A Formal Analysis of Complex Type Flaw Attacks on Security Protocols *

Han Gao[1], Chiara Bodei[2], and Pierpaolo Degano[2]

[1] Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads bldg 322, DK-2800 Kongens Lyngby, Denmark. hg@imm.dtu.dk

[2] Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, I-56127, Pisa, Italy. {chiara,degano}@di.unipi.it

Abstract. A simple type confusion attack occurs in a security protocol, when a principal interprets data of one type as data of another. These attacks can be successfully prevented by "tagging" types of each field of a message. Complex type confusions occur instead when tags can be confused with data and when fields or sub-segments of fields may be confused with concatenations of fields of other types. Capturing these kinds of confusions is not easy in a process calculus setting, where it is generally assumed that messages are correctly interpreted. In this paper, we model in the process calculus LySa only the misinterpretation due to the confusion of a concatenation of fields with a single field, by extending the notation of one-to-one variable binding to many-to-one binding. We further present a formal way of detecting these possible misinterpretations, based on a Control Flow Analysis for this version of the calculus. The analysis over-approximates all the possible behaviour of a protocol, including those effected by these type confusions. As an example, we considered the amended Needham-Schroeder symmetric protocol, where we succeed in detecting the type confusion that lead to a complex type flaw attacks it is subject to. Therefore, the analysis can capture potential type confusions of this kind on security protocols, besides other security properties such as confidentiality, freshness and message authentication.

1 Introduction

In the last decades, formal analyses of cryptographic protocols have been widely studied and many formal methods have been put forward. Usually, protocol specification is given at a very high level of abstraction and several implementation aspects, such as the cryptographic ones, are abstracted away. Despite the abstract working hypotheses, many attacks have been found that are independent of these aspects. Sometimes, this abstract view is not completely adequate, though. At a high level, a message in a protocol consists of fields: each represents some value, such as the name of a principal, a nonce or a key. This structure can be easily modelled by a process calculus. Nevertheless, at a more concrete level,

* This work has been partially supported by the project SENSORIA.

a message is nothing but a raw sequence of bits. In this view, the recipient of a message has to decide the interpretation of the bit string, i.e. how to decompose the string into substrings to be associated to the expected fields (of the expected length) of the message. The message comes with no indication on its arity and on the types of its components. This source of ambiguity can be exploited by an intruder that can fool the recipient into accepting as valid a message different from the expected one. A *type confusion attack* arises in this case.

A *simple* type confusion occurs when a field is confused with another [16]. The current preventing techniques [13] consists in systematically associating message fields with tags representing their intended type. On message reception, honest participants check tags so that fields with different types cannot be mixed up. As stated by Meadows [17], though, simple tags could not suffice for more complex type confusion cases: "in which tags may be confused with data, and terms of pieces of terms of one type may be confused with concatenations of terms of several other types." Tags should also provide the length of tagged fields.

Here, we are interested in semantically capturing attacks that occur when a concatenation of fields is confused with a single field [24]. Suppose, e.g. that the message pair (A, N), where A is a principal identity and N is a fresh nonce, is interpreted as a key K, from the receiver of the message. For simplicity, we call them *complex type confusion attacks.* This level of granularity is difficult to capture with a standard process calculus. An alternative could be separating control from data, as in [1], and using equational theories on data; this however makes mechanical analysis more expensive. In a standard process algebraic framework, there is no way to confuse a term (A, N) with a term K. The term is assumed to abstractly model a message, plugging in the model the hypothesis that the message is correctly interpreted. In concrete implementation this confusion is instead possible, provided that the two strings have the same length.

As a concrete example, consider the Amended Needham Schroeder symmetric key protocol [9]. It aims at distributing a new session key K between two agents, Alice (A) and Bob (B), via a trusted server S. Initially each agent is assumed to share a long term key, K_A and K_B resp., with the server. The protocol narration is reported in Fig. 1 (a). In messages 1 and 2, A initiates the protocol with B.

$$
\begin{array}{l}
1.\ A \to B : A \\
2.\ B \to A : \{A, N_B\}_{K_B} \\
3.\ A \to S :\ A, B, N_A, \{A, N_B\}_{K_B} \\
4.\ S \to A :\ \{N_A, B, K, \{K, N_B, A\}_{K_B}\}_{K_A} \\
5.\ A \to B : \{K, N_B, A\}_{K_B} \\
6.\ B \to A : \{N\}_K \\
7.\ A \to B : \{N-1\}_K \\
\qquad\qquad\qquad (a)
\end{array}
\qquad
\begin{array}{lll}
1. & A \to B : & A \\
2. & B \to A : & \{A, N_B\}_{K_B} \\
3. & A \to S : & A, B, N_A, \{A, N_B\}_{K_B} \\
1'. & M \to A : & N_A, B, K' \\
2'. & A \to M : & \{N_A, B, K', N'_A\}_{K_A} \\
4. & M(S) \to A : & \{N_A, B, K', N'_A\}_{K_A} \\
5. & A \to M(B) : & N'_A \\
6. & M(B) \to A : & \{N\}_{K'} \\
7. & A \to M : & \{N-1\}_{K'} \\
 & & (b)
\end{array}
$$

Fig. 1. Amended Needham-Schroeder Symmetric Protocol: Protocol Narration (a) and Type Flaw Attack (b)

In message 3 S generates a new session key K, that is distributed in messages 4 and 5. Nonces created by A and B are used to check freshness of the new key. Finally, messages 6 and 7 are for mutual authentication of A and B: B generates a new nonce N and exchanges it with A, encrypted with the new session key K.

The protocol is vulnerable to a complex type flaw attack, discovered by Long [14] and shown in Fig. 1 (b). It requires two instances of the protocol, running in parallel. In one, A plays the roles of initiator and in the other that of responder. In the first instance, A initiates the protocol with B. In the meantime, the attacker, M, initiates the second instance with A and sends the triple N_A, B, K' to A (in step $1'$). The nonce N_A is a copy from step 3 in the first instance and K' is a faked key generated by the attacker. A will generate and send out the encryption of the received fields, N_A, B, K', and a nonce N'_A. The attacker $M(S)$ impersonates S and replays this message to A in the first instance. A decrypts this message, checks the nonce N_A and the identity B, and accepts K' as the session key, which is actually generated by the attacker. After the challenge and response steps (6 and 7), A will communicate with M using the faked key K'.

Our idea is to explore complex type confusion attacks, by getting closer to the implementation, without crossing the comfortable borders of process calculi. To this aim, we formally model the possible misinterpretations between terms and concatenations of terms. More precisely, we extend the notation of one-to-one variable binding to many-to-one binding in the process calculus LYSA [5], that we use to model security protocols. The Control Flow Analysis soundly over-approximates the behaviour of protocols, by collecting the set of messages that can be sent over the network, and by recording which values variables may be bound to. Moreover, at each binding occurrence of a variable, the analysis checks whether there is any many-to-one binding possible and records it as a binding violation. The approach is able to detect complex type confusions possibly leading to attacks in cryptographic protocols. Other security properties can be addressed in the same framework, by just changing the values of interest of the Control Flow Analysis, while its core does not change.

The paper is organized as follows. In Section 2, we present the syntax and semantics of the LYSA calculus. In Section 3, we introduce the Control Flow Analysis and we describe the Dolev-Yao attacker used in our setting. Moreover, we conduct an experiment to analyse the amended Needham-Schoreder symmetric key protocol. Section 4 concludes the paper.

