Implementing Control Flow Analysis for Security Protocols

Mikael Buchholtz*

DRAFT of October 1, 2003

The control flow analysis of the process calculus LySA [3] provides a way of analysing authenticity properties in security protocols. This report describes the implementation of the analysis and proceeds by transforming the analysis into formulae of Alternation-free Least Fixed Point logic. These formulae serve as the input language of the Succinct Solver [8] that, thus, can be used to compute the analysis result.

1. Introduction

This report describes the technical results in the development of the implementation of the control flow analysis for the process calculus LySA [3]. The implementation has been used to analyse a number of authentication protocols as reported in [3]. The overall implementation strategy is to provide a *generation function*, $\mathcal{G}(P)$, that given a LySA process P returns a formula in Alternation-free Least Fixed Point logic (ALFP). ALFP is a powerful fragment of first-order predicate logic and an ALFP formula interpreted over a finite universe serves as the input to the Succinct Solver [8]. In return the Succinct Solver gives the smallest interpretation of the predicates that satisfies the formula. By encoding the analysis components as predicates and describing their inter-dependencies as specified by the analysis in ALFP, the Succinct Solver, thus, provides a tool for computing the analysis components.

This implementation strategy requires that notation is changed from the format of the original analysis in [3] into ALFP and this transformation is the topic of Section 6. Before

*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.

getting that far, however, a number of other points requires more care and are handled separately in Section 2 to 5. Each of these sections transforms the former analysis into a new analysis resulting, finally, in the generation function, $\mathcal{G}(P)$, in Section 6.

Each new analysis is shown to be *sound* with respect to the previous analysis; that is, it is shown that the new analysis also provides a solution to the previous analysis. Each soundness results is stated as a lemma and the overall soundness of the implementation is shown by combining the lemmata in Section 9. There it is also conjectured that the converse property of soundness, called *completeness*, also holds.

The development described in this report closely follows a corresponding development for the Spi-calculus [2] described in [7]. Resemblances and differences from this development are noted throughout this report. One difference is this the development in [7] applies a number of techniques to gain a low complexity according to a rather pessimistic cost measure. In this report, the focus is instead on developing an implementation that closely corresponds to the original analysis as stated in [3]. Thus, optimisations for speed and space consumption of the solving procedure is not a part of this work though practical experiments shows that the implementation runs in cubic time in the size of the process that is analysed.

2. The Starting Point

2.1. LySA

LySA [3] is a process calculus for security protocols. It uses terms, E, to describe names, variables, and *symmetric key encryption*, respectively, according to the following grammar:

$$E \ ::= \ n \quad | \quad x \quad | \quad \{E_1, \cdots, E_k\}^{\ell}_{E_0}[\mathsf{dest}\ \mathcal{L}]$$

Here, the sequence of terms $E_1, \cdots, E_k$ are encrypted under the key E_0. These encrypted terms are annotated with a so-called crypto-point ℓ to denote the place of encryption as well as a set of crypto-points $\mathcal{L}$ where the term is intended to be decrypted.

Processes are formed from the grammar

$$\begin{array}{rcl} P & ::= & \langle E_1, \cdots, E_k \rangle . P \quad | \quad (E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k) . P \quad | \\ & & \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k\}^{\ell}_{E_0}\ [\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P \quad | \\ & & (\nu\, n)P \quad | \quad !\, P \quad | \quad P_1 | P_2 \quad | \quad \mathsf{0} \end{array}$$

The process $\langle E_1, \cdots, E_k \rangle . P$ sends a sequence of k terms on a global network before proceeding as the process P. The process $(E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k) . P$ receives such a k-ary message and matches the first j terms in the message against the terms $E_1, \cdots, E_j$. If they are component-wise equal then the variables $x_{j+1}, \cdots, x_k$ become bound in P to the remaining $k - j$ terms and P is then executed. This kind of *pattern matching* also

E	$\in$	$\mathcal{T}$	Terms
P	$\in$	$\mathcal{P}$	Processes
x	$\in$	$\mathcal{X}$	Variables
n	$\in$	$\mathcal{N}$	Names
ℓ	$\in$	$\mathcal{C}$	Crypto-points

Table 1: Types used in the syntax.

occurs in decryption decrypt E as $\{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k\}^{\ell}_{E_0}$ [orig $\mathcal{L}$] in P where the term E is attempted to be decrypted. Corresponding to encryption, this construct is annotated with a crypto-point ℓ to denote the place of decryption along with a set of crypto-points $\mathcal{L}$ describing where E is indented to have been encrypted. The remaining process expressions are restriction, replication, parallel composition, and the terminated process, respectively. The calculus has a reduction semantics, which is defined in [3]. To be precise about types of the syntactic categories and the meta-variables used to range over them these are summarised in Table 1.

2.2. The Original Control Flow Analysis

The original version of the control flow analysis from [3] is restated in Table 3. The overall idea of the analysis is to track the messages that are communicated in an analysis component κ along with a component ρ holding the possible values that variables may become bound to. The version of the analysis given in Table 3 incorporates a reference monitor: while tracking the control flow information, the analysis also checks whether the annotations of encryption and decryption points are violated. If that is the case, the error component ψ reports this by including the pairs of crypto-points for which the annotations do not hold.

Table 2 summarise the types of the analysis components. Note that since the semantics of LySA, as given in [3], applies α-renaming of variables and names, the analysis uses a notion of *canonical* names and variables. The notation $\lfloor x \rfloor$ signifies the canonical variable of x while $\lfloor \mathcal{X} \rfloor$ is the set of canonical variables. A similar notation is used for names. Furthermore, the analysis uses a set membership operator, E, which ignores the annotations on encrypted terms.

ρ	:	$\lfloor \mathcal{X} \rfloor \to \wp(\mathcal{V})$	Variable environment
κ	$\in$	$\wp(\mathcal{V}^*)$	Network component
θ	$\in$	$\wp(\mathcal{V})$	Local term cache
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
V	$\in$	$\mathcal{V}$	Values – terms without variables

Table 2: Components used in the original analysis $\models$.

$$\frac{\lfloor n \rfloor \in \theta}{\rho \models n : \theta} \qquad \frac{\rho(\lfloor x \rfloor) \subseteq \theta}{\rho \models x : \theta}$$

$$\frac{\begin{array}{c}\wedge_{i=0}^{k}\, \rho \models E_i : \theta_i \wedge \\ \forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \theta_i \Rightarrow \{V_1, \cdots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\ \mathcal{L}] \in \theta\end{array}}{\rho \models \{E_1, \cdots, E_k\}_{E_0}^{\ell}[\mathsf{dest}\ \mathcal{L}] : \theta}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{k}\, \rho \models E_i : \theta_i \wedge \\ \forall V_1, \cdots, V_k : \wedge_{i=1}^{k} V_i \in \theta_i \Rightarrow \langle V_1, \cdots, V_k \rangle \in \kappa \wedge \\ \rho, \kappa \models P : \psi\end{array}}{\rho, \kappa \models \langle E_1, \cdots, E_k \rangle.\, P : \psi}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{j}\, \rho \models E_i : \theta_i \wedge \\ \forall \langle V_1, \cdots, V_k \rangle \in \kappa : \wedge_{i=1}^{j} V_i \in \theta_i \Rightarrow \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \\ \rho, \kappa \models P : \psi\end{array}}{\rho, \kappa \models (E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k).\, P : \psi}$$

$$\frac{\begin{array}{c}\rho \models E : \theta \quad \wedge \quad \wedge_{i=0}^{j}\, \rho \models E_i : \theta_i \wedge \\ \forall \{V_1, \cdots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\ \mathcal{L}] \in \theta : \wedge_{i=0}^{j} V_i \in \theta_i \Rightarrow \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \\ (\neg(\ell \in \mathcal{L}' \wedge \ell' \in \mathcal{L}) \Rightarrow (\ell, \ell') \in \psi) \wedge \\ \rho, \kappa \models P : \psi\end{array}}{\rho, \kappa \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k\}_{E_0}^{\ell'}\ [\mathsf{orig}\ \mathcal{L}']\ \mathsf{in}\ P : \psi}$$

$$\frac{\rho, \kappa \models P : \psi}{\rho, \kappa \models (\nu\, n)P : \psi} \qquad \frac{\rho, \kappa \models P : \psi}{\rho, \kappa \models !\, P : \psi}$$

$$\frac{\rho, \kappa \models P_1 : \psi \ \wedge\ \rho, \kappa \models P_2 : \psi}{\rho, \kappa \models P_1 | P_2 : \psi} \qquad \rho, \kappa \models 0 : \psi$$

Table 3: The original analysis of terms, $\rho \models E : \theta$, and processes, $\rho, \kappa \models P : \psi$.

In the following sections the formulation of the analysis is changes in order to provide ALFP formulae that correspond to the original analysis. To distinguish the individual analyses, the $\models$-symbols used in analysis judgements will be annotated with different superscripts. The same goes for the analysis components except in the tables where confusion is unlikely to occur.

3. From Succinct to Verbose

The analysis in Table 3 is a *succinct* Flow Logic [9] meaning that there are analysis components placed on the right-hand side of the judgements. That is true both for the component θ in the judgements for terms, $\rho \models E : \theta$, and the component ψ in the judgements for processes, $\rho, \kappa \models P : \psi$. This Flow Logic style, in effect, creates a local scope for the analysis components θ and ψ. The ALFP logic that the analysis will be transformed into does, however, not provide scoping mechanisms for predicates. Since the analysis components will be transformed into predicates the first step will be to ensure that all analysis components have global scope. This can be achieved by transforming the Flow Logic into *verbose* form [9] where all components are placed on the left-hand side of the judgement.

The technique for transforming a succinct Flow Logic into a verbose one is standard [9] and proceeds by adding labels in the syntax at all the places where succinct judgements are used. Secondly, the succinct component, say θ, is replaced with a global mapping, say θ^v, that for each label returns the corresponding succinct component.

By inspecting the analysis in Table 3, it is clear that the judgements of terms are used at all applied occurrences of terms. Consequently, the syntax of LySA is instrumented with labels, $l \in Lab$, at all such occurrences yielding the syntax of labelled LySA:

$$
\begin{array}{rcl}
E & ::= & n^l \quad | \quad x^l \quad | \quad \{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}]^l \\
P & ::= & \langle E_1^{l_1}, \cdots, E_k^{l_k}\rangle.\, P \quad | \quad (E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k).\, P \quad | \\
 & & \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k\}^{\ell}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P \quad | \\
 & & (\nu\, n)P \quad | \quad !\,P \quad | \quad P_1|P_2 \quad | \quad \mathsf{0}
\end{array}
$$

The semantics of labelled LySA completely ignores the labels and, consequently, the

ρ	:	$\lfloor \mathcal{X} \rfloor \to \wp(\mathcal{V})$	Variable environment
κ	$\in$	$\wp(\mathcal{V}^*)$	Network component
θ^v	:	$Lab \to \wp(\mathcal{V})$	Global term cache
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
l	$\in$	Lab	Labels on terms

Table 4: Components used in the verbose analysis $\models^v$.

$$\frac{\lfloor n \rfloor \in \theta(l)}{\rho, \theta \models^v n^l} \qquad \frac{\rho(\lfloor x \rfloor) \subseteq \theta(l)}{\rho, \theta \models^v x^l}$$

$$\frac{\begin{array}{c}\wedge_{i=0}^{k}\, \rho, \theta \models^v E_i^{l_i} \wedge \\ \forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \theta(l_i) \;\Rightarrow\; \{V_1, \cdots, V_k\}_{V_0}^{\ell}[\mathsf{dest}\ \mathcal{L}] \in \theta(l)\end{array}}{\rho, \theta \models^v \{E_1^{l_1}, \cdots, E_k^{l_k}\}_{E_0^{l_0}}^{\ell}[\mathsf{dest}\ \mathcal{L}]^l}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{k}\, \rho, \theta \models^v E_i^{l_i} \wedge \\ \forall V_1, \cdots, V_k :\ \wedge_{i=1}^{k} V_i \in \theta(l_i) \Rightarrow \langle V_1, \cdots, V_k \rangle \in \kappa \wedge \\ \rho, \kappa, \psi, \theta \models^v P\end{array}}{\rho, \kappa, \psi, \theta \models^v \langle E_1^{l_1}, \cdots, E_k^{l_k} \rangle . P}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{j}\, \rho, \theta \models^v E_i^{l_i} \wedge \\ \forall \langle V_1, \cdots, V_k \rangle \in \kappa :\ \wedge_{i=1}^{j} V_i \in \theta(l_i) \Rightarrow \quad (\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \\ \rho, \kappa, \psi, \theta \models^v P)\end{array}}{\rho, \kappa, \psi, \theta \models^v (E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k) . P}$$

$$\frac{\begin{array}{c}\rho, \theta \models^v E^l \quad \wedge \wedge_{i=0}^{j}\, \rho, \theta \models^v E_i^{l_i} \wedge \\ \forall\ \{V_1, \cdots, V_k\}_{V_0}^{\ell'}[\mathsf{dest}\ \mathcal{L}'] \in \theta(l) :\ \wedge_{i=0}^{j} V_i \in \theta(l_i) \Rightarrow \\ (\wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \\ (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L}) \Rightarrow (\ell', \ell) \in \psi \wedge \\ \rho, \kappa, \psi, \theta \models^v P)\end{array}}{\rho, \kappa, \psi, \theta \models^v \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k\}_{E_0^{l_0}}^{\ell}\ [\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P}$$

$$\frac{\rho, \kappa, \psi, \theta \models^v P}{\rho, \kappa, \psi, \theta \models^v (\nu\, n)P} \qquad \frac{\rho, \kappa, \psi, \theta \models^v P}{\rho, \kappa, \psi, \theta \models^v\ !P}$$

$$\frac{\rho, \kappa, \psi, \theta \models^v P_1 \ \wedge\ \rho, \kappa, \psi, \theta \models^v P_2}{\rho, \kappa, \psi, \theta \models^v P_1 | P_2} \qquad \rho, \kappa, \psi, \theta \models^v 0$$

Table 5: Verbose formulation of the analysis. Note that θ is referred to as θ^v elsewhere in this report.

two variants of the calculus are equivalent in every practical aspect. When distinction between the two variants is necessary $\lfloor\!\lfloor E^l \rfloor\!\rfloor$ signifies a LySA term that is the result of removing labels from the labelled LySA term E^l while $\lfloor\!\lfloor P \rfloor\!\rfloor$ signifies the homomorphic extension of this function to processes.

