International Journal of Information Security manuscript No.
(will be inserted by the editor)

A Calculus for Control Flow Analysis of Security Protocols

Mikael Buchholtz, Hanne Riis Nielson, Flemming Nielson⋆

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

Received: date / Revised version: date

Abstract The design of a process calculus for analysing security protocols is governed by three factors: how to express the security protocol in a precise and faithful manner, how to accommodate the variety of attack scenarios, and how to utilise the strengths (and limit the weaknesses) of the underlying analysis methodology. We pursue an analysis methodology based on control flow analysis in flow logic style and we have previously shown its ability to analyse a variety of security protocols [7]. This paper develops a calculus, LySa$^{\mathrm{NS}}$, that allows for much greater control and clarity in the description of attack scenarios, that gives a more flexible format for expressing protocols, and that at the same time allows to circumvent some of the "false positives" arising in [7].

1 Introduction

Security protocols are used to establish secure communication in untrusted computer networks. A security protocol describes a sequence of messages, which should be exchanged between network nodes, or principals, in order to achieve some security goal intended by the protocol. Such protocols usually rely on cryptographic techniques to prevent undesired tampering with messages and are sometimes called cryptographic protocols.

Figure 1 depicts a typical scenario for the use of a security protocol. Here a number of principals, $I_1, I_2, \ldots$, are connected to a common network, known as the *ether*, over which they exchange messages using the protocol. Apart from these principals, there may also be trusted third parties or servers, S, malicious attackers, M, and other agents that have access to the ether.

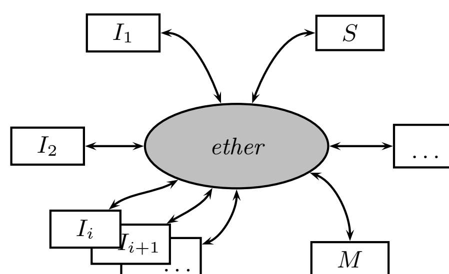

Fig. 1 A network scenario making use of a security protocol.

We are interested in ensuring that the security goals are met even in a "worst-case scenario". Such a scenario would for example allow an arbitrary number of principals to be running an arbitrary number of instances of the protocol at the same time through a network populated by an arbitrary number of attackers. The setup depicted in Figure 1 is rather classical in that it has a fixed structure, and the attackers are operating as separate entities rather than being embedded in the principals.

Our goal is to develop a calculus that will allow us to describe various classes of attack scenarios rather than having to rely on a fixed setup as the one in Figure 1. We may then study protocols e.g. together with an outside attacker or together with "dishonest principals" that appear to the rest of the system as genuine principals but contain code for implementing actions not expected of principals (as might be the result of a genuine principal being infected with a virus), or any combination of these attack scenarios.

In future work we also anticipate the need to consider dynamically changing networks in order to model future and emerging wireless technologies and mobile infrastructure. We believe that the foundations laid in this work will prove sufficiently flexible to deal also with these issues, e.g. by the incorporation of mobility primitives,

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

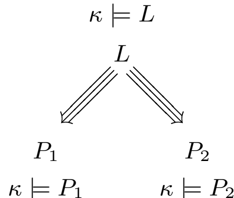

Fig. 2 Instantiation, $L \Rrightarrow P_i$, and analysis, $\kappa \models L$ and $\kappa \models P_i$ (for $i = 1, 2$). The analysis component κ stays invariant under the instantiation.

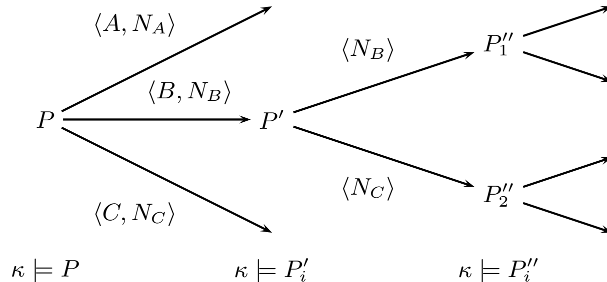

Fig. 3 Semantics, $P \to P'$, and analysis, $\kappa \models P$. The analysis component κ stays invariant under the semantic executions. (Here κ will include all the message sent on the network, i.e. $\{\langle A, N_A \rangle, \langle B, N_B \rangle, \langle C, N_C \rangle, \langle N_B \rangle, \langle N_C \rangle\} \subseteq \kappa$.)

but this development is well beyond the scope of the present paper.

1.1 Protocols and Attack Scenarios

The first step of our approach is to encode protocols in a suitable programming formalism in the form of a process calculus. To be a bit more precise the process calculus has two “levels”:

- The meta-level (to be denoted L) describes an overall system scenario that we have in mind, whereas
- the object-level (to be denoted P) describes a concrete system falling within a given system scenario.

Because of the ellipses $(\cdots)$ in Figure 1, the figure should be viewed as an informal way of expressing a meta-level process. The formal meta-level process will make clear a number of issues regarding the protocol as well as the places where the system is open to attack; hence this will be the step where most care is needed in modelling security protocols.

As part of our development we will formalise an instantiation relation that non-deterministically evolves a meta-level process into an object-level process ($L \Rrightarrow P$) as sketched in Figure 2; we will also formalise a semantics that expresses how object-level processes execute into each other ($P \to P'$) and as sketched in Figure 3 there may be many different executions of the protocol depending on the principals involved.

1.2 Control Flow Analysis

The second step of our approach amounts to carrying out an analysis of processes at both meta-level and object-level. For this we rely on techniques from control and data flow analysis [30] that allow us to make fully automatic analyses. As such, these techniques show promise not only as methods for general protocol analysis but also as techniques that may be used to validate actual implementations of security protocols.

To be more specific, the central piece of information that the analysis captures is the set of messages that are sent on the network. In the analysis they are collected in an analysis component called κ. Technically, the analysis is given as judgements of the form

$$\kappa \models L \qquad\qquad \kappa \models P$$

that describe that κ is a valid analysis of the meta-level process L and of the object-level process P, respectively. Typical analyses will contain more analysis components but they are not needed at the current level of exposition.

These judgements should be viewed as conditions for when the analysis information correctly describes the process; hence they are boolean valued predicates intended to give true. This is quite similar to *type checking*: given a type and a process does the type describe the process? At a later stage in the development we will then be able to rely on techniques from control and data flow analysis that allow us to make fully automatic analyses with a low computational complexity; hence we will be able to compute the best κ given L or P. This is quite similar to *type inference* (except that for some type systems the computational complexity is intractable).

We need to ensure that the analysis is semantically correct and there are two components to this. First we need to ensure that if κ describes a meta-level process L (i.e. $\kappa \models L$) and if the meta-level process can be instantiated to an object-level process P (i.e. $L \Rrightarrow P$), then κ also describes the object-level process P (i.e. $\kappa \models P$); this is depicted in Figure 2. Secondly, we need to ensure that if κ describes an object-level process P (i.e. $\kappa \models P$) and if the object-level process P executes into another object-level process P' (i.e. $P \to P'$) then κ also describes the object-level process P' (i.e. $\kappa \models P'$); this is illustrated in Figure 3.

Let us concentrate our explanation around the more familiar case of Figure 3 which goes under the name “subject reduction”. The idea is that we want to find one particular κ that describes the process during all steps of all executions. This is done by letting κ be a *conservative over-approximation* to the set of messages that are communicated on the network. Thus, κ will be a component that contains all the messages that are actually sent on the network but, possibly, also may include

additional messages even though they, in fact, will not be communicated. We often refer to values that "unnecessarily" end up in the analysis result as *false positives* and we say that an analysis is more precise than another if it gives fewer false positives.

It is important to stress that the approximations are *conservative* in the sense that they cannot give *false negatives*. That is, it will never be the case that the process communicates a message that does no show up in κ. In other words, the analysis always "errs on the safe side" and hence it is the absence of offending information in κ that will allow us to guarantee a security property of a protocol.

1.3 Security Properties

Analysis of security protocols amounts to ensuring certain security goals in all instances of an attack scenario. To give a simple and typical example, an attack scenario often takes the form of an attacker M running in parallel with the genuine part P of the system. Rather than writing $M \mid P$, where P is known but M is not, we shall make use of a meta-level process $L = (\bullet \mid P)$. The control flow analysis of L will then have to take care of all potential attackers M that may be placed for $\bullet$.