2 The LySa Calculus

The LySa calculus [5] is a process calculus, designed especially for modelling cryptographic protocols in the tradition of the π- [20] and Spi- [2] calculi. It differs from these essentially in two aspects: (1) the absence of channels: all processes have only access to a single global communication channel, the network; (2) the inclusion of pattern matching *into* the language constructs where values can become bound to values, i.e. into input and into decryption (while usually there is a separate construct).

Syntax In LYSA, the basic building blocks are values, $V \in Val$, which correspond to *closed* terms, i.e. terms without free variables. Values are used to represent keys, nonces, encrypted messages, etc. Syntactically, they are described by expressions $E \in Expr$ (or terms) that may either be variables, names, or encryptions. Variables and names come from two disjoint sets Var, ranged over by x, and $Name$, ranged over by n, respectively. Finally, expressions may be encryptions of a k-tuple of other expressions, in which case, E_0 is the key used to perform the encryption. LYSA expressions are, in turn, used to construct LYSA processes $P \in Proc$ as shown below. Here, we assume perfect cryptography.

$$
\begin{aligned}
E ::= {} & n \mid x \mid \{E_1, \ldots, E_k\}_{E_0} \\
P ::= {} & \langle E_1, \ldots, E_k \rangle.P \mid (E_1, \ldots, E_j; x_{j+1}, \ldots, x_k)^l.P \\
& \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_k\}^l_{E_0}\ \mathsf{in}\ P \mid \\
& (\nu\, n)P \mid P_1 | P_2 \mid !P \mid 0
\end{aligned}
$$

The set of free variables, resp. free names, of a term or a process is defined in the standard way. As usual we omit the trailing 0 of processes. The label l from a denumerable set Lab ($l \in Lab$) in the input and in the decryption constructs uniquely identifies each input and decryption point, resp., and is mechanically attached.

In addition to the classical constructs for composing processes, LYSA contains an input and a decryption construct with pattern matching. Patterns are in the form $(E_1, \cdots, E_j; x_{j+1}, \cdots, x_k)$ and are matched against k-tuples of values $\langle E'_1, \cdots, E'_k \rangle$. The intuition is that the matching succeeds when the first $1 \leq i \leq j$ values E'_i pairwise correspond to the values E_i, and the effect is to bind the remaining $k - j$ values to the variables $x_{j+1}, \cdots, x_k$. Syntactically, this is indicated by a semi-colon that separates the components where matching is performed from those where only binding takes place. For example, let $P = \mathsf{decrypt}\ \{y\}_K\ \mathsf{as}\ \{x;\}^l_K\ \mathsf{in}\ P'$ and $Q = \mathsf{decrypt}\ \{y\}_K\ \mathsf{as}\ \{;x\}^l_K\ \mathsf{in}\ Q'$. While the decryption in P succeeds only if x matches y, the one in Q always does, binding x to y.

Extended LYSA As seen above, in LYSA, values are passed around among processes through pattern matching and variable binding. This is the way to model how principals acquire knowledge from the network, by reading messages (or performing decryptions), provided they have certain format forms. A requirement for pattern matching is that patterns and expressions are of the same length: processes only receive (or decrypt) messages, whose length is exactly as expected and each variable is binding to one single value, later on as *one-to-one* binding. We shall relax this constraint, because it implicitly prevents us from modelling complex type confusions, i.e. the possibility to accept a concatenation of fields as a single one. Consider the complex type flaw attack on the amended Needham-Schroder protocol, shown in the Introduction. The principal A, in the role of responder, is fooled by accepting N_A, B, K' as the identity of the initiator and generates the encryption $\{N_A, B, K', N'_A\}_{K_A}$, which will be replayed by the attacker later on in the first instance. In LYSA, A's input can be roughly expressed as $(; x_b)$, as she is expecting a single field representing the identity of the initiator of the protocol. Because of the length requirement, though, x_b can only be binding to a single value and not to a concatenation of values, such as the (N_A, B, K') object of the output of the attacker.

To model complex type confusions, we need to allow a pattern matching to succeed also in the cases in which the length of lists is different. The extension of the notation of pattern matching and variable binding will be referred as *many-to-one* binding. Patterns are then allowed to be matched against expressions with *at least* the same number of elements. A single variable can then be bound also to a concatenation of values. Since there may be more values than variables, we partition the values into groups (or lists) such that there are the same number of value groups and variables. Now, each group of values is bound to the corresponding variable. In this new setting, the pattern in A's input $(; x_b)$ can instead successfully match the expression in the faked output of the attacker $\langle N_A, B, K' \rangle$ and result in the binding of x_b to the value (N_A, B, K').

We need some auxiliary definitions first. The domain of *single values* is built from the following grammar and represents closed expressions (i.e. without free variables), where each value is a singleton, i.e. it is not a list of values. In other words, no many-to-one binding has affected the expression. These are the values used in the original LYSA semantics.

$val \ni v ::= n \mid \{v_1, \ldots, v_k\}_{v_0}$.

General values are closed expressions, where each value V can be a list of values $(V_1, .., V_n)$. These values are used to represent expressions closed after at least one many-to-one-binding and are the values our semantics handles.

$Val \ni V ::= v \mid (V_1, .., V_n) \mid \{V_1, .., V_n\}_{V_0}$

To perform meaningful matching operations between lists of general values, we first flatten them, thus obtaining *flattened values* that can be either single values v or encryptions of general values.

$Flat \ni T := v \mid \{V_1, .., V_n\}_{V_0}$

Flattening is obtained by using the following $Flatten$ function $Fl : Val \to Flat$

- $Fl(v) = v$;
- $Fl((V_1, .., V_n)) = Fl(V_1), ..., Fl(V_n)$;
- $Fl(\{V_1, .., V_n\}_{V_0}) = \{V_1, .., V_n\}_{V_0}$.

Example 1. $Fl(((n_1, n_2), (\{m_1, (m_2, m_3)\}_{m_0}))) = n_1, n_2, \{m_1, (m_2, m_3)\}_{m_0}$

The idea is that encryptions cannot be directly flattened when belonging to a list of general values. Their contents are instead flattened when received and analysed in the decryption phase.

To perform many-to-one bindings, we resort to a partition operator $\prod_k$ that, given a list of flattened values $(T_1, \ldots, T_n)$, returns all the possible partitions composed by k *non-empty* groups (or lists) of flattened values. For simplicity, we use $\widetilde{T}$ to represent a list of flattened values $(T_1, \ldots, T_j)$

$$\prod_k (T_1, \ldots, T_n) = \{(\widetilde{T_1}, ..., \widetilde{T_k}) \mid \forall i : \widetilde{T_i} \neq \emptyset \wedge Fl((\widetilde{T_1}, ..., \widetilde{T_k})) = (T_1, \ldots, T_n)\} \text{ if } n \geq k$$

Note that the function is only defined if $n \geq k$, in which case it returns a set of lists satisfying the condition. Now binding k variables $x_1, ..., x_k$ to n flattened values amounts to partitioning the values into k (the number of variables) non-empty lists of flattened values, $(\widetilde{T_1}, ..., \widetilde{T_k}) \in \prod_k(T_1, \ldots, T_n)$, and binding variables x_i to the corresponding list $\widetilde{T_i}$.