To anticipate the development in Section 5, the sets of crypto-points in annotations have also been instrumented with labels $l_s \in Lab$. These labels will be ignored for now and to avoid confusion they are left out until Section 5 where they are needed.

The succinct formulation of the judgements for terms that uses θ will, thus, be transformed into a verbose formulation that uses θ^v as described above. The analysis component θ^v will sometimes be referred to as a *global term cache*, since it for each label caches the values that the term with that label may evaluate to.

The succinct formulation of the observation predicate ψ may be rewritten in a similar manner by introducing a "global observation cache" to map new labels added the syntax to the values of the observation predicate at that point. However, it turns out that ψ actually already is a *global* component in the original formulation of the analysis and can, therefore, equally well be written in verbose form.

With these two modifications the analysis uses the components summarised in Table 4 while the analysis itself is given in Table 5. Note that this analysis is written in the same Flow Logic style as the initial analysis of the Spi-calculus [7, Table 1].

The following lemma expresses that this verbose analysis is sound with respect to the original analysis:

Lemma 1 *The analysis* $\models^v$ *in Table 5 is* sound *with respect to the original analysis* $\models$ *in Table 3. That is*

(1) If $\rho, \theta^v \models^v E^l$ *then* $\rho \models \lfloor\!\lfloor E^l \rfloor\!\rfloor : \theta^v(l)$

(2) If $\rho, \kappa, \psi, \theta^v \models^v P$ *then* $\rho, \kappa \models \lfloor\!\lfloor P \rfloor\!\rfloor : \psi$.

Proof Both (1) and (2) can be proven by straightforward structural induction in the definition of $\models^v$ in Table 5. **Case (1)** The cases of names and variables are base cases while the induction hypothesis must be used in the case of encrypted terms. **Case (2)** rely on (1) in the case of output, input, and decryption. The details of the proof are given in Appendix A.1. □

4. From Infinite to Finite

The analysis is specified over the *infinite* set of values, $\mathcal{V}$, that contain infinitely many terms built from names and encryption. This poses a problem for the implementation

strategy, since the analysis should be transformed into ALFP formulae interpreted over a *finite* universe. This section addresses this problem by encoding sets of terms into *tree grammars* and give a new analysis that represent sets of terms by the finite set of grammar rules.

Example 1 *As an example of where infinite sets occur in the analysis consider the* LySA *process (ignoring for the moment the annotations of crypto-points):*

$$P \quad \stackrel{\text{def}}{=} \quad \langle n^{l_1} \rangle.\,\mathsf{0} \mid !\,(;\ x).\,\langle \{x^{l_2}\}^{l_4}_{k^{l_3}} \rangle.\,\mathsf{0}$$

The process sends the terms $n, \{n\}_k, \{\{n\}_k\}_k, \{\{\{n\}_k\}_k\}_k, \ldots$ *over the network and in doing so it binds the variable* x *to each of these values. Consequently, the analysis* $\rho, \kappa, \psi, \theta \models^v P$ *specifies that the infinite set* $\{\lfloor n \rfloor, \{\lfloor n \rfloor\}_{\lfloor k \rfloor}, \{\{\lfloor n \rfloor\}_{\lfloor k \rfloor}\}_{\lfloor k \rfloor}, \ldots\}$ *must be a subset of* $\rho(\lfloor x \rfloor)$. □

The transformation of the analysis into a finite representation goes in two steps. The first observation concerns the way that sets of values are used by the analysis $\models^v$ in Table 5: for each term, E^l, the analysis of terms specifies the set of values $\theta^v(l)$ that the term may evaluate to. The analysis of processes also collects sets of values in ρ and κ but these sets are manipulated in "blocks of values" that correspond to the sets in θ^v. It is, for example, always the case that if one of the values from $\theta^v(l)$ is required to be in $\rho(\lfloor x \rfloor)$ then all other values of $\theta^v(l)$ are also required to be in $\rho(\lfloor x \rfloor)$.

As a consequence of this first observation, the analysis of processes in Table 5 can be transformed into an equivalent analysis where ρ and κ collects sets of *blocks of values* rather than sets of values. The actual values in these blocks are still collected in θ^v and a blocks can, therefore, be represented by the label that is needed to access the values in θ^v. For example, $\rho : \lfloor \mathcal{X} \rfloor \to \wp(\mathcal{V})$ can be represented by $\rho^f : \lfloor \mathcal{X} \rfloor \to \wp(Lab)$. The intuition is then that even though $\rho^f(\lfloor x \rfloor)$ is a set of labels then it actually describes the set of values given as the union of applying θ^v to each of the labels in $\rho^f(\lfloor x \rfloor)$, i.e.

$$\rho(\lfloor x \rfloor) = \bigcup_{l \in \rho^f(\lfloor x \rfloor)} \theta^v(l)$$

The rule for the analysis of variables, for example, can then be written as

$$\frac{\forall l' \in \rho^f(\lfloor x \rfloor) : \theta^v(l') \subseteq \theta^v(l)}{\rho^f, \theta^v \models x^l}$$

which is equivalent to the rule in Table 5, though each of the elements l' in ρ^f now has to find their values through θ^v.

Similar modifications can be made to κ throughout, thereby letting $\kappa^f \in \wp(Lab^*)$. In the resulting analysis, sets of values (i.e. possible infinite sets of terms) only appear in

θ^v so the further development will concentrate on a finite representation of this *one* component. This will be the second step of the transformation to a finite analysis. To prepare for this development some theory of terms is reviewed in the next section primarily based on [5].

4.1. Terms, Tree Languages, and Tree Grammars

A (single sorted) *signature*, Σ, is a finite set of *function symbols* that each are associated with a natural number from $\mathbb{N}_0$ called an *arity*. Signatures will be written using the notation $\{f_i, g_j, \cdots, h_k\}$ meaning that f is a function symbol with arity i while g is a function symbol with arity j, etc.

Terms are built by applying function symbols to other terms and sometimes also to elements of an arbitrary set X. The set of terms over a signature Σ and a set X is defined inductively as the smallest set, $T(\Sigma, X)$, such that

$$T(\Sigma, X) = X \cup \{f(t_1, \cdots, t_k) \mid f_k \in \Sigma \wedge t_1 \in T(\Sigma, X) \wedge \cdots \wedge t_k \in T(\Sigma, X)\}$$

If the arity of a function symbol, f, is 0 then the element $f()$ in $T(\Sigma, X)$ is called a *constant* and is often written simply as f. The set of terms generated purely from application of function symbols to other function symbols, i.e. $T(\Sigma, \emptyset)$, is called the set of *ground terms* or the *free term algebra* over Σ.

A set of ground terms may be regarded as formal language over terms. In this context terms are called *trees* and, consequently, these formal languages are called *tree languages*. A tree language can be generated from a tree grammar. A (regular, normalised) *tree grammar*, G, is a quadruple (N, Σ, R, S) where

- N is a set of non-terminals,
- Σ is a signature,
- R is a *finite* mapping of rules with functionality $N \rightarrow \wp(B(\Sigma, N))$ where the set $B(\Sigma, N)$ is defined below, and
- S is a start symbol from N.

If an element u is in $R(A)$ then the pair (A, u) is called a *rule* in the grammar and is often written as $A \rightarrow u$. Sometimes A will be referred to as the *head* of the rule while u is the *body* of rule. The set $B(\Sigma, X)$ of "rule bodies" is the subset of $T(\Sigma, X)$ where all function symbols are applied only to elements of X but not applied recursively to terms:

$$B(\Sigma, X) \stackrel{\text{def}}{=} \{f(A_1, \cdots, A_k) \mid f_k \in \Sigma \wedge A_1 \in X \wedge \cdots \wedge A_k \in X)\}$$

Note that unlike $T(\Sigma, X)$ the set X is not included in $B(\Sigma, X)$.

A tree grammar $G = (N, \Sigma, R, S)$ can be used to generate a set of ground terms. The set generated by starting from a non-terminal A for which there is a rule $A \to u$ in R is found by recursively substituting bodies of rules into u until a ground term is found. This set is denoted $L(G, A)$ and is defined inductively as the smallest set satisfying

$$L(G, A) = \{f(t_1, \ldots, t_k) \mid f(A_1, \ldots, A_k) \in R(A) \wedge t_1 \in L(G, A_1) \wedge \ldots \wedge t_k \in L(G, A_k)\}$$

Notice that $L(G, A)$ is indeed a subset of $T(\Sigma, \emptyset)$. Since, the rule mapping R is the only component of the grammar G that is mentioned by the above definition the set is sometimes written $L(R, A)$ instead of $L(G, A)$. The tree languages *generated* by the tree grammar $G = (N, \Sigma, R, S)$ is the set of ground terms found by starting at the start symbol S:

$$L(G) \stackrel{\text{def}}{=} L(G, S)$$

Readers familiar with tree automata may be interested to recall that languages generated by a normalised, regular tree grammar, so-called *regular tree languages*, are equivalent to *recognisable tree languages* i.e. languages recognised by a bottom-up tree automaton. These languages are closed under union, intersection, and complementation [5, Theorem 5].

4.2. Tree Grammars for the Analysis

The analysis uses sets of ground terms in the component $\theta^v : Lab \to \wp(\mathcal{V})$. In particular, $\theta^v(l)$ denotes the set of ground terms that the LySA term E^l may evaluate to. The idea is to get a finite representation of the analysis by representing this set of terms as a tree grammar G such that $L(G)$ is precisely the set $\theta^v(l)$. Each set in the range of θ^v will be represented by its own unique tree grammar, G_l, in a component θ^f such that:

$$\theta^v(l) = L(\theta^f(l))$$

To precisely describe the tree grammars used by the analysis a signature is needed. Consider the set

$$S \stackrel{\text{def}}{=} \{n_0 \mid n \in \mathcal{N}\} \cup \{enc_k^{\ell,\mathcal{L}} \mid k \in \mathbb{N} \wedge \ell \in \mathcal{C} \wedge \mathcal{L} \in \wp(\mathcal{C})\}$$

where names are regarded as constants (i.e. 0-ary function symbols) and enc is a k-ary function symbol denoting encrypted terms annotated with crypto-points. Unfortunately, S is not a signature, since it is an *infinite* set. Luckily, the analysis of any specific process P only uses terms over a finite subset S. More precisely, the analysis of a process P uses terms generated by the signature

$$\begin{array}{lll} \Sigma_{\text{LySA}} \stackrel{\text{def}}{=} & \{\lfloor n \rfloor_0 & \mid n \text{ is a name used in } P\} \cup \\ & \{enc_{k+1}^{\ell,\mathcal{L}} & \mid k \text{ is the arity of an encryption annotated with } \ell \text{ and } \mathcal{L} \text{ in } P\} \end{array}$$

Often $enc^{\ell,\mathcal{L}}(A_0, A_1, \cdots, A_k)$ will be written as $\{A_1, \cdots, A_k\}^{\ell}_{A_0}[\mathsf{dest}\ \mathcal{L}]$.

The tree grammars used in the analysis will represent the evaluation of labelled LySA terms and the set of non-terminals for these tree grammars is chosen to be set of these labels, Lab. The grammars will then have rules where the head is a label and the body is from $B(\Sigma_{\text{LySA}}, Lab)$ i.e. rules will be on the form

$$\begin{array}{lcll} l & \to & \lfloor n \rfloor & \text{or} \\ l & \to & \{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] & \end{array}$$

The tree grammars used in the analysis will on the form:

$$\theta^f(l) = G_l = (Lab, \Sigma_{\text{LySA}}, R_l, l)$$

To lighten the notation one may disregard the set Lab and Σ_{LySA} since they are easily derived given a specific process. Note also that the start symbol of any grammar G_l always is the label l and can therefore be left implicit. That means that only the rules, R_l, need to be mentioned in the analysis.

The rules of the grammars will be specified in the analysis of terms. The grammar G_l for a name n^l simply contains grammar rule $l \to \lfloor n \rfloor$ in R_l. The grammar, G_l, for an encrypted term $\{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}]^l$ should include a rule $l \to \{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}]$ for the term itself in R_l as well as rules for all the subterms $E_0^{l_0}, \cdots, E_k^{l_k}$ and their subterms etc. Additionally, the grammars $G_{l_0}, \cdots, G_{l_k}$ for the subterms should *also* contain rules for each subterm and, in fact, the rules will be *exactly the same* as for the rules for the subterm in G_l. For example, if the rule mapping R_l contains a rule $l_0 \to u_0$ then the rule mapping R_{l_0} will contain the exact same rule.

The tree grammar G_l that describe the evaluation of a variable x^l must meet the requirement from analysis that uses blocks of values as sketched on page 8. This amounts to requiring that

$$\forall l' \in \rho^f(\lfloor x \rfloor) : L(G_{l'}) \subseteq L(G_l)$$

The tree grammar G_l will include rules in R_l on form $l \to u$. The above subset requirement can be achieved by having rules in R_l of the form $l \to u$ whenever $l' \to u$ is in $R_{l'}$ as well as copying all rules for non-terminals mentioned in u from $R_{l'}$ to R_l. That is, the analysis of a variable x^l should require that

$$\forall l' \in \rho^f(\lfloor x \rfloor) : \forall u \in R_{l'}(l') : u \in R_l(l)$$

as well as having a requirement for copying rules of subterms.

Notice that both the grammars for encryption and for variables require that rules are copied between grammars. In fact, the exact same rules will be duplicated in all the relevant grammars. That is, if both R_{l_1} and R_{l_2} define rules for some label l then they define exactly the same rules for l (i.e. $R_{l_1}(l) = R_{l_2}(l)$). Therefore, it is unnecessary

to keep copies of all the rule mappings for the different grammars and instead these mappings are merged into one mapping, γ, representing in effect all the grammars.

Remember that the intuition of θ^f is that $\theta^f(l)$ should return the grammar $G_l = (Lab, \Sigma_{\text{LySA}}, R_l, l)$. However, all the components in the grammar can either be left implicit or be represented by the on rule mapping γ so θ^f will be dispensed with all together.

4.3. The Finite Analysis

The components in the finite analysis are given in Table 6. As describes earlier, the analysis uses ρ^f and κ^f to collect the labels of LySA terms rather than the actual values that the terms may evaluate to. The set of values that the terms may evaluate to is given by tree grammars that all are represented by the rules in γ.

The analysis is given in Table 7 where the analysis of LySA terms specifies the rules of the tree grammars in the component γ. Note that the component θ^f is not a part of the analysis specification but is implicitly represented by the rules in γ.