The traditional approach to this problem is to adapt the famous Dolev-Yao conditions [13] to the setting at hand. The formal correctness of the Dolev-Yao conditions, as captured by the notion of "hardest attacker" [32,31,7], is that when κ describes $\bullet$ then it also describes any attacker M, i.e. $\kappa \models \bullet$ implies $\kappa \models M$ (and that for some choice of M this is actually a biimplication). This resembles what Cervesato shows in [11] though his approach works directly on *the semantics* of Multiset Rewriting systems (MSR) while our approach uses *a finitary analysis* that additionally leads to a decidable validation procedure.

The next step then amounts to utilising the control flow information for validating the security goals of the protocols. For properties related to *secrecy* this is rather straightforward:

1. Partition the values into secret and public.
2. Calculate κ for the protocol together with the Dolev-Yao condition as indicated by the meta-level process describing the protocol scenario.
3. Secrecy is guaranteed if no secret values are in clear in κ.

For properties related to *authenticity* and *integrity* a little more work is needed. In [7] annotations on the origins and destinations of encryptions were used to ensure authenticity issues; similar annotations can be added to the calculus developed here. Also there is the prospect of developing more flow-dependent analyses that more directly deal with transaction based authentication; but this is beyond the scope of the present paper.

2 State of the Art

The literature contains very many approaches to the modelling and analysis of cryptographic protocols using notations ranging from process calculi to domain specific languages and using techniques ranging from logical theories over type systems to flow analysis. It is hardly possible to give full credit to all of these approaches in a short paper and in this overview we limit our attention to some of the more influential developments based on process calculi, type systems, and control flow analysis.

2.1 Calculi for Security Protocols

The use of process calculi for security protocol analysis initially proved its worth when Lowe [21] found a new attack on Needham-Schroeder's public key protocol [28] by encoding and analysing it in CSP. Following this initial work numerous other calculi have been used for modelling and analysing security protocols. For example:

- VSPA [17] is a value passing variant of CCS [26] extended to incorporate two security levels.
- The Spi-calculus [4] extends the π-calculus [27] with cryptographic primitives.
- The Applied π-calculus [3] extends the π-calculus with a general notion of terms.
- LySa [7] is a variant of the Spi-calculus with pattern matching.

An obvious strength of process calculi for modelling security protocols is their inherent handling of concurrency and communication. Usually, communication takes place on named channels and one or more designated channels are used to model the ether to which attackers have access. All other channels may be used freely for local or secret communication or for "signals" used in analysis. The use of additional channels requires a small argument to ensure that these do not constitute covert channels. In LySa, on the other hand, there is only *one global ether* for communication and, hence, local communication is excluded.

Though concurrency and communication consistently model principals connected to a common network none of the calculi directly incorporate a notion of principals. In particular, there is no syntactic way to distinguish local, concurrent computations taking place at a principal from global communication on the ether.

Cryptographic primitives, such as encryption, signatures, and hash values, are important ingredients for modelling security protocols. In the calculi they give rise to *terms* where constants and variables are composed using function symbols. For example, an encrypted message m under a key k may be modelled as $E(k, m)$, or the hash value of v may be modelled as $H(v)$. VSPA and CSP allow terms to be composed and *decomposed* freely. This means that attackers have to behave in a

disciplined manner so they will not decompose a message such as $E(k, m)$ (i.e. decrypt it) without knowing the key. In contrast, all other calculi handles this problem by making such unwanted manipulations of terms semantically impossible.

The Applied π-calculus, for example, introduces a general notion of terms and equips the term algebra with an equational theory. This equivalence relation is used to semantically describe the interdependency of various function symbols and an attacker is free to do everything within the limitations of this semantics. Quite similar is the calculus in [2] where the interdependency of function symbols is described as constructors and destructors.

In the Spi-calculus and in LySa a fixed set of term constructs are chosen and their meaning is "hard-wired" into the semantics. Since there are no additional requirements on the terms they are elements of a free term algebra, and this simplifies the analysis of the calculus.

2.2 Analysis of Security Protocols

In previous work [7,8], we have applied techniques from control flow analysis to get an *automated* validation procedure for security protocols. Other automated analysis techniques that work with language based formalisms have objectives quite similar to ours.

For example, the type systems [1,19] for the Spi-calculus can also be automated and *type checking* may be done in polynomial time in the size of the process. However, *type inference* seems significantly more expensive (i.e. to take exponential time). In comparison, our approach is more in the flavour of type inference while retaining a polynomial worst-case complexity.

The general idea in these type system approaches is that any process that type checks will have a particular property. Similarly to our approach these techniques are also semantically incomplete, so there exists processes with this nice property that may fail to type check. For example, in [19] protocols must use a certain kind of nonce handshake to type check though there are plenty of other ways to attain the desired authenticity property.

Another type system based approach from [2] is shown to correspond to protocol analysis based on Horn-clauses [5,6]. This may be seen as an implementation of type inference that apparently terminates quickly on many examples, though termination in general is not guaranteed. Interestingly, our implementations [33,7] also use Horn-clauses though they are guaranteed to terminate in polynomial time.

Many classical analysis techniques for process calculi rely on relating processes by means of equivalence or refinement relations. In manual approaches, reasoning with such relations can be used to show quite strong security properties, e.g. [4,18]. Equivalence and refinement relations are also central to a number of automated approaches.

t	$::=$	n	Name
	$\mid$	x	Variable
	$\mid$	$\mathsf{T}(t_1, \cdots, t_k)$	Tuple
	$\mid$	$\ldots$	

Table 1 Some of the basic terms.

The analyses of CSP [21,37], for example, use *refinement relations* between processes to specify security properties. The refinements are automatically checked by state space exploration, and though it is only done for limited scenarios, with few principals, few runs, etc. the approach is frequently successful [24]. Information flow analysis [16] of VSPA is another example and here security properties are formulated in terms of process equivalences. It has been adapted to the analysis of security protocols [17,14] and gives an automatic validation procedure which is exponential in the size of the processes though a compositional algorithm [17] often behaves better in practice.

3 Principled Language Design

We are now ready to embark on the design of LySa$^{\text{NS}}$ that addresses the three main goals of this paper as announced in the abstract: to express the security protocol in a precise and faithful manner, to accommodate the variety of attack scenarios, and to utilise the strengths (and limit the weaknesses) of the underlying analysis methodology.

In this section, we describe the basic building blocks of LySa$^{\text{NS}}$ and the motivation for including them; this draws upon our needs as presented in Section 1 and existing ingredients as surveyed in Section 2. Some features of the calculus, like parallel composition and recursion, are sufficiently standard that we will postpone their treatment to Section 4, where the formal syntax and semantics is presented. In Section 5 we shall justify that the last of our design goals is meet and there we briefly outline an analysis of LySa$^{\text{NS}}$. Appendix A contains a number of elaborate examples of the use of LySa$^{\text{NS}}$, Appendix B contains the formal definition of a notion of well-formed processes, while Appendices C and D contain additional formal definitions for Sections 4 and 5 and are published in electronic form.

3.1 Terms and Patterns

To model cryptographic primitives, we use terms, t, which consist of names, n, variables, x, and composite terms as shown in Table 1. As in the Spi-calculus and in LySa we use a fixed number of designated function symbols (or constructors) for constructing composite terms; we shall see in Section 3.3 that this suffices for directly capturing the main security mechanisms: shared key encryption,

p	$::=$	n	Matches name
	$\mid$	x	Matches value of variable
	$\mid$	$\mathsf{T}(p_1, \cdots, p_k)$	Matches tuple when $p_1, \cdots, p_k$ match components
	$\mid$	$_$	Matches anything
	$\mid$	$p\%x$	Binds variable x when p matches
	$\mid$	$\ldots$	

Table 2 Some of the basic patterns.

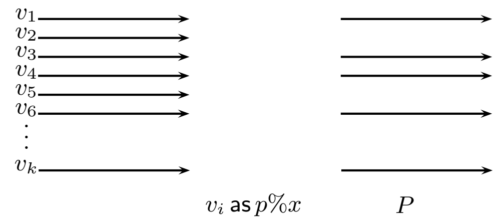

Fig. 4 Pattern matching only succeeds on some values; here it fails on v_2 and v_5.

public key encryption, private key signatures and hash values. For now we focus on the simple case of tuples which are constructed using the function symbol T, i.e. $\mathsf{T}(t_1, \cdots, t_k)$ represents a tuple of the k terms $t_1, \cdots, t_k$.

