et Vérification. SPORE Security Protocol Open REpository. `http://www.lsv.ens-cachan.fr/spore/`.

[15] Andrew Gordon and Alan Jeffrey. Types and effects for asymmetric cryptographic protocols. In *15th IEEE Computer Security Foundations Workshop (CSFW 15), Cape Breton*, pages 77–91. IEEE Press, 2002.

[16] Anders Moen Hagalisletto. Attacks are Protocols Too. In *Proceedings of The Second International Conference on Availability, Reliability and Security (IEEE ARES 2007)*, pages 1197 – 1206. Workshop WAIS 2007.

[17] Anders Moen Hagalisletto. Errors in Attacks on Authentication Protocols. In *Proceedings of the Second International Conference on Availability, Reliability and Security (ARES 2007)*, pages 223 – 229. IEEE Computer Society.

[18] Anders Moen Hagalisletto. Validating Attacks on Authentication Protocols. In *Proceedings of the 12th IEEE Symposium on Computers and Communications - (ISCC 2007)*.

[19] Anders Moen Hagalisletto. Automated Refinement of Security Protocols. In *Workshop SSN in the Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium*, 2006.

[20] Tzonelih Hwang, Narn-Yih Lee, Chuan-Ming Li, Ming-Yung Ko, and Yung-Hsiang Chen. Two attacks on Neuman-Stubblebine authentication protocols. *Information Processing Letter*, 53(2):103–107, 1995.

[21] Romain Janvier, Yassine Lakhnech, and Laurent Mazaré. Relating the symbolic and computational models of security protocols using hashes. In *Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, pages 67–89, 2006.

[22] Mahadev Satyanarayanan. Integrating security in a large distributed system. *ACM Trans. Comput. Syst.*, 7(3):247–280, 1989.

[23] Dawn Song, Sergey Berezin, and Adrian Perrig. Athena, a Novel Approach to Efficient Automatic Security Protocol Analysis. *Journal of Computer Security*, 9(1,2):47–74, 2001.

[24] Thomas Y. C. Woo and Simon S. Lam. A Lesson on Authentication Protocol Design. *Operating Systems Review*, 28(3):24–37, 1994.

Anders Moen Hagalisletto was born in 1969 and received his Bachelor's degree in mathematics and philosophy in 1993, Master's degree in philosophy in 1997, and Master's degree in computer science in 1999. He is currently working toward the Ph.D. degree. He has been working on topics in mathematical logic, in particular provability logic, and was cofounder and coeditor of Nordic Journal of Philosophical Logic. From 1999 to 2002 he was empoyed as research scientist at Norwegian Computing Center, and in 2001 he held a position as university lecturer at the Institute for Computer Science, University of Oslo. His current research interests include industrial appilcations of formal methods, in particular automated software engineering; Petri Nets models of railway systems and tools for testing and analyzing security protocols. He is currently working as a research scientist at Birkeland Innovation on building tools for automated integration of lightweight applications. He is also employed as a reseach scientist (20 % position) at Norwegian Computing Center.