Example 2. Consider the successful matching of $(m; x_1, x_2)$ against (m, n_1, n_2, n_3). Since $\prod_2(n_1, n_2, n_3) = \{((n_1), (n_2, n_3)), ((n_1, n_2), (n_3))\}$, it results in two possible effects (recall that for each i, $\widetilde{T_i}$ must be non-empty), i.e.

- binding variable x_1 to (n_1) and binding variable x_2 to (n_2, n_3), or
- binding variable x_1 to (n_1, n_2) and binding variable x_2 to (n_3).

Finally, we define the relation $=^F$ as the least equivalence over Val and (by overloading the symbol) and over $Flat$ that includes:

- $v =^F v'$ iff $v = v'$;
- $(V_1, ..., V_k) =^F (V'_1, ..., V'_n)$ iff $Fl(V_1, ..., V_k) = Fl(V'_1, ..., V'_n)$;
- $\{V_1, ..., V_k\}_{V_0} =^F \{V'_1, ..., V'_n\}_{V'_0}$ iff $Fl(V_1, ..., V_k) = Fl(V'_1, ..., V'_n)$ and $Fl(V_0) = Fl(V'_0)$;

Semantics LYSA has a reduction semantics, based on a standard structural equivalence. The *reduction relation* $\rightarrow_\mathcal{R}$ is the least relation on closed processes that satisfies the rules in Tab. 1. It uses a standard notion of structural congruence $\equiv$.

At run time, the complex type confusions are checked by a reference monitor, which aborts when there is a possibility that a concatenation of values is bound to a single variable. We consider two variants of the *reduction relation* $\rightarrow_\mathcal{R}$, graphically identified by a different instantiation of the relation $\mathcal{R}$, which decorates the transition relation. The first variant takes advantage of checks on type confusions, while the other one discards them: essentially, the first semantics checks for the presence of complex type confusions. More precisely, the reference monitor performs its checks at each binding occurrence, i.e. when the pattern $V_1, \ldots, V_k$ is matched against $V'_1, \ldots, V'_k; x_j, ..., x_t$. Both the lists of values are flattened and result in s values $T_1, \ldots, T_s$, len values $T'_1, \ldots, T'_{len}$, resp. The reference monitor checks whether the length of the list $len + (t - j)$ of the flattened values of the pattern, corresponds to the length s of the list of the general values to match against it. If $(len + t - j) = s$ then there is a one-to-one correspondence between variables and flattened values. Otherwise, then there exists at least a variable x_i, which may bind to a list of more than one value. Formally:

- the reference monitor semantics, $P \rightarrow_{\mathsf{RM}} P'$, takes $\mathcal{R} = \mathsf{RM}(s, len + t - j)$ true when $s = len + t - j$, where s and $len + t - j$ are defined as above;
- the standard semantics, $P \rightarrow P'$ takes $\mathcal{R}$ to be universally true.

The rule (Com) in the Tab. 1 states when an output $\langle V_1, \ldots, V_k\rangle.P$ is matched by an input $(V'_1, \ldots, V'_j; x_{j+1}, \ldots, x_t)^l.P'$. It requires that: (i) the first j general values of the input pattern $V'_1, \ldots, V'_j$ are flattened into len flattened values $T'_1, \ldots, T'_{len}$; (ii) the general values $V_1, \ldots, V_k$ in the output tuple are flattened into s flattened values $T_1, \ldots, T_s$; (iii) if $s \geq (len + t - j)$ and the first len values of $T_1, \ldots, T_s$ pairwise match with $T'_1, \ldots, T'_{len}$ then the matching succeeds; (iv) in

(Com)

$$\frac{\wedge_{i=1}^{len} T_i =^F T_i' \ \wedge \ \mathcal{R}(s, len + t - j)}{\langle V_1, \ldots, V_k \rangle.P \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_t)^l.P' \rightarrow_{\mathcal{R}} P \mid P'[\widetilde{T}_{j+1}/x_{j+1}, \ldots, \widetilde{T}_t/x_t]}$$

where (A) holds

(Dec)

$$\frac{V_0 =^F V_0' \ \wedge \ \wedge_{i=1}^{len} T_i =^F T_i' \ \wedge \ \mathcal{R}(s, len + t - j)}{\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}, \ldots, x_t\}^l_{V_0'}\ \mathsf{in}\ P \rightarrow_{\mathcal{R}} P[\widetilde{T}_{j+1}/x_{j+1}, \ldots, \widetilde{T}_t/x_t]}$$

where (A) holds

(New)

$$\frac{P \rightarrow_{\mathcal{R}} P'}{(\nu\ n)P \rightarrow_{\mathcal{R}} (\nu\ n)P'}$$

(Par)

$$\frac{P_1 \rightarrow_{\mathcal{R}} P_1'}{P_1 \mid P_2 \rightarrow_{\mathcal{R}} P_1' \mid P_2}$$

(Congr)

$$\frac{P \equiv P' \ \wedge \ P' \rightarrow_{\mathcal{R}} P'' \wedge P'' \equiv P'''}{P \rightarrow_{\mathcal{R}} P'''}$$

$$A = \left\{ \begin{array}{ll} Fl((V_1, \ldots, V_k)) & = T_1, \ldots, T_{len}, T_{len+1}, \ldots, T_s \\ Fl(V_1', \ldots, V_j')) & = T_1', \ldots, T_{len}' \\ (\widetilde{T}_{j+1}, \ldots, \widetilde{T}_t) & \in \prod_{t-j}(T_{len+1}, \ldots, T_s) \end{array} \right\}$$

Table 1. Operational Semantics; $P \rightarrow_{\mathcal{R}} P'$, parameterised on $\mathcal{R}$.

this case, the remaining values $T_{len+1}, \ldots, T_s$ are partitioned into a sequence of non-empty lists $\widetilde{T}_i$, whose number is equal to the one of the variables (i.e. $t - j$), computed by the operator $\prod_{t-j}$. Furthermore, the reference monitor checks for the possibility of many-to-one binding, i.e. checks whether $s \geq (len+t-j)$. If this is the case, it aborts the execution. Note that, if instead $s = (len + t - j)$, then $Fl(V_1, \ldots, V_k) = V_1, \ldots, V_k$, $Fl(V_1', \ldots, V_j') = V_1', \ldots, V_j'$, $k = s$, and $j = len$.

The rule (Dec) performs pattern matching and variable binding in the same way as in (Com), with the following additional requirement: the keys for encryption and decryption have to be equal, i.e. $V_0 =^F V_0'$. Similarly, the reference monitor aborts the execution if many-to-one binding occurs.

The rules (New), (Par) and (Congr) are standard, where the (Congr) rule also makes use of structural equivalence $\equiv$.

As for the dynamic property of the process, we say that a process is *complex type coherent*, when there is no complex type confusions, i.e. there is no many-to-one binding in any of its executions. Consequently, the reference monitor will never stop any execution step.

Definition 1 (Complex Type Coherence). *A process P is* complex type coherent *if for all the executions $P \rightarrow^* P' \rightarrow P''$ whenever $P' \rightarrow P''$ is derived using either axiom*

- (Com) on $\langle V_1, \ldots, V_k \rangle.Q \mid (V_1', \ldots, V_j'; x_{j+1}, \ldots, x_t)^l.Q'$ or

- (Dec) on $\mathsf{decrypt}\ \{V_1, \ldots, V_k\}_{V_0}\ \mathsf{as}\ \{V_1', \ldots, V_j'; x_{j+1}, \ldots, x_t\}^l_{V_0'}$ in Q

it is always the case that $s = len + t - j$, where $Fl(V_p, \ldots, V_k) = T_p, ..., T_s$ and $Fl(V_p', \ldots, V_j') = T_p, ..., T_{len}$ with $p = 1$ ($p = 0$) in the case of (Com), (Dec), respectively.