In order to decompose a composite term and extract the individual components we shall use pattern matching since it proved to interact well with the analysis of LySa [7]. This is an alternative to the explicit use of destructors and is by no means novel when considering security protocols (see e.g. [23,18]). The pattern matching mechanism suggested here is, however, much more general than what has previously been suggested in process calculi and it has been carefully tailored to assist our flow analysis as we will see in Sections 3.2 and 5. We shall write

$$t \,\mathsf{as}\, p.\ P$$

to denote the attempted matching of a term t against a pattern p before continuing with the remainder P of the computation. As usual there are two aspects of pattern matching: (1) it is a test that only succeeds on terms of a suitable form and hence may prevent P from being executed, and (2) it is an extraction operation that maps constituent components to variables that may be used in P. We shall decide to follow the tradition of most process calculi and refrain from using a notation like $t \,\mathsf{as}\, P \,\mathsf{or}\, Q$ that includes an explicit clause, Q, for error handling.

The first aspect of matching is handled by effectively allowing terms to be used as patterns as shown in Table 2. For example, the pattern n matches the term n while the pattern $\mathsf{T}(n_1, n_2)$ matches exactly the pair $\mathsf{T}(n_1, n_2)$. Furthermore, a pattern may be an applied occurrence of a variable x and only matches the value that the variable is bound to. For example, the pattern matching

$$n \,\mathsf{as}\, x$$

will succeed if x is bound to the name n at run-time and it will fail otherwise. A pattern can also be a wild-card, written as underscore '_' that matches any term. Hence, the matching $t \,\mathsf{as}\, _$ always succeeds. As we shall see in Section 3.3 we will need to be careful with the wild-card pattern when dealing with the cryptographic primitives.

The second aspect of matching is handled by letting patterns contain defining occurrences of variables. Inspired by the notation of [23] we write $p\%x$ for a pattern that on success will bind the matched value to the variable x. For example, the pattern matching

$$n \,\mathsf{as}\, _\%x$$

will bind n to the variable x because the wild-card pattern always succeeds. The %-notation syntactically distinguishes binding occurrences of variables from applied occurrences of variables and its scope extends as far to the right as possible.

It is clear from Tables 1 and 2 that for each composite term there is a corresponding composite pattern. The composite patterns provide a mechanism for decomposing terms by demanding subpatterns to be matched. Thus a tuple $\mathsf{T}(t_1, \cdots, t_k)$ successfully matches the pattern $\mathsf{T}(p_1, \cdots, p_k)$ whenever all the terms $t_1, \cdots, t_k$ successfully match the respective subpatterns $p_1, \cdots, p_k$. As an example, the matching

$$\mathsf{T}(n_1, n_2) \,\mathsf{as}\, \mathsf{T}(n_1, _\%x)$$

succeeds since the first component of $\mathsf{T}(n_1, n_2)$ matches the value n_1. At the same time the tuple will be decomposed and the variable x will be bound to the second component, i.e. x will be bound to n_2.

3.2 Pattern Matching and the Analysis

In the explanation above we have made it clear that in some runs of the matching construct, $t \,\mathsf{as}\, p.P$, the test will succeed and the process P will be executed; in others the test will fail and the process P will be prevented from executing.

Turning to the analysis we do not consider a single run of the matching construct but rather *all runs* at the same time as explained in Section 1.2. If there is at least one value for which the test succeeds, then the process P following the pattern matching must be analysed as well. However, pattern matching is also used to bind values to variables. This means that in addition to the network component κ we shall need an environment ρ for keeping track of which values may be bound to what variables. This component should contain all the successful bindings to the variables and, in the interest of a precise analysis, we should avoid to include bindings from unsuccessful matchings. For example, when analysing the situation of Figure 4 we would expect that the analysis

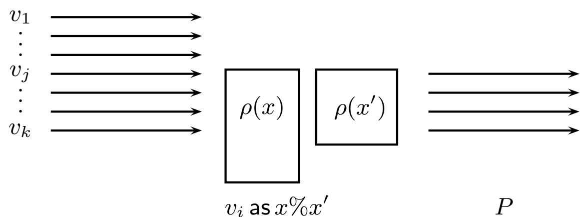

Fig. 5 The analysis may allow a more precise set of values for x' over those of x in the context of analysing the pattern $x\%x'$.

information $\rho(x)$ associated with x contains v_1, v_3, v_4, v_6 etc. but we would prefer it not to include v_2 and v_5.

To understand this point it is useful to consider the pattern $x\%x'$. Semantically it will bind x' to the value that matches x; in other words, x' will be equal to x in all successful matchings and hence it might seem that the simpler pattern x would serve equally well. It is when we consider the analysis that we see the merits of using the pattern $x\%x'$. So suppose that the values $v_1, \cdots, v_k$ are matched against x in different runs but that the matching only succeeds for the values $v_j, \cdots, v_k$ as illustrated by Figure 5. Then $\rho(x)$ will contain the values $v_j, \cdots, v_k$ but it may also contain other values like $v_{k+1}, \cdots, v_n$. However, the analysis can be constructed such that $\rho(x')$ only contains the values that successfully match at this particular point and hence $\rho(x')$ may be a strict subset of $\rho(x)$; this will allow us to obtain a considerably more precise analysis.

3.3 Cryptographic Primitives

We now return to the interesting problem of how best to represent the cryptographic primitives in as direct a way as possible. We shall consider shared key and public key encryption, private key signatures, and hashing. In each case we shall define the syntax of terms (to be summarised in Table 3) and we shall take care to define the syntax of patterns in such a way that we capture the security mechanisms in an appropriate manner (to be summarised in Table 4).

Shared key cryptography (or symmetric key cryptography) is used to encrypt values under a key, which is shared in the sense that it is used both for encryption and decryption. We model an encrypted value as the composite term

$$\mathsf{E}_{t_0}(t_1, \cdots, t_k)$$

meaning that the terms $t_1, \cdots, t_k$ have been encrypted under the key t_0.

Example 1 In the Needham-Schroeder symmetric key protocol (see [28] or Appendix A.3) the server S constructs an encrypted message to be sent to A to assist him in establishing the desired communication link with B. In our calculus this encrypted message can be written as

$$\mathsf{E}_{K_A}(N_A, B, K_{AB}, \mathsf{E}_{K_B}(K_{AB}, A))$$

where K_A is the master key shared between S and A, K_B is the master key shared between S and B, K_{AB} is the newly constructed session key to be shared between A and B, and N_A is a nonce to guard against replay attacks. □

The secrecy of values encrypted under a shared key relies on the assumption that finding any one of t_0, or $t_1, \cdots, t_k$ from $\mathsf{E}_{t_0}(t_1, \cdots, t_k)$ is a *computationally hard* problem unless the key t_0 is known. In our approach we shall model computationally hard problems as being *semantically impossible*; in other words we shall be using perfect cryptography.

Syntactically, decryption amounts to pattern matching against a composite pattern of the form

$$\mathsf{E}_{p_0}(p_1, \cdots, p_k)$$

Semantically, it is ensured that the pattern matching only succeeds if t_0 matches the pattern p_0. To correctly model perfect cryptography it is therefore essential that we *restrict* the syntax of the pattern p_0 by demanding that it contains *no wild-card patterns*. As an example, if p_0 was the pattern $_\%x$ then not only would decryption always succeed but the key would be bound to x and could be used also for subsequent encryption.

Example 2 Continuing Example 1 the encrypted message can be decrypted by the recipient using the pattern:

$$\mathsf{E}_{K_A}(N_A, B, _\%x_K, _\%x)$$

The pattern introduces variables x_K and x for the components not already known and hence x_K will be bound to the session key K_{AB} and x will bound to the encrypted message $\mathsf{E}_{K_B}(K_{AB}, A)$. □

Public key cryptography (or asymmetric cryptography) is used to encrypt values under a public key, m^+, whereas decryption is performed with respect to a private key, m^-. We model an encrypted value as the composite term

$$\mathsf{P}_{m^+}(t_1, \cdots, t_k)$$

meaning that the terms $t_1, \cdots, t_k$ have been encrypted under the public key m^+.

