Analysis of LySA-calculus with explicit confidentiality annotations

Han Gao, Hanne Riis Nielson
Informatics and Mathematical Modelling, Technical University of Denmark
Richard Petersens Plads bldg 322, DK2800 Kgs. Lyngby, Denmark
{hg,riis}@imm.dtu.dk

Abstract

Recently there has been an increased research interest in applying process calculi in the verification of cryptographic protocols due to their ability to formally model protocols. This work presents LySA with explicit confidentiality annotations for indicating the expected behavior of target protocols. A static analysis approach is developed for analyzing protocols specified in the extended LySA. The proposed approach will over-approximate the possible executions of protocols while keeping track of all messages communicated over the network, and furthermore it will capture the potential malicious activities performed by attackers as specified by the confidentiality annotations. The proposed analysis approach is fully automatic without the need of human intervention and has been applied successfully to a number of protocols.

1. Introduction

Secure communication over the Internet has been of rapidly growing interest. Assuring correctness of communication protocols by analysis and verification has been considered a way to improve communication security. The challenges of protocol analysis and verification are twofold. Firstly, protocols are often described informally using protocol narrations, which are imprecise about the finer details concerning the deployment of the protocol, and do not completely specify the actions internal to the principals [1]. Secondly, protocol definitions provide syntax and semantics of inter-operation, but does not specify the purpose of each message element used in a protocol, thereby making it difficult to capture the intension of communication behaviors.

In [3], a control flow analysis is developed to validate security properties against various attacks. The control flow analysis collects every single message that might be learnt by the attacker. The secrecy property is checked by inspecting this information. However, this does not always suffice, in particular, the attacker may break a protocol's execution without knowing any "secret". Thanks to the confidentiality annotations, our analysis is able to compute a much wider range of information, which greatly facilities the precise reasoning of a protocol's behavior. This is illustrated by the example of section 4.

This work reports our research that extends LySA calculus with explicitly annotations, specifying the confidentiality intentions of protocols. Based on the extended LySA, a control flow analysis is proposed, which approximates all the behaviors during the execution of protocols, including tracking the set of messages that are transferred over the network and recording the potential values of variables. Our study shows that, due to the over-approximative nature of the analysis, we are able to capture any malicious activities expressed in terms of annotation violations registered in an error component of the analysis. The proposed analysis can be applied to a wide range of cryptographic protocols. In contrast to model checking approaches, the analysis is fully automatic and termination is always guaranteed.

The paper is organized as follows. In Section 2, we present the LySA calculus with explicit confidentiality annotations. We introduce a control flow analysis in Section 3 and show how to deal with the hostile environment in Section 4. In Section 5 we conclude with an assessment of our approach and discuss some perspectives for future work.

2. The extended LySA-calculus

LySA is a process algebra, in the tradition of the π- [6] and Spi- [2] calculus. It inherits most of the features of its predecessors, while still keeping its own characteristics. One characteristic is the absence of channels: in LySA all processes have only access to a single global communication channel, the ether. The second characteristic is that tests associated with input and decryption are expressed using pattern matching.

Syntax LySA consists of terms and processes. The syntax of terms E and processes P is given in the Table 1.

$E ::=$	$terms$
n	name ($n \in \mathcal{N}$)
x	variable ($x \in \mathcal{X}$)
$\{E_1, \ldots, E_k\}^l_{E_0}[within\ \mathcal{L}]$	symmetric encryption ($k \geq 0$)
$P ::=$	$processes$
0	nil
$P_1 \vert P_2$	parallel composition
$(\nu\ n^l[within\ \mathcal{L}])P$	restriction
$!P$	replication
$decrypt\ E\ as\ \{E_1, \ldots, E_j; x_{j+1}^{l_{j+1}}[from\ \mathcal{L}_{j+1}], \ldots, x_k^{l_k}[from\ \mathcal{L}_k]\}_{E_0}\ in\ P$	symmetric decryption (with matching)
$(E_1, \ldots, E_j; x_{j+1}^{l_{j+1}}[from\ \mathcal{L}_{j+1}], \ldots, x_k^{l_k}[from\ \mathcal{L}_k]).P$	input (with matching)
$\langle E_1, \ldots, E_k \rangle.P$	output

Table 1. Syntax of extended LySA

Here $\mathcal{N}$ and $\mathcal{X}$ denote sets of names and variables, respectively. The name n is used to represent keys, challenges as well as names of principals. Encryptions are tuples of terms $E_1, \ldots, E_k$ encrypted under a shared key represented by the term E_0. We add to each encryption a label, $l \in Lab$, serving as an unique identity. Each encryption is decorated by an annotation $[within\ \mathcal{L}]$, where $\mathcal{L} \subseteq Lab$ indicates a set of labels of the variables to which the encryption may be bound. As we shall see shortly, this information will be used to specify the confidentiality intention of the protocol.