3 The Control Flow Analysis

Our analysis aims at safely over-approximating how a protocol behaves and when the reference monitor may abort the computation.

The Control Flow Analysis describes a protocol behaviour by collecting all the communications that a process may participate in. In particular, the analysis records which value tuples may flow over the network (see the analysis component κ below) and which value variables may be bound to (component ρ). This gives information on bindings due to pattern matching. Moreover, at each binding occurrence, the Control Flow Analysis checks whether there is any many-to-one binding possible, and records it as a binding violation (component ψ). Formally, the approximation, or *estimate*, is a triple (ρ, κ, ψ) (respectively, a pair (ρ, θ) when analysing an expression E) that satisfies the judgements defined by the axioms and rules in Tab. 2.

Analysis of Expressions For each expression E, our analysis will determine a superset of the possible values it may evaluate to. For this, the analysis keeps track of the potential values of variables, by recording them into the global *abstract environment*:

• $\rho : \mathcal{X} \rightarrow \mathcal{P}(Val)$ that maps the variables to the sets of general values that they may be bound to, i.e. if $a \in \rho(x)$ then x may take the value a.

The judgement for expressions takes the form $\rho \models E : \vartheta$ where $\vartheta \subseteq Val^*$ is an acceptable *estimate* (i.e. a sound over-approximation) of the set of general value lists that E may evaluate to in the environment ρ. The judgement is defined by the axioms and rules in the upper part of Tab. 2. Basically, the rules demand that ϑ contains all the value lists associated with the components of a term, e.g. a name n evaluates to the set ϑ, provided that n belongs to ϑ; similarly for a variable x, provided that ϑ includes the set of value lists $\rho(x)$ to which x is associated with.

The rule (Enc) (i) checks the validity of estimates θ_i for each expression E_i; (ii) requires that all the values $T_1, ..., T_s$ obtained by flattening the k-tuples $V_1, ..., V_k$, such that $V_i \in \theta_i$, are collected into values of the form $(\{T_1, \cdots, T_s\}^l_{V_0})$, (iii) requires these values to belong to ϑ.

Analysis of Processes In the analysis of processes, we focus on which tuples of values can flow on the network:

• $\kappa \subseteq \mathcal{P}(Val^*)$, the *abstract network environment*, includes all the tuples forming a message that may flow on the network, e.g. if the tuple $\langle a, b \rangle$ belongs to κ then it can be sent on the network.

The judgement for processes has the form: $(\rho, \kappa) \models P : \psi$, where ψ is the possibly empty set of "error messages" of the form l, indicating a binding violation at the point labelled l. We prove in Theorem 2 below that when $\psi = \emptyset$ we may do without the reference monitor. The judgement is defined by the axioms and rules in the lower part of Tab. 2 (where $A \Rightarrow B$ means that B is analysed only when A is evaluated to be *true*) and are explained below.

CFA Rules Explanation The rule for *output* (Out), computes all the messages that can be obtained by flattening all the general values to which sub-expressions may be evaluated. The use of the flatten function makes sure that each message is plain-structured, i.e. redundant parentheses are dropped.

$$\text{(Name)}\ \frac{(n) \in \vartheta}{\rho \models n : \vartheta} \qquad \text{(Var)}\ \frac{\rho(x) \subseteq \vartheta}{\rho \models x : \vartheta}$$

$$\text{(Enc)}\ \frac{\begin{array}{c} \wedge_{i=0}^{k}\ \rho \models E_i : \vartheta_i\ \wedge \\ \forall V_0, \ldots, V_k : \wedge_{i=0}^{k}\ V_i \in \vartheta_i\ \wedge\ Fl(V_1, ..., V_k) = T_1, ..., T_s \Rightarrow \\ (\{T_1, ..., T_s\}_{V_0}) \in \vartheta \end{array}}{\rho \models \{E_1, \ldots, E_k\}_{E_0} : \vartheta}$$

$$\text{(Out)}\ \frac{\begin{array}{c} \wedge_{i=1}^{k}\ \rho \models E_i : \vartheta_i\ \wedge \\ \forall V_1, \ldots, V_k : \wedge_{i=1}^{k} V_i \in \vartheta_i\ \wedge\ Fl(V_1, ..., V_k) = T_1, ..., T_s \Rightarrow \\ \langle T_1, ..., T_s \rangle \in \kappa\ \wedge\ (\rho, \kappa) \models P : \psi \end{array}}{(\rho, \kappa) \models \langle E_1, \ldots, E_k \rangle.P : \psi}$$

$$\text{(In)}\ \frac{\begin{array}{c} \wedge_{i=1}^{j}\ \rho \models E_i : \vartheta_1\ \wedge \\ \forall V'_1, \ldots, V'_j : \wedge_{i=1}^{j} V'_i \in \vartheta_i\ \wedge\ Fl(V'_1, ..., V'_j) = T'_1, ..., T'_{len} \Rightarrow \\ \forall \langle T_1, ..., T_s \rangle \in \kappa : T_1, ..., T_{len} =^F T'_1, ..., T'_{len} \Rightarrow \\ \forall (\widetilde{T}_{j+1}, \ldots, \widetilde{T}_t) \in \prod_{t-j}(T_{len+1}, \ldots, T_k) \Rightarrow \\ (\wedge_{i=j+1}^{t}\ \widetilde{T}_i \in \rho(x_i)\ \wedge (s > len + t - j) \Rightarrow l \in \psi\ \wedge\ (\rho, \kappa) \models P : \psi) \end{array}}{\begin{array}{c} (\rho, \kappa) \models (E_1, \ldots, E_j; x_{j+1}, \ldots, x_t)^l.P : \psi \\ \text{where } s \geq len + t - j \end{array}}$$

$$\text{(Dec)}\ \frac{\begin{array}{c} \rho \models E : \vartheta\ \wedge\ \wedge_{i=0}^{j}\ \rho \models E_i : \vartheta_i \wedge \\ \forall V'_0, \ldots, V'_j : \wedge_{i=0}^{j} V'_i \in \vartheta_i\ \wedge\ Fl(V'_1, ..., V'_j) = T'_1, ..., T'_{len} \Rightarrow \\ \forall \{T_1, \ldots, T_s\}_{V_0} \in \vartheta : T_1, ..., T_{len} =^F T'_1, ..., T'_{len} \Rightarrow \\ \forall (\widetilde{T}'_{j+1}, \ldots, \widetilde{T}'_t) \in \prod_{t-j}(T_{len+1}, \ldots, T_k) \Rightarrow \\ (\wedge_{i=j+1}^{t}\ \widetilde{T}'_i \in \rho(x_i)\ \wedge (s > len + t - j) \Rightarrow l \in \psi\ \wedge\ (\rho, \kappa) \models P : \psi) \end{array}}{\begin{array}{c} (\rho, \kappa) \models \mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_j; x_{j+1}, \ldots, x_t\}_{E_0}^{l}\ \mathsf{in}\ P : \psi \\ \text{where } s \geq len + t - j \end{array}}$$

$$\text{(New)}\ \frac{(\rho, \kappa) \models P : \psi}{(\rho, \kappa) \models (\nu\ n)P : \psi} \qquad \text{(Par)}\ \frac{(\rho, \kappa) \models P_1 : \psi\ \wedge\ (\rho, \kappa) \models P_2 : \psi}{(\rho, \kappa) \models P_1 | P_2 : \psi}$$

$$\text{(Rep)}\ \frac{(\rho, \kappa) \models P : \psi}{(\rho, \kappa) \models !P : \psi} \qquad \text{(Nil)}\ \ (\rho, \kappa) \models 0 : \psi$$