The secrecy of values encrypted under a public key relies on the assumption that finding any one of m^+ or $t_1, \cdots, t_k$ from $\mathsf{P}_{m^+}(t_1, \cdots, t_k)$ is a computationally hard problem unless the private key m^- is known.[1] Additionally

[1] Some implementations of public key cryptography include hints about the public key m^+ in the ciphertext. To model such implementations one may want to include m^+ in clear along with the ciphertext.

it is assumed that finding the private key m^- is computationally hard even if the public key m^+ is already known. As above our approach will be based on perfect cryptography.

Syntactically, decryption amounts to pattern matching against a composite pattern of the form:

$$\mathsf{P}_{m^-}(p_1, \cdots, p_k)$$

Semantically, it is ensured that the pattern matching only succeeds if the public encryption key m^+ is in bijective correspondence with the private decryption key m^-. Semantically, we only allow a bijective correspondence to exist between pairs of names such as m^+ and m^-. Thus, all public and private keys will, in effect, be names and in Section 3.4 we describe how such a correspondence may be introduced.

We allow a slightly more general syntax where terms have the form $\mathsf{P}_{t_0}(t_1, \cdots, t_k)$ and patterns have the form $\mathsf{P}_{p_0}(p_1, \cdots, p_k)$ and we again have to restrict the pattern p_0 so that it contains no wild-card patterns. Since all key pairs consist of names, however, we will semantically ensure that no decryption can succeed if composite terms are used for keys.

Example 3 Let us consider one of the messages exchanged in the Needham-Schroeder public key protocol [28] given in full in Appendix A.4. The principal A sends a message to B encrypted with B's public key K_B^+:

$$\mathsf{P}_{K_B^+}(N_A, A)$$

To decrypt the message B can use the pattern

$$\mathsf{P}_{K_B^-}(_\%x_{N_A}, _\%x_A)$$

and the variables x_{N_A} and x_A will be bound to the nonce created by A and the name of A. □

Signatures. Sometimes digital signatures are presented as the dual notion of public key encryption: one uses the private key for encryption and the public key for decryption. In our view this is an oversimplification due to the symmetry between encryption and decryption in the RSA approach [36] and we observe that this symmetry is indeed not inherent in the ElGamal [15] approach.

So we shall decide to be very careful in modelling digital signatures in the proper abstract way. We write

$$\mathsf{S}_{m^-}(t_1, \cdots, t_k)$$

for the sequence $t_1, \cdots, t_k$ of messages signed under the private key m^-.

The corresponding pattern used for checking the signature is

$$\mathsf{S}_{p_0}(p_1, \cdots p_k)$$

Conceptually, this should be viewed as a test that only succeeds when the correct signature has been used in which case the content of the signed message may be decomposed. It is important to stress that knowledge of the key is *not* required in order to learn the content of a message. For example, in

$$\mathsf{S}_{m^-}(n) \;\mathsf{as}\; \mathsf{S}_{_}(_\%x)$$

the name n is bound to x even though it was signed under a private key, m^-, for which the public key, m^+, was unknown. On the other hand, even though the pattern matching succeeds, it does not convey any trust in the authenticity of the message.

To obtain trust in the authenticity of the message, knowledge of the key is essential. For example, in

$$\mathsf{S}_{m^-}(n) \;\mathsf{as}\; \mathsf{S}_{m^+}(_\%x)$$

we not only get access to the contents of the message but get the additional assurance that the message was signed with the private key, m^-, corresponding to the public key, m^+, known by us. Thus if the owner of the key pair (m^-, m^+) has exercised due care in only making the public key known to others we can indeed be sure of the authenticity of the message.

At the syntactic level this means that the pattern p_0 allowed in signature verification is more permissive than for public key cryptography. In particular we shall have to allow the pattern to be a wild-card pattern. However, there are still restrictions that need to be enforced, such as disallowing a pattern like $_\%x$, because it would allow us to learn a new key.

Example 4 Let us return to the Needham-Schroeder public key protocol of Example 3. In other messages the principal A obtains its knowledge about B's public key through a signed message sent by the server:

$$\mathsf{S}_{K_S^-}(K_B^+, B)$$

The principal A can extract the public key using the pattern

$$\mathsf{S}_{K_S^+}(_\%x_{K_B}, B)$$

and A will then be entitled to trust that x_{K_B} is bound to B's public key and that only B will be able to decrypt a message of the form $\mathsf{P}_{x_{K_B}}(N_A, A)$. □

Cryptographic hash values. We shall use the term

$$\mathsf{H}(t_1, \cdots, t_k)$$

for the one-way cryptographic hash value calculated from the sequence of terms $t_1, \cdots, t_k$. Correspondingly, there is a pattern

$$\mathsf{H}(p_1, \cdots, p_k)$$

that allows to check hash values.

However, the very idea of a cryptographic hash value is that it is "one-way" i.e. that it should not be possible to learn the parts from which it is composed. This once more calls for demanding that patterns must not include a wild-card — this time for all the subpatterns $p_1, \cdots, p_k$.

Example 5 Consider a version of the Otway-Rees protocol [35] based on hashing [25] (given in full in Appendix A.5). Here the server S sends a message to principal B containing among others the two messages:

$$\cdots, \mathsf{E}_{K_B}(K_{AB}),\ \mathsf{H}(K_B, N_B, A, \mathsf{E}_{K_B}(K_{AB})), \cdots$$

At this point in the protocol, B already has knowledge of N_B and A and of course it also knows the master key K_B so we can use the pattern

$$\cdots, \mathsf{E}_{K_B}(_\%x_{K_{AB}})\%x,\ \mathsf{H}(K_B, N_B, A, x), \cdots$$

to extract not only the session key (in the variable $x_{K_{AB}}$) but also to check on the hash value (using the encrypted value $\mathsf{E}_{K_B}(K_{AB})$ bound to x). □

3.4 Names and Name Generation

All keys, whether private, public or shared, must be introduced in such a way that we can control which part of the system that has initial knowledge about the keys. Following the π-calculus tradition, we shall use restrictions for this. So $(\nu\, m)P$ may be used to construct a new fresh symmetric key m that is initially only known in P.

Private and public keys are modelled in the calculus as two names m^- and m^+, respectively, and we will need to ensure that there is a clear bijective correspondence between them. Hence we introduce a new kind of restriction operator $(\nu^{\pm} m)\, P$ that introduces the *two* names m^+ and m^- in the process P at the same time.

Semantically speaking, all names in the calculus consist of a *base name*, m, and a *tag*, τ, and is typically written as m^τ. The base name is simply an identifier such as S, K_A, or pk_B from some infinite set. On the other hand, there are finitely many tags and they are used to describe what the name may be used for: if the name is a public key then it will be tagged with $+$, i.e. it will be a name m^+, while it will be tagged with $-$ if it is a private key. Since all names need a tag, we introduce the tag ε that will be used on names that are neither a public nor a private key. Names may be α-renamed but this must be done in such a way that only the base name is changed whereas the tag is unchanged. However, the details of the tagging scheme should not distract us when presenting the overall design of our calculus. We therefore write names as m rather than m^τ when the choice of tag is not crucial.

The restriction operator $(\nu^{\pm} m)\, P$, thus, introduces two names with the same base name m but with different tags. Similarly to the π-calculus, the semantics will ensure that the names cannot be forged and they are therefore suitable to be used as keys. The only other restriction operator allowed is $(\nu^{\varepsilon} m)\, P$. The fact that the sets of tags $\{+, -\}$ and $\{\varepsilon\}$ used by the two restriction operators are disjoint turns out to simplify some of the more technical parts of the development and for this reason we do not want to allow more permissive forms of restriction.

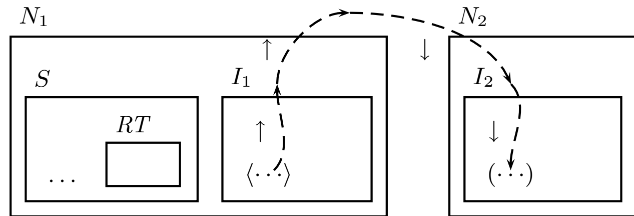

Fig. 6 Principal I_1 sends a message to principal I_2 situated in another sub-network.

