Automated Analysis of Infinite Scenarios

Mikael Buchholtz*

Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads, bldg. 321, DK-2800 Kgs. Lyngby, mib@imm.dtu.dk

Abstract. The security of a network protocol crucially relies on the scenario in which the protocol is deployed. This paper describes syntactic constructs for modelling network scenarios and presents an automated analysis tool, which can guarantee that security properties hold in all of the (infinitely many) instances of a scenario. The tool is based on control flow analysis of the process calculus LySa and is applied to the Bauer, Berson, and Feiertag protocol where is reveals a previously undocumented problem, which occurs in some scenarios but not in other.

1 Introduction

The security of any network protocol is not only determined by the behaviour the protocol itself. The security additionally relies upon the scenario in which the protocol is deployed. A protocol that suffers from a parallel session attack is an example of this: It may be possible to show that the protocol is secure — even when it is under attack — in the case that only a single session of the protocol is deployed on the network. However, when multiple session are present on the network, the parallel session attack can occur because the attacker uses messages from one session to perform the attack on another session.

This paper presents an automated analysis tool that focuses on deployment scenarios for security protocols. Protocols will be modelled in the process calculus LySa [3] and analysed with a control flow analysis that can guarantee confidentiality and authentication properties. The syntax of a LySa process, P, has features for modelling cryptography, nonce generation, message passing, etc. These features are well-suited to model the internal behaviour of individual principals. In this paper, we furthermore want to model the scenarios in which these principals appear. To this end, LySa is extended with a meta-level that contains various indexing constructs. For example, the meta-level has an indexing parallel composition, $|_{i \in S}\ M$, that describes a number of meta-level processes M in parallel. These processes only differ in their index i and, thus, the construct can be used to describe principals $A_1, A_2, A_3, \ldots$ that only differ in their identity but otherwise follow the same protocol. A key idea in having the meta-level is that it should *not* simply be a syntactic shorthand for succinct modelling of

* This work is funded by the Information Society Technologies programme of the European Commission, Future and Emerging Technologies, under the IST-2001-32072 project DEGAS.

one specific instance of a scenario. Instead, a meta-level process specifies all the different instances that a protocol may be deployed in.

Each instance of a scenario will be described by an object-level process P, which is an ordinary LySa process without indexing constructs. Instantiation is described by an instantiation relation, $M \Rrightarrow P$, and meta-level process may instantiate many different object-level processes. In fact, a meta-level is allowed to instantiate to infinitely many different object-level processes, thus, describing an infinitely large scenario.

A second step is to extend the control flow analysis from [3] to cover also meta-level processes. The control flow analysis is an automatable analysis technique that works by over-approximating the behaviour of a process. Thus, the extended meta-level analysis provides an automated analysis of scenarios and even copes with the fact that they may be infinitely large.

1.1 Contribution of this paper

This paper contributes with version 2 of the LySatool. Version 1 of the tool [7] is the implementation of the control flow analysis of object-level LySa [3]. The second version of the LySatool includes the analysis of the meta-level and, hence, caters for analysis of arbitrarily large scenarios. The overall idea of adding an analysable meta-level was first suggested in [8] but for a different calculus. This paper extends that work in several respects: it (1) gives an implementation of a meta-level analysis. To take advantage of an existing implementation of an object-level analysis the idea of a meta-level has been applied on the LySa calculus; (2) gives a more detailed treatment of the correctness of the analysis. The simpler format of LySa over the calculus in [8] simplifies the work needed to do this; (3) makes the meta-level useful both to check confidentiality *and* authentication properties; and (4) uses the LySatool to find a previously undiscovered problem in a classical key establishment protocol [1].

2 LySa and the Meta-Level

The process calculus LySa [3] is tailored to model network protocols that attain their security by means of cryptography. LySa models perfect cryptography, i.e. that successful decryption of ciphertext is only possible if the correct key is known. As of [4], LySa caters for both symmetric and asymmetric key cryptography. To keep the presentation simple, only symmetric key cryptography is presented in this paper. However, all the results in this paper, including the LySatool implementation, work also for asymmetric cryptography.

2.1 Syntax

The syntax of LySa is given in Figure 1. The syntax of expression $E \in \mathit{Expr}$ and processes $M \in \mathit{MProc}$ up until the horizontal double line corresponds closely to the syntax of LySa presented in [3]. This subset of processes is sometimes

$$
\begin{array}{lll}
mx & ::= & x_{\overline{i}} \\
E & ::= & n_{\overline{i}} \quad | \quad mx \quad | \quad \{E_1, \ldots, E_k\}_{E_0} \\
M & ::= & \langle E_1, \ldots, E_k \rangle.M \quad | \quad (E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k).M \quad | \\
 & & \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k\}_{E_0}\ \mathsf{in}\ M \quad | \\
 & & (\nu\, n_{\overline{i}})\, M \quad | \quad !M \quad | \quad M_1 \mid M_2 \quad | \quad \mathsf{0} \quad | \\
\hline
 & & \mathsf{let}\ X \subseteq S\ \mathsf{in}\ M \quad | \quad |_{i \in S}\ M \quad | \quad (\nu_{\overline{i} \in \overline{S}}\ n_{\overline{a}\overline{i}})M
\end{array}
$$

Fig. 1. The syntax of LySa including the meta-level.

referred to as *object-level* processes taken from the set *Proc* and is ranged over by P. The syntax below the horizontal double line describes the new meta-level constructs.

The basic building blocks of LySa are values, which syntactically are described by expressions built over distinct countable sets of indexed names $n_{\overline{i}} \in \mathit{Name}$ and indexed variables $x_{\overline{i}} \in \mathit{Var}$. A sequence of indexes $\overline{i} = i_1 \ldots i_k$ (for $k \geq 0$) has each index i taken from a countable set *Index*. At the object-level, the indices are simply seen as a syntactic concatenation of $\overline{i}$ to n or x. At the meta-level, on the other hand, indices play a crucial role for the indexing constructs. A k-tuple of expressions, $E_1, \ldots, E_k$, may be encrypted under a key E_0 by the encryption expression $\{E_1, \ldots, E_k\}_{E_0}$.

2.2 Object-Level Semantics

Only the object-level of LySa has a dynamic semantics. This semantics describes how an object-level process, P, evolves in a step-by-step fashion. The semantics is formalised by a reduction relation, $P \rightarrow P'$, defined as the smallest relation satisfying the rules in Figure 2. The semantics ranges over values $V \in \mathit{Val}$, which are expressions without variables. The meaning of the object-level constructs and their formal semantics is explained in the following.