Table 2. Analysis of terms; $\rho \models E : \vartheta$, and processes: $(\rho, \kappa) \models P : \psi$

More precisely, it (i) checks the validity of estimates θ_i for each expression E_i; (ii) requires that all the values obtained by flattening the k-tuples $V_1, ..., V_k$, such that $V_i \in \theta_i$, can flow on the network, i.e. that they are in the component ρ; (iii) requires that the estimate (ρ, κ, ψ) is valid also for the continuation process P. Suppose e.g. to analyse $\langle A, N_A \rangle.0$. In this case, we have that $\rho \models A : \{(A)\}$, $\rho \models N_A : \{(N_A)\}$, $Fl((A), (N_A)) = A, N_A$ and $\langle A, N_A \rangle \in \kappa$. Suppose instead to have $\langle A, x_A\rangle.P$ and $\rho(x_A) = \{(N_A), (N'_A)\}$. In this case we have $Fl((A), (N_A)) = A, N_A$, $Fl((A), (N'_A)) = A, N'_A$, $\langle A, N_A\rangle \in \kappa$ and also $\langle A, N'_A\rangle \in \kappa$.

The rule for *input* (In) basically looks up in κ for matched tuples and performs variable binding before analysing the continuation process. This is done in the following steps: the rule (i) evaluates the first j expressions, whose results are general values, V'_i. These are flattened into a list of values $T'_1, ..., T'_{len}$ in order to perform the pattern matching. Then, the rule (ii) checks whether the first len values of any message $\langle T_1, ..., T_s\rangle$ in κ (i.e. any message predicted to flow on the network) matches the values from previous step, i.e. $T'_1, ..., T'_{len}$. Also, the rule (iii) partitions the remaining $T_{len+1}, ..., T_s$ values of the tuple $\langle T_1, ..., T_s\rangle$ in all the possible ways to obtain $t - j$ lists of flattened values $\widetilde{T_i}$ and requires each list is bound to the corresponding variable $\widetilde{T_i} \in \rho(x_i)$. The rule (iv) checks whether the flattened pattern and the flattened value are of the same length. If this is not the case, the final step should be in putting l in the error component ψ. Finally, the rule (v) analyses the continuation process. Suppose to analyse the process $(A, x_A; x, x_B).0$, where $\langle A, N_A, B, N_B\rangle \in \kappa$ and $(N_A) \in \rho(x_A)$. Concretising the rule (Inp) gives $j = 2, t = 2$ and the followings,

$$\frac{\begin{array}{ll}
\rho \models A : \vartheta_1 \wedge \rho \models x_A : \vartheta_2 & \text{yielding } \vartheta_1 \ni (A) \text{ and } \vartheta_2 \ni (N_A) \\
\forall V'_1, V'_2 : V'_1 \in \vartheta_1 \wedge V'_2 \in \vartheta_2 \wedge & \text{taking } V'_1 = (A) \text{ and } V'_2 = (N_A)\ \wedge \\
Fl(V'_1, V'_2) = T'_1, ..., T'_{len} & len = 2 \text{ and } T'_1, ..., T'_{len} = A, N_A \\
\forall \langle T_1, \ldots, T_s\rangle \in \kappa : & \text{if } \langle A, N_A, B, N_B\rangle \in \kappa \text{ and } s = 4 \\
 & \text{i.e. } T_1 = A, T_2 = N_A, T_3 = B, T_4 = N_B \\
T_1, \ldots, T_{len} =^F T'_1, \ldots, T'_{len} \Rightarrow & T_1, T_2 =^F T'_1, T'_2 = A, N_A \\
\forall (\widetilde{T_3}, \widetilde{T_4}) \in \prod_2(T_3, T_4) \Rightarrow & \prod_2(T_3, T_4) = \prod_2(B, N_B) = \{((B), (N_B))\} \\
\quad (\widetilde{T_3} \in \rho(x) \wedge \widetilde{T_4} \in \rho(x_A) \wedge & \text{gives } (B) \in \rho(x) \wedge (N_B) \in \rho(x_B) \\
\quad (s > len + t - j) \Rightarrow l \in \psi \wedge & \text{and } 4 = 4 \text{ does not require } l \notin \psi \\
\quad (\rho, \kappa) \models 0 : \psi) & \mathit{true}
\end{array}}{(\rho, \kappa) \models (A, x_A; x, x_B)^l.0 : \psi}$$

In particular, $((B), (N_B)) \in \prod_2(B, N_B)$ implies that $(B) \in \rho(x)$ and $(N_B) \in \rho(x_B)$. Suppose to have also that $\langle A, N_A, B, N_B, K\rangle \in \kappa$. In this case, $((B), (N_B, K)) \in \prod_2(B, N_B, K)$ and therefore $(B) \in \rho(x)$ and $(N_B, K) \in \rho(x_B)$ and also $((B, N_B), K) \in \prod_2(B, N_B, K)$ and therefore $(B, N_B) \in \rho(x)$ and $(K) \in \rho(x_B)$. More precisely:

$$\frac{\begin{array}{ll}
\rho \models A : \vartheta_1 \wedge \rho \models x_A : \vartheta_2 & \text{yielding } \vartheta_1 \ni (A) \text{ and } \vartheta_2 \ni (N_A) \\
\forall V'_1, V'_2 : V'_1 \in \vartheta_1 \wedge V'_2 \in \vartheta_2 \wedge & \text{taking } V'_1 = (A) \text{ and } V'_2 = (N_A)\ \wedge \\
Fl(V'_1, V'_2) = T'_1, ..., T'_{len} & len = 2 \text{ and } T'_1, ..., T'_{len} = A, N_A \\
\forall \langle T_1, \ldots, T_s\rangle \in \kappa : & \text{if } \langle A, N_A, B, N_B, K\rangle \in \kappa \text{ and } s = 5 \\
 & \text{i.e. } T_1 = A, T_2 = N_A, T_3 = B, T_4 = N_B, T_5 = K \\
T_1, \ldots, T_{len} =^F T'_1, \ldots, T'_{len} \Rightarrow & T_1, T_2 =^F T'_1, T'_2 = A, N_A \\
\forall (\widetilde{T_3}, \widetilde{T_4}) \in \prod_2(T_3, T_4, T_5) \Rightarrow & \prod_2(T_3, T_4, T_5) = \prod_2(B, N_B, K) = \\
 & \{((B), (N_B, K)), ((B, N_B), (K))\} \\
\quad (\widetilde{T_3} \in \rho(x) \wedge \widetilde{T_4} \in \rho(x_A) & \text{gives } (B), (B, N_B) \in \rho(x) \text{ and } (K), (N_B, K) \in \rho(x_B) \\
\quad (s > len + t - j) \Rightarrow l \in \psi \wedge & 5 > 2 + 4 - 2 \text{ requires } l \in \psi \\
\quad (\rho, \kappa) \models 0 : \psi & \mathit{true}
\end{array}}{(\rho, \kappa) \models (A, x_A; x, x_B)^l.0 : \psi}$$