3.5 Principals and Communication

The notion of a principal is a central aspect of security protocols and should consequently also be a central aspect in any formalism designed to model such protocols. As noted in Section 2 other process calculi, however, neglect having a clear notion of a principal as a distinct entity in the calculus. We instead choose to model principals explicitly by introducing a notion of *a boundary* as known from Mobile Ambients [10]. A principal named n that is running the process P will be written as $n\,[\,P\,]$. Using this notation we can e.g. specify the scenario depicted in Figure 1 as the process

$$I_1\,[\,P_1\,] \mid I_2\,[\,P_2\,] \mid \cdots \mid S\,[\,P_S\,] \mid \cdots \mid M\,[\,P_M\,]$$

which clearly expresses that process P_1 is running as a principal called I_1 while the process P_S is running as a principal called S, etc.

Another example of network boundaries is illustrated on Figure 6 where two sub-networks N_1 and N_2 contain different principals. The network N_1, for example, contains a principal I_1 and a server S that, in turn, contains a routing table RT. In this way, the ambient boundaries are also able to describe various kinds of structure on a network.

Position Based Communication. Mobile Ambients [10] only allows communication to take place locally within a single ambient. This form of communication seems too restrictive for our purposes since we use ambients to represent principals and inter-principal communication is a main focus. One solution is to adopt a π-calculus notion of channels that offers great flexibility but in a way that does not respect the boundaries; indeed, what is a boundary worth if the boundary does not offer some degree of isolation? Instead we adopt *position based* communication that describes how messages are routed relatively to the current position. For example, on Figure 6 the principal I_1 might want to communicate with the principal I_2; given the topology the message would have to go up, up, down and down.

We shall use η for describing such a *communication path* (e.g. $\eta = \uparrow.\uparrow.N_2.I_2$) and in principle we could allow communication paths to be elements of a regular

language defined over directions and names. For our immediate goal of modelling security protocols in scenarios like the one in Figure 1 it suffices with just two types of communication path. One allows communication from a principal to one of its neighbours (including itself), written $\updownarrow$, and the other allows local communication within a principal, written $\circ$. Hence, in the sequel η will be either $\updownarrow$ or $\circ$. Examples of other communication directions used for other purposes may be found e.g. in Boxed Ambients [9].

Output works by sending sequences of terms, i.e. it is polyadic, and is written:

$$\langle t_1, \cdots, t_k \rangle^\eta$$

Input receives sequences of terms and binds variables accordingly. For this we use the variable binding mechanisms already introduced, namely pattern matching, and write it as:

$$(p_1, \cdots, p_k)^\eta$$

This is similar to, although much more flexible, than the approach taken in LySa.

Example 6 A server, such as the one in Figure 1, may have a database of all the keys, K_i, that it shares with the principles I_i. One way of coding this is as a process

$$\mathit{Database} = !\langle I_1, K_1 \rangle^\circ \mid !\langle I_2, K_2 \rangle^\circ \mid \ldots$$

that uses *local* communication to send pairs consisting of a principal name and the corresponding key; the use of ! denotes replication and means that the pairs of values can be sent as many times as necessary. Look-up in this database can then be done as local input such as $(I_1, _\%x^K)^\circ$ that will bind x^K to K_1 when placed in parallel with *Database*.

The encoding of databases uses local communication only and, hence, cannot be confused with the global communication used for the encoding of the messages from the protocol itself. □

In terms of security this means that our analysis of security protocols will always permit attacks due to communication (unless of course the right cryptographic techniques are used as part of the communication); with the advent of mobile and wireless infrastructures we believe this to be the better design over some of the alternatives surveyed in Section 2.

3.6 Attack Scenarios

All the features described above are concerned with modelling specific principals engaged in a particular run of the protocol and this constitutes the so-called *object-level* of the calculus. This addresses two of our design goals concerned with faithful modelling of protocols while having their analysis in mind. The remaining design goal is to accommodate the variety of different scenarios in which the protocol may be deployed and will be the topic of this section.

Scenarios. To describe scenarios, we introduce a *meta-level*. Meta-level processes, L, extend the object-level with a number of indexing constructs. First, it introduces indexing sets, S, declared by the construct:

$$\mathsf{let}\ X = S\ \mathsf{in}\ L$$

The indexing sets are allowed to be infinite and can then be used in a construct $(\nu^{\cdots}_{i \in X}\ m_i)$ for creating a family of new names and in a parallel construct $\|_{i \in X}\ L_i$ for creating a family of parallel processes. Further, all names and variables may be indexed with one or more indices.

This allows us to specify the network scenario of Figure 1 as follows (with $\mathbb{N}$ being the natural numbers):

$$\begin{array}{ll} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} & \bullet \\ & \mid (\nu^\varepsilon_{i \in X}\ I_i)\,(\nu^\varepsilon S) \\ & \quad (\|_{i \in X}\ I_i\,[\,P_i\,])\ \mid\ S\,[\,P_S\,] \end{array}$$

Here $\bullet$ denotes the presence of the attacker and we use $(\nu^\varepsilon_{i \in X}\ I_i)$ to create the names I_i of the principals whereas the object-level construct $(\nu^\varepsilon S)$ is used to create the name of the server. The attacker, the principals, and the server all run in parallel; the names of the principals and the server are initially only known to the honest part of the scenario.

We may model a variation of the scenario where for example I_0 is a dishonest principal. To do so we single out the treatment of I_0:

$$\begin{array}{ll} \mathsf{let}\ X = \mathbb{N} \setminus \{0\}\ \mathsf{in} & \bullet \\ & \mid (\nu^\varepsilon_{i \in X}\ I_i)\,(\nu^\varepsilon S) \\ & \quad I_0\,[\,P_0 \mid \bullet\,]\ \mid \\ & \quad (\|_{i \in X}\ I_i\,[\,P_i\,])\ \mid\ S\,[\,P_S\,] \end{array}$$

where again $\bullet$ indicates the potential actions of an attacker. This has far reaching consequences because now the names of all principals are known to the attacker and there is no way to control how the names will be used.

It should be clear that a meta-level process specifies an attack scenario i.e. it specifies all the different object-level processes in which the protocol may be deployed. Though there will typically be infinitely many object-level processes that fall within this scenario (there may e.g. be infinitely many combinations of principals that could use the protocol) each such object-level process will be finite.

To make this relationship clear we shall write $L \Rrightarrow P$ (L is instantiated by P) whenever the object-level process P is one of the finite processes falling within the attack scenario described by the meta-level process L. The meta-level processes will have no dynamic semantics, i.e. they cannot themselves be executed. Instead they will be instantiated to object-level processes that may then be executed.

The meta-level can be used to specify scenarios with an arbitrary number of principals. For example, the construct $\mathsf{let}\ X = S\ \mathsf{in}\ L$ may be instantiated by any object-level process obtained by choosing a finite subset of S

and using this set for X when instantiating L. To see this in practice, consider the meta-level process of only honest principals found above. This process may be instantiated to the following object-level process by taking $X = \{1, 2\}$:

$$\begin{array}{l} P_{attacker} \mid \\ (\nu^\varepsilon I_1)\,(\nu^\varepsilon I_2)\,(\nu^\varepsilon S) \\ \quad I_1\,[\,P_1\,] \mid I_2\,[\,P_2\,] \mid S\,[\,P_S\,] \end{array}$$

Here $P_{attacker}$ can be any attacker process and P_1 and P_2 are appropriately instantiated versions of P_i. Similarly, the meta-level process may also be instantiated using any other subset of $\mathbb{N}$.

Indexed names. In order to control the way in which the processes P_1 and P_2 are obtained from P_i we shall allow to use indexed names. As an example, suppose that $I_i\,[\,P_i\,]$ takes the form

$$I_i\,[\,(\nu^\varepsilon m)\,\langle m^\varepsilon\rangle^\updownarrow\,]$$

and that $X = \{1, 2\}$; then the corresponding instantiation is

$$I_1\,[\,(\nu^\varepsilon m)\,\langle m^\varepsilon\rangle^\updownarrow\,] \quad\mid\quad I_2\,[\,(\nu^\varepsilon m)\,\langle m^\varepsilon\rangle^\updownarrow\,]$$

where each of I_1 and I_2 generates a new name m that is sent on the ether. Semantically, the two occurrences of m are clearly distinct; from the point of view of the analysis they are much harder to distinguish unless we force the two occurrences to be α-renamed apart.