In LySa a tuple of values can be communicated over a global network. Sending a messages is done by synchronous output $\langle V_1, \ldots, V_k \rangle.P_1$ that matches a pattern-matching input $(V_1, \ldots, V_j;\ mx_{j+1}, \ldots mx_k).P_2$ as described by the rule (Com). If the first j values in output and input (until the semi-colon) are identical then last $k - j$ values in output are component-wise bound to the variables in input. This binding takes place through a substitution, which may apply α-renaming to avoid capturing bound names. An encrypted value may be decrypted as described by the rule (SDec). The key V_0 used for encryption and decryption must the same and the first j values inside the encryption are pattern-matched. The process $(\nu\, n_{\overline{i}})\, P$ restricts the scope of $n_{\overline{i}}$ to be P, only. Apart from communication, parallel composition is interleaved as described by (Par). The inactive process, $\mathsf{0}$, cannot evolve and consequently it is not mentioned in Figure 2. Finally, the rule (Congr) may bring processes on a form where they match the

$$\text{(Com)}\ \langle V_1, \ldots, V_k\rangle.P_1 \mid (V_1, \ldots, V_j;\ mx_{j+1}, \ldots, mx_k).P_2 \to P_1 \mid P_2[mx_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, mx_k \overset{\alpha}{\mapsto} V_k]$$

$$\text{(SDec)}\ \mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1, \ldots, V_j;\ mx_{j+1}, \ldots, mx_k\}_{V_0}\ \mathsf{in}\ P \to P[mx_{j+1} \overset{\alpha}{\mapsto} V_{j+1}, \ldots, mx_k \overset{\alpha}{\mapsto} V_k]$$

$$\text{(New)}\ \frac{P \to P'}{(\nu\, n_{\bar{i}})\, P \to (\nu\, n_{\bar{i}})\, P'} \qquad \text{(Par)}\ \frac{P_1 \to P_1'}{P_1 \mid P_2 \to P_1' \mid P_2}$$

$$\text{(Congr)}\ \frac{P \equiv P'' \quad P'' \to P''' \quad P''' \equiv P'}{P \to P'}$$

Fig. 2. The reduction relation; $P \to P'$.

other rules by applying the structural congruence $P \equiv P'$. This relation is as usual the least congruence on object-level processes where

- parallel composition is associative, commutative with $\mathsf{0}$ as neutral element,
- $!P$ describes replication of P i.e. $!P \equiv !P \mid P$,
- restriction has capture-avoiding scope extrusion, and
- names may undergo disciplined α-conversion.

The notion of *disciplined* α-conversion is used solely for the benefit of the analysis as discussed in Section 3. For the sake of the analysis, the set of names will be partition into equivalences classes. *Disciplined α-conversion* requires that α-conversion only takes place within the same equivalence class. Each of these classes contain countably many element and consequently disciplining does not effect the expressive power of the semantics.

Example 1. A repeated nonce handshake between two principals A and B that initially share a key K may be modelled in LySa as the object-level process

$$(\nu\, K)\,(\quad !(\nu\, n)\,\langle A, B, n\rangle.(B, A;\ x).\mathsf{decrypt}\ x\ \mathsf{as}\ \{n;\ \}_K\ \mathsf{in}\ \mathsf{0} \\ \mid !(A, B;\ y).\langle B, A, \{y\}_K\rangle.\mathsf{0}\ \ )$$

Note in particular that the semi-colon is placed *after* the nonce n when the variable x is decrypted. This means pattern-matching takes place and decryption only succeeds if x is indeed bound to the nonce n encrypted under the key K.

2.3 Meta-Level Semantics

The meta-level has no dynamic semantics as such. Instead, a meta-level process M specifies a scenario, which is made up of a set of object-level processes. The meta-level process M is said to *instantiate to* an object-level process P, written $M \Rrightarrow P$, whenever P is in the set described by M. The set of object-level

$$\text{(IOut)}\ \frac{M \Rrightarrow P}{\langle E_1, \ldots, E_k\rangle.M \Rrightarrow \langle E_1, \ldots, E_k\rangle.P}$$

$$\text{(IInp)}\ \frac{M \Rrightarrow P}{(E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k).M \Rrightarrow (E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k).P}$$

$$\text{(ISDec)}\ \frac{M \Rrightarrow P}{\begin{array}{l}\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k\}_{E_0}\ \mathsf{in}\ M \Rrightarrow \\ \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k\}_{E_0}\ \mathsf{in}\ P\end{array}}$$

$$\text{(INew)}\ \frac{M \Rrightarrow P}{(\nu\, n_{\overline{a}})\, M \Rrightarrow (\nu\, n_{\overline{a}})\, P} \qquad \text{(IRep)}\ \frac{M \Rrightarrow P}{!M \Rrightarrow !P}$$

$$\text{(IPar)}\ \frac{M_1 \Rrightarrow P_1 \qquad M_2 \Rrightarrow P_2}{M_1 \mid M_2 \Rrightarrow P_1 \mid P_2} \qquad \text{(INil)}\ \mathsf{0} \Rrightarrow \mathsf{0}$$

$$\text{(ILet)}\ \frac{M[X \mapsto S'] \Rrightarrow P}{\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M \Rrightarrow P} \qquad \text{if } S' \subseteq_{fin} S$$

$$\text{(IIPar)}\ \frac{M[i \mapsto a_1] \Rrightarrow P_1 \quad \ldots \quad M[i \mapsto a_k] \Rrightarrow P_k}{|_{i \in \{a_1, \ldots, a_k\}}\ M \Rrightarrow P_1 \mid \ldots \mid P_k}$$

$$\text{(IINew)}\ \frac{M \Rrightarrow P}{(\nu_{\overline{i} \in \{\overline{a_1}, \ldots, \overline{a_k}\}}\ n_{\overline{a}\overline{i}})\, M \Rrightarrow (\nu\, n_{\overline{a}\overline{a_1}}) \ldots (\nu\, n_{\overline{a}\overline{a_k}})\, P}$$

Fig. 3. The instantiation relation; $M \Rrightarrow P$

processes may be infinite but each object-level process P in this set will itself be a finite process.

The instantiation relation is defined in Figure 3. All object-level processes instantiate to themselves with any subprocesses instantiated as well. In the indexing meta-level constructs S is a set of indexes from $\mathcal{P}(Index)$. The syntax of index sets, S, is left unspecified but include set identifiers $X \in SetId$ as placeholder for an index set. The process $\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M$ declares such a set identifier X to stand for some arbitrary subset of S for use inside M. This is the key mechanism, which lets instantiation describe *sets* of object-level processes. Note that the rule (ILet) requires X to become bound to a *finite* subset of S. This is done to ensure that all object-level processes are finite when instantiation is performed. The rule (IPar) instantiates an indexed parallel to the parallel composition of the a finite number of processes that have the index i taken from the index set $\{a_1, \ldots, a_k\}$. This index set is required to be finite, which is again done to attain a finite object-level process. Finally, the indexed restriction $(\nu_{\overline{i} \in \overline{S}}\ n_{\overline{a}\overline{i}})M$ instantiates to restrictions of all the names $n_{\overline{a}\overline{i}}$ where $\overline{i}$ is substituted with elements from $\overline{S}$ as described by the rule (IINew).