The analysis of output simply specifies that the labels of all the LySA terms that are sent should be in κ^f. Correspondingly, input specifies that labels from κ^f should be in ρ^f when variables may become bound. The pattern matching that takes place in input specifies that two terms $E_1^{l_1}$ and $E_2^{l_2}$ may match if the intersection of the sets of values that they may evaluate to is non-empty; that is when $L(\gamma, l_1) \cap L(\gamma, l_2)$ is non-empty. The specification of matching in Table 5 uses the membership operator $\in$ that ignores annotations on terms. In the same spirit an intersection operator $\barwedge$ is introduced as

$$S_1 \barwedge S_2 = \{V \in S_1 \cup S_2 \mid V \in S_1 \wedge V \in S_2\}$$

and this operator in used for the intersection used for matching. The definition says that if V is in S_1 then the annotations should be ignored when membership of S_2 is checked. The modifications of the analysis of decryption are similar to the changes made in input.

ρ^f	:	$\lfloor \mathcal{X} \rfloor \to \wp(Lab)$	Variable environment
κ^f	$\in$	$\wp(Lab^*)$	Network component
γ	:	$Lab \to B(\Sigma_{\text{LySA}}, Lab)$	Rules in tree grammars
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
l	$\in$	Lab	Labels on terms / non-terminals
u	$\in$	$B(\Sigma_{\text{LySA}}, Lab)$	Bodies of tree grammar rules

Table 6: Components used in the finite analysis $\models^f$.

$$\frac{\lfloor n \rfloor \in \gamma(l)}{\rho,\gamma \models^f n^l} \qquad \frac{\forall l' \in \rho(\lfloor x \rfloor) : \gamma(l') \subseteq \gamma(l)}{\rho,\gamma \models^f x^l}$$

$$\frac{\wedge_{i=0}^{k}\ \rho,\gamma \models^f E_i^{l_i} \wedge \{l_1,\cdots,l_k\}_{l_0}^{\ell}[\mathsf{dest}\ \mathcal{L}] \in \gamma(l)}{\rho,\gamma \models^f \{E_1^{l_1},\cdots,E_k^{l_k}\}_{E_0^{l_0}}^{\ell}[\mathsf{dest}\ \mathcal{L}]^l}$$

$$\frac{\wedge_{i=1}^{k}\ \rho,\gamma \models^f E_i^{l_i} \wedge \langle l_1,\cdots,l_k\rangle \in \kappa \wedge \rho,\kappa,\psi,\gamma \models^f P}{\rho,\kappa,\psi,\gamma \models^f \langle E_1^{l_1},\cdots,E_k^{l_k}\rangle.P}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{j}\ \rho,\gamma \models^f E_i^{l_i} \wedge \\ \forall \langle l_1',\cdots,l_k'\rangle \in \kappa :\ (\wedge_{i=1}^{j}\ L(\gamma,l_i') \circledcirc L(\gamma,l_i) \neq \emptyset) \Rightarrow\ (\wedge_{i=j+1}^{k}\ l_i' \in \rho(\lfloor x_i \rfloor) \wedge \\ \rho,\kappa,\psi,\gamma \models^f P)\end{array}}{\rho,\kappa,\psi,\gamma \models^f (E_1^{l_1},\cdots,E_j^{l_j};\ x_{j+1},\cdots,x_k).P}$$

$$\frac{\begin{array}{c}\rho,\gamma \models^f E^l \ \wedge \wedge_{i=0}^{j}\ \rho,\gamma \models^f E_i^{l_i} \wedge \\ \forall\ \{l_1',\cdots,l_k'\}_{l_0'}^{\ell'}[\mathsf{dest}\ \mathcal{L}'] \in \gamma(l) :\ (\wedge_{i=0}^{j}\ L(\gamma,l_i') \circledcirc L(\gamma,l_i) \neq \emptyset)\ \Rightarrow \\ (\wedge_{i=j+1}^{k}\ l_i' \in \rho(\lfloor x_i \rfloor) \wedge \\ (\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L}) \Rightarrow (\ell',\ell) \in \psi \wedge \\ \rho,\kappa,\psi,\gamma \models^f P)\end{array}}{\rho,\kappa,\psi,\gamma \models^f \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1},\cdots,E_j^{l_j};\ x_{j+1},\cdots,x_k\}_{E_0^{l_0}}^{\ell}\ [\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P}$$

$$\frac{\rho,\kappa,\psi,\gamma \models^f P}{\rho,\kappa,\psi,\gamma \models^f (\nu\, n)P} \qquad \frac{\rho,\kappa,\psi,\gamma \models^f P}{\rho,\kappa,\psi,\gamma \models^f\ !P}$$

$$\frac{\rho,\kappa,\psi,\gamma \models^f P_1 \wedge \rho,\kappa,\psi,\gamma \models^f P_2}{\rho,\kappa,\psi,\gamma \models^f P_1|P_2} \qquad \rho,\kappa,\psi,\gamma \models^f 0$$

Table 7: Finite representation of the analysis. Note that ρ and κ are referred to as ρ^f and κ^f, respectively, elsewhere in this report.

Example 2 *Recall the process from Example 1 that on execution may generate arbitrarily deep encryption:*

$$P \quad \stackrel{\text{def}}{=} \quad \langle n^{l_1} \rangle.\,0 \mid !\,(;\ x).\,\langle \{x^{l_2}\}^{l_4}_{k^{l_3}} \rangle.\,0$$

The finite analysis $\models^f$ *in Table 7 specifies that*

$$\begin{array}{lcl} \kappa & \supseteq & \{l_1, l_4\} \\ \rho(x) & \supseteq & \{l_1, l_4\} \end{array} \qquad\qquad \begin{array}{ll} \gamma: & l_1 \to \lfloor n \rfloor \\ & l_2 \to \lfloor n \rfloor \\ & l_2 \to \{l_2\}_{l_3} \\ & l_3 \to \lfloor k \rfloor \\ & l_4 \to \{l_2\}_{l_3} \end{array}$$

Note in particular, that the grammar for l_2 *is "created" by "copying" the bodies,* u*, of the rules where the head is in* $\rho(x)$ *into bodies of rules* $l_2 \to u$*. This creates a circularity in the rule for* l_2 *such that* $L(\gamma, l_2)$ *is the infinite set* $\{\lfloor n \rfloor, \{\lfloor n \rfloor\}_{\lfloor k \rfloor}, \{\{\lfloor n \rfloor\}_{\lfloor k \rfloor}\}_{\lfloor k \rfloor}, \ldots\}$ *i.e. precisely the set that* x *may evaluate to in Example 1.* □

The step of going from an infinite domain to a finite domain using tree grammars was introduced in the analysis of the Spi-calculus [7]. There, however, the analysis components corresponding to ρ and κ contain bodies of the grammar rules (i.e. elements of $B(\Sigma, Lab)$) rather than the heads of the rules (i.e. elements of Lab) as the components do here. The main motivation for this difference is that matching in LySA takes place both at decryption *and at input* rather than only at decryption as in the Spi-calculus. The analysis of matching requires checks of non-empty intersection of two tree languages and with the current definitions these checks require the use of non-terminals (i.e. labels). By letting κ contain such non-terminals the matching can be preformed in a analogue way both in input and in decryption, which seem nice since they identical in the original analysis.

Lemma 2 *The analysis* $\models^f$ *in Table 7 is* sound *with respect to the analysis* $\models^v$ *in Table 5. That is*

(1) if $\rho^f, \gamma \models^f E$ *then* $\rho, \theta^v \models^v E$ *where for all* l *and all* x

$$\theta^v(l) = L(\gamma, l)$$
$$\rho(\lfloor x \rfloor) = \bigcup_{l' \in \rho^f(\lfloor x \rfloor)} \theta^v(l')$$

(2) if $\rho^f, \kappa^f, \psi, \gamma \models^f P$ *then* $\rho, \kappa, \psi, \theta^v \models^v P$ *where* ρ *and* θ^v *are as in (1) and*

$$\kappa = \{\langle V_1, \cdots, V_k \rangle \mid \langle l_1, \cdots l_k \rangle \in \kappa^f \wedge V_1 \in \theta^v(l_1) \wedge \cdots \wedge V_k \in \theta^v(l_k)\}$$

Proof The proof goes by structural induction on terms and processes in the definition of $\models^f$. The proof uses the definition of $L(\gamma, l)$ to establish the desired result as shown in detail in Appendix A.2. □

5. Removing Polyvariance

Communication and encryption in LySA are polyadic in the sense that they work on *sequences* of terms. For the analysis, this means that e.g. κ^f contains *sequences* of labels. The predicate symbols of an ALFP formula is, however, interpreted over an *unstructured* universe by the Succinct Solver[1]. Thus, sequences have to be encoded directly into predicates of a *fixed arity*. Furthermore, the tree grammars rules in γ have a body that contains sequences of information, which also has to be encoded into predicates.

The basic idea is to let the analysis work over *families* of predicates where each member of the family represent a sequence of a specific length. Though the families will be infinite, only finitely many predicates of a family are be needed to analyse a specific process.

Recall from [3, Section 5] that a process P is said to have type $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}_\mathsf{Enc})$ whenever (1) it has no free variables, (2) its free names are in $\mathcal{N}_\mathsf{f}$, (3) all the arities used for sending or receiving are in $\mathcal{A}_\kappa$, and (4) all the arities used for encryption or decryption are in $\mathcal{A}_\mathsf{Enc}$. Clearly, all the sets $\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa$, and $\mathcal{A}_\mathsf{Enc}$ will be finite and only predicates based on the last two sets are needed for an analysis of P.

5.1. Communication

The analysis describes the network component κ by a family of relations:

$$\kappa^p_k = \{m \mid m \text{ is in } \kappa^f \text{ and has length } k\}$$

such that $\kappa^p_k \subseteq \wp(Lab^k)$. The transformation of the analysis $\models^f$ into an analysis $\models^p$ that uses the family of relations is quite simple, since in $\models^p$ the analysis of output and input only refers to sequences in κ^f with the same length. Thus, the analysis $\models^f$ simply need to refer to the corresponding κ^p_k instead of κ^f. The analysis of a process of type $(\mathcal{N}_\mathsf{f}, \mathcal{A}_\kappa, \mathcal{A}_\mathsf{Enc})$ will, consequently, only use the (finitely many) relations κ_k where $k \in \mathcal{A}_\kappa$.

5.2. Encrypted Values

Encrypted values are represented in the tree grammar rules of $\gamma : Lab \to \wp(B(\Sigma_{\mathrm{LySA}}, Lab))$. The bodies of the rules are either a canonical name $\lfloor n \rfloor$ or a polyadic k-ary encryption $\{l_1, \ldots, l_k\}^\ell_{l_0}[\mathsf{dest}\ \mathcal{L}]$. These encryptions will be represented by a family of auxiliary relations, σ_k, such that an element in σ_k represents a sequence of k terms encrypted.

[1] The implementation of the Succinct Solver has an automatic encoding of structured terms into relations over an unstructured universe that is not used by the analysis presented in this report.

A first suggestion for an encoding the k-ary term $\{l_1, \ldots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}]$ is to let σ_k contain the tuple $(l, l_0, l_1, \cdots, l_k, \ell, \mathcal{L})$. Here, the first element, l, serves as a unique pointer into σ_k, while the remaining elements are simply the content of the term. Now, γ will be changed into a component γ^f that contains the canonical names as before but instead of encryptions contains unique pointers, l, into σ_k.

The encoding works fine to get rid of the polyvariance and an analysis of a given program of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_{\kappa}, \mathcal{A}_{\mathsf{Enc}})$ will only need the (finitely many) auxiliary relations σ_k where $k \in \mathcal{A}_{\mathsf{Enc}}$. However, the last element, $\mathcal{L}$, is *a set* of crypto-points and since the Succinct Solver works on unstructured universes it cannot directly represent these sets. Hence, yet another encoding is needed. The idea is to introduce a unique label in the syntax at all sets of crypto-points as already mentioned in Section 3 and add a cache $\delta : Lab \to \wp(\mathcal{C})$ to the analysis, which stores the sets of crypto-points for each label. The transformation from the previous analysis $\models^p$ will, for example, let

$$\{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}^{l_s}] \in \gamma(l)$$

be translated into

$$l \in \gamma^f(l)\ \ \wedge\ \ (l, l_0, \cdots, l_k, \ell, l_s) \in \sigma_k\ \ \wedge\ \ \wedge_{\ell' \in \mathcal{L}}\, \ell' \in \delta(l_s)$$

A similar encoding is used to represent the sets of crypto-points in the annotations of decryption.

5.3. The Non-polyvariant Analysis

The components used in the analysis with polyvariance removed are shown in Table 8. The analysis specification is given in Table 9 and in the judgements κ and σ denote the *families* of relations representing messages and encryption, respectively.

In the analysis of the Spi-calculus [7] sequences are encoded into binary/ternary relations representing, in effect, a linked list. This encoding gives good time-complexity with relation to a rather pessimistic complexity measure though it in practise is likely to be slower that the encoding presented here.