To achieve this, we let $I_i\,[\,P_i\,]$ take the form

$$I_i\,[\,(\nu^\varepsilon m_i)\,\langle m_i^\varepsilon\rangle^\updownarrow\,]$$

where names are indexed; taking $X = \{1, 2\}$ the corresponding instantiation is

$$I_1\,[\,(\nu^\varepsilon m_1)\,\langle m_1^\varepsilon\rangle^\updownarrow\,] \quad\mid\quad I_2\,[\,(\nu^\varepsilon m_2)\,\langle m_2^\varepsilon\rangle^\updownarrow\,]$$

and now an analysis is likely to provide more precise results by taking advantage of the fact that the names are not the same i.e. that they have been "α-renamed apart".

Similarly, the analysis may more easily distinguish between different variables if they are also α-renamed apart and for this reason, we allow indices on variables as well as on names.

In closing it is important to stress that many of the considerations of how to choose the syntax so as to assist the analysis are of a general nature; in other words, they are also likely to assist reasoning based on other approaches than the control flow analysis approach taken in this paper (e.g. type systems).

4 Syntax and Formal Semantics

Having presented the main ingredients and design considerations for LySa$^{\text{NS}}$ we need to synthesise a formal syntax and to present its formal semantics.

t	$::=$	n	Name
	$\mid$	x	Variable
	$\mid$	$\mathsf{T}(t_1, \cdots, t_k)$	Tuple
	$\mid$	$\mathsf{H}(t_1, \cdots, t_k)$	Hash value
	$\mid$	$\mathsf{E}_{t_0}(t_1, \cdots, t_k)$	Shared key encryption
	$\mid$	$\mathsf{P}_{t_0}(t_1, \cdots, t_k)$	Public key encryption
	$\mid$	$\mathsf{S}_{t_0}(t_1, \cdots, t_k)$	Private key signature

Table 3 Terms; t.

p	$::=$	n	Matches name
	$\mid$	x	Matches value of variable
	$\mid$	$_$	Matches anything
	$\mid$	$p\%x$	Bind variable when p matches
	$\mid$	$\mathsf{T}(p_1, \cdots, p_k)$	Matches tuple
	$\mid$	$\mathsf{H}(cp_1, \cdots, cp_k)$	Checks hash value
	$\mid$	$\mathsf{E}_{cp_0}(p_1, \cdots, p_k)$	Shared key decryption
	$\mid$	$\mathsf{P}_{cp_0}(p_1, \cdots, p_k)$	Public key decryption
	$\mid$	$\mathsf{S}_{sp_0}(p_1, \cdots, p_k)$	Verify signature
cp	$::=$	$n \mid x \mid p\%x \mid \ldots$	Constructive pattern — as p but no wild-card
sp	$::=$	$_ \mid cp$	Signature pattern

Table 4 Patterns; p.

4.1 Terms and Patterns

Syntax. The basic building blocks of the calculus are names $n \in \mathsf{Name}$ and variables $x \in \mathsf{Var}$; both Name and Var are infinite sets. Names are tagged and may be written m^τ though we often leave out the tag when it is unimportant. Both names and variables may be indexed with zero, one, or more indices. We often write a sequence of indices as $\bar{i}$ instead of the more elaborate $i_1 \cdots i_k$. Indices are primarily of interest for the meta-level and we leave them out entirely at other places where they are unimportant.

The syntax of terms, $t \in \mathsf{Term}$, is given in Table 3. Patterns, $p \in \mathsf{Pat}$, closely correspond to terms with the addition of a wild-card and a variable binding mechanism as shown in Table 4. As mentioned earlier, there are restrictions on which patterns that are allowed in decryptions, in verification of signatures and in checking of hash values. Syntactically, this is handled by adopting two restricted classes of patterns. A *constructive pattern*, *cp*, is as an ordinary pattern except that it is not allowed to use the wild-card '_'. A *signature pattern*, *sp*, is slightly more liberal as it is either a wild-card or a constructive pattern; thus in contrast to the general patterns the wild-card can only occur at the top-level. We believe that these decisions present the right way to model the standard security mechanisms.

Semantics. In the tradition of process calculi we shall employ a reduction semantics on closed processes. Hence

(Name) $\theta \vdash n \rhd n : \theta$	(Var) $\theta \vdash v \rhd x : \theta$ if $v = \theta x$	(Wild) $\theta \vdash v \rhd _ : \theta$

$$\text{(Tup)}\quad \frac{\wedge_{i=1}^{k}\, \theta_{i-1} \vdash v_i \rhd p_i : \theta_i}{\theta_0 \vdash \mathsf{T}(v_1, \cdots, v_k) \rhd \mathsf{T}(p_1, \cdots, p_k) : \theta_k} \qquad \text{(Hash)}\quad \frac{\wedge_{i=1}^{k}\, \theta_{i-1} \vdash v_i \rhd p_i : \theta_i}{\theta_0 \vdash \mathsf{H}(v_1, \cdots, v_k) \rhd \mathsf{H}(p_1, \cdots, p_k) : \theta_k}$$

$$\text{(SDec)}\quad \frac{\theta \vdash v_0 \rhd p_0 : \theta_0, \quad \wedge_{i=1}^{k}\theta_{i-1} \vdash v_i \rhd p_i : \theta_i}{\theta \vdash \mathsf{E}_{v_0}(v_1, \cdots, v_k) \rhd \mathsf{E}_{p_0}(p_1, \cdots, p_k) : \theta_k} \qquad \text{(PDec)}\quad \frac{\theta \vdash m^- \rhd p_0 : \theta_0, \quad \wedge_{i=1}^{k}\theta_{i-1} \vdash v_i \rhd p_i : \theta_i}{\theta \vdash \mathsf{P}_{m^+}(v_1, \cdots, v_k) \rhd \mathsf{P}_{p_0}(p_1, \cdots, p_k) : \theta_k}$$

$$\text{(Sign)}\quad \frac{\theta \vdash m^+ \rhd p_0 : \theta_0, \quad \wedge_{i=1}^{k}\theta_{i-1} \vdash v_i \rhd p_i : \theta_i}{\theta \vdash \mathsf{S}_{m^-}(v_1, \cdots, v_k) \rhd \mathsf{S}_{p_0}(p_1, \cdots, p_k) : \theta_k} \qquad \text{(Bind)}\quad \frac{\theta \vdash v \rhd p : \theta'}{\theta \vdash v \rhd p\%x : \theta'[x \mapsto v]}$$

Table 5 Semantics of pattern matching; $\theta \vdash v \rhd p : \theta'$.

the semantics of terms will be values, ranged over by $v \in \mathsf{Val}$, which are merely terms with no variables.

Pattern matching needs to produce a substitution, or an environment,

$$\theta : \mathsf{Var} \to \mathsf{Val}$$

for recording the values bound to variables as a result of the matching. We write $[\,]$ for the empty environment and $\theta[x \mapsto v]$ for the environment that is as θ except that it maps x to v. Eventually these environments will be used to substitute values for variables in the process following the matching.

When a value v is matched against a pattern p it will be relative to an environment that maps all *applied occurrences* of variables in p to their values. Furthermore, the matching will give rise to an extension θ' of the environment that additionally maps the *defining occurrences* of the variables in p to their values. Thus, the judgements of the semantics take the form:

$$\theta \vdash v \rhd p : \theta'$$

The semantics of pattern matching is shown in Table 5. It imposes a left-to-right scoping of variables such that a binding occurrence affects applied occurrences occurring to the right. In the case of public key decryption and verification of digital signatures we must take care to ensure that the keys used are indeed tagged as required and that they belong to the same key pair.

Example 7 The semantics of pattern matching is best explained through a few examples. The name A successfully matches the pattern x in the environment $[x \mapsto A]$:

$$[x \mapsto A] \vdash A \rhd x : [x \mapsto A]$$

Since no new variables are bound in the matching the environment is *not* updated. The name B successfully matches the pattern $_\%x$:

$$\theta \vdash B \rhd _\%x : \theta[x \mapsto B]$$

Here θ needs to be updated to record that x is mapped to B.