Example 2. The nonce handshake in Example 1 describes scenario where precisely two principals are present. Below the meta-level constructs are used to describe the same nonce handshake but this time in a more general scenario:

$$\begin{array}{l}\mathsf{let}\ X \subseteq \mathbb{N}\ \mathsf{in}\ \mathsf{let}\ Y \subseteq \mathbb{N}\ \mathsf{in}\ (\nu_{ij \in X \times Y}\ K_{ij})(\\ \quad |_{i \in X}\, |_{j \in Y}\ \ !(\nu\, n_{ij})\, \langle A_i, B_j, n_{ij} \rangle.(B_j, A_i;\, x_{ij}).\mathsf{decrypt}\ x_{ij}\ \mathsf{as}\ \{n_{ij};\, \}_{K_{ij}}\ \mathsf{in}\ 0 \\ \quad |\ |_{j \in Y}\, |_{i \in X}\ \ !(A_i, B_j;\, y_{ij}).\langle B_j, A_i, \{y_{ij}\}_{K_{ij}} \rangle.0\ \ )\end{array}$$

The first line declares the set identifiers X and Y to be subset of the natural numbers and, thus, the meta-level process describes all instances where *any* number of A_i's initiates a nonce handshake with any number of B_j's. Note also that the parts that describe the internals of each A_i and B_j closely correspond to the object-level processes in Example 1. The scenario, on the other hand is described by the meta-level constructs.

2.4 Binders and Substitution

The restriction operator $(\nu\, n_{\tilde{i}})\, M$ is a *binder* of the name $n_{\tilde{i}}$. In general, any kind of substitution of elements in the syntax respects binders and only substitutes free (i.e. unbound) instances of elements. Also input and decryption are binders of variables, the let-construct a binder of set identifiers, indexed parallel is a binder of the index i, while indexed restriction too is a binder of names.

Names that are not bound by any binder are said to be *free* names and they play an important role in the analysis attackers as discussed in Section 4. It is completely standard to define a function $\mathrm{fn}(P)$ that finds the free names of the object-level process P. For the meta-level, we define a function, $\mathrm{mfn}(M)$ that returns the most free names there can be in any instance of M. That is, mfn satisfies that if $M \Rrightarrow P$ then $\mathrm{fn}(P) \subseteq \mathrm{mfn}(M)$.

Names in $\mathrm{mfn}(M)$ do not need to be free in every instance of M but will not be a problem with the way $\mathrm{mfn}(M)$ is use in Section 4. This is as oppose to [8], where only a restricted class of processes were treated, namely the ones where names are either free in all instances or in none. Thus, the approach taken here considers a more general class of processes than in [8].

3 The Control Flow Analysis

The aim of a control flow analysis is to statically predict the behaviour of a process. Since the behaviour of an object-level process is given by its reduction semantics, the correctness of the analysis of object-level processes will as usual be given by a subject reduction result. A meta-level process, on the other hand, has no dynamic behaviour in itself. Instead, the control flow analysis will predict the behaviour of all the object-level processes that a meta-level process instantiates to. The correctness of the meta-level analysis will therefore show that the analysis is preserved by instantiation.

The control flow analysis can also be used to analyse processes under attack but the discussion of this is postponed until Section 4. An overall trademark of

the analysis is that it works by finding conservative over-approximations to the behaviour of a process. This means that *any* actual behaviour of a process will be reflected in the analysis result but the converse does not necessarily hold. With respect to security, this means that the analysis can be used to *guarantee the absence of attacks*. However, the analysis cannot be used to guarantee the presence of an attack because a possible attack reported by the analysis may be a consequences of approximation.

3.1 Equivalence Classes for Dealing with Infinities

One of the challenges when making an efficiently computable, automated analysis of the behaviour of a process is the infinity of values that may occur in the execution of the process. For example, at the object-level a replicated restriction, such as in $!(\nu\, n)\, \langle n \rangle.P$, may semantically produce an infinity of names by α-converting the name n. Also, at the meta-level an indexing parallel, such as $|_{i \in X}\ \langle n_i \rangle.P$, may instantiate to infinitely different names if X represents an infinite set.

To deal with this infinity, the set of values, Val, will be partitioned into *finitely* many equivalence classes written $\lfloor Val \rfloor$. It is important to stress, that this partitioning is made purely for the benefit of the analysis and will in no way effect the semantic behaviour a process. The analysis will record representatives of these equivalence classes, so-called canonical values, which are written $\lfloor V \rfloor$. As a consequence, the analysis is only capable of distinguishing two values V_1 and V_2 if they belong to different equivalence classes i.e. if $\lfloor V_1 \rfloor \neq \lfloor V_2 \rfloor$. The analysis is carefully designed such that any "mistakes" that arise because it cannot correctly distinguish two values will lead to over-approximation.

3.2 The Object-Level Analysis

The object-level control flow analysis aims at giving an account of the messages communicated on the network during any execution of an object-level process. Messages communicate by the polyadic output are recorded in an analysis component $\kappa \in \mathcal{P}(\lfloor Val \rfloor^*)$by a set of tuples of *canonical* values, thereby, benefitting from the finite partitioning of the value domain. It is also practical to have an analysis component $\rho : \lfloor Var \rfloor \to \mathcal{P}(\lfloor Val \rfloor)$that records the set of values that variables may become bound to during the execution of a process. The control flow analysis is specified using the Flow Logic [18] framework as a predicate

$$\rho, \kappa \models_{\Gamma} P$$

that holds precisely when ρ, κ is an analysis result that correctly describes the behaviour of the object-level process P. The predicate is defined inductively in the structure of processes in Figure 4. The environment Γ is needed only for the meta-level analysis presented in Section 3.3 and is often ignored when discussing the object-level.