In addition to the classical constructs for composing processes, LySA also contains an input construct with matching and a decryption operation with matching. The idea behind the matching is that of a tuple we allow the match on a prefix of the tuple to values and then bind the remaining values to variables. At each of these binding occurrence we specify a set of labels, $[from\ \mathcal{L}]$, pointing to the values that may be bound to the variable. These annotations play a central role when specifying the confidentiality intension of the protocol.

Semantics Due to lack of space we omit a formal specification of the semantics of LySA with the confidentiality annotation.

Example We shall use the Wide Mouthed Frog protocol [4] (WMF) to illustrate how to encode protocols in our calculus.

WMF is a symmetric key management protocol aiming at establishing a secret session key K_{ab} between the two principals A and B sharing secret master keys K_A and K_B, respectively, with a trusted server S. The protocol is specified by the following narration:

$$\begin{array}{lll} 1. & A \rightarrow S: & \{B, K_{ab}\}_{K_A} \\ 2. & S \rightarrow B: & \{A, K_{ab}\}_{K_B} \\ 3. & A \rightarrow B: & \{Msg\}_{K_{ab}} \end{array}$$

In the first message A sends to S its name, and then a fresh key K_{ab} and the name of the intended receiver B, encrypted under the key K_A. In the second one, S forwards the key and the sender name A to B, encrypted under the key K_B. Finally, A sends B the message Msg encrypted under the session key K_{ab}.

The LySA specification of the WMF protocol is:

$$\begin{array}{ll} 1. & (\upsilon\ K_{ab}^{l_{Kab}}[within\ \{l_k, l_{k'}\}]) \\ & \quad \langle A, S, \{B, K_{ab}\}^{l_1}_{K_A}[within\ Lab]\rangle. \\ 3. & \quad (\upsilon\ Msg^{l_{Msg}}[within\ \{l_{zm}\}]) \\ & \quad \langle A, B, \{Msg\}^{l_2}_{K_{ab}}[within\ Lab]\rangle.0 \\ 2'. & \mid (S, B; y^{l_y}[from\ Lab]). \\ 2''. & \quad decrypt\ y\ as\ \{A; k^{l_k}[from\ \{l_{Kab}\}]\}_{K_B}\ in \\ 3'. & \quad\quad (A, B; z^{l_z}[from\ Lab]). \\ 3''. & \quad\quad decrypt\ z\ as\ \{; z_m^{l_{zm}}[from\ \{l_{Msg}\}]\}_k\ in\ 0 \\ 1'. & \mid (A, S; p^{l_p}[from\ Lab]). \\ 1''. & \quad decrypt\ p\ as\ \{B; k'^{l_{k'}}[from\ \{l_{Kab}\}]\}_{K_A}\ in \\ 2. & \quad\quad \langle S, B, \{A, k'\}^{l_3}_{K_B}[within\ Lab]\rangle.0 \end{array}$$

During the execution of the protocol, leaking some crucial values may cause the protocol to be broken. Therefore we have to carefully check the flow of those values, which, in this protocol, include K_{ab}, Msg. K_{ab} is first received by S and bound to the variable k' (in step $1''$) and then received by B and bound to variable k (in step $2''$). It is natural to require that k and k' are the only variables allowed to bind to K_{ab}, whereas K_{ab}, on the other hand, is only allowed to be bound to k and k'. This intension is clarified by: (i) K_{ab}

has the annotation $[within\ \{l_k, l_{k'}\}]$ and (2) the annotations of k' and k are $[from\ \{l_{Kab}\}]$.

Similarly for Msg, the annotations of Msg and z_m clearly specify that only binding Msg to z_m is allowed in the protocol execution.