The rule for *decryption* (Dec) is similar to (In): the values to be matched are those obtained by evaluating the expression E; while the matching ones are the terms inside decryption. If the check succeeds then variables are bound and the continuation process P is analysed. Moreover, the rule checks the possibility of many-to-one binding: the component ψ must contain the label l corresponding to the decryption. Suppose e.g. to have $\mathsf{decrypt}\ E\ \mathsf{as}\ \{E_1, \ldots, E_2; x_3, \ldots, x_4\}^l_{E_0}\ \mathsf{in}\ P$, with $E = \{A, N_A, B, N_B\}_K$, $E_0 = K, E_1 = A$, $E_2 = x_A$ and $\rho(x_A) = \{(N_A)\}$. Then we have that $\rho \models A : \{(A)\}$, $\rho \models x_A : \{(N_A)\}$ and $Fl((A),(N_A)) = A, N_A$. Then $((B),(N_B)) \in \prod_2(B, N_B)$ implies that $(B) \in \rho(x_3)$ and $(N_B) \in \rho(x_4)$. Suppose to have instead $E = \{A, N_A, B, N_B, K_0\}_K$, then $((B),(N_B, K_0)) \in \prod_2(B, N_B, K_0)$ and therefore $(B) \in \rho(x_3)$ and $(N_B, K_0) \in \rho(x_4)$ and also $((B, N_B),(K_0)) \in \prod_2(B, N_B, K_0)$ and therefore $(B, N_B) \in \rho(x_3)$ and $(K_0) \in \rho(x_4)$. Furthermore $l \in \psi$.

The rule (Nil) does not restrict the estimate, while the rules (New), (Par) and (Rep) ensure that the estimate also holds for the immediate sub-processes.

Semantics Properties Our analysis is correct with respect to the operational semantics of LYSA. The detailed proofs are omitted due to space limitations and can be found in [4].

We have the following results. The first states that estimates are resistant to substitution of closed terms for variables, and it holds for both extended terms and processes. The second one says that estimates respect $\equiv$.

Lemma 1. *1. (a)* $\rho \models E : \vartheta \wedge (T_1, \ldots, T_k) \in \rho(x)$ *imply* $\rho \models E[T_1, \ldots, T_k/x] : \vartheta$
(b) $(\rho, \kappa) \models P : \psi \ \wedge\ (T_1, \ldots, T_k) \in \rho(x)$ *imply* $(\rho, \kappa) \models P[T_1, \ldots, T_k/x] : \psi$
2. If $P \equiv Q$ *and* $(\rho, \kappa) \models P$ *then* $(\rho, \kappa) \models Q$

Our analysis is semantically correct regardless of the way the semantics is parameterised, furthermore the reference monitor semantics cannot stop the execution of P when ψ is empty. The proof is by induction on the inference of $P \to Q$.

Theorem 1. ***(Subject reduction)*** *If* $P \to Q$ *and* $(\rho, \kappa) \models P : \psi$ *then* $(\rho, \kappa) \models Q : \psi$. *Additionally, if* $\psi = \emptyset$ *then* $P \to_{\mathsf{RM}} Q$.

The next theorem shows that our analysis correctly predicts when we can safely do without the reference monitor. We shall say that the reference monitor RM *cannot abort* a process P when there exist no Q, Q' such that $P \to^* Q \to Q'$ and $P \to^*_{\mathsf{RM}} Q \not\to_{\mathsf{RM}}$. (As usual, * stands for the transitive and reflexive closure of the relation in question, and we omit the string of labels in this case; while $Q \not\to_{\mathsf{RM}}$ stands for $\nexists Q' : Q \to_{\mathsf{RM}} Q'$.) We then have:

Theorem 2. ***(Static check for reference monitor)***

If $(\rho, \kappa) \models P : \psi$ *and* $\psi = \emptyset$ *then RM cannot abort* P.

Modelling the Attackers In a protocol execution, several principals exchange messages over an open network, which is therefore vulnerable to a malicious attacker. We assume it is an active Dolev-Yao attacker [10]: it can eavesdrop, and replay, encrypt, decrypt, generate messages providing that the necessary information is within his knowledge, that it increases while interacting with the network. This attacker can be modelled in LYSA as a process running in parallel with the protocol process. Formally, we shall have $P_{sys} \mid P_\bullet$, where P_{sys} represents the protocol process and $P_\bullet$ is some *arbitrary* attacker. To get an account of the infinitely many attackers, the overall idea is to find a formula $\mathcal{F}$ (for a similar treatment see [5]) that characterizes $P_\bullet$: this means that whenever a triple (ρ, κ, ψ) satisfies it, then $(\rho, \kappa) \models P_\bullet : \psi$ and this holds for all attackers, in particular for the hardest one [21]. Intuitively, the formula $\mathcal{F}$ has to mimic how $P_\bullet$ is analysed. The attacker process is parameterised on some attributes of P_{sys}, e.g. the length of all the encryptions that occurred and all the messages sent over the network. In the formula, the names and variables the attacker uses are apart from the ones used by P_{sys}. We can then postulate a new distinguished name $n_\bullet$ (variable $z_\bullet$) in which the names (variables, resp.) of the attacker are coalesced; therefore $n_\bullet$ may represent any name generated by the attacker, while $\rho(z_\bullet)$ represents the attacker knowledge. It is possible to prove that if an estimate of a process P with $\psi = \emptyset$ satisfies the attacker formula than RM does not abort the execution of $P|\ Q$, regardless of the choice of the attacker Q. Further details are in [4, 5].

Implementation Following [5], the implementation can be obtained along the lines that first transform the analysis into a logically equivalent formulation written in Alternation free Least Fixed Point logic (ALFP) [22], and then followed by using the Succinct Solver [22], which computes the least interpretation of the predicate symbols in a given ALFP formula.

3.1 Validation of the Amended Needham-Schroeder Protocol

Here, we will show that the analysis applied to the Amended Needham-Schroeder protocol, successfully captures the complex type confusion leading to the attack, presented in the Introduction.

In LYSA, each instance of the protocol is modelled as three processes, A, B and S, running in parallel within the scope of the shared keys. To allow the complex type confusion to arise, we put two instances together, and add indices to names and variables used in each instance in order to tell them apart, namely

$$P_{NS} \;=\; (\nu\, K_A)(\nu\, K_B)(A_1 \mid A_2 \mid B_1 \mid B_2 \mid S)$$

To save space, processes without indices are shown in Tab. 3. For clarity, each message begins with the pair of principals involved in the exchange. In LYSA we do not have other data constructors than encryption, but the predecessor operation can be modelled by an encryption with the key PRED that is also known to the attacker. For the sake of readability, we directly use $N - 1$. We can apply our analysis and check that $(\rho, \kappa) \models P_{NS} : \psi$, where ρ, κ and ψ have the non-empty entries (only the interesting ones) listed in Tab. 3.