$$
\begin{array}{lll}
\text{(AN)} & \rho \models n_{\bar{i}} : \vartheta & \text{iff } \lfloor n_{\bar{i}} \rfloor \in \vartheta \\
\text{(AVar)} & \rho \models x_{\bar{i}} : \vartheta & \text{iff } \rho(\lfloor x_{\bar{i}} \rfloor) \subseteq \vartheta \\
\text{(ASEnc)} & \rho \models \{E_1, \ldots, E_k\}_{E_0} : \vartheta & \text{iff } \wedge_{i=0}^{k} \rho \models E_i : \vartheta_i \wedge \\
 & & \quad \forall U_0 \in \vartheta_0 \ldots U_k \in \vartheta_k : \{U_1, \ldots, U_k\}_{U_0} \in \vartheta \\
\text{(AOut)} & \rho, \kappa \models_\Gamma \langle E_1, \ldots, E_k \rangle.M & \text{iff } \wedge_{i=1}^{k} \rho \models E_i : \vartheta_i \wedge \\
 & & \quad \forall U_1 \in \vartheta_1 \ldots U_k \in \vartheta_k : U_1 \ldots U_k \in \kappa \wedge \\
 & & \quad \rho, \kappa \models_\Gamma M \\
\text{(AInp)} & \rho, \kappa \models_\Gamma (E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k).M & \\
 & & \text{iff } \wedge_{i=1}^{j} \rho \models E_i : \vartheta_i \wedge \\
 & & \quad \forall U_1 \ldots U_k \in \kappa : \wedge_{i=1}^{j} U_i \in \vartheta_i \Rightarrow \\
 & & \quad\quad (\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor mx_i \rfloor) \wedge \rho, \kappa \models_\Gamma M) \\
\text{(ASDec)} & \rho, \kappa \models_\Gamma \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j;\ mx_{j+1}, \ldots, mx_k\}_{E_0}\ \mathsf{in}\ M & \\
 & & \text{iff } \rho \models E : \vartheta \wedge \wedge_{i=0}^{j} \rho \models E_i : \vartheta_i \wedge \\
 & & \quad \forall \{U_1, \ldots, U_k\}_{U_0} \in \vartheta : \wedge_{i=0}^{j} U_i \in \vartheta_i \Rightarrow \\
 & & \quad\quad (\wedge_{i=j+1}^{k} U_i \in \rho(\lfloor mx_i \rfloor) \wedge \rho, \kappa \models_\Gamma M) \\
\text{(ANew)} & \rho, \kappa \models_\Gamma (\nu\, n_{\bar{i}})\, M & \text{iff } \rho, \kappa \models_\Gamma M \\
\text{(ARep)} & \rho, \kappa \models_\Gamma !M & \text{iff } \rho, \kappa \models_\Gamma M \\
\text{(APar)} & \rho, \kappa \models_\Gamma M_1 \mid M_2 & \text{iff } \rho, \kappa \models_\Gamma M_1 \wedge \rho, \kappa \models_\Gamma M_2 \\
\text{(ANil)} & \rho, \kappa \models_\Gamma \mathsf{0} & \text{iff true} \\
\hline
\text{(ALet)} & \rho, \kappa \models_\Gamma \mathsf{let}\ X \subseteq S\ \mathsf{in}\ M & \text{iff } \rho, \kappa \models_{\Gamma[X \mapsto S']} M \\
 & & \quad \text{where } S' \subseteq_{fin} \Gamma(S) \text{ and } \lfloor S' \rfloor = \lfloor \Gamma(S) \rfloor \\
\text{(AIPar)} & \rho, \kappa \models_\Gamma |_{i \in S}\ M & \text{iff } \wedge_{a \in \Gamma(S)}\ \rho, \kappa \models_\Gamma M[i \mapsto a] \\
\text{(AINew)} & \rho, \kappa \models_\Gamma (\nu_{\bar{i} \in \overline{S}}\, n_{\overline{a}\bar{i}}) M & \text{iff } \rho, \kappa \models_\Gamma M
\end{array}
$$

Fig. 4. Analysis of LySa expressions, $\rho \models E : \vartheta$, and object-level and meta-level processes $\rho, \kappa \models_\Gamma M$.

The object-level part of the analysis in Figure 4 is essentially the control flow analysis from [3]. It relies on an auxiliary predicate $\rho \models E : \vartheta$ defined on expression. Conceptually, $\vartheta \in \mathcal{P}(\lfloor Val \rfloor)$ contains the values that E may evaluate to in some execution: (AN) names may evaluate to their canonical name; (AVar) variables may evaluate to the values recorded in the analysis component ρ; (ASEnc) encryption expression may evaluate to any encryption generated by recursive evaluation of subexpressions.

The rule for k-ary output (AOut) evaluates all expressions using the auxiliary predicate $\rho \models E_i : \vartheta_i$ and ensures that all combinations of their evaluations are recorded as k-tuples in κ. Correspondingly, k-ary input succeeds according to the analysis rule (AInp) for all k-tuples in κ where the first j values correspond to what the j first expressions may evaluate to. This takes care of the analysis of pattern-matching. If it is deemed successful then the remaining $k - j$ values are required to be component-wise recorded in $\rho(\lfloor mx_i \rfloor)$ thereby ensuring that

the analysis records possible variable bindings. The rule (ASDec) for decryption follows the same idea as for pattern-matching input though here the candidates are fund by evaluating the expression E. The remaining rules for the object-level analysis are standard. One may coin the fact that the analysis only distinguished names up to their canonical assignment:

Lemma 1 (Invariance of canonical names). *If* $\rho, \kappa \models P$ *and* $\lfloor n \rfloor = \lfloor n' \rfloor$ *then* $\rho, \kappa \models P[n \mapsto n']$.

Proof. The lemma is a direct consequence of the fact that the analysis only records canonical names. The proof proceeds by straightforward by induction in the definition of the analysis with the only interesting case being the rule (AN) though it too is straightforward because $\lfloor n \rfloor = \lfloor n[n \mapsto n'] \rfloor = \lfloor n \rfloor$.

The main technical result about the correctness of the object-level analysis is that the analysis correctly captures the behaviour of all executions of an object-level process. This is formulated as a standard subject reduction result:

Lemma 2. *If* $\rho, \kappa \models P$ *and* $P \to P'$ *then* $\rho, \kappa \models P'$.

Proof. The proof proceeds by structural induction in the reduction step $P \to P'$. The proof uses auxiliary lemmata about invariance of structural congruence and substitution of variables for values in ρ. The details may be found in [3].

3.3 The Meta-Level

The analysis of the meta-level constructs are given in the last there rules in Figure 4. The rules makes use of the environment $\Gamma : (\mathit{SetId} \cup \mathcal{P}(\mathit{Index})) \to \mathcal{P}(\mathit{Index})$ to record declarations of set identifiers. It is implicitly assumed that every index set S maps to itself i.e. that $\Gamma(S) = S$ for all $S \in \mathcal{P}(\mathit{Index})$.

The rule (ALet) updates Γ for the set identifier X declared in the let-construct. However, the analysis only keeps track of indices up to a finite, canonical partitioning of the index sets. Therefore it suffices to update Γ with a finite subset S' that belongs to the same equivalence class as the set S, which is declared in the let-construct. Thus, Γ will map all set identifiers to finite sets, which makes it easy to implement Γ. The rule (AIPar) makes the conjunction of the analysis of the process M where i has been substituted for the indexes in $\Gamma(S)$. This substitution corresponds to what happens semantically in (IIPar) while the conjunction is analogue to the analysis of binary parallel composition in (APar). The rule (AINew) ignores the restriction similarly to the rule (ANew).

The fact that the meta-level analysis is invariant up to the canonical partitioning of index sets is captured by the following lemma:

Lemma 3 (Invariance of canonical indices). *Let* $\lfloor a_1 \rfloor = \lfloor a_2 \rfloor$. *Then* $\rho, \kappa \models_\Gamma M[i \mapsto a_1]$ *if and only if* $\rho, \kappa \models_\Gamma M[i \mapsto a_2]$.