Unlike K_{ab} and Msg, some values are not necessary to keep secret, e.g. encryptions $\{A, k'\}^{l_3}_{K_B}$ and $\{B, K_{ab}\}^{l_1}_{K_A}$, which are then allowed to be bound to any variable. If we use Lab as a shortcut to the set containing all the labels occurred in the protocol specification, their annotations are $[within\ Lab]$.

3. Control Flow Analysis

The aim of the analysis is to give a safe over-approximation of all possible behaviors of target protocols; this will include the possible messages communicated over the network and the possible value bindings of the variables. At each variable binding occurrence, the analysis will check whether the binding is allowed according to the annotations. Each illegal binding will be regarded as a violation and recorded in an error component of the analysis.

Analysis of terms For each term E, the analysis will determine a superset of the possible *abstract values* it may evaluate to. The judgement for expressions takes the form $\rho, \sigma \models_\gamma E : \vartheta$ where $\vartheta \subseteq \mathcal{P}((Lab \times \mathcal{P}(Lab))^*)$ is an acceptable estimate of the set of *abstract values*.

The *abstract values* of variables and encryptions are of the different form and therefore collected in two different global environments:

- $\rho : Lab \to \mathcal{P}((Lab) \times \mathcal{P}(Lab))$ maps the label of each variable to its potential *abstract values*, expressed as sets of pairs. Each pair consists of a label of value it may be bound to and the associated $within/from$ label sets.

- $\sigma : Lab \to \mathcal{P}((Lab \times \mathcal{P}(Lab))^*)$ maps the label of each encryption to its *abstract values*. An encryption can be viewed as composed by sub-terms, consequently its *abstract values* preserves the original structure but with each sub-term replaced by its *abstract values*.

Besides ρ and σ, another environment γ is employed in the analysis for recording the label and annotation of each simple term (i.e. name and variable).

- $\gamma : (N \cup V) \to \mathcal{P}((Lab) \times \mathcal{P}(Lab))$ maps the label of each simple term to a pair of its label and $within/from$ label set.

Analysis of processes The judgement for processes has the form: $(\rho, \kappa, \sigma) \models_\gamma P : \psi$ expressing that ρ, κ, σ and ψ are valid analysis estimates of process P.

ψ is the error-component which collects an over-approximation of the $within/from$ violations.

- $\psi \subseteq \mathcal{P}(Lab \times Lab)$: if $(l_i, l'_i) \in \psi$ and assume that these two labels are pointing to n and x then either n is not allowed to be bound to x or x is not allowed to bind to n when value binding takes place.

In the analysis of a process P we also collect which *abstract values* may flow on the network. To achieve this we make use of the abstract environment κ:

- $\kappa \subseteq \mathcal{P}((Lab \times \mathcal{P}(Lab))^*)$: includes all the message tuples that may flow on the network, where each element of the messages is represented by its *abstract value*.

The judgement is defined by the rules listed in the Table 2 and is explained below.

The first three rules in Table 2 describe that the analysis of terms returns sets of *abstract values*, ϑ, that they may evaluate to. This is used in the rule for output: first, all the expressions are evaluated and then it is required that all the combination of the *abstract values* found by this evaluation is recorded in κ. Finally, the continuation process must be analyzed.

The rule for input incorporates pattern matching and violation capturing. The pattern matching is dealt with by first evaluating all the of first j expressions in the input to be the sets ϑ_i for $i = 1, \ldots, j$. Next, if any of the sequences of length k in κ are such that the first j *abstract values* component-wise are included in ϑ_i then the match is concluded to possibly be successful. In this case, the remaining values of the k-tuple must be recorded in ρ as possible binding of the variables. At this point, the analysis checks the *within/from* assertions of the *abstract values* and variables to determine whether the binding is allowed by the annotations. Any violation will be recorded in ψ before the continuation process can be analyzed. Notice that the continuation process is analyzed base on the updated γ, which includes the label and *from* annotation of each variable occurred in the input.