ρ^f	:	$\lfloor \mathcal{X} \rfloor \to \wp(Lab)$	Variable environment
κ^p_k	$\in$	$\wp(Lab^k)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
γ^p	:	$Lab \to \wp(\lfloor \mathcal{N} \rfloor \cup Lab)$	Rules in tree grammars
σ_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Symmetric encryptions
δ	:	$Lab \to \wp(\mathcal{C})$	Sets of crypto-points
l	$\in$	Lab	Labels on terms and sets of crypto-points

Table 8: Components used in the analysis $\models^p$ where polyvariance is removed.

$$\frac{\lfloor n \rfloor \in \gamma(l)}{\rho,\gamma,\sigma,\delta \models^p n^l} \qquad \frac{\forall l' \in \rho(\lfloor x \rfloor) : \gamma(l') \subseteq \gamma(l)}{\rho,\gamma,\sigma,\delta \models^p x^l}$$

$$\frac{\wedge_{i=0}^{k}\, \rho,\gamma,\sigma,\delta \models^p E_i^{l_i} \wedge (l,l_0,l_1,\cdots,l_k,\ell,l_s) \in \sigma_k \wedge l \in \gamma(l) \wedge \wedge_{\ell' \in \mathcal{L}}\, \ell' \in \delta(l_s)}{\rho,\gamma,\sigma,\delta \models^p \{E_1^{l_1},\cdots,E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}]^l}$$

$$\frac{\wedge_{i=1}^{k}\, \rho,\gamma,\sigma,\delta \models^p E_i^{l_i} \wedge (l_1,\cdots,l_k) \in \kappa_k \wedge \rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P}{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p \langle E_1^{l_1},\cdots,E_k^{l_k}\rangle.\, P}$$

$$\frac{\begin{array}{c}\wedge_{i=1}^{j}\, \rho,\gamma,\sigma,\delta \models^p E_i^{l_i} \ \wedge \\ \forall (l'_1,\cdots,l'_k) \in \kappa_k :\ (\wedge_{i=1}^{j}\, L(\gamma,l'_i) \,\text{⋒}\, L(\gamma,l_i) \neq \emptyset) \Rightarrow \ (\wedge_{i=j+1}^{k}\, l'_i \in \rho(\lfloor x_i \rfloor)\ \wedge \\ \rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P)\end{array}}{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p (E_1^{l_1},\cdots,E_j^{l_j};\ x_{j+1},\cdots,x_k).\, P}$$

$$\frac{\begin{array}{c}\rho,\gamma,\sigma,\delta \models^p E^l \ \ \wedge \wedge_{\ell' \in \mathcal{L}}\, \ell' \in \delta(l_s) \ \ \wedge \wedge_{i=0}^{j}\, \rho,\gamma,\sigma,\delta \models^p E_i^{l_i} \ \wedge \\ \forall\ (l',l'_0,\cdots,l'_k,\ell',l'_s) \in \sigma_k : l' \in \gamma(l) \Rightarrow\ (\wedge_{i=0}^{j}\, L(\gamma,l'_i) \,\text{⋒}\, L(\gamma,l_i) \neq \emptyset)\ \Rightarrow \\ (\ \wedge_{i=j+1}^{k}\, l'_i \in \rho(\lfloor x_i \rfloor)\ \wedge \\ (\ell \notin \delta(l'_s) \vee \ell' \notin \delta(l_s)) \Rightarrow (\ell',\ell) \in \psi\ \wedge \\ \rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P)\end{array}}{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1},\cdots,E_j^{l_j};\ x_{j+1},\cdots,x_k\}^{\ell}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P}$$

$$\frac{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P}{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p (\nu\, n)P} \qquad \frac{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P}{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p\, !P}$$

$$\frac{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P_1 \ \wedge\ \rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P_2}{\rho,\kappa,\psi,\gamma,\sigma,\delta \models^p P_1|P_2} \qquad \rho,\kappa,\psi,\gamma,\sigma,\delta \models^p 0$$

Table 9: The analysis $\models^p$ with polyvariance removed. Note that ρ, κ, and γ are known as ρ^f, κ^p, and γ^p, respectively, elsewhere in this report.

Soundness of the analysis mainly concerns mapping the representation of encryptions in σ_k and δ back to bodies in the rules of γ.

Lemma 3 *The analysis* $\models^p$ *in Table 9 is* sound *with respect to the analysis* $\models^f$ *in Table 7. That is:*

(1) If $\rho^f, \gamma^p, \sigma, \delta \models^p E$ *then* $\rho^f, \gamma \models^f E$ *where for all* l

$$\begin{array}{rcll} \gamma(l) & = & \{ \lfloor n \rfloor \mid \lfloor n \rfloor \in \gamma^p(l) \} \cup & \\ & & \{ \{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] \mid & l' \in \gamma^p(l) \wedge \\ & & & (l', l_0, \cdots, l_k, \ell, l_s) \in \sigma_k \wedge \mathcal{L} = \delta(l_s)\} \end{array}$$

(2) If $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \delta \models^p P$ *then* $\rho^f, \kappa^f, \psi, \gamma \models^f P$ *where* γ *is as above and*

$$\kappa^f = \bigcup_{i \in \mathbb{N}} \kappa^p_i$$

Proof The proof goes by structural induction and uses the uniqueness of labels, l_s, at sets of crypto-points, $\mathcal{L}$, to ensure that $\delta(l_s)$ is precisely the set $\mathcal{L}$. A detailed proof is given in Appendix A.3. □

6. Generating ALFP Logic Formulae

The next step is to create a generation function, $\mathcal{G}$, which takes a process and returns an ALFP formula. The function is defined on the structure of terms and processes by taking each of the rules in the analysis $\models^p$ in Table 9 and giving an ALFP formula equivalent to the hypothesis of these rules. Such an equivalent formula is relatively easy to find, since the analysis $\models^p$ is specified using analysis components over elements from a finite, unstructured universe.

The generation function is given in Table 11 and uses the convention that all relations (that is all analysis components that are powersets) are interpreted as predicates with the

ρ^g	$\in$	$\wp(\lfloor \mathcal{X} \rfloor \times Lab)$	Variable environment
κ^p_k	$\in$	$\wp(Lab^k)$	Network component
ψ	$\in$	$\wp(\mathcal{C} \times \mathcal{C})$	Error component
γ^g	$\in$	$\wp(Lab \times (\lfloor \mathcal{N} \rfloor \cup Lab))$	Rules in tree grammars
σ_k	$\in$	$\wp(Lab^{k+2} \times \mathcal{C} \times Lab)$	Symmetric encryptions
δ^g	$\in$	$\wp(Lab \times \mathcal{C})$	Sets of crypto-points
l	$\in$	Lab	Labels on terms and set of crypto-points

Table 10: Predicates used in the ALFP formula generated by $\mathcal{G}(P)$.

$$
\begin{array}{lcl}
\mathcal{G}(n^l) & = & \gamma(l, \lfloor n \rfloor) \\
\mathcal{G}(x^l) & = & \forall l' : \rho(\lfloor x \rfloor, l') \Rightarrow \forall u : \gamma(l', u) \Rightarrow \gamma(l, u) \\
\mathcal{G}((\{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}])^l) & = & (\wedge_{i=0}^{k} \mathcal{G}(E_i^{l_i})) \wedge \\
 & & \sigma_k(l, l_0, \cdots, l_k, \ell, l_s) \wedge \gamma(l, l) \wedge \wedge_{\ell' \in \mathcal{L}}\ \delta(l_s, \ell')
\end{array}
$$

$$
\begin{array}{lcl}
\mathcal{G}(0) & = & \text{true} \\
\mathcal{G}(\langle E_1^{l_1}, \cdots, E_k^{l_k} \rangle . P) & = & \wedge_{i=1}^{k}\ \mathcal{G}(E_i^{l_i}) \wedge \kappa_k(l_1, \cdots, l_k) \wedge \mathcal{G}(P) \\
\mathcal{G}((E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k).P) = & & \wedge_{i=1}^{j}\ \mathcal{G}(E_i^{l_i}) \wedge \\
 & & \forall l'_1, \cdots, l'_k : \kappa_k(l'_1, \cdots, l'_k) \Rightarrow \\
 & & (\wedge_{i=1}^{j}\ NEI(l'_i, l_i)) \Rightarrow\ \ (\wedge_{i=j+1}^{k}\ \rho(\lfloor x_i \rfloor, l'_i)\ \wedge \mathcal{G}(P)) \\
\mathcal{G}(P_1 | P_2) & = & \mathcal{G}(P_1) \wedge \mathcal{G}(P_2) \\
\mathcal{G}((\nu\, n)P) & = & \mathcal{G}(P) \\
\mathcal{G}(!\, P) & = & \mathcal{G}(P) \\
\mathcal{G}(\mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k\}^{\ell}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P) = & & \\
 & & \mathcal{G}(E^l)\ \ \wedge \wedge_{\ell' \in \mathcal{L}}\ \delta(l_s, \ell')\ \ \wedge \wedge_{i=0}^{j}\ \mathcal{G}(E_i^{l_i}) \wedge \\
 & & \forall l', l'_0, \cdots, l'_k, \ell', l'_s : \sigma_k(l', l'_0, \cdots, l'_k, \ell', l'_s) \wedge \gamma(l, l') \Rightarrow \\
 & & (\wedge_{i=0}^{j}\ NEI(l'_i, l_i)) \Rightarrow \\
 & & \quad (\wedge_{i=j+1}^{k}\ \rho(\lfloor x_i \rfloor, l'_i) \wedge \\
 & & \quad (\neg\delta(l'_s, \ell) \vee \neg\delta(l_s, \ell')) \Rightarrow \psi(\ell', \ell) \wedge \\
 & & \quad \mathcal{G}(P))
\end{array}
$$

$$
\begin{array}{l}
\forall l_1, l_1 : (\exists u : \mathcal{N}_{\mathsf{c}}(u) \wedge \gamma(l_1, u) \wedge \gamma(l_2, u)) \Rightarrow NEI(l_1, l_2) \\
\wedge\wedge_{k \in \mathcal{A}_{\mathsf{Enc}}} \\
\forall l_1, l_2 : (\exists l'_1, l_{10}, \cdots, l_{1k}, \ell_1, l_{1s}, l'_2, l_{20}, \cdots, l_{2k}, \ell_2, l_{2s} : \\
\qquad \sigma_k(l'_1, l_{10}, \cdots, l_{1k}, \ell_1, l_{1s}) \wedge \\
\qquad \sigma_k(l'_2, l_{20}, \cdots, l_{2k}, \ell_2, l_{2s}) \wedge \\
\qquad NEI(l_{10}, l_{20}) \wedge \ldots \wedge NEI(l_{1k}, l_{2k}) \wedge \\
\qquad \gamma(l_1, l'_1) \wedge \gamma(l_2, l'_2)) \\
\qquad\qquad \Rightarrow NEI(l_1, l_2)
\end{array}
$$

Table 11: Generation of the ALFP logic formulae. The specification of the predicate *NEI* is added to the formula produces by $\mathcal{G}(P)$ predicate when P is of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_{\kappa}, \mathcal{A}_{\mathsf{Enc}})$ and uses the canonical names in $\mathcal{N}_{\mathsf{c}}$. Note that ρ, κ, γ, and δ are known as $\rho^g, \kappa^p, \gamma^g$, and δ^g, respectively, elsewhere in this report.

notational convention that set membership $e \in R$ is written as the predicate $R(e)$. Three of the components in the analysis $\models^p$ are not relations but rather functions with type $f : A \to \wp(B)$ as shown in Table 8. Such functions can be represented isomorphically by a relation $R \in \wp(A \times B)$ and the functions

$$rel(f)(a) = \{(a, b) \mid b \in f(a)\} \qquad \text{and its inverse} \qquad fun(R)(a) = \{b \mid R(a, b)\}$$

may be used to map between the two domains. The new domains of the analysis components are given in Table 10 while the formulation of the generation function is in Table 11. For the components where the domain has changed from functions to relations the following transformations have been performed on the specification of the analysis:

- Membership, such as $b \in f(a)$ is written $f(a, b)$.
- Subset, such as $f(a) \subseteq f(b)$ is extended to $\forall x : f(a, x) \Rightarrow f(b, x)$.
- Quantifications are expanded so e.g. $\forall x \in f(y) : \ldots$ is written $\forall x, y : f(y, x) \Rightarrow \ldots$

Finally, the analysis $\models^p$ in Table 9 contains tests of non-emptiness of language intersection such as

$$L(\gamma^g, l_1) \circledcirc L(\gamma^g, l_2) \neq \emptyset$$

Following closely the development from the analysis of the Spi-calculus in [7] this test is axiomatised in a *global* auxiliary predicate *NEI*. The axiomatisation is given in Table 11 and is defined on the structure of values in the body of the grammar rules in γ^p, namely: names represented by $\lfloor n \rfloor$ and encryptions represented by labels. Note that the first part of the axiomatisation uses the set of canonical names, $\mathcal{N}_c$, used in the process that is analysed. The second part recursively checks all sub-components but leaves out checks on the crypto-points since this is an axiomatisation of $\circledcirc$ and not ordinary set intersection $\cap$.

As a final comment to Table 11, note that for clarity of the presentation the generation function does not adhere to the strict *syntactic* rules for stratified negation required by the Succinct Solver though these will be enforced in the implementation as described in Section 8.

Lemma 4 *A solution to the formula generated by the function $\mathcal{G}$ defined in Table 11 is* sound *with respect an analysis result which satisfies the analysis $\models^p$ defined in Table 9:*

(1) If $(\rho^g, \gamma^g, \sigma, \delta^g)$ satisfies $\mathcal{G}(E)$ and the formula for NEI then $\rho^f, \gamma^p, \sigma, \delta \models^p E$ where $\rho^f = fun(\rho^g)$, $\gamma^p = fun(\gamma^g)$, and $\delta = fun(\delta^g)$.

(2) If $(\rho^g, \kappa^p, \psi, \gamma^g, \sigma, \delta^g)$ satisfies $\mathcal{G}(P)$ as well as the formula for NEI then also $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \delta \models^p P$ where ρ^f, γ^p, and δ are as above.

Proof That $NEI(l_1, l_2)$ is an axiomatisation of $L(\gamma, l_1) \Subset L(\gamma, l_2)$ can be seen by expanding the definitions of the latter and getting the first.

The rest of the proof proceeds by structural induction on terms and processes and involves only straightforward expansions of definitions of set membership, set inclusion, etc. and simple rewriting of the formula. □

7. Dolev-Yao Attacker

A protocol, specified as a LySA process, will be analysed together with a so-called Dolev-Yao attacker. In [3], the attacker is specified as a formula, $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$, that describes the capabilities of the attacker through its impact on the analysis components. The intend is that the analysis components should satisfy both the analysis specified in Table 3 as well as the formula $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ and this intend is formalised as Theorem 5 in [3].

One implementation strategy for the attacker is to let the formula $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ undergo the same transformations as the analysis did in Sections 3 to 6. This strategy, however, involves a rather heavy proof burden since one must prove both that each transformation gives an equivalent formula as well as prove that the transformations correspond to the transformation done for the analysis.

Another implementation strategy, which is pursued in this report, is to analyse a process Q_{hard} for which the analysis is equivalent to formula $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$. This implementation strategy was originally applied in [6]. That it is plausible to adapt this strategy to LySA is suggested by Theorem 4 in [3] that reads

> "There exists an attacker Q_{hard} of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$ such that the formula $\rho, \kappa \models \overline{Q_{hard}} : \psi$ is equivalent to the formula $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_\kappa, \mathcal{A}_{\mathsf{Enc}})$."