Proof. The substitution only modifies names and variables so it is sufficient to note that the analysis uses their canonical representatives and that $\lfloor n_{\bar{i}}[i \mapsto a_1] \rfloor = \lfloor n_{\bar{i}}[i \mapsto a_2] \rfloor$ as well as $\lfloor x_{\bar{i}}[i \mapsto a_1] \rfloor = \lfloor x_{\bar{i}}[i \mapsto a_2] \rfloor$. Thus, the analysis of $M[i \mapsto a_1]$ and $M[i \mapsto a_2]$ will be equivalent for all M.

The aim of the meta-level analysis is to capture the behaviour of an entire scenario as described by a meta-level process M. This is captured by the following main result about the correctness of the meta-level analysis:

Theorem 1 (Correctness of instantiation). *If* $\rho, \kappa \models_\Gamma M$ *and* $M\Gamma \Rrightarrow P$ *then* $\rho, \kappa \models P$.

Proof. The proof is by induction in M over the cases in the instantiation relation defined in Figure 3. The only interesting cases are the three meta-level constructs. The cases for the object-level constructs follow from the induction hypothesis.

3.4 Implementation

The goal of implementing the control flow analysis is to attain an analysis result for a process. That is, given a process M the implementation provides ρ, κ such that $\rho, \kappa \models_{[]} M$. The implementation of the control flow analysis in the LySatool works in two steps: (1) a generation function $\mathcal{G}(M)$ produces a formula that corresponds to the analysis predicate defined in Figure 4, and (2) a standard solver [17] is used to find an interpretation, which satisfies the formula.

The main challenge when the object-level was implemented in the LySatool version 1 [7] was that the analysis is specified over infinite sets of terms (which denote encryption). The implementation solves this by encoding sets of terms as regular tree grammars and manipulates a finite number of grammar rules akin to the strategy proposed in [16].

The second version of the LySatool is based on version 1 and indeed much of the code is reused. The main modification is to extend the constraint generation function $\mathcal{G}$ from ranging over object-level processes, only, to range over meta-level constructs as well. The definition of this function closely follows the three last rules in Figure 4 by taking the left-hand side of the iff as an argument to the function and returning the right hand-side. In the case of the $\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M$, one has to choose a finite set that is within the same equivalence class as S. In the implementation this is the point where one chooses the partitioning of these indexing sets as a parameter for controlling the precision of the meta-level analysis. The analysis results presented in Section 4 and Section 6 have been attained using this version 2 of the LySatool.

4 The Attacker

The goal of the LySatool is to validate security properties of a LySa process. Consequently, the main focus is on analysing process under attack from malicious parties also populating the network. The object-level of LySa has been designed such that this attack setup can be described as simple parallel composition: if P is an object-level process describing a protocol and $P_\bullet$ is an attacker then the behaviour of the process $P \mid P_\bullet$comprises all the attacks that $P_\bullet$ may launch over the network on P.

Instead of having to analyse all the infinitely many attackers $P_\bullet$ in parallel with P we follow ideas from [15]: It suffices to analyse a *single* process, P_{hard}, to get an account of the behaviour of all attackers.

Lemma 4 (Existence of a hardest attacker). *There exists process P_{hard} with the property that: for all attacker processes $P_\bullet$*

$$\rho, \kappa \models P \mid P_{hard} \quad \textit{implies} \quad \rho, \kappa \models P \mid P_\bullet$$

Proof. The proof is by construction of P_{hard} and subsequent induction in $P_\bullet$. The proof also relies on restricting the attention to attackers that only use the same arities as P for communication and encryption. Details can be found in [3].

By the subject reduction result in Lemma 2 it then follows that $\rho, \kappa \models P \mid P_{hard}$ gives an account of how P behaves under attack from any possible attacker, which is allowed by the semantics of LySa. This analysis result gives an account of the behaviour of the attacker. Note also that because $P_\bullet$ is placed in parallel with P it has access to all the free names in P. The hardest attacker, P_{hard}, therefore takes $\mathrm{fn}(P)$ as a parameter.

When choosing the canonical partitioning of names and variables, a special equivalence class is reserved for names and variables at the attacker. Representatives of these equivalence classes are denoted $n_\bullet$ and $x_\bullet$, respectively. For example, $\rho(x_\bullet)$ is an over-approximation of all the values that any variable in an attacker may become bound to i.e. it represents the knowledge of the attacker.

4.1 The Attacker at the Meta-Level

When we want to evaluate the security of a scenario described by a meta-level process M we must consider the possibility that each of the object-level process P, where $M \Rrightarrow P$, may be under attack. The aim of the analysis is, hence, to guarantee that no such attacks can occur on any instance of M.

In order to analyse all instances of a meta-level process under attack, we can once more rely directly on the hardest attacker P_{hard}. By design every object-level process instantiates to itself. Consequently, adding P_{hard} at the meta-level means that it always instantiates to be a hardest attacker at the object-level. This is made clear by the following theorem:

Theorem 2 (Attacker at the meta-level). *If $\rho, \kappa \models_\Gamma M \mid P_{hard}$ and $M\Gamma \Rrightarrow P$ then $\rho, \kappa \models P \mid P_\bullet$ for all attacker processes $P_\bullet$.*

Proof. From (APar) then $\rho, \kappa \models_\Gamma M$ and $\rho, \kappa \models_\Gamma P_{hard}$. By Theorem 1 then $\rho, \kappa \models P$. Furthermore, because P_{hard} is an object-level process then $P_{hard}\Gamma = P_{hard}$ and $P_{hard} \Rrightarrow P_{hard}$ so by Theorem 1 it follows that also $\rho, \kappa \models P_{hard}$. Consequently, by (APar) then $\rho, \kappa \models P \mid P_{hard}$ and finally by Lemma 4 $\rho, \kappa \models P \mid P_\bullet$ for all attackers $P_\bullet$.

At the meta-level, the function $\mathrm{mfn}(M)$ is used to provide the set of free names to P_{hard}. This set may actually be larger than the free names in some

specific instance of M meaning that the attacker may increase its power because it has access to too many names compared to what occurs semantically. However, if no attacks are reported by the analysis when the attacker has this extra power then no attacks can occur semantically, either.

Example 3. We refer to the scenario for the nonce handshake from Example 2 as the process M. Taking $\lfloor \mathbb{N} \rfloor = \{1\}$ the analysis result of $\rho, \kappa \models_{[]} M \mid P_{hard}$ as reported by the LySatool reveals that

$$\rho(x_\bullet) \cap \mathit{Name} = \{\lfloor n_{11} \rfloor, \lfloor A_1 \rfloor, \lfloor B_1 \rfloor\}$$

The index 1 in the analysis result is a canonical representative of any element in $\mathbb{N}$. Thus, the attacker may learn any nonce n_{ij} as well as the identities of any principal A_i and B_j for $i, j \in \mathbb{N}$. On the other hand, the analysis guarantees that the keys K_{ij} are confidential because $\lfloor K_{11} \rfloor$ is not in $\rho(x_\bullet)$. Since the analysis is an over-approximation this means that the attacker cannot learn the keys.