For composite values, the left-to-right scoping may sometimes prove useful. For example, the message m in the tuple $\mathsf{T}(k, \mathsf{E}_k(m))$ may successfully be decrypted using the pattern $\mathsf{T}(_\%x, \mathsf{E}_x(_\%y))$ since the matching of the first part of the tuple results in a environment $[x \mapsto k]$ that is used when matching the second part of the tuple.[2]

In the case of public key decryption, a value $\mathsf{P}_{k^+}(A)$ encrypted under a public key, matches a pattern of the form $\mathsf{P}_{k^-}(A)$:

$$\theta \vdash \mathsf{P}_{k^+}(A) \rhd \mathsf{P}_{k^-}(A) : \theta$$

In particular, the first premise of the rule (PDec) succeeds since

$$\theta \vdash k^- \rhd k^- : \theta$$

The semantics of signatures works in a similar fashion.□

4.2 Object-Level Processes

Syntax. Processes are built from patterns and terms using the grammar in Table 6. Names are bound by restriction that may either bind one name, m^{ε}, or two names, m^+ and m^-, depending on the choice of T. Variables may be bound in patterns and therefore pattern matching and input both serve as binders of variables. A name or a variable that is not bound is called free and we write $\mathsf{fn}(P)$ and $\mathsf{fv}(P)$ for the free names and variables, respectively, of the process P. The definitions of these

[2] Note that a message such as $\mathsf{T}(\mathsf{E}_k(m), k)$ cannot benefit from the left-to-right scoping. It may instead be rearranged to fit the above format or, alternatively, it may be decomposed in two subsequent matchings, the first one matching $\mathsf{T}(\mathsf{E}_k(m), k)$ against $\mathsf{T}(_\%z, _\%x)$ and the next one matching z against $\mathsf{E}_x(_\%y)$.

P	$::=$	$P_1 \mid P_2$	Parallel composition
	$\mid$	$!P$	Replication
	$\mid$	$\mathsf{0}$	Terminated process
	$\mid$	$(\nu^T m)\, P$	Restriction/name generation
	$\mid$	$n\,[\,P\,]$	Ambient/principal
	$\mid$	$t \,\mathsf{as}\, p.P$	Pattern matching
	$\mid$	$\langle t_1, \cdots, t_k\rangle^\eta.P$	Output
	$\mid$	$(p_1, \cdots, p_k)^\eta.P$	Input
T	$::=$	ε	General names
	$\mid$	$\pm$	Public/private key pair
η	$::=$	$\circ$	Local communication
	$\mid$	$\updownarrow$	Global communication

Table 6 Processes; P.

functions may be found in the electronic Appendix C and they are standard except that fv has to cater for the left-to-right scoping of variables defined in patterns. For example, in the pattern $\mathsf{T}(p_1, p_2)$ variables defined in p_1 are not free in p_2. Consequently, $\mathsf{fv}(\mathsf{T}(p_1, p_2))$ is given as

$$\mathsf{fv}(\mathsf{T}(p_1, p_2)) = \mathsf{fv}(p_1) \cup (\mathsf{fv}(p_2) \setminus \mathsf{dv}(p_1))$$

where $\mathsf{dv}(p)$ gives the variables defined in p i.e. the defining variables in subpatterns of p that have the form $p'\%x$.

Semantics. The semantics uses α-conversion of variables and names; as already explained, names keep their tag unchanged and only have their base name modified. As an example, the bound name m^+ may be substituted for n^+ but not for n^-.

Substitutions are extended homomorphically to terms, patterns, and processes and we write $P\theta$ for the process that is as P except that all free variables that are defined by θ are replaced by their values. As usual, substitutions perform α-conversion of bound names in P when necessary to avoid capture of the names in θ.

The semantics of processes is then given by a reduction relation $P \rightarrow P'$ that describes how the process P evolves into the process P'. The reduction relation is given as the smallest relation on closed processes that fulfils the rules in Table 7. We shall now briefly comment on some of the rules.

The semantics of pattern matching is used in the rule (Match). Since the semantics of processes is substitution based all variables in the term t will have been substituted with their values before the pattern matching is performed. The rule (Com) says that any two processes directly next to each other may communicate using either local communication or global communication; the latter corresponds to a principal that talks to itself via the network. On the other hand, two principals residing at different principals may only communicate using global communication as expressed in the rule (Glob). In order to bring processes on a form where they match the rules in Table 7, a structural congruence is defined in Table 8. The structural congruence is used in the rule (Con), which as the remaining rules is standard. In the definition of the structural congruence we slightly abuse notation by pretending that T is a set of tags: ε is $\{\varepsilon\}$ and $\pm$ is $\{+, -\}$.

4.3 Canonical Names, Variables, and Indices

As mentioned above the semantics makes extensive use of α-renaming. This presents an obstacle to the analysis because it would naturally collect sets of names and this becomes meaningless when the identity of names is not preserved. To solve this problem, which is standard for congruence based semantics, we conceive that every name n has a *canonical name*, written $\lfloor n \rfloor$, that does not change when α-renaming is performed. (We sometimes say that we performed disciplined α-renaming to stress this point.) Thus, if n is α-renamed to m then their canonical names must be same, i.e. $\lfloor n \rfloor = \lfloor m \rfloor$. Similarly, each variable x has a canonical variable, written $\lfloor x \rfloor$, that is stable under α-renaming.

We take care, that tags are not changed in the process of finding a canonical name and, thus, $\lfloor m^\tau \rfloor$ preserves the tag τ. Furthermore, the indexed names and variables inherit the canonical structure of the index sets. Thus $\lfloor m_i \rfloor = \lfloor n_j \rfloor$ holds if and only if both $\lfloor i \rfloor = \lfloor j \rfloor$ and $\lfloor m \rfloor = \lfloor n \rfloor$; and similarly for variables.

Canonical names and variables are, thus, a central, though standard, aspect in an analysis of congruence based semantics. Semantic correctness is ensured by calculating over-approximations of what can happen when one name is substituted for the other and vice versa. The assignment of canonical names and variables will consequently be a key parameter in controlling the precision of the analysis: the more different canonical names and variables there are, the more distinguishing power the analysis has i.e. the more precise it becomes.

As a novelty, we additionally use assignment of canonical indices in indexing sets to decide the precision of the analysis of meta-level processes. As described earlier, indexing sets, S, of the meta-level may be infinite but in order to cope with this in the analysis we shall require that there is a corresponding *finite* set of *canonical indices*, $\lfloor S \rfloor$.

As an example the set $\mathbb{N}$ of indices of principals of a previous example will have to be mapped to a finite set of canonical indices; if $\lfloor i \rfloor = \lfloor j \rfloor$ then the analysis will not be able to tell the corresponding principals I_i and I_j apart. However, if $\lfloor i \rfloor \neq \lfloor j \rfloor$ then the analysis gives information differentiating I_i and I_j.

4.4 Meta-Level Processes

Indices on names and variables are particularly of interest for the meta-level. When a meta-level process is instantiated the intent is that all indices will be replaced

$$\text{(Par)}\ \frac{P_1 \to P_1'}{P_1 \mid P_2 \to P_1' \mid P_2} \qquad \text{(New)}\ \frac{P \to P'}{(\nu^T m)\, P \to (\nu^T m)\, P'}$$

$$\text{(Amb)}\ \frac{P \to P'}{n\,[\,P\,] \to n\,[\,P'\,]} \qquad \text{(Match)}\ \frac{\emptyset \vdash t \rhd p : \theta}{t\ \mathsf{as}\, p.P \to P\theta}$$

$$\text{(Com)}\ \frac{\emptyset \vdash t_1 \rhd p_1 : \theta_1 \quad \theta_1 \vdash t_2 \rhd p_2 : \theta_2 \quad \cdots \quad \theta_{k-1} \vdash t_k \rhd p_k : \theta_k}{\langle t_1, \cdots, t_k\rangle^\eta.P_1 \mid (p_1, \cdots, p_k)^\eta.P_2 \to P_1 \mid P_2\theta_k}$$

$$\text{(Glob)}\ \frac{\emptyset \vdash t_1 \rhd p_1 : \theta_1 \quad \theta_1 \vdash t_2 \rhd p_2 : \theta_2 \quad \cdots \quad \theta_{k-1} \vdash t_k \rhd p_k : \theta_k}{n_1\,[\,\langle t_1, \cdots, t_k\rangle^\updownarrow.P_1 \mid Q_1\,] \mid n_2\,[\,(p_1, \cdots, p_k)^\updownarrow.P_2 \mid Q_2\,] \to n_1\,[\,P_1 \mid Q_1\,] \mid n_2\,[\,P_2\theta_k \mid Q_2\,]}$$

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

Table 7 Reduction relation for processes; $P \to P'$.