Furthermore, the proof of Theorem 4 in [3] gives such a process defined as[2]

$$Q_{hard} \stackrel{\text{def}}{=} !\,(|_{k\in\mathcal{A}_\kappa} Q_1^k \mid |_{k\in\mathcal{A}_{\mathsf{Enc}}} Q_2^k \mid |_{k\in\mathcal{A}_{\mathsf{Enc}}} Q_3^k \mid |_{k\in\mathcal{A}_\kappa} Q_4^k \mid Q_5)$$

where

$$\begin{aligned}
Q_1^k &= (;\ z_1, \cdots, z_k).\,\mathsf{0}\\
Q_2^k &= (;\ z).\,(;\ z_0).\,\mathsf{decrypt}\ z\ \mathsf{as}\ \{;\ z_1, \cdots, z_k\}^{\ell_\bullet}_{z_0}\ [\mathsf{orig}\ \mathcal{C}]\ \mathsf{in}\ \mathsf{0}\\
Q_3^k &= (;\ z_0).\cdots(;\ z_k).\,\langle\{z_1, \cdots, z_k\}^{\ell_\bullet}_{z_0}[\mathsf{dest}\ \mathcal{C}]\rangle.\,\mathsf{0}\\
Q_4^k &= (;\ z_1).\cdots(;\ z_k).\,\langle z_1, \cdots, z_k\rangle.\,\mathsf{0}\\
Q_5^k &= \langle n_\bullet\rangle.\,\mathsf{0} \mid \langle n_1\rangle.\,\mathsf{0} \mid \cdots \mid \langle n_m\rangle.\,\mathsf{0} \mid (;\ z).\,\mathsf{0}
\end{aligned}$$

and it is assumed that $1 \in \mathcal{A}_\kappa$, all the variables $z, z_0, z_1, \cdots, z_k$ all have the canonical representative $z_\bullet$, and $\mathcal{N}_{\mathsf{f}} = \{n_1, \cdots, n_m\}$.

[2] Certain typos from [3] have been corrected in the definition of Q_{hard}.

An ALFP formula, which is equivalent to $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$, can then be found by taking $\mathcal{G}$ on Q_{hard}. However, $\mathcal{G}$ is defined on labelled LySA where labels are added at the applied occurrences of variables as well as at sets of crypto-points. To transform Q_{hard} into a labelled LySA process with an equivalent analysis two unique labels, $l_\bullet$ and $l_{\mathcal{C}}$, are added giving the following process

$$R_{hard} \stackrel{\text{def}}{=} !\,(|_{k\in\mathcal{A}_\kappa} R_1^k \mid |_{k\in\mathcal{A}_{\mathsf{Enc}}} R_2^k \mid |_{k\in\mathcal{A}_{\mathsf{Enc}}} R_3^k \mid |_{k\in\mathcal{A}_\kappa} R_4^k \mid R_5)$$

where

$$\begin{aligned}
R_1^k &= (;\ z_1,\cdots,z_k).\,0\\
R_2^k &= (;\ z).\,(;\ z_0).\,\mathsf{decrypt}\ z^{l_\bullet}\ \mathsf{as}\ \{;\ z_1,\cdots,z_k\}^{\ell_\bullet}_{z_0^{l_\bullet}}\ [\mathsf{orig}\ \mathcal{C}^{l_{\mathcal{C}}}]\ \mathsf{in}\ 0\\
R_3^k &= (;\ z_0).\cdots(;\ z_k).\,\langle\{z_1^{l_\bullet},\cdots,z_k^{l_\bullet}\}^{\ell_\bullet}_{z_0^{l_\bullet}}[\mathsf{dest}\ \mathcal{C}^{l_{\mathcal{C}}}]^{l_\bullet}\rangle.\,0\\
R_4^k &= (;\ z_1).\cdots(;\ z_k).\,\langle z_1^{l_\bullet},\cdots,z_k^{l_\bullet}\rangle.\,0\\
R_5^k &= \langle n_\bullet{}^{l_\bullet}\rangle.\,0 \mid \langle n_1^{l_\bullet}\rangle.\,0 \mid \cdots \mid \langle n_m^{l_\bullet}\rangle.\,0 \mid (;\ z^{l_\bullet}).\,0
\end{aligned}$$

The reason for adding only two labels is that the knowledge of the attacker then does not need to be duplicated. This also gives the primary challenge in arguing that the analysis of R_{hard} is equivalent to the analysis of Q_{hard}. That this indeed the case is stated in Section 9 and the proofs uses the following two lemmata. The first lemma states that the insertion of only one label, $l_\bullet$, suffices:

Lemma 5 *Let R'_{hard} be as R_{hard} except that all labels $l_\bullet$ are replaced by unique labels. The the analysis of R'_{hard} is then logically equivalent to the analysis of R_{hard}:*

$$\rho,\kappa,\psi,\theta^v \models^v R'_{hard}\psi \qquad \textit{if and only if} \qquad \rho,\kappa,\psi,\theta^v \models^v R_{hard}.$$

Proof The proof in Appendix A.4 proceeds by applying the analysis $\models^v$ in Table 5 to R_{hard} and R'_{hard} and showing that the resulting formulae are equivalent. □

The second point is that the analysis does not change because R_{hard} only uses one label for all sets of crypto-points.

Lemma 6 *Let R''_{hard} be as R_{hard} except that all labels $l_{\mathcal{C}}$ are replaced with unique labels. The formula for the analysis of R''_{hard} is the logically equivalent to the analysis of R_{hard}:*

$$\rho^f,\kappa^p,\psi,\gamma^p,\sigma,\delta \models^p R''_{hard} \qquad \textit{if and only if} \qquad \rho^f,\kappa^p,\psi,\gamma^p,\sigma,\delta \models^p R_{hard}$$

Proof By inspecting the analysis $\models^p$ in Table 9 is becomes clear that the analyses of the two processes have syntactically identical requirements except for formula including δ. In these formula, the analysis of R_{hard} referrers to $\delta(l_{\mathcal{C}})$ exactly at places where the analysis of R''_{hard} referrers to either $\delta(l_i^e)$ or $\delta(l_i^d)$. The formula in the analysis of R_{hard} requires $\delta(l_{\mathcal{C}}) = \mathcal{C}$. Similarly, the analysis of R''_{hard} requires that $\delta(l_i^e) = \delta(l_i^d) = \mathcal{C}$ for

all $i \in \mathcal{A}_{\mathsf{Enc}}$. Thus, the two formula are syntactical identical except for renaming $\delta(l_i^e)$ and $\delta(l_i^d)$ to $\delta(l_{\mathcal{C}})$ and all these sets are required to contain exactly the same elements. Clearly, the two formulae are then logically equivalent. □

8. Implementation

The analysis is implemented in SML/NJ [1] as an ML function, `analyse`, that computes an ASCII version of an ALFP formula on the basis of Table 11 and the attacker R_{hard} in Section 7.

The function `analyse` takes as input an ML data structure which represents a labelled LySA process P of type $(\mathcal{N}_{\mathsf{f}}, \mathcal{A}_{\kappa}, \mathcal{A}_{\mathsf{Enc}})$. In return it gives an ALFP formula that is the conjunction of

1. The result of $\mathcal{G}(P)$.
2. The result of $\mathcal{G}(R_{hard})$ on a version of R_{hard} where $\mathcal{A}_{\kappa}$ and $\mathcal{A}_{\mathsf{Enc}}$ is taken from the type of P.
3. The formula *NEI* taking $\mathcal{A}_{\mathsf{Enc}}$ to be the set given by the type of P. The auxiliary set $\mathcal{N}_{\mathsf{c}}$ used in the specification of *NEI* in Table 11 is taken to be all the names in P together with the name $n_{\bullet}$.

The resulting ASCII version of the ALFP formula can directly can be given to the Succinct Solver that in return computes the smallest analysis components that satisfy the formula i.e. the analysis of P together with an arbitrary attacker.

The Succinct Solver requires that negation is stratified and this is enforced in a strict syntactical way: all negated queries on predicates in the i^{th} stratum must be placed syntactically *after* any clauses which may add elements into relations in strata less than i.

The analysis in Table 11 uses negation to test whether elements are missing from δ in the clause for decryption. To adhere to the stratification requirements these tests have to be placed at the end of all the clauses that may add elements to δ. This is done in two steps: first, add all elements in the tests into an auxiliary relation *PQD* (Pending Queries to δ). This is done by replacing a clause such as

$$(\neg\delta(l'_s, \ell) \vee \neg\delta(l_s, \ell')) \Rightarrow \psi(\ell', \ell)$$

with the clause

$$PQD(l'_s, \ell, l_s, \ell')$$

Second, at the *end of all clauses* the following clause is added

$$\forall l'_s, \ell, l_s, \ell' : PQD(l'_s, \ell, l_s, \ell') \Rightarrow \\ ((\neg\delta(l'_s, \ell) \vee \neg\delta(l_s, \ell')) \Rightarrow \psi(\ell', \ell))$$

With these modification ML implementation closely follows the analysis given in Table 11.

9. Conclusion

The overall soundness of the implementation can now be summarised in the following theorem:

Theorem 1 (Soundness of the analysis) *The generation function in Table 11 is* sound *with respect to original analysis in Table 3. That is,*

(1) If $(\rho^g, \gamma^g, \sigma, \delta^g)$ *satisfies* $\mathcal{G}(E)$ *and the formula for NEI then* $\rho, \kappa \models [\![E]\!] : \theta$

(2) If $(\rho^g, \kappa^p, \psi, \gamma^g, \sigma, \delta^g)$ *satisfies* $\mathcal{G}(P)$ *as well as the formula for NEI then* $\rho, \kappa \models [\![P]\!] : \psi$.

where ρ, κ *and* θ *are found by composition of the transformations in Lemma 1 to 4.*

Proof The theorem follows from Lemma 1 to 4. □

Additionally it is conjectured that the converse property of soundness, called completeness, also holds. This seems reasonable, since the analysis of the Spi-calculus in [7] follows a similar implementation strategy and it is indeed complete.

Completeness requires that all labels on terms and sets of crypto-points are *unique*. For this purpose, the function $\lceil E \rceil$ is introduced to denote a labelled LySA term that is the result of adding unique labels to the LySA term E. This function is extended homomorphically to processes.

Conjecture 1 (Completeness of the analysis) *The generation function in Table 11 is* complete *with respect to original analysis in Table 3. That is,*

(1) If $\rho, \kappa \models E : \theta$ *then there exists a* $(\rho^g, \gamma^g, \sigma, \delta^g)$ *that satisfies* $\mathcal{G}(\lceil E \rceil)$ *and the formula for NEI.*

(2) If $\rho, \kappa \models P : \psi$ *then there exists a* $(\rho^g, \kappa^p, \psi, \gamma^g, \sigma, \delta^g)$ *that satisfies* $\mathcal{G}(\lceil P \rceil)$ *and the formula for NEI.*

Furthermore, the analysis $\models, \models^v, \models^f, \models^p$, *and* $\mathcal{G}$ *are all complete with respect to the previous analysis.*

Similar to the correctness of the analysis, the correctness of the implementation of the attacker can also be split into a soundness and a completeness result:

Theorem 2 (Soundness of the attacker) $\mathcal{G}(R_{hard})$ *implies* $\mathcal{F}^{DY}_{RM}$.

Proof A consequence of the soundness Lemma 4 is that $\mathcal{G}(R_{hard})$ implies that there is an analysis $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \delta \models^p R_{hard}$. By Lemma 6 this analysis is equivalent to an analysis $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \delta \models^p R'_{hard}$ where the labels on the crypto-points are unique. Then by the soundness Lemmas 3, 2, and 1 there is also an analysis $\rho, \kappa \models [\![R_{hard}]\!] : \psi$ which by Theorem 4 in [3] is equivalent to $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$. □

Conjecture 2 (Completness of the attacker) $\mathcal{F}^{DY}_{RM}$ *implies* $\mathcal{G}(R_{hard})$.

Proof By Theorem 4 in [3] then $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ implies $\rho, \kappa \models Q_{hard} : \psi$ which by Conjecture 1 implies $\rho, \kappa, \psi, \theta^v \models^v [\![Q_{hard}]\!]$. By Lemma 5 then also $\rho, \kappa, \psi, \theta^v \models^v R_{hard}$ noting that the labels on sets of crypto-points are ignored in this analysis. By Conjecture 1 and Lemma 6 then also $\mathcal{G}(R_{hard})$. □

This concludes the technical description of the implementation of the analysis of LySA. Apart from providing such an implementation, this report also serve a witness that the implementation strategy described for the Spi-calculus in [7] carries over to other related calculi such as LySA. The main difference between LySA and the Spi-calculus is that LySA introduces pattern matching. In the implementation this has lead to choosing different domains for ρ and κ than in [7] when implementing the tree grammar encoding. Furthermore, this report contains a more detailed explanation of the tree grammar encoding than [7] and it describes an implementation of a Dolev-Yao style attacker.

References

[1] Standard ML of New Jersey. http://www.smlnj.org.

[2] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols – The Spi calculus. *Information and Computation*, 148(1):1–70, 1999.

[3] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Automatic validation of protocol narration. In *Proceedings of the 16th Computer Security Foundations Workshop (CSFW 2003)*, pages 126–140. IEEE Computer Society Press, 2003.

[4] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Efficient mechanical analysis of protocol narration. Manuscript.

[5] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. http://www.grappa.univ-lille3.fr/tata/, 27th September 2002.

[6] F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in Mobile Ambients. In *CONCUR 1999 – Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 463–477. Springer Verlag, 1999.

[7] F. Nielson, H. Riis Nielson, and H. Seidl. Cryptographic analysis in cubic time. *Electronic Notes in Theoretical Computer Science*, 62, 2002.

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

[9] 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 *Lecture Notes in Computer Science*, pages 223–244. Springer Verlag, 2002.

A. Proofs

A.1. Proof of Lemma 1

The proof proceeds by structural induction in terms and processes. The *induction hypothesis* is that Lemma 1 holds for all subterm or subprocesses of the expression considered.

Case (1)

(Names) Assume $\rho, \theta^v \models^v n^l$. Then $\lfloor n \rfloor \in \theta^v(l)$ by the definition of $\models^v$ in Table 5. Using the definition of $\models$ in Table 3 allows to conclude that $\rho \models n : \theta^v(l)$ i.e. that $\rho \models [\![n^l]\!] : \theta^v(l)$ as required.

(Variables) Assume $\rho, \theta^v \models^v x^l$. Then $\rho(\lfloor x \rfloor) \subseteq \theta^v(l)$ by the definition of $\models^v$. The definition of $\models$ can now be used to conclude that $\rho \models [\![x]\!] : \theta(l)$ as required.

(Encryption) Let $E \stackrel{\text{def}}{=} \{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}]$ and assume that $\rho, \theta^v \models^v E^l$. Using the definition of $\models^v$ then $\wedge_{i=0}^k\ \rho, \theta^v \models^v E_i^{l_i}$ so by the induction hypothesis also $\wedge_{i=0}^k \rho \models [\![E_i^{l_i}]\!] : \theta^v(l_i)$. Using again the definition of $\models^v$ it also holds

$$\forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^k V_i \in \theta^v(l_i) \Rightarrow \{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \in \theta^v(l)$$

which is precisely what is needed for to conclude that $\rho \models [\![E^l]\!] : \theta(l)$ as required using the definition of $\models$. This concludes the proof of Lemma 1 case (1). Note that case (2) has not been used to prove this part of the lemma.