5 Security Properties

As discussed in Example 3, the analysis can guarantee *confidentiality* properties. To find out whether a particular value V is confidential one simply inspects the analysis result. If $\lfloor V \rfloor$ is not in $\rho(x_\bullet)$ then the analysis guarantees that no attacker can ever bind V to any of its variables.

The analysis of [3] is furthermore able to guarantee destination and origin *authentication*. This property considers the places where cryptography is applied to ensure that a message can only reach a particular principal. The property of destination and origin authentication is specified by annotations of the form

$$[\mathsf{at}\, c\, \mathsf{dest}\, C\,] \qquad \text{and} \qquad [\mathsf{at}\, c\, \mathsf{orig}\, C\,]$$

at all points of encryption and decryption, respectively. Here $c \in CP$ is a crypto-point that marks the point in the syntax (akin to a line-number). Encryptions are furthermore annotated with a set $C \subseteq CP$ of destination crypto-points where the encrypted values are intended to be decrypted. Symmetrically, decryptions are annotated with a set C of crypto-points where successfully decrypted value are intended to have been encrypted.

Semantically annotations are void i.e. they do not interfere with the semantic behaviour of a process. A process P is said to *guarantee dynamic authentication* none of these intentions are broken in any execution of the process. That is, P guarantees dynamic authentication if there are no reduction steps derived using the rule (SDec) of the form $\mathsf{decrypt}\{V_1, \ldots, V_k\}_{V_0}[\mathsf{at}\, c'\ \mathsf{dest}\, C'\,]\ \mathsf{as}\{V_1, \ldots, V_j; mx_{j+1}, \ldots, mx_k\}_{V_0}[\mathsf{at}\, c\, \mathsf{orig}\, C\,]\, \mathsf{in}\, P \to P'$ such that $c \notin C'$ or $c' \notin C$.

The main result of [3] is that an extension of the object-level analysis presented in Figure 4 is capable of analysing whether an object-level process guarantees dynamic authentication. The extension of the analysis essentially boils down to adding a check of whether $c \notin C'$ or $c' \notin C$ in the rule (SDec) for analysis of decryption. If the analysis finds no errors in these checks then a process P is said to *guarantee static authentication*. The main result of [3] is that

Lemma 5. *If P guarantees static authentication then P guarantees dynamic authentication.*

Proof. The proof relies on the fact that the analysis over-approximates the dynamic behaviour of P and thereby also the potential authentication errors that may be reported. The details are [3].

5.1 Authentication at the Meta-Level

To further refine the authentication property for the meta-level, crypto-points are equipped with indices analogue to indices on names and variables. That is, crypto-points will be of the form $c_{\bar{i}}$. Crypto-points are also made subject to a notion of canonicity because the meta-level analysis only distinguishes elements up to the canonical partitioning of index sets.

The meta-level analysis is now capable of checking destination and origin authentication up to the partitioning into equivalence classes. Conceptually, the analysis guarantees that messages only reach principals within a certain equivalence class. The meta-level analysis is extended by adding a check of whether $\lfloor c_{\bar{i}} \rfloor \notin \lfloor C' \rfloor$ or $\lfloor c'_{\bar{i}} \rfloor \notin \lfloor C \rfloor$ in the rule (SDec). If no violations of the authentication properties are found by these checks in the meta-level analysis of M then M *guarantees static authentication*. The check for static authentication by the meta-level analysis suffices guarantee the authentication properties for all object-level process that M instantiates to:

Theorem 3 (Authentication at the meta-level). *If M guarantees static authentication and $M\Gamma \Rrightarrow P$ then P guarantees dynamic authentication.*

Proof. The theorem follows immediately from Theorem 1 and the fact that indexed crypto-points are subject to canonicity as well as Lemma 5.

6 An Example Protocol

To illustrate the usefulness of the meta-level we analyse a protocol by Bauer, Berson, and Feiertag [1]. According to a recent survey [5] there are no known attacks on the protocol and, furthermore, the protocol is the basis one of the key establishment mechanisms in an ISO/IEC standard [13]. The protocol makes use of a server with which each principal initially shares a key KS_i. In the first two messages of the protocol, fresh nonces na_{ij} and nb_{ij} produced by principal I_i and I_j, respectively, are sent to the server along with the identities of the principals. The server generates a new session key K_{ij}, which is returned encrypted to each principal along with their own nonce and the identity of the other principal. The

protocol may be encoded as a meta-level scenario in the following way:

$$
\begin{array}{ll}
\mathsf{let}\ X \subseteq S_0\ \mathsf{in}\,\mathsf{let}\ Y \subseteq S_1\ \mathsf{in}\,(\nu_{i\in X\cup Y}\ KS_i) & \\
\quad |_{i\in X}\,|_{j\in Y} & !(\nu\, na_{ij})\,\langle I_i, na_{ij}\rangle. \\
 & (;\, xa_{ij}).\mathsf{decrypt}\ xa_{ij}\ \mathsf{as}\ \{I_j, na_{ij};\, xk_{ij}\}_{KS_i}[\mathsf{at}\, a_{ij}\ \mathsf{orig}\,\{s2_{ij}\}\,]\,\mathsf{in}\,0 \\
|\,|_{j\in Y}\,|_{i\in X} & !(I_i;\, yn_{ij}). \\
 & (\nu\, nb_{ij})\,\langle I_j, I_i, yn_{ij}, nb_{ij}\rangle. \\
 & (;\, yb_{ij}, ya_{ij}).\mathsf{decrypt}\ yb_{ij}\ \mathsf{as}\ \{I_i, nb_{ij};\, yk_{ij}\}_{KS_j}[\mathsf{at}\, b_{ij}\ \mathsf{orig}\,\{s1_{ij}\}\,]\,\mathsf{in} \\
 & \langle ya_{ij}\rangle.0 \\
|\,|_{i\in X}\,|_{j\in Y} & !(I_j, I_i;\, za_{ij}, zb_{ij}).(\nu\, K_{ij})\,\langle\{I_i, zb_{ij}, K_{ij}\}_{KS_j}[\mathsf{at}\, s1_{ij}\ \mathsf{dest}\,\{b_{ij}\}\,], \\
 & \qquad\qquad \{I_j, za_{ij}, K_{ij}\}_{KS_i}[\mathsf{at}\, s2_{ij}\ \mathsf{dest}\,\{a_{ij}\}\,]\rangle.0
\end{array}
$$

Annotations are added to declare that the two encryptions made at the server are intended for the correct responder and initiator of the protocol, only.

One scenario can be described by taking $S_0 = \{0\}$ and $S_1 = \{1\}$. Then the meta-level process describes a scenario where principal I_0 repeatedly initiates the protocol with I_1. When choosing $\lfloor S_0 \rfloor \neq \lfloor S_1 \rfloor$, the analysis guarantees static authentication. That is, the analysis guarantees that the messages containing the session keys will only be delivered to the correct principals.