The rule for analysis of decryption is quite similar to the analysis of input. Only, here the *abstract values* to be matched are found by evaluating the expression E into the set ϑ and then matching is performed against any k-ary encryption expression associated with ϑ. Notice that the key is matched due to the indices starting from 0 rather than from 1 as in communication. The analysis is identical to the analysis of input once the successfully matching values have been determined, meaning that whether the bindings are allowed is examined and violations are recorded.

$$\frac{\forall (l,\mathcal{L}) \in \gamma(n) : (l,\mathcal{L}) \in \vartheta}{\rho,\sigma \models_\gamma n : \vartheta} \qquad \frac{\forall (l,\mathcal{L}) \in \gamma(x) : \rho(l) \subseteq \vartheta}{\rho,\sigma \models_\gamma x : \vartheta}$$

$$\frac{\begin{array}{c}\wedge_{i=0}^{k}\, \rho,\sigma \models_\gamma E_i : \vartheta_i \ \wedge\ \forall (l_0,\mathcal{L}_0),\ldots,(l_k,\mathcal{L}_k) : \wedge_{i=0}^{k}\,(l_i,\mathcal{L}_i) \in \vartheta_i \Rightarrow \\ (l,\mathcal{L}) \in \vartheta \ \wedge\ \{(l_1,\mathcal{L}_1),\ldots,(l_k,\mathcal{L}_k)\}_{(l_0,\mathcal{L}_0)} \in \sigma(l)\end{array}}{\rho,\sigma \models_\gamma \{E_1,\ldots,E_k\}^{l}_{E_0}[within\ \mathcal{L}] : \vartheta}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{k}\, \rho,\sigma \models_\gamma E_i : \vartheta_i \ \wedge \forall (l_1,\mathcal{L}_1),\ldots,(l_k,\mathcal{L}_k) : \wedge_{i=1}^{k}(l_i,\mathcal{L}_i) \in \vartheta_i \Rightarrow \\ \langle (l_1,\mathcal{L}_1),\ldots,(l_k,\mathcal{L}_k)\rangle \in \kappa \ \wedge\ (\rho,\kappa,\sigma) \models_\gamma P : \psi\end{array}}{(\rho,\kappa,\sigma) \models_\gamma \langle E_1,\ldots,E_k\rangle.P : \psi}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{j}\, \rho,\sigma \models_\gamma E_i : \vartheta_i \ \wedge\ \forall \langle (l'_1,\mathcal{L}'_1),\ldots,(l'_k,\mathcal{L}'_k)\rangle \in \kappa : \wedge_{i=1}^{j}\,(l'_i,\mathcal{L}'_i) \in \vartheta_i \Rightarrow \\ \wedge_{i=j+1}^{k}\,(l'_i,\mathcal{L}'_i) \in \rho(l_i) \ \wedge \wedge_{i=j+1}^{k} \neg RM(l_i,\mathcal{L}_i,l'_i,\mathcal{L}'_i) \Rightarrow (l_i,l'_i) \in \psi \ \wedge\ (\rho,\kappa,\sigma) \models_{\gamma'} P : \psi\end{array}}{\begin{array}{c}(\rho,\kappa,\sigma) \models_\gamma (E_1,\ldots,E_j; x_{j+1}^{l_{j+1}}[from\ \mathcal{L}_{j+1}],\ldots,x_k^{l_k}[from\ \mathcal{L}_k]).P : \psi \\ \text{where } \gamma' = \gamma[x_{j+1} \mapsto (l_{j+1},\mathcal{L}_{j+1}),\ldots,x_k \mapsto (l_k,\mathcal{L}_k)]\end{array}}$$

$$\frac{\begin{array}{c}\rho,\sigma \models_\gamma E : \vartheta \ \wedge\ \wedge_{i=0}^{j}\ \rho,\sigma \models_\gamma E_i : \vartheta_i \ \wedge \\ \forall (l',\mathcal{L}'),(l'_0,\mathcal{L}'_0),\ldots,(l'_k,\mathcal{L}'_k) : (l',\mathcal{L}') \in \vartheta \ \wedge\ \{(l'_1,\mathcal{L}'_1),\ldots,(l'_k,\mathcal{L}'_k)\}_{(l'_0,\mathcal{L}'_0)} \in \sigma(l') \ \wedge\ \wedge_{i=0}^{j}\ (l'_i,\mathcal{L}'_i) \in \vartheta_i \Rightarrow \\ \wedge_{i=j+1}^{k}\,(l'_i,\mathcal{L}'_i) \in \rho(l_i) \ \wedge \wedge_{i=j+1}^{k} \neg RM(l_i,\mathcal{L}_i,l'_i,\mathcal{L}'_i) \Rightarrow (l_i,l'_i) \in \psi \ \wedge\ (\rho,\kappa,\sigma) \models_{\gamma'} P : \psi\end{array}}{\begin{array}{c}(\rho,\kappa,\sigma) \models_\gamma decrypt\ E\ as\ \{E_1,\ldots,E_j; x_{j+1}^{l_{j+1}}[from\ \mathcal{L}_{j+1}],\ldots,x_k^{l_k}[from\ \mathcal{L}_k]\}_{E_0}.P : \psi \\ \text{where } \gamma' = \gamma[x_{j+1} \mapsto (l_{j+1},\mathcal{L}_{j+1}),\ldots,x_k \mapsto (l_k,\mathcal{L}_k)]\end{array}}$$