Case (2)

(Output) Assume $\rho, \kappa, \psi, \theta^v \models^v \langle E_1^{l_1}, \cdots, E_k^{l_k} \rangle . P$. Then from definition of $\models^v$ the three conjuncts in the hypothesis of the rule for output hold. From the conjunct $\wedge_{i=1}^k\ \rho, \theta^v \models^v E_i^{l_i}$ and Lemma 1 case (1) then $\wedge_{i=1}^k\ \rho \models [\![E_i]\!] : \theta^v(l_i)$ and from $\rho, \kappa, \psi, \theta^v \models^v P$ and the induction hypothesis then $\rho, \kappa \models [\![P]\!] : \psi$. The final conjunct is the implication

$$\forall V_1, \cdots, V_k : \wedge_{i=1}^k V_i \in \theta^v(l_i) \Rightarrow \langle V_1, \cdots, V_k \rangle \in \kappa$$

which is precisely the last conjunct needed to conclude $\rho, \kappa \models \langle [\![E_1]\!], \cdots, [\![E_k]\!] \rangle . [\![P]\!] : \psi$ by using the definition of $\models$. Since there are no extra labels in this process then also $\rho, \kappa \models [\![\langle E_1, \cdots, E_k \rangle . P]\!] : \psi$ which is the required result.

(Input and decryption) are analogue to output.

(Restriction) Assume $\rho, \kappa, \psi, \theta^v \models^v (\nu\, n)P$. Then $\rho, \kappa, \psi, \theta^v \models^v P$ using the definition of $\models^v$ so by the induction hypothesis $\rho, \kappa \models [\![P]\!] : \psi$. Using the definition of $\models$, one may conclude that $\rho, \kappa \models (\nu\, n)[\![P]\!] : \psi$ and since there are labels on n then also $\rho, \kappa \models [\![(\nu\, n)P]\!] : \psi$ as required.

(Replication and Parallel Composition) are similar to restriction while the case for the *(Terminated Process)* trivially holds. This concludes the proof of Lemma 1. □

A.2. Proof of Lemma 2

The proof proceeds by structural induction on terms and processes over judgements of $\models^f$. The *induction hypothesis* is then that Lemma 2 holds for all terms and processes that are immediate constituents of the term or process considered.

Case (1)

(Names) Assume $\rho^f, \gamma \models^f n^l$. Then $\lfloor n \rfloor \in \gamma(l)$ by the definition of $\models^f$ in Table 7. Then define $\theta^v(l) \stackrel{\text{def}}{=} L(\gamma, l) \supseteq \{f() \mid f() \in \gamma(l)\} \supseteq \{\lfloor n \rfloor\}$. In other words, $\lfloor n \rfloor \in \theta^v(l)$ so by the definition of $\models^v$ in Table 5 then $\rho, \theta^v \models^v n^l$ for any choice of ρ and in particular for the one in Lemma 2.

(Variables) Assume $\rho^f, \gamma \models^f x^l$. Then $\forall l' \in \rho^f(\lfloor x \rfloor) : \gamma(l') \subseteq \gamma(l)$ by the definition of $\models^f$ which is the same as $(\bigcup_{l' \in \rho^f(\lfloor x \rfloor)} \gamma(l')) \subseteq \gamma(l)$. This states that the bodies of rules in the grammar that have l as a head is going to be a superset of the bodies in all the rules with l' as a head. Consequently, the tree languages generated grammars are also going to be supersets, i.e. $(\bigcup_{l' \in \rho^f(\lfloor x \rfloor)} L(\gamma, l')) \subseteq L(\gamma, l)$. With the definitions of ρ and θ^v from Lemma 2 it then follows that $\rho(\lfloor x \rfloor) \subseteq \theta^v(l)$ and consequently that $\rho, \theta^v \models^v x^l$ by the definition of $\models^v$.

(Encryption) Let $E = \{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}]$ and assume $\rho^f, \gamma \models^f E^l$. Then by the definition of $\models^f$ it holds that (a) $\wedge_{i=0}^k \rho^f, \gamma \models^f E_i^{l_i}$ and (b) $\{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] \in \gamma(l)$.

From (a) and the induction hypothesis it is known that $\wedge_{i=0}^k \rho, \theta^v \models^v E_i^{l_i}$ which is the first requirement for $\models^v$ to be an analysis of E^l.

Using (b) and the definition of $L(\gamma, l)$ it holds that

$$\{\ \{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \mid V_0 \in L(\gamma, l_0) \wedge \cdots \wedge V_k\} \in L(\gamma, l_k) \subseteq L(\gamma, l)$$

With the definition of θ^v in Lemma 2 this is the same as

$$\{\ \{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \mid V_0 \in \theta^v(l_0) \wedge \cdots \wedge V_k \in \theta^v(l_k)\} \subseteq \theta^v(l)$$

or equivalently

$$\forall V_0 \in \theta^v(l_0), \cdots, V_k \in \theta^v(l_k) : \{V_1, \cdots, V_k\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \in \theta^v(l)$$

This is precisely the second requirement in $\models^v$ and conclusively it holds that $\rho, \theta^v \models^v E^l$. This concludes the proof of case (1).

Case (2)

(Output) Assume $\rho^f, \kappa^f, \psi, \gamma \models^f \langle E_1^{l_1}, \cdots, E_k^{l_k}\rangle.P$. Then by the definition of $\models^f$ in Table 7 (a) $\wedge_{i=1}^k \rho^f, \gamma \models^f E_i^{l_i}$, (b) $\langle l_1, \cdots, l_k\rangle \in \kappa^f$, and (c) $\rho^f, \kappa^f, \psi, \gamma \models^f P$. Using Lemma 2 case (1) and the induction hypothesis there are analysis of $\models^v$ corresponding to (a) and (c), respectively. From (b) and the definition of κ in Lemma 2 then

$$\{\langle V_1, \cdots, V_k\rangle \mid V_1 \in \theta^v(l_1) \wedge \cdots \wedge V_k \in \theta^v(l_k)\} \subseteq \kappa$$

or equivalently

$$\forall V_1 \in \theta^v(l_1), \cdots, V_k \in \theta^v(l_k) : \langle V_1, \cdots, V_k\rangle \in \kappa$$

This is precisely the last requirement that is needed to use the definition of $\models^v$ in Table 5 to conclude that $\rho, \kappa, \psi, \theta^v \models^v \langle E_1^{l_1}, \cdots, E_k^{l_k}\rangle.P$.

(Input) Assume $\rho^f, \kappa^f, \psi, \gamma \models^f (E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k).P$. Then by the definition of $\models^f$ it holds that (a) $\wedge_{i=1}^j \rho^f, \gamma \models^f E_i^{l_i}$,

$$\text{(b)} \qquad \forall \langle l'_1, \cdots, l'_k\rangle \in \kappa^f : (\wedge_{i=1}^j L(\gamma, l'_i) \circledR L(\gamma, l_i) \neq \emptyset) \Rightarrow (c)$$

where

$$\text{(c)} \qquad \wedge_{i=j+1}^k\ l'_i \in \rho^f(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi, \gamma \models^f P$$

The formula (b) may be rephrased (using the definition of $\circledR$ and θ^v) to

$$\forall \langle l'_1, \cdots, l'_k\rangle \in \kappa^f : (\wedge_{i=1}^j \exists V_i : V_i \in \theta^v(l'_i) \wedge V_i \in \theta^v(l_i)) \Rightarrow (c)$$

or equivalently to

$$\forall \langle l'_1, \cdots, l'_k\rangle \in \kappa^f : \forall V_1 \in \theta^v(l'_1), \cdots, V_j \in \theta^v(l'_j) : (\wedge_{i=1}^j V_i \in \theta^v(l_i)) \Rightarrow (c)$$

Next, first observe that if $V \in \theta^v(l)$ and $l \in \rho(\lfloor x \rfloor)$ then $V \in \cup_{l' \in \rho^f(\lfloor x \rfloor)} \theta^v(l')$; that is $V \in \rho(\lfloor x \rfloor)$ where ρ is as defined in Lemma 2. The formula above may then be rewritten to

$$\begin{aligned}\forall \langle l'_1, \cdots, l'_k\rangle \in \kappa^f : \forall V_1 \in \theta^v(l'_1), \cdots, V_j \in \theta^v(l'_j), V_{j+1} \in \theta^v(l'_{j+1}), \cdots, V_k \in \theta^v(l'_k) :\\ (\wedge_{i=1}^j V_i \in \theta^v(l_i)) \Rightarrow \wedge_{i=j+1}^k V'_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi, \gamma \models^f P\end{aligned}$$

taking advantage of $V_{j+1}, \cdots, V_k$ being free in the hypothesis of the implication. Finally, by using the definition of κ in Lemma 2 and the induction hypothesis it is clear that

$$\forall \langle V'_1, \cdots, V'_k\rangle \in \kappa : (\wedge_{i=1}^j V_i \in \theta^v(l_i)) \Rightarrow \wedge_{i=j+1}^k V'_i \in \rho(\lfloor x_i \rfloor) \wedge \rho, \kappa, \psi, \theta \models^v P$$

Using the definition of $\models^v$ on this formula together with Lemma 2 case (1) for (a) leads to the desired conclusion that $\rho, \kappa, \psi, \theta^v \models^v (E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k).P$.

(Decryption, restriction, replication, parallel composition, terminated process) The interesting parts of decryption are analogue to input while the remaining cases are straightforward. This concludes the proof of Lemma 2. □

A.3. Proof of Lemma 3

The proof proceeds by structural induction over judgements of $\models^p$.

Case (1)

(Names) Assume $\rho^f, \gamma^p, \sigma, \delta \models^p n^l$. Then $\lfloor n \rfloor \in \gamma^p(l)$ and with γ defined as in Lemma 3 also $\lfloor n \rfloor \in \gamma(l)$, which can be used to conclude $\rho^f, \gamma \models^f n^l$.

(Variables) Assume $\rho^f, \gamma^p, \sigma, \delta \models^p x^l$. Then $\forall l' \in \rho^f(\lfloor x \rfloor) : \gamma^p(l') \subseteq \gamma^p(l)$. First, note that with the definition γ in Lemma 3 it clearly holds that $\gamma^p(l_1) \subseteq \gamma^p(l_2)$ implies that $\gamma(l_1) \subseteq \gamma(l_2)$ for any l_1 and l_2. In particular it holds for the choices of l' and l above so $\forall l' \in \rho^f(\lfloor x \rfloor) : \gamma(l') \subseteq \gamma(l)$, which is what is needed to conclude that $\rho^f, \gamma \models^f x^l$ by the definition of $\models^p$.

(Encryption) Assume $\rho^f, \gamma^p, \sigma, \delta \models^p \{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}]^l$. From the definition of $\models^p$ then (a) $\wedge_{i=0}^k \rho^f, \gamma^p, \sigma, \delta \models^p E_i^{l_i}$, which by the induction hypothesis can be used to conclude $\wedge_{i=0}^k \rho^f, \gamma \models^f E_i^{l_i}$. Additionally, (b) $(l, l_0, l_1, \cdots, l_k, \ell, l_s) \in \sigma_k$, (c) $l \in \gamma^p(l)$, and (d) $\wedge_{\ell' \in \mathcal{L}}\ \ell' \in \delta(l_s)$. Note that since l_s is unique then $\delta(l_s)$ is *precisely* the set $\mathcal{L}$. Together with the definition of γ in Lemma 3 (b), (c), and (d) can be used to conclude that

$$\{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] \in \gamma(l)$$

This, together with (a) may be used to conclude that $\rho^f, \gamma \models^f \{E_1^{l_1}, \cdots, E_k^{l_k}\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}]^l$.

Case (2)

(Output, Input) Simply note that if some m is in κ_k then also $m \in \kappa_k \cup X$ for some arbitrary set X. In particular, if $m \in \kappa_k$ then also $m \in \kappa$ with the definition of κ in Lemma 3. This, together with straightforward application of Lemma 3 case (1) and the induction hypothesis is enough to conclude the proof for output and input.

(Decryption) Assume that $\rho^f, \kappa^p, \psi, \gamma^p, \sigma, \delta \models^p P$ for

$$P \stackrel{\text{def}}{=} \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{E_1^{l_1}, \cdots, E_j^{l_j};\ x_{j+1}, \cdots, x_k\}^{\ell}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P'$$

Then the hypothesis in the rule for decryption also hold in the definition $\models^p$ in Table 9 so it e.g. holds that $\wedge_{\ell' \in \mathcal{L}}\ \ell' \in \delta(l_s)$ and since l_s is unique then $\delta(l_s) = \mathcal{L}$. The precondition of first implication in the hypothesis of the rule for decryption holds precisely when $\forall\ (l', l'_0, \cdots, l'_k, \ell', l'_s) \in \sigma_k : l' \in \gamma^p(l)$ holds. Using the definition of γ in Lemma 3 then it also holds that

$$\{l'_1, \cdots, l'_k\}^{\ell'}_{l'_0}[\mathsf{dest}\ \delta(l'_s)] \in \gamma(l)$$

For the choices of l'_i where the precondition of the second implication also holds, i.e. where $L(\gamma, l'_i) \barwedge L(\gamma, l_i) \neq \emptyset$ holds, then the conclusion of the implications hold. This is

exactly as the corresponding conclusion in the rule for decryption in the analysis $\models^f$ in Table 7 noting that $\mathcal{L} = \delta(l_s)$ and that $\mathcal{L}' = \delta(l'_s)$. Finally, the induction hypothesis and Lemma 3 case (1) are needed to conclude the proof.

(Restriction, replication, parallel composition, and replication) are straightforward and this concludes the proof of Lemma 3. □

A.4. Proof of Lemma 5

Proof The proof proceeds by applying the analysis $\models^v$ in Table 5 to both R_{hard} and R'_{hard} and showing that the resulting formulae are logically equivalent.

The processes R_{hard} and R'_{hard} both consist of replication and parallel composition of subprocesses. The analyses $\models^v$, thus, result in formulae that are conjunction of the analyses of the respective subprocesses. The analyses of the two processes are, thus, equivalent if the conjunction of the analysis of each of the subprocesses are equivalent:

(1) Consider the respective analysis of $R'^k_1 = R^k_1 = (;\ z_1, \cdots, z_k).\,0$. Since the processes are identical their analyses are clearly logically equivalent.