Taking instead $S_0 = S_1 = \mathbb{N}$ the encoding represents a scenario where every principal I_i can use the protocol with every principal I_j. This scenario includes the case where a pair of principals uses the protocol in both directions at the same time. For this scenario the analysis no longer guarantees static authentication. In fact, the scenario does not satisfy dynamic authentication because the authentication property may be violated. The attack occurs precisely when the protocol is used in both directions at the same time. As an end result of the attack, two principals, say I_1 and I_2, may end up sharing a session key K_{12} but both of them will think that the key came from a protocol session they themselves initiated. The problem is easy to fix: one simply needs to ensure that the two encryptions made at the server do not have the same format. The analysis guarantees authentication when the messages in one of these encryptions are rearranged.

7 Conclusion

7.1 Related Work

The meta-level analysis bears some resemblance to a result shown by Comon-Lundh and Cortier [10] that says that it suffices to consider a limited number of principals when analysing a protocol. The result is shown by projecting the semantic behaviour of all principals onto this limited number of principals. Our meta-level analysis can also be seen as projection the behaviour of different principals. However, the projection is onto the canonical values in the analysis result rather that onto the semantic behaviour as in [10]. Thus, our results are provided by a computable analysis that features a syntactic meta-level, which furthermore allows a flexible modelling of different scenarios.

The idea of having additional syntax that describes scenarios can also be found in frameworks such as Casper [14], CAPSL [11], etc. The main difference from our approach is that they use syntactic unfolding i.e. that their scenarios undergo a syntactic transformation (corresponding to our instantiation) *before* analysis takes place.

It is also be appropriate to mention that our object-level analysis is related to the approaches in [2, 12, 9]. The reader is referred e.g. to [3, 6] for a detailed comparison with these approaches. However, none of these approaches deal with scenarios that is the topic of this paper.

7.2 Summary

When discussing the security of a protocol it is vital to consider the scenario in which the protocol will be deployed. This paper puts the focus on these deployment scenarios by extending the process calculus LySa with a meta-level. This meta-level contains language primitives that caters for a flexible description of scenarios. We have shown that it is viable to make a control flow analysis directly on the meta-level that, also in practice, is capable of guaranteeing both confidentiality and authentication properties. The analysis has been implemented with relatively minor effort by relying on a previous implementation of the object-level analysis. The result is version 2 of the LySatool, which has proven its worth by finding a previously unreported problem in a classical security protocol. The LySatool is freely available at

http://www.imm.dtu.dk/cs_LySa/lysatool

where the full analysis results for examples in this paper can also be found.

Acknowledgements. The idea of having an analysable meta-level came up when writing [8] with Flemming Nielson and Hanne Riis Nielson. Many other ideas concerning LySa come from them as well as Chiara Bodei and Pierpaolo Degano.

References

1. R. K. Bauer, T. A. Berson, and R. J. Feiertag. A key distribution protocol using event markers. *ACM Transactions on Computer Systems*, 1(3):249 – 255, 1983.
2. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *CSFW 2001*, pages 82–96. IEEE, 2001.
3. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *CSFW 2003*, pages 126–140. IEEE, 2003.
4. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Static validation of security protocols. *JSC*, 2004. To appear. Preliminary version at www.imm.dtu.dk/pubdb/views/edoc_download.php/3199/pdf/imm3199.pdf.
5. C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishment.* Springer, 2003.
6. M. Buchholtz. Automated analysis of security in networking systems. Ph. D. thesis proposal. Available from http://www.imm.dtu.dk/∼mib/thesis/, December 2004.

7. M. Buchholtz. Implementing control flow analysis for security protocols. DEGAS Report WP6-IMM-I00-Pub-003, Draft 2003.
8. M. Buchholtz, F. Nielson, and H. Riis Nielson. A calculus for control flow analysis of security protocols. *IJIS*, 2(3-4):145–167, 2004.
9. M. Bugliesi, R. Focardi, and M. Maffei. Compositional analysis of authentication protocols. In *ESOP 2004*, volume 2986 of *LNCS*, pages 140–154. Springer, 2004.
10. H. Comon-Lundh and V. Cortier. Security properties: Two agents are sufficient. In *ESOP 2003*, number 2618 in LNCS, pages 99–113. Springer, 2003.
11. G. Denker, J. Millen, and H. Rueß. The CAPSL integrated protocol environment. Technical Report SRI-CLS-2000-02, SRI International, 2000.
12. A. D. Gordon and A. Jeffrey. Authenticity by Typing for Security Protocols. In *CSFW 2001*, pages 145 –159. IEEE, 2001.
13. Information technology - security techniques - key management - part 2. mechanisms using symmetric techniques ISO/IEC 11770-2. International Standard, 1996.
14. G. Lowe. Casper: A compiler for the analysis of security protocols. *JSC*, 6(1):53–84, 1998.
15. F. Nielson, H. Riis Nielson, and R. R. Hansen. Validating firewalls using Flow Logics. *TCS*, 283(2):381–418, 2002.
16. F. Nielson, H. Riis Nielson, and H. Seidl. Cryptographic analysis in cubic time. In *TOSCA 2001*, volume 62 of *ENTCS*. Elsevier, 2001.
17. F. Nielson, H. Riis Nielson, and H. Seidl. A succinct solver for ALFP. *NJC*, 9:335–372, 2002.
18. H. Riis Nielson and F. Nielson. Flow Logic: a multi-paradigmatic approach to static analysis. In *The Essence of Computation: Complexity, Analysis, Transformation*, volume 2566 of *LNCS*, pages 223–244. Springer, 2002.

This appendix is *not* intended to be part of the paper. It contains the proof of a theorem that has been omitted for the sake of space.

A Proof of Theorem 1

The proof of Theorem 1 uses a lemma that concerns the way let-constructs are analysed. The idea when analysing the let-construct is to conduct the analysis with the largest possible set of canonical indices. This suffices because it also covers the analysis of processes where smaller subsets are chosen. This can formally be stated as the lemma:

Lemma 6 (Subset in let-declaration). *If* $\lfloor S_2 \rfloor \subseteq \lfloor S_1 \rfloor$ *then* $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} M$ *implies* $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} M$.