$$(\rho,\kappa,\sigma) \models_\gamma 0 : \psi \qquad \frac{(\rho,\kappa,\sigma) \models_\gamma P_1 : \psi \ \wedge\ (\rho,\kappa,\sigma) \models_\gamma P_2 : \psi}{(\rho,\kappa,\sigma) \models_\gamma P_1|P_2 : \psi}$$

$$\frac{(\rho,\kappa,\sigma) \models_{\gamma[n \mapsto (l,\mathcal{L})]} P : \psi}{(\rho,\kappa,\sigma) \models_\gamma (\upsilon\ n^l[within\ \mathcal{L}])P : \psi} \qquad \frac{(\rho,\kappa,\sigma) \models_\gamma P : \psi}{(\rho,\kappa,\sigma) \models_\gamma !P : \psi}$$

$$RM(l,\mathcal{L},l',\mathcal{L}') \stackrel{def}{=} l \in \mathcal{L}' \ \wedge\ l' \in \mathcal{L}$$

Table 2. Analysis of terms and processes

The rules for restriction first updates γ to contain the label and annotation information of n and then requires that ρ, κ and σ are valid analysis estimates of process P given the updated γ.

The rules for the inactive process, parallel composition and replication are straightforward.

Example Initially, γ has entries:

$$\begin{array}{lllll}\gamma: & A \mapsto (l_A, Lab) & B \mapsto (l_B, Lab) & S \mapsto (l_S, Lab) \\ & K_A \mapsto (l_{Ka}, \emptyset) & K_B \mapsto (l_{Kb}, \emptyset)\end{array}$$

In order to assure the secrecy, the long term keys, K_A and K_B are not allowed to be bound to any variable. So both $\gamma(K_A)$ and $\gamma(K_B)$ have $\emptyset$ in their second component. On the other hand, the names A, B and S of the principals will be known to every body so we do not impose any restriction on the bindings.

The analysis of the WMF protocol gives:

$$(\rho,\kappa,\sigma) \models_\gamma WMF : \emptyset$$

where ρ, κ and σ have these non-empty entries

$$\begin{array}{lllll}\rho: & l_y & \mapsto \{(l_3, Lab)\} & l_k & \mapsto \{(l_{Kab}, \{l_k, l_{k'}\})\} \\ & l_z & \mapsto \{(l_2, Lab)\} & l_{zm} & \mapsto \{(l_{Msg}, \{l_{zm}\})\} \\ & l_p & \mapsto \{(l_1, Lab)\} & l_{k'} & \mapsto \{(l_{Kab}, \{l_k, l_{k'}\})\}\end{array}$$

$$\begin{array}{ll}\kappa: & \{\langle (l_A, Lab), (l_S, Lab), (l_1, Lab)\rangle\} \cup \\ & \{\langle (l_A, Lab), (l_B, Lab), (l_2, Lab)\rangle\} \cup \\ & \{\langle (l_S, Lab), (l_B, Lab), (l_3, Lab)\rangle\}\end{array}$$

$$\begin{array}{ll}\sigma: & l_1 \mapsto \{(l_B, Lab), (l_{Kab}, \{l_k, l_{k'}\})\}_{(l_{Ka}, \emptyset)} \\ & l_2 \mapsto \{(l_{Msg}, \{l_{zm}\})\}_{(l_{Kab}, \{l_k, l_{k'}\})} \\ & l_3 \mapsto \{(l_A, Lab), (l_{Kab}, \{l_k, l_{k'}\})\}_{(l_{Kb}, \emptyset)}\end{array}$$

As mentioned before, ρ maps the label of each variable to its *abstract value*. The analysis indicates that e.g. variable y is bound to the encryption with label l_3. By inspecting the entries of κ, it can be concluded that total three message tuples may flow on the network. For example, $\langle (l_A, Lab), (l_S, Lab), (l_1, Lab)\rangle$ corresponds to the message that A sends to S an encryption labelled l_1. Furthermore we can deduce the *abstract value* of l_1 by looking up the σ: l_1 points to an encryption encrypted using the

value K_a and the encrypted values are identified by l_B and l_{Kab}.