(2) Both R'^k_2 and R^k_2 start by receiving a 1-ary message and binding it to z. The analysis $\models^v$ of this input gives that

$$\forall \langle V_1 \rangle \in \kappa : V_1 \in \rho(z_\bullet)$$

and that the remaining process should be analysed. That is, all 1-ary messages in κ will be in $\rho(z_\bullet)$. Note also, that since all the variables in R'_{hard} (and in R_{hard}) have the same canonical variable the analysis will use $\rho(z_\bullet)$ in the analysis of *all subprocesses*, thus, sharing this information between them. The second input in R'^k_2 and R^k_2 also have identical analyses.

The decryption in R'^k_2 gives the formula

$$\begin{array}{l} \rho(z_\bullet) \subseteq \theta(l_1) \wedge \rho(z_\bullet) \subseteq \theta(l_2) \wedge \\ \forall \{V_1, \cdots, V_k\}^{\ell'}_{V_0}[\mathsf{dest}\ \mathcal{L}'] \in \theta(l_1) : V_0 \in \theta(l_2) \Rightarrow \ \wedge^k_{i=1} V_i \in \rho(z_\bullet) \wedge \\ \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ell_\bullet \notin \mathcal{L}' \vee \ell \notin \mathcal{C}) \Rightarrow (\ell', \ell_\bullet) \in \psi \end{array}$$

where l_1 and l_2 are unique labels that will only be used in this formula. The analysis of R_{hard} gives an identical formula except that $l_\bullet$ is used in the place for l_1 and l_2. Note that $\theta(l_1)$ and $\theta(l_2)$ contain the same sets and may therefore be replaced by $\theta(l_\bullet)$ provided that no additional requirements are put on $\theta(l_\bullet)$. That this is the case is argued throughout the rest of the proof.

(3) Both R'^k_3 and R^k_3 start with k identical inputs that, of course, have identical analysis.

The analysis of $R_3'^k$ gives the formula

$$\begin{array}{l}
\wedge_{i=0}^{k}\ \forall\langle V\rangle \in \kappa : V \in \rho(z_\bullet) \wedge \\
\wedge_{i=0}^{k}\ \rho(z_\bullet) \subseteq \theta(l_{3i}) \wedge \\
\forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k}\ V_i \in \theta(l_{3i}) \Rightarrow \{V_0, \cdots, V_k\}_{V_0}^{\ell_\bullet}[\mathsf{dest}\ \mathcal{C}] \in \theta(l_4) \wedge \\
\forall V : V \in \theta(l_4) \Rightarrow \langle V\rangle \in \kappa
\end{array}$$

where $l_{30}, l_{31}, \cdots, l_{3k}$, and l_4 are unique labels not used in any other formula. Notice that every element in $\theta(l_4)$ is also required to be in $\rho(z_\bullet)$ since they must be 1-ary messages of κ. The formula for the analysis of R_3^k is identical to the formula above except that all the labels have been replaced by $l_\bullet$:

$$\begin{array}{l}
\wedge_{i=0}^{k}\ \forall\langle V\rangle \in \kappa : V \in \rho(z_\bullet) \wedge \\
\wedge_{i=0}^{k}\ \rho(z_\bullet) \subseteq \theta(l_\bullet) \wedge \\
\forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k}\ V_i \in \theta(l_\bullet) \Rightarrow \{V_0, \cdots, V_k\}_{V_0}^{\ell_\bullet}[\mathsf{dest}\ \mathcal{C}] \in \theta(l_\bullet) \wedge \\
\forall V : V \in \theta(l_\bullet) \Rightarrow \langle V\rangle \in \kappa
\end{array}$$

The first part of the formula requires that 1-ary messages in κ must be in $\rho(z_\bullet)$ and also be in $\theta(l_\bullet)$. These requirements are similar to the requirements from the analysis of R_2^k. The second part additionally requires that $\{V_0, \cdots, V_k\}_{V_0}^{\ell_\bullet}[\mathsf{dest}\ \mathcal{C}]$ is in $\theta(l_\bullet)$ but since this value is also required to be a 1-ary message in κ the formula is equivalent to the analysis of $R_3'^k$.

(4) The analysis of $R_4'^k$ is

$$\begin{array}{l}
\wedge_{i=1}^{k}\ \forall\langle V\rangle \in \kappa : V \in \rho(z_\bullet) \wedge \\
\wedge_{i=1}^{k}\ \rho(z_\bullet) \subseteq \theta(l_{5i}) \wedge \\
\forall V_1, \cdots, V_k : V_i \in \theta(l_{5i}) \Rightarrow \langle V_1, \cdots, V_k\rangle \in \kappa
\end{array}$$

where $l_{51}, \cdots l_{5k}$ are unique labels not used in any other formula. The analysis of R_4^k is identical except that all labels have been replaced with $l_\bullet$. Since all $\theta(l_{5i})$ contain the exact same elements as $\theta(l_\bullet)$ and there are no new requirements on $\theta(l_\bullet)$ then the analysis of R_4^k is equivalent with the analysis of $R_4'^k$.

(5) The analysis of $R_5'^k$ gives

$$\begin{array}{l}
(n_\bullet \in \theta(l_6) \wedge \forall V \in \theta(l_6) : \langle V\rangle \in \kappa) \wedge \\
(\wedge_{i=1}^{m}\ \lfloor n_i \rfloor \in \theta(l_{7i}) \wedge \forall V \in \theta(l_{7i}) : \langle V\rangle \in \kappa) \wedge \\
\forall\langle V\rangle \in \kappa : V \in \rho(z_\bullet))
\end{array}$$

where $l_6, l_{71}, \cdots, l_{7m}$, and l_8 are unique labels. Changing the labels in the formula to $l_\bullet$ gives an equivalent formula (in conjunction with the formula for the analysis of the other subprocesses) since all elements in $\theta(l_i)$ are also 1-ary messages in κ and they will therefore be in $\rho(z_\bullet)$ and eventually $\theta(l_\bullet)$ anyway.

This concludes the proof of Lemma 5. □

B. Asymmetric Cryptography

In [4], LySA is extended with *asymmetric cryptography*. This appendix covers the extensions made to the implementation to incorporate this new feature.

B.1. The New Stating Point

The syntax for LySA terms, E, is extended with public and private keys as well as asymmetric key encryption and becomes:

$$E \ ::= \ \ldots \quad | \quad m^+ \quad | \quad m^- \quad | \quad \{\lfloor E_1, \cdots, E_k \rfloor\}^{\ell}_{E_0}[\mathsf{dest}\ \mathcal{L}]$$

The syntax for processes, P, is extended with a restriction operator that restricts a key pair, i.e. the two names m^+ and m^-, along with a construct for asymmetric key decryption:

$$P \ ::= \ \ldots \quad | \quad (\nu_{\pm}\ m)P \quad | \quad \mathsf{decrypt}\ E\ \mathsf{as}\ \{\lfloor E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k \rfloor\}^{\ell}_{E_0}\ [\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P$$

The analysis uses the same analysis components as before, i.e. the ones in Table 2, though the set of values $\mathcal{V}$ is extended to contain asymmetric encryptions. The extensions to the specification of the analysis are shown in Table 12.

$$\frac{\lfloor m^+ \rfloor \in \theta}{\rho \ \models m^+ : \theta} \qquad \frac{\lfloor m^- \rfloor \in \theta}{\rho \ \models m^- : \theta}$$

$$\frac{\begin{array}{c}\wedge_{i=0}^{k}\, \rho \ \models E_i : \theta_i \ \wedge \\ \forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \theta_i \ \Rightarrow \ \{\lfloor V_1, \cdots, V_k \rfloor\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \in \theta\end{array}}{\rho \ \models \{\lfloor E_1, \cdots, E_k \rfloor\}^{\ell}_{E_0}[\mathsf{dest}\ \mathcal{L}] : \theta}$$

$$\frac{\rho, \kappa \ \models P : \psi}{\rho, \kappa \ \models (\nu_{\pm}\ m)P : \psi}$$

$$\frac{\begin{array}{r}\rho \ \models E : \theta \ \ \wedge \ \ \wedge_{i=0}^{j}\, \rho \ \models E_i : \theta_i \ \wedge \\ \forall\, \{\lfloor V_1, \cdots, V_k \rfloor\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \in \theta \ : \forall V_0' \in \theta_0 : \forall (m^+, m^-) : \\ \{V_0, V_0'\} = \{m^+, m^-\} \wedge \wedge_{i=1}^{j} V_i \in \theta_i \ \Rightarrow \ \wedge_{i=j+1}^{k}\, V_i \in \rho(\lfloor x_i \rfloor) \ \wedge \\ (\neg(\ell \in \mathcal{L}' \wedge \ell' \in \mathcal{L}) \Rightarrow (\ell, \ell') \in \psi) \ \wedge \\ \rho, \kappa \ \models P : \psi\end{array}}{\rho, \kappa \ \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{\lfloor E_1, \cdots, E_j;\ x_{j+1}, \cdots, x_k \rfloor\}^{\ell'}_{E_0}\ [\mathsf{orig}\ \mathcal{L}']\ \mathsf{in}\ P : \psi}$$

Table 12: Extensions to the original analysis in Table 3.

B.2. From Succinct to Verbose

Applied occurrences of terms and sets of crypto-points in annotation are all labelled, yielding the following syntax:

$$
\begin{array}{lcl}
E & ::= & \ldots \quad | \quad m^{+l} \quad | \quad m^{-l} \quad | \quad \{\!|E_1, \cdots, E_k|\!\}^{\ell}_{E_0}[\mathsf{dest}\ \mathcal{L}^{l_s}]^{l} \\
P & ::= & \ldots \quad | \quad (\nu_{\pm}\ m)P \quad | \\
 & & \mathsf{decrypt}\ E^{l}\ \mathsf{as}\ \{\!|E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}, \cdots, x_k|\!\}^{\ell}_{E_0^{l_j}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P
\end{array}
$$

The analysis of Table 12 is rewritten into succinct form using the labels at terms to refer to the global term cache θ^v as shown in Table 13. The analysis components used by this analysis are the same as before i.e. the ones in Table 4.

$$
\frac{\lfloor m^{+} \rfloor \in \theta(l)}{\rho, \theta \models^{v} m^{+l}} \qquad \frac{\lfloor m^{-} \rfloor \in \theta(l)}{\rho, \theta \models^{v} m^{-l}}
$$

$$
\frac{\begin{array}{c} \wedge_{i=0}^{k}\, \rho, \theta \models^{v} E_i^{l_i} \wedge \\ \forall V_0, V_1, \cdots, V_k : \wedge_{i=0}^{k} V_i \in \theta_i \ \Rightarrow\ \{\!|V_1, \cdots, V_k|\!\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \in \theta(l) \end{array}}{\rho, \theta \models^{v} \{\!|E_1^{l_1}, \cdots, E_k^{l_k}|\!\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}]^{l}}
$$

$$
\frac{\rho, \kappa, \psi, \theta \models^{v} P}{\rho, \kappa, \psi, \theta \models^{v} (\nu_{\pm}\ m)P}
$$

$$
\frac{\begin{array}{l} \rho, \theta \models^{v} E^{l} \quad \wedge \quad \wedge_{i=0}^{j}\, \rho, \theta \models^{v} E_i^{l_i} \wedge \\ \forall\, \{\!|V_1, \cdots, V_k|\!\}^{\ell}_{V_0}[\mathsf{dest}\ \mathcal{L}] \in \theta(l) : \forall V_0' \in \theta(l_0) : \forall (m^{+}, m^{-}) : \\ \qquad \{V_0, V_0'\} = \{m^{+}, m^{-}\} \wedge \wedge_{i=1}^{j} V_i \in \theta(l_i) \ \Rightarrow\ \wedge_{i=j+1}^{k} V_i \in \rho(\lfloor x_i \rfloor) \wedge \\ \qquad\qquad ((\ell \notin \mathcal{L}' \vee \ell' \notin \mathcal{L}) \Rightarrow (\ell, \ell') \in \psi) \wedge \\ \qquad\qquad \rho, \kappa, \psi, \theta \models^{v} P \end{array}}{\rho, \kappa, \psi, \theta \models^{v} \mathsf{decrypt}\ E^{l}\ \mathsf{as}\ \{\!|E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}, \cdots, x_k|\!\}^{\ell'}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}']\ \mathsf{in}\ P}
$$

Table 13: Extensions to the verbose analysis in Table 5. Note that θ is referred to as θ^v elsewhere in this report.