Proof. The proof proceeds by induction in the structure of M.
Case $\mathsf{let}\ X' \subseteq S\ \mathsf{in}\ M$. Assume that $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} \mathsf{let}\ X' \subseteq S\ \mathsf{in}\ M$ i.e. by (ALet)

$$\rho, \kappa \models_{\Gamma[X \mapsto S_1][X' \mapsto S']} M$$

for some S' such that $S' \subseteq_{fin} \Gamma(S)$ and $\lfloor S' \rfloor = \lfloor \Gamma(S) \rfloor$. Now assume that $X = X'$. Then the inner substitution of X is overwritten by $[X' \mapsto S']$ so

$$\begin{aligned} \rho, \kappa \models_{\Gamma[X \mapsto S_1][X' \mapsto S']} M \text{ iff } & \rho, \kappa \models_{\Gamma[X \mapsto S_2][X' \mapsto S']} M \\ \text{iff } & \rho, \kappa \models_{\Gamma[X \mapsto S_2]} \mathsf{let}\ X' \subseteq S\ \mathsf{in}\ M \end{aligned}$$

as required. Alternatively assume that $X \neq X'$. Then the order of substitutions does not matter. Using this and the induction hypothesis (IH) one may derive

$$\begin{aligned} \rho, \kappa \models_{\Gamma[X \mapsto S_1][X' \mapsto S']} M \text{ iff } & \rho, \kappa \models_{\Gamma[X' \mapsto S'][X \mapsto S_1]} M \\ \text{implies } & \rho, \kappa \models_{\Gamma[X' \mapsto S'][X \mapsto S_2]} M \text{ (by IH)} \\ \text{iff } & \rho, \kappa \models_{\Gamma[X \mapsto S_2][X' \mapsto S']} M \end{aligned}$$

which allows to conclude that $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} \mathsf{let}\ X' \subseteq S\ \mathsf{in}\ M$ as required.
Case $|_{i \in S}\ M$. First notice that if $S \neq X$ then $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} |_{i \in S}\ M$ implies $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} |_{i \in S}\ M$ simply by applying the induction hypothesis for the analysis of M. Next assume that $S = X$, which gives that

$$\rho, \kappa \models_{\Gamma[X \mapsto S_1]} |_{i \in S}\ M \quad \text{iff} \quad \wedge_{a_1 \in S_1}\ \rho, \kappa \models_{\Gamma[X \mapsto S_1]} M[i \mapsto a_1]$$

From the assumption that $\lfloor S_2 \rfloor \subseteq \lfloor S_1 \rfloor$ it is known that for every $a_2 \in S_2$ there is a corresponding $a_1 \in S_1$ such that $\lfloor a_2 \rfloor = \lfloor a_1 \rfloor$. By Lemma 3 then it holds that $\rho, \kappa \models_{\Gamma[X \mapsto S_1]} M[i \mapsto a_2]$ for all $a_2 \in S_2$. This together with the induction hypothesis allows to conclude

$$\wedge_{a_2 \in S_2}\ \rho, \kappa \models_{\Gamma[X \mapsto S_2]} M[i \mapsto a_2]$$

which is precisely $\rho, \kappa \models_{\Gamma[X \mapsto S_2]} |_{i \in S}\ M$ as required.

The remaining cases are straightforward and follow by applying the induction hypothesis because the analysis does not directly use Γ in these cases.

Using this lemma, we can now proceed to the proof of Theorem 1:

Proof. The proof proceeds by induction in the structure of M.
Case $\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M$. First, calculate

$$(\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M)\Gamma = \mathsf{let}\ X \subseteq \Gamma(S)\ \mathsf{in}\ M(\Gamma \setminus X)$$

where $\Gamma \setminus X$ is as Γ except that $\Gamma(X)$ is undefined. Next, assume that $(\mathsf{let}\ X \subseteq S\ \mathsf{in}\ M)\Gamma \Rrightarrow P$ which according to (ILet) in Table 3 happens because

$$(M(\Gamma \setminus X))[X \mapsto S'] \Rrightarrow P$$

for some $S' \subseteq_{fin} \Gamma(S)$. Because X is undefined in $\Gamma \setminus X$ this is the same as

$$M(\Gamma[X \mapsto S']) \Rrightarrow P$$

Next, assume that $\rho, \kappa \models_\Gamma \mathsf{let}\ X \subseteq S\ \mathsf{in}\ M$ i.e. from (ALet) that

$$\rho, \kappa \models_{\Gamma[X \mapsto S'']} M$$

where $S'' \subseteq_{fin} \Gamma(S)$ and $\lfloor S'' \rfloor = \lfloor \Gamma(S) \rfloor$. Notice that $\lfloor S' \rfloor \subseteq \lfloor S'' \rfloor$ so by Lemma 6

$$\rho, \kappa \models_{\Gamma[X \mapsto S']} M$$

From the induction hypothesis it then follows that $\rho, \kappa \models P$ as required.
Case $|_{i \in S}\ M$. Assume $\rho, \kappa \models_\Gamma |_{i \in S}\ M$ i.e. from (AIPar) that

$$\wedge_{a \in \Gamma(S)}\, \rho, \kappa \models_\Gamma M[i \mapsto a]$$

Furthermore, let $\Gamma(S) = \{a_1, \ldots, a_k\}$ for some arbitrary set $\{a_1, \ldots, a_k\}$. Next, assume that $(|_{i \in S}\ M)\Gamma \Rrightarrow P_1 \mid \ldots \mid P_k$ by (IPar). Noting that $(|_{i \in S}\ M)\Gamma = |_{i \in \Gamma(S)}\ M\Gamma$ and using (IIPar) this means that

$$M\Gamma[i \mapsto a_j] \Rrightarrow P_j$$

for each $a_j \in \Gamma(S)$. Since the two substitutions Γ and $[i \mapsto a_j]$ range over different domains the order of the substitution of does not matter. Thus, it also holds that for all $a_j \in \Gamma(S)$ that

$$(M[i \mapsto a_j])\Gamma \Rrightarrow P_j$$

The induction hypothesis can be applied k times to establish that

$$\rho, \kappa \models P_1 \wedge \ldots \wedge \rho, \kappa \models P_k$$

which by (APar) from Table 4 applied k times give precisely $\rho, \kappa \models P_1 \mid \ldots \mid P_k$ as required.
Case $(\nu_{\overline{i \in S}}\, n_{\overline{ai}})M$. Assume that $\rho, \kappa \models_\Gamma (\nu_{\overline{i \in S}}\, n_{\overline{ai}})M$ i.e. that

$$\rho, \kappa \models_\Gamma M$$

Let $\Gamma(\overline{S}) = \{\overline{a_1}, \ldots, \overline{a_k}\}$ and note that $((\nu_{\bar{i}\in\overline{S}}\ n_{\overline{a}\bar{i}})M)\Gamma = (\nu_{\bar{i}\in\Gamma(\overline{S})}\ n_{\overline{a}\bar{i}})M\Gamma$. Next assume that $((\nu_{\bar{i}\in\overline{S}}\ n_{\overline{a}\bar{i}})M)\Gamma \Rrightarrow (\nu\, n_{\overline{aa_1}}) \ldots (\nu\, n_{\overline{aa_k}})\, P$, which according to (IINew) happens because

$$M\Gamma \Rrightarrow P$$

The induction hypothesis applies to give that $\rho, \kappa \models P$, which by (ANew) from Table 4 is the same as $\rho, \kappa \models (\nu\, n_{\overline{aa_1}}) \ldots (\nu\, n_{\overline{aa_k}})\, P$ as requires.

The remaining cases for the object-level syntax are straightforward because the substitution Γ does not modify anything in the object-level syntax.