The message exchanges of the regular run (the first instance) performed by A and B are correctly reflected by the analysis. In step 1, B receives the tuple sent by A and binds variable y_a^1 to the value (A), as predicted by $(A) \in \rho(y_a^1)$. In step 2, B generates a nonce N_B^1, encrypts it together to the value of y_a^1 and

Initiator $A:$	Responder $B:$
$/*1*/\ \langle A, B, A\rangle.$	$/*1*/\ (A, B; y_a)^{l_6}.$
$/*2*/\ (B, A; x_{enc})^{l_1}.$	$/*2*/\ (\nu\ N_B)\ \ \langle B, A, \{y_a, N_B\}_{K_B}\rangle.$
$/*3*/\ (\nu\ N_A)\langle A, S, A, B, N_A, x_{enc}\rangle.$	
$/*4*/\ (S, A; x_z)^{l_2}.$	
decrypt x_z as $\{N_A, B; x_k, x_y\}^{l_3}_{K_A}$ in	$/*5*/\ (A, B; y_{enc})^{l_7}.$
$/*5*/\ \langle A, B, x_y\rangle.$	decrypt y_{enc} as $\{N_B, A; y_k\}^{l_8}_{K_B}$ in
$/*6*/\ (B, A; x_{no})^{l_4}.$	$/*6*/\ (\nu\ N_0)\langle B, A, \{N_0\}_{y_k}\rangle.$
decrypt x_{n0} as $\{; x_n\}^{l_5}_{x_k}$ in	$/*7*/\ (A, B; y_{n0})^{l_9}.$
$/*7*/\ \langle A, B, \{x_n - 1\}_{x_k}\rangle.0$	decrypt y_{n0} as $\{N_0 - 1\}^{l_{10}}_{y_k}$ in 0
Server $S:$	
$/*3*/\ (A, S, A, B; z_{na}, z_{enc})^{l_{11}}.$	
decrypt z_{enc} as $\{A; z_{nb}\}^{l_{12}}_{K_B}$ in	
$/*4*/\ (\nu\ K)$	
$\langle S, A, \{z_{na}, B, K, \{z_{nb}, A, K\}_{K_B}\}_{K_A}\rangle.0$	

$(A) \in \rho(y^1_a)$	$(B) \in \rho(z_\bullet)$	$(n_\bullet) \in \rho(x^1_k)$
$(\{A, N^1_B\}_{K_B}) \in \rho(x^1_{enc})$	$(N^1_A) \in \rho(z_\bullet)$	$\langle A, B, N^1_A, B, n_\bullet, N^2_A\rangle \in \kappa$
$(N^2_A) \in \rho(x^1_y)$	$\langle A, B, N^1_A, B, n_\bullet\rangle \in \kappa$	$(N^1_A, B, n_\bullet, N^2_A) \in \rho(x^1_z)$
$\langle A, B, N^1_A, \{A, N^1_B\}_{K_B}\rangle \in \kappa$	$(N^1_A, B, n_\bullet) \in \rho(y^2_a)$	$l_6 \in \psi$

Table 3. Amended Needham-Schroeder protocol: specification (above); some analysis results (below).

sends it out to the network. A reads this message, binds the variable x^1_{enc} to the value $(\{A, N^1_B\}_{K_B})$, as reflected by $(\{A, N^1_B\}_{K_B}) \in \rho(x^1_{enc})$; then, in step 3, it generates N^1_A and sends it to S as a plain-text, together with x^1_{enc} as predicted by $\langle A, S, N^1_A, \{A, N^1_B\}_{K_B}\rangle \in \kappa$, and so on.

Moreover, the non-empty error component ψ shows that a many-to-one binding may happen in the decryption with label l_6 and therefore suggests a possible complex type confusion leading to a complex type flaw attack.

By studying the contents of the analysis components ρ and κ, we can rebuild the attack sequence. Since $\langle A, S, N^1_A, \{A, N^1_B\}_{K_B}\rangle \in \kappa$, then $(N^1_A) \in \rho(z_\bullet)$. This corresponds to the fact that the attacker, able to intercept messages on the net, can learn N^1_A. The entry $\langle A, B, N^1_A, B, n_\bullet\rangle \in \kappa$ reflects that the attacker is able to constructs and sends to A a new message $(N^1_A, B, n_\bullet)$ to initiate the second instance, where $(n_\bullet)$ is within its knowledge. The entry $(N^1_A, B, n_\bullet)$ in $\rho(y^2_a)$ corresponds to the fact A receives this message, by binding y^2_a to the value $(N^1_A, B, n_\bullet)$. This is a many-to-one binding, detected by the analysis, as reported by the error component: $l_6 \in \psi$. Afterwards, A encrypts what she has received with a new nonce N^2_A and sends it out, as indicated by $\langle A, B, N^1_A, B, n_\bullet, N^2_A\rangle \in \kappa$. The attacker replays this to A, who takes it as the message from S in the step 4 of the first instance $((N^1_A, B, n_\bullet, N^2_A) \in \rho(x^1_z))$. The entry $(n_\bullet) \in \rho(x^1_k)$ reflects that in decrypting message 4, A binds x^1_k to the concatenation of values $(n_\bullet)$ to be used as the session key. After completing the challenge and response in step 6 and 7, A then believes she is talking to B using the session key K, but indeed she is talking to the attacker using $(n_\bullet)$ as the new key. This exactly corresponds to the complex type flaw attack shown before.

The protocol can be modified such that each principal use different keys for different roles, i.e. all the principals taking the initiator's role A_i share a master key K_A^i with the server and all the principals taking the responder's role B_j share K_B^j with the server. In this case, the analysis holds for $\psi = \emptyset$ and thereby it guarantees the absence of complex type confusions attacks.

Here, only two sessions are taken into account. However, as in [5], the protocol can be modelled in a way that multiple principals are participating in the protocol at the same time and therefore mimic the scenario that several sessions are running together. Due to space limitation, further details are skipped here.

4 Conclusion

We say that a complex type confusion attack happens when a concatenation of fields in a message is interpreted as a single field. This kind of attack is not easy to deal with in a process algebraic setting, because message specifications are given at a high level: the focus is on their contents and not on their structure. In this paper, we extended the notation of variable binding in the process calculus LYSA from one-to-one to many-to-one binding, thus making it easier to model the scenario where a list of fields is confused with a single field. The semantics of the extended LYSA makes use of a reference monitor to capture the possible many-to-one bindings at run time. We mechanise the search for complex type confusions by defining a Control Flow Analysis for the extended LYSA calculus. It checks at each input and decryption place whether a many-to-one binding may happen. The analysis ensures that, if no such binding is possible, then the process is not subject to complex type flaw attacks at run time. As far as the attacker is concerned, we adopted the standard notion from Dolev-Yao threat model [10], and we enriched it to deal with the new kind of variable binding.

We applied our Control Flow Analysis to the Amended Needham-Schroeder Protocol (as shown in Section 3), to Otway-Rees [23], Yahalom [8] (not reported, because of lack of space). It has confirmed that we can successfully detect the complex type confusions leading to type flaw attacks on those protocols. This detection is done in a purely mechanical and static way. The analysis also confirms the complex type flaw attacks on a version of the Neuman-Stubblebine protocol, found in [27].

The technique presented here is for detecting *complex* type flaw attacks only. *Simple* type flaw attacks, i.e. two single fields of different types are confused with each other, not considered here, have been addressed instead in [6], under a framework similar to the present one. Besides the type tags, several kinds of annotations for LYSA has been developed for validating various security properties, e.g. confidentiality [12], freshness [11] and message authentication [5]. They can be easily combined with the annotations introduced here, thus giving more comprehensive results.