B.3. From Infinite to Finite

The tree grammar encoding works as before though the signature for terms has to be extended to include asymmetric encryptions. That is, Σ_{LySA} becomes

$$\begin{array}{l}\Sigma_{\text{LySA}} \stackrel{\text{def}}{=} \ldots \cup \\ \{\lfloor m^+ \rfloor_0 \mid \ m^+ \text{ is used in the processes } P \text{ that is analysed } \} \cup \\ \{\lfloor m^- \rfloor_0 \mid \ m^- \text{ is used in } P \ \} \cup \\ \{ aenc^{\ell,\mathcal{L}}_{k+1} \mid \ k \text{ is the arity of an } \textit{asymmetric} \text{ encryption annotated with } \ell \text{ and } \mathcal{L} \text{ in } P \ \}\end{array}$$

and as before $aenc^{\ell,\mathcal{L}}(A_0, A_1, \cdots, A_k)$ will typically be written $\{\!|A_1, \cdots, A_k|\!\}^{\ell}_{A_0}[\mathsf{dest}\ \mathcal{L}]$. Thus, rules in a tree grammar over Σ_{LySA} where the non-terminals are taken from the set of labels, Lab, will have the form:

$$\begin{array}{lcl} l & \rightarrow & \lfloor n \rfloor \\ l & \rightarrow & \lfloor m^+ \rfloor \\ l & \rightarrow & \lfloor m^- \rfloor \end{array} \qquad \text{or} \qquad \begin{array}{lcl} l & \rightarrow & \{l_1, \cdots, l_k\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] \\ l & \rightarrow & \{\!|l_1, \cdots, l_k|\!\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] \end{array}$$

The encoding into tree grammars follows closely the ideas presented in Section 4 and the extensions to the analysis are given in Table 14.

$$\frac{\lfloor m^+ \rfloor \in \gamma(l)}{\rho, \gamma \ \models^f m^{+l}} \qquad \frac{\lfloor m^- \rfloor \in \gamma(l)}{\rho, \gamma \ \models^f m^{-l}}$$

$$\frac{\wedge^k_{i=0}\ \rho, \gamma \ \models^f E^{l_i}_i \quad \wedge \quad \{\!|l_1, \cdots, l_k|\!\}^{\ell}_{l_0}[\mathsf{dest}\ \mathcal{L}] \in \gamma(l)}{\rho, \gamma \ \models^f \{\!|E^{l_1}_1, \cdots, E^{l_k}_k|\!\}^{\ell}_{E^{l_0}_0}[\mathsf{dest}\ \mathcal{L}]^l}$$

$$\frac{\rho, \kappa, \psi, \gamma \ \models^f P}{\rho, \kappa, \psi, \gamma \ \models^f (\nu_{\pm}\ m)P}$$

$$\frac{\begin{array}{r}\rho, \gamma \ \models^f E^l \quad \wedge \quad \wedge^j_{i=0}\ \rho, \gamma \ \models^f E^{l_i}_i \ \wedge \\ \forall\ \{\!|l'_1, \cdots, l'_k|\!\}^{\ell'}_{l'_0}[\mathsf{dest}\ \mathcal{L}'] \in \gamma(l) \ : \forall V_0 \in \gamma(l_0) : \forall V'_0 \in \gamma(l'_0) : \forall (m^+, m^-) : \\ \{V_0, V'_0\} = \{m^+, m^-\} \wedge \\ (\wedge^j_{i=1} L(\gamma, l'_i) \ \text{ⓡ} \ L(\gamma, l_i) \neq \emptyset) \quad \Rightarrow \quad \wedge^k_{i=j+1}\ l'_i \in \rho(\lfloor x_i \rfloor) \ \wedge \\ ((\ell' \notin \mathcal{L} \vee \ell \notin \mathcal{L}') \Rightarrow (\ell', \ell) \in \psi) \ \wedge \\ \rho, \kappa, \psi, \gamma \ \models^f P\end{array}}{\rho, \kappa, \psi, \gamma \ \models^f \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{\!|E^{l_1}_1, \cdots, E^{l_j}_j; x_{j+1}, \cdots, x_k|\!\}^{\ell}_{E^{l_0}_0}\ [\mathsf{orig}\ \mathcal{L}]\ \mathsf{in}\ P}$$

Table 14: Extensions to the finite analysis in Table 7. Note that ρ and κ are referred to as ρ^f and κ^f, respectively, elsewhere in this report.

B.4. Non-polyadic Analysis

The polyvariance in asymmetric encryption is remove from the analysis in the way described in Section 5. In particular, a family of new relations α_k is introduced to store asymmetric encryptions. A k-ary asymmetric encryption is stored in the relation

$$\alpha_k \in \wp(Lab^{k+2} \times \mathcal{C} \times Lab)$$

analogue to the k-ary symmetric encryptions stored in σ_k that was introduced in Table 8.

The sets of crypto-points are again kept in the analysis component δ and by assuming that all labels at sets of crypto-points are unique no confusion will occur by using the same component as before. With these modifications the non-polyadic analysis is as given in Table 15.

$$\frac{\lfloor m^+ \rfloor \in \gamma(l)}{\rho,\gamma,\sigma,\alpha,\delta \models^p m^{+l}} \qquad \frac{\lfloor m^- \rfloor \in \gamma(l)}{\rho,\gamma,\sigma,\alpha,\delta \models^p m^{-l}}$$

$$\frac{\wedge_{i=0}^{k}\ \rho,\gamma,\sigma,\alpha,\delta \models^p E_i^{l_i} \quad \wedge \quad (l,l_0,l_1,\cdots,l_k,\ell,l_s) \in \alpha_k \quad \wedge \quad l \in \gamma(l) \quad \wedge \quad \wedge_{\ell' \in \mathcal{L}}\ \ell' \in \delta(l_s)}{\rho,\gamma,\sigma,\alpha,\delta \models^p \{\!|E_1^{l_1},\cdots,E_k^{l_k}|\!\}^{\ell}_{E_0^{l_0}}[\mathsf{dest}\ \mathcal{L}^{l_s}]^{l}}$$

$$\frac{\rho,\kappa,\psi,\gamma,\sigma,\alpha,\delta \models^p P}{\rho,\kappa,\psi,\gamma,\sigma,\alpha,\delta \models^p (\nu_{\pm}\ m)P}$$

$$\frac{\begin{array}{r}\rho,\gamma,\sigma,\alpha,\delta \models^p E^l \quad \wedge\ \wedge_{\ell' \in \mathcal{L}}\ \ell' \in \delta(l_s) \quad \wedge\ \wedge_{i=0}^{j}\ \rho,\gamma,\sigma,\alpha,\delta \models^p E_i^{l_i} \wedge \\ \forall\,(l',l'_0,l'_1,\cdots,l'_k,\ell',l'_s) \in \alpha_k : l' \in \gamma(l)\ \Rightarrow \forall V_0 \in \gamma(l_0) : \forall V'_0 \in \gamma(l'_0) : \forall (m^+,m^-) : \\ \{V_0,V'_0\} = \{m^+,m^-\} \wedge \\ (\wedge_{i=1}^{j} L(\gamma,l'_i) \barwedge L(\gamma,l_i) \neq \emptyset) \quad \Rightarrow \quad \wedge_{i=j+1}^{k}\ l'_i \in \rho(\lfloor x_i \rfloor) \wedge \\ ((\ell' \notin \delta(l_s)) \vee \ell \notin \delta(l'_s) \Rightarrow (\ell',\ell) \in \psi) \wedge \\ \rho,\kappa,\psi,\gamma,\sigma,\alpha,\delta \models^p P\end{array}}{\rho,\kappa,\psi,\gamma,\sigma,\alpha,\delta \models^p \mathsf{decrypt}\ E^l\ \mathsf{as}\ \{\!|E_1^{l_1},\cdots,E_j^{l_j};\ x_{j+1},\cdots,x_k|\!\}^{\ell}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P}$$

Table 15: Extensions to the non-polyvariant analysis in Table 9. Note that ρ, κ, and γ are referred to as ρ^f, κ^p, and γ^p, respectively, elsewhere in this report.

B.5. The Generation Function

The generation function $\mathcal{G}$ is extended with clauses for the new syntactic constructs and its definition is found by transforming the hypothesis of the rules in Table 15 into ALFP. As in Section 6, all analysis components are transformed into sets of Cartesian products

and notational conventions are changed so that set membership, etc. are written as statements about predicates.

One notable challenge remains: the query on set of key pairs in the rule for asymmetric decryption. The matching in of the content of the encrypted messages is performed as usual by consulting the auxiliary relation *NEI*. The matching of the key, on the other hand, is performed by requiring that two elements from the grammars of the l_0 and l_0' are a key pair. This corresponds to "unfolding" the definition of non-empty intersection, i.e. of *NEI*, and checking that there is a key pair in the intersection. Note that the recursive part of finding the non-empty intersection is not needed since all asymmetric keys are names.

$$
\begin{array}{ll}
\mathcal{G}(m^{+^l}) & = \gamma(l, \lfloor m^+ \rfloor) \wedge KP(m^+, m^-) \wedge KP(m^-, m^+) \\
\mathcal{G}(m^{-^l}) & = \gamma(l, \lfloor m^- \rfloor) \wedge KP(m^-, m^+) \wedge KP(m^+, m^-) \\
\mathcal{G}(\{\!|E_1^{l_1}, \cdots, E_k^{l_k}|\!\}^{\ell}_{E_0^{l_0}} [\mathsf{dest}\ \mathcal{L}^{l_s}]^l) & = \wedge_{i=0}^{k} \mathcal{G}(E_i^{l_i}) \wedge \\
 & \quad \alpha_k(l, l_0, l_1, \cdots, l_k, \ell, l_s) \quad \wedge \quad \gamma(l,l) \quad \wedge \quad \wedge_{\ell' \in \mathcal{L}} \quad \delta(l_s, \ell')
\end{array}
$$

$$
\begin{array}{ll}
\mathcal{G}((\nu_{\pm}\ m)P) & = \mathcal{G}(P) \\
\mathcal{G}(\mathsf{decrypt}\ E^l\ \mathsf{as}\ \{\!|E_1^{l_1}, \cdots, E_j^{l_j}; x_{j+1}, \cdots, x_k|\!\}^{\ell}_{E_0^{l_0}}\ [\mathsf{orig}\ \mathcal{L}^{l_s}]\ \mathsf{in}\ P) = & \\
 & \quad \mathcal{G}(E^l) \quad \wedge \wedge_{\ell' \in \mathcal{L}}\ \in \delta(l_s, \ell') \quad \wedge\ \wedge_{i=0}^{j}\ \mathcal{G}(E_i^{l_i})\ \wedge \\
 & \quad \forall\ l', l_0, l_1', \cdots, l_k', \ell', l_s' : \alpha_l(l', l_0', l_1', \cdots, l_k', \ell', l_s') \wedge \gamma(l, l')\ \Rightarrow \\
 & \quad \forall V_0, V_0' : \gamma(l_0, V_0) \wedge \gamma(l_0', V_0') \Rightarrow \\
 & \quad\ KP(V_0, V_0') \wedge \\
 & \quad\ \wedge_{i=1}^{j}\ NEI(l_i', l_i) \Rightarrow \wedge_{i=j+1}^{k}\ \rho(\lfloor x_i \rfloor, l_i')\ \wedge \\
 & \qquad\qquad ((\neg\delta(l_s, \ell') \vee \neg\delta(l_s', \ell)) \Rightarrow (\ell', \ell) \in \psi)\ \wedge \\
 & \qquad\qquad \mathcal{G}(P)
\end{array}
$$

$$
\begin{array}{l}
\ldots \\
\wedge\wedge_{k \in \mathcal{A}_{\mathsf{AEnc}}} \\
\forall l_1, l_2 : (\exists l_1', l_{10}, \cdots, l_{1k}, \ell_1, l_{1s}, l_2', l_{20}, \cdots, l_{2k}, \ell_2, l_{2s} : \\
\qquad \alpha_k(l_1', l_{10}, \cdots, l_{1k}, \ell_1, l_{1s}) \wedge \\
\qquad \alpha_k(l_2', l_{20}, \cdots, l_{2k}, \ell_2, l_{2s}) \wedge \\
\qquad NEI(l_{10}, l_{20}) \wedge \ldots \wedge NEI(l_{1k}, l_{2k}) \wedge \\
\qquad \gamma(l_1, l_1') \wedge \gamma(l_2, l_2')) \\
\qquad\qquad \Rightarrow NEI(l_1, l_2)
\end{array}
$$

Table 16: Extensions to the generation function analysis in Table 11. Note that ρ, κ, γ, and δ are referred to as $\rho^g, \kappa^p, \gamma^g$, and δ^g, respectively, elsewhere in this report.

The axiomatisation of the auxiliary predicate *NEI* is extended to include asymmetric

encryption. That is, asymmetric encryptions in α_k are consulted and when all subcomponents match then the labels of two asymmetric encryptions are required to be in *NEI*. This is done for all k in the set $\mathcal{A}_{\mathsf{AEnc}}$ of arities of asymmetric encryptions or asymmetric decryptions in the program that is analysed.

The implementation follows the lines presented in Section 8 and also for asymmetric decryption queries to δ are stratified in the syntactic way required by the Succinct Solver.

B.6. Dolev-Yao Attacker

In [4], the formula for the attacker, $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ is extended with the capability use asymmetric encryption and asymmetric decryption as well as access to private and free names that represent asymmetric keys. Consequently, in the proof of Theorem 4 in [4] the attacker Q_{hard} (given in Section 7) is extended with the following processes:

$$
\begin{aligned}
Q_6^k &= (;\ z).\,(;\ z_0).\,\mathsf{decrypt}\ z\ \mathsf{as}\ \{\!|;\ z_1,\cdots,z_k|\!\}^{\ell_\bullet}_{z_0}\ [\mathsf{orig}\ \mathcal{C}]\ \mathsf{in}\ 0\\
Q_7^k &= (;\ z_0).\cdots(;\ z_k).\,\langle\{\!|z_1,\cdots,z_k|\!\}^{\ell_\bullet}_{z_0}[\mathsf{dest}\ \mathcal{C}]\rangle.\,0\\
Q_8^k &= \langle m^+_\bullet\rangle.\,0 \mid \langle m^+_\bullet\rangle.\,0 \mid \langle m^+_1\rangle.\,0 \mid \langle m^-_1\rangle.\,0 \mid \cdots \mid \langle m^+_p\rangle.\,0 \mid \langle m^-_p\rangle.\,0 \mid (;\ z).\,0
\end{aligned}
$$

where the free names that represent asymmetric keys in the program that is analysed are $\{m^+_1, m^-_1, \cdots, m^+_p, m^-_p\}$ and $m^+_\bullet$ and $m^-_\bullet$ are two distinct canonical names.

A labelled LySA process, R_{hard}, for which $\mathcal{G}(R_{hard})$ gives a formula equivalent to the new $\mathcal{F}^{\mathsf{DY}}_{\mathsf{RM}}$ can the be found by adding labels $l_\bullet$ and $l_{\mathcal{C}}$ to the processes above similarly to what was done in Section 7. The resulting process is:

$$
\begin{aligned}
R_6^k &= (;\ z).\,(;\ z_0).\,\mathsf{decrypt}\ z^{l_\bullet}\ \mathsf{as}\ \{\!|;\ z_1,\cdots,z_k|\!\}^{\ell_\bullet}_{z_0^{l_\bullet}}\ [\mathsf{orig}\ \mathcal{C}^{l_{\mathcal{C}}}]\ \mathsf{in}\ 0\\
R_7^k &= (;\ z_0).\cdots(;\ z_k).\,\langle\{\!|z_1^{l_\bullet},\cdots,z_k^{l_\bullet}|\!\}^{\ell_\bullet}_{z_0^{l_\bullet}}[\mathsf{dest}\ \mathcal{C}^{l_\bullet}]\rangle.\,0\\
R_8^k &= \langle m^{+\,l_\bullet}_\bullet\rangle.\,0 \mid \langle m^{+\,l_\bullet}_\bullet\rangle.\,0 \mid \langle m^{+\,l_\bullet}_1\rangle.\,0 \mid \langle m^{-\,l_\bullet}_1\rangle.\,0 \mid \cdots \mid \langle m^{+\,l_\bullet}_p\rangle.\,0 \mid \langle m^{-\,l_\bullet}_p\rangle.\,0 \mid (;\ z).\,0
\end{aligned}
$$

These additions to the attacker ends the description of the modifications needed to cater for asymmetric encryption.