4. Modelling the Attacker

Protocols are always executed in an environment where malicious attackers may exist. For describing the capability of the attacker, we adopt the Dolev-Yao condition. The attacker can perform the following actions: (1) Eavesdrop all messages sent on the network; (2) Decrypt messages if he knows the key; (3) Construct new encryption using the keys known; (4) Construct and send to the network new messages and (5) Generate his own names.

Example We analyze the WMF protocol and the least solution has an empty ψ-component, i.c. $\psi = \emptyset$.

The session key, however, is not secret any long if a long term key, say K_B, is leaked. In this case, the analysis result has a non-empty ψ-component, which contains the following pairs: $(l_k, l_\bullet)$ and $(l_{zm}, l_\bullet)$[1]. Since K_B is leaked, now the attacker is able to decrypt y and learns the value of k, which is indicated by $(l_k, l_\bullet)$. As a result, he can continue to decrypt z and hence knows the value of z_m. This is captured by $(l_{zm}, l_\bullet)$.

These experiments show that the explicit confidentiality annotations greatly help the analysis to capture unwanted variable bindings and hence detect malicious behaviors.

5. Conclusion

We have extended LySA with *within/from* assertions and show that protocol narrations may be formalized as extended LySA processes such that the static analysis presented before can pinpoint a wide variety of confidentiality errors in communication protocols.

The advantages of the proposed approach is twofold. Firstly, we identify the need for clarifying the protocol intensions, i.e. "Is message *Msg* sent from *A* intended for *B*?", with the use of confidentiality annotations. The annotations provide LySA calculus extra syntax and semantics that can be reasoned about protocol behaviors and capture malicious behaviors.

Secondly, we show how to take advantages of the annotations when specifying the control flow analysis. Facilitated by the annotations, the control flow analysis is able to pinpoint the secrecy and integrity of each term, thus provides more detail information about protocol executions. Those information could be combined with other security mechanism, say trust management, to form a more complex system.

Future work will focus on the following aspects:

The calculus LySA calculus was developed after the Spi-calculus but it has been designed to facilitate obtaining useful information from a relatively unsophisticated static analysis, especially control flow analysis. Extensions of the LySA calculus enables the analysis to deal directly with some of the security properties.

As far as analysis is concerned, a perfect view of cryptography is taken and only attacks can be expressed in extended LySA are considered. In the context of conference key protocols, *power* and *mod* functions are usually employed to compute the group keys. Since these functions are not present in the extended LySA we cannot model the protocols using them. Therefore we would like add additional constructs to the extended LySA in order to handel mathematics operations.

The security properties LySA has its own build-in mechanism to handle authentication properties based on *origin/destination* annotations. In this paper, we have focused on the analysis of confidentiality properties which is facilitated by adding *within/from* annotations to the syntax of LySA. Moving further in the direction of annotations we may add beliefs in the style of BAN logic [4]. For example, we would like to add annotations to the creation of new nonces and keys about their intended use, e.g. $(\upsilon\ n[A \rightarrow B])$ and $(key[A \rightarrow B])$ might denote that the creation of a nonce n is intended to establish an authentic connection from A to B and the key key is used to encrypt messages sending from A to B.

References

[1] M. Abadi. Security protocols and specifications. *In Proc. of FoSSaCS'99*, pages 1–13, 1999.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols the spi calculus. *SRC - Research Report*, 149, 1998.

[3] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Automatic validation of protocol narration. *Proceedings of the 16th Computer Security Foundations Workshop*, pages 126–140, 2003.

[4] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, pages 18–36, 1990.

[5] F.Nielson, H.Seidl, and H.R.Nielson. A succinct solver for alfp. *Nordic Journal of Computing*, 9:335–372, 2002.

[6] R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge University Press, 1999.

[7] H. R. Nielson and F.Nielson. Flow logic: A multi-paradigmatic approach to static analysis. *Lecture Notes in Computer Science*, 2566:223–244, 2002.

[1] here $l_\bullet$ is the label of the variables in the attacker's knowledge