Usual formal frameworks for the verification of security protocols need to be suitably extended for modelling complex type flaw confusions. Extensions include the possibility to decompose and rebuild message components, that we obtain by playing with single, general and flattened values. In [7], for instance, the author uses a concatenation operator to glue together different components in messages. The approach is based on linear logic and it is capable of finding the complex type flaw attack on the Otway-Rees protocol. Meadows [17, 18] approach is more general and can address also even more complex type confusions, e.g. those due to the confusion between pieces of fields of one type with pieces of another. The author, using the GDOI protocol as running example, develops a model of types that assumes differing capacities for checking types by principals. Moreover, Meadows presents a procedure to determine whether the types of two messages can be confused, then also evaluating the probability of possible misinterpretations. In [15], using the AVISPA [3] model checking tool, type flaw attacks of the GDOI protocol are captured. Furthermore, by using the Object-Z schema calculus [28, 14] the authors verify the attacks at a lower level and find which are the low-level assumptions that lead to the attacks and which are the requirements that prevent them. Type confusions are captured also in [19], by using an efficient Prolog based constraint solver. The above settings, especially the ones in [17, 18, 15], are more general than our, e.g. they capture more involved kinds of type confusions in a complex setting, like the one of the GDOI protocol. Our work represents a first step in modelling lower level features of protocol specifications in a process algebraic setting, like the ones that lead to type confusions. The idea is to only perform the refinement of the high-level specifications necessary to capture the low-level feature of interest. Our control flow analysis procedure always guarantees termination, even though it only offers an approximation of protocols behaviour and of their dynamic properties. Due to the nature of the over-approximation, false positives may happen, as some of the many-to-one bindings are not necessary leading to a complex type flaw attack. By taking the bit length of each field into account, i.e. using them as thresholds like in [25, 26], may greatly reduce the number of false positives. For example, assuming that a nonce, N, is always represented by 8 bits, an agent's name, A, by 8 bits, and a key, K, by 12 bits, the concatenation of A and N will be never confused with K and therefore it can be ruled out. In this paper we focussed on a particular kind of confusions, leaving other kind of type confusions for future work. We could use one-to-many bindings to deal with the case in which pieces of fields are confused with each other. We also would like to move to the multi-protocol setting, where the assumptions adopted in each protocol could be different, but messages could be easily confused, typically, because of the re-use of keys.

References

1. M. Abadi, C. Fournet. Mobile values, new names, and secure communication. *POPL*: 104-115,2001.
2. M. Abadi and A. D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1): 1-70, 1999.

3. A. Armando et Al. The AVISPA tool for the automated validation of internet security protocols and applications. *In Proc. of the 17th International Conference on Computer-Aided Verification (CAV)*, LNCS 3576, pp. 281-285, Springer, 2005.
4. C. Bodei,H. Gao, and P. Degano. A Formal Analysis of Complex Type Flaw Attacks on Security Protocols. TR-08-03, Pisa University.
5. C. Bodei, M. Buchholtz, P. Degano, F. Nielson and H. Riis Nielson. Static Validation of Security Protocols. *Journal of Computer Security*, 13(3): 347 - 390, 2005.
6. C. Bodei, P. Degano, H. Gao and L. Brodo. Detecting and Preventing Type Flaws: a Control Flow Analysis with tags. *In Proc. of 5th International Workshop on Security Issues in Concurrency (SecCO)*, ENTCS, 2007.
7. M. Bozzano. A Logic-Based Approach to Model Checking of Parameterized and Infinite-State Systems. PhD Thesis, DISI, University of Genova, 2002.
8. M. Burrows, M. Abadi and R. Needham. A Logic of Authentication. *TR 39*, Digital Systems Research Center, February, 1989.
9. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0, 1997. http://www.cs.york.ac.uk/~jac/papers/drareviewps.ps.
10. D. Dolev and A. C. Yao. On the Security of Public Key Protocols. IEEE TIT, IT-29(12):198-208, 1983.
11. H. Gao, C. Bodei, P. Degano, and H. Riis Nielson. A Formal Analysis for Capturing Replay Attacks in Cryptographic Protocols. *In Proc. of the 12th Annual Asian Computing Science Conference (ASIAN)*, LNCS 4846: 150-165, Springer, 2007.
12. H. Gao and H. Riis Nielson. Analysis of LySa calculus with explicit confidentiality annotations. *In Proc. of Advanced Information Networking and Applications (AINA)*, IEEE Computer Society, 2005.
13. J. Heather, G. Lowe and S. Schneider. How to prevent type flaw attacks on security protocols. *In Proc. of the 13th Computer Security Foundations Workshop (CSFW)*, IEEE Computer Society Press, 2000.
14. B. W. Long. Formal verification of a type flaw attack on a security protocol using Object-Z. *In 4th International Conference of B and Z Users, ZB*, LNCS 3455: 319–333, 2005.
15. B. W. Long, Colin J. Fidge David A. Carrington. Cross-layer verification of type flaw attacks on security protocols. *In Proc. of the 30th Australasian conference on Computer science* Volume 62, 2007.
16. C. Meadows. Analyzing the Needham-Schroeder public key protocol: A comparison of two approaches. *In Proc. of European Symposium on Research in Computer Security.* Springer, 2006.
17. C. Meadows. Identifying potential type confusion in authenticated messages. *In Proc. of Workshop on Foundation of Computer Security (FCS)*, pp. 75-84, 2002. Copenhagen, Denmark, DIKU TR 02/12.
18. C. Meadows. A procedure for verifying security against type confusion attacks. *In Proc. of the 16th Workshop on Foundation of Computer Security (CSFW)*, 2003.
19. J. Millen, V. Shmatikov. Constraint Solving for Bounded-Process Cryptographic Protocol Analysis. *ACM Conference on Computer and Communications Security*, 2001: 166-175.
20. R. Milner. Communicating and mobile systems: the π-calculus. Cambridge University Press, 1999.
21. F. Nielson, H. Riis Nielson and R.R. Hansen. Validating firewalls using flow logics. *Theor. Comput. Sci.* 283(2): 381-418, 2002.
22. F. Nielson, H. Seidl, and H. R. Nielson. A Succinct Solver for ALFP. *Nordic Journal of Computing*, 9:335-372, 2002.
23. D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8-10, 1987.
24. E. Snekkenes. Roles in cryptographic protocols. *In Proc. of the Computer Security Symposium on Research in Security and Privacy*, pp. 105-119. IEEE Computer Society Press, 1992.
25. S. Stubblebine and V. Gligor, On Message Integrity in Cryptographic Protocols, *IEEE Computer Society Symposium on Research in Security and Privacy*, 1992, pp. 85-104.
26. S. Stubblebine and V. Gligor, Protocol Design for Integrity Protection, *IEEE Computer Society Symposium on Research in Security and Privacy*, 1993, pp. 41-53.
27. P. Syverson and C. Meadows. Formal requirements for key distribution protocols. In *Advances in Cryptology - EUROCRYPT*, LNCS 950: 320-331. Springer, 1994.
28. J.B. Wordsworth. Software development with Z - A practical approach to formal methods in software engineering. *International Computer Science Series.* Addison-Wesley Publishers Ltd., London, 1992.