$P \equiv P$	$P_1 \mid P_2 \equiv P_2 \mid P_1$
$P_1 \equiv P_2 \Rightarrow P_2 \equiv P_1$	$(P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$
$P_1 \equiv P_2 \wedge P_2 \equiv P_3 \Rightarrow P_1 \equiv P_3$	$P \mid \mathsf{0} \equiv P$
$P_1 \equiv P_2$ if P_1 and P_2 are α-equivalent	$!P \equiv P \mid !P$
$P_1 \equiv P_2 \Rightarrow P_1 \mid P_3 \equiv P_2 \mid P_3$	$(\nu^T m)\, \mathsf{0} \equiv \mathsf{0}$
$P_1 \equiv P_2 \Rightarrow !P_1 \equiv !P_2$	$(\nu^T m)\,(P_1 \mid P_2) \equiv P_1 \mid (\nu^T m)\, P_2 \quad$ if $\{m^\tau \mid \tau \in T\} \cap \mathsf{fn}(P_1) = \emptyset$
$P_1 \equiv P_2 \Rightarrow (\nu^T m)\, P_1 \equiv (\nu^T m)\, P_2$	$(\nu^{T_1} m_1)\,(\nu^{T_2} m_2)\, P \equiv (\nu^{T_2} m_2)\,(\nu^{T_1} m_1)\, P$
$P_1 \equiv P_2 \Rightarrow n\,[\,P_1\,] \equiv n\,[\,P_2\,]$	$n\,[\,(\nu^T m)\, P\,] \equiv (\nu^T m)\, n\,[\,P\,] \quad$ if $n \notin \{m^\tau \mid \tau \in T\}$

Table 8 Structural congruence for processes; $P \equiv P'$.

L	$::=$	$\mathsf{let}\ X = S\ \mathsf{in}\ L$	Declare indexing set
	$\mid$	$(\nu^T_{\bar{i} \in \overline{X}}\ m_{\bar{j}\bar{i}})\, L$	Indexing restriction
	$\mid$	$\Vert_{i \in X}\ L$	Indexing parallel
	$\mid$	$(\nu^T m)\, L$	Ordinary restriction
	$\mid$	$L_1 \mid L_2$	Ordinary parallel
	$\mid$	$n\,[\,L\,]$	Ambient
	$\mid$	P	Object-level process
	$\mid$	$\bullet$	Open to attack

Table 10 Meta-level processes; L.

by elements from suitable indexing sets. We can view the result of this substitution as an injective syntactic concatenation of indices with the names and the variables, respectively. Conceptually, we may then think of the resulting object-level process as not containing indices at all.

Syntax. Indexing sets are introduced by a let-construct as shown in the grammar for meta-level processes in Table 10. We leave the details of the notation for indexing sets unspecified (i.e. the syntax of S is unspecified) and in examples we use standard notation for sets and operations on them.

We have indexed versions of the constructs for restriction and parallel composition. For restriction the indexed base name $m_{\bar{j}\bar{i}}$ has an index composed of $\bar{j}$ and $\bar{i}$. Here, $\bar{j}$ is a string of indices that have been defined previously, while $\bar{i}$ is a string of indices from the current indexing sets $\overline{X} = X_1 \cdots X_k$. In indexed parallel composition we define a parallel process for each element in the indexing set.

Additionally, meta-level processes can be constructed from object-level processes, ordinary parallel composition and restriction, and the ambient construct. Finally, the symbol $\bullet$ signifies the places where the attacker may be placed and hence where the system is open to attack.

Well-formedness. The syntax is overly liberal in a number of respects and when analysing systems this may pose undesired complications. As an example, indexing sets may be used without having been defined, and indices may be used that have not been previously introduced. More fundamentally, we wish to be able to understand the scope rules also at the meta-level; in particular, we would like to avoid the possibility that a name is free in one instantiation and bound in another.

$$(\text{Let})\quad \frac{\Gamma[X \mapsto S'], \mathcal{N} \vdash L \Rrightarrow P}{\Gamma, \mathcal{N} \vdash \mathsf{let}\ X = S\ \mathsf{in}\ L \Rrightarrow P} \quad \mathsf{if}\ S' \subseteq_{\text{fin}} S$$

$$(\text{INew})\quad \frac{\Gamma, \mathcal{N} \cup \{m^\tau \mid m \in \{m_{\overline{aa_1}} \cdots m_{\overline{aa_j}}\} \wedge \tau \in T\} \vdash L \Rrightarrow P}{\Gamma, \mathcal{N} \vdash (\nu^T_{i \in \overline{X}}\ m_{\overline{ai}})\, L \Rrightarrow (\nu^T m_{\overline{aa_1}}) \cdots (\nu^T m_{\overline{aa_j}})\, P} \quad \mathsf{if}\ \Gamma(\overline{X}) = \{\overline{a_1}, \cdots, \overline{a_j}\}$$

$$(\text{IPar})\quad \frac{\Gamma, \mathcal{N} \vdash L\langle i \mapsto a_1\rangle \Rrightarrow P_1 \quad \cdots \quad \Gamma, \mathcal{N} \vdash L\langle i \mapsto a_j\rangle \Rrightarrow P_j}{\Gamma, \mathcal{N} \vdash \|_{i \in X}\ L \Rrightarrow P_1 \mid \cdots \mid P_j} \quad \mathsf{if}\ \Gamma(X) = \{a_1, \cdots, a_j\}$$

$$(\text{MNew})\quad \frac{\Gamma, \mathcal{N} \cup \{m^\tau \mid \tau \in T\} \vdash L \Rrightarrow P}{\Gamma, \mathcal{N} \vdash (\nu^T m)\, L \Rrightarrow (\nu^T m)\, P} \qquad (\text{MPar})\quad \frac{\Gamma, \mathcal{N} \vdash L_1 \Rrightarrow P_1 \qquad \Gamma, \mathcal{N} \vdash L_2 \Rrightarrow P_2}{\Gamma, \mathcal{N} \vdash L_1 \mid L_2 \Rrightarrow P_1 \mid P_2}$$

$$(\text{MAmb})\quad \frac{\Gamma, \mathcal{N} \vdash L \Rrightarrow P}{\Gamma, \mathcal{N} \vdash n\,[\,L\,] \Rrightarrow n\,[\,P\,]} \qquad (\text{Proc})\quad \Gamma, \mathcal{N} \vdash P \Rrightarrow P$$

(Atcr) $\Gamma, \mathcal{N} \vdash \bullet \Rrightarrow P$ for an arbitrary P such that

- there are no free variables in P,
- all free names in P come from $\mathcal{N}$, and
- all variables created in P have $\mathsf{x}_\bullet$ as canonical variable,
- all names generated in P have $\mathsf{n}^\varepsilon_\bullet$, $\mathsf{n}^+_\bullet$, or $\mathsf{n}^-_\bullet$ as canonical name.

Table 9 Instantiation relation between meta-level processes and object-level processes; $\Gamma, \mathcal{N} \vdash L \Rrightarrow P$

This motivates introducing a well-formedness requirement that ensures that all object-level or meta-level processes are fully closed; this means that not only are there no free variables, also there are no free names. This does not present a limitation in our ability to deal with open systems because our approach is to insert $\bullet$ at all points where the system is open to attack.

The well-formedness requirement is formalised as well-formedness predicates on each of the syntactic categories. The formalisation may be found in Appendix B and in the remainder of this paper we tacitly assume that all syntactic constructs are well-formed.

Instantiation. The semantics of meta-level processes is given by an *instantiation relation*

$$\Gamma, \mathcal{N} \vdash L \Rrightarrow P$$

that holds between a meta-level process L and any one of possibly infinitely many object-level processes P. The component Γ is a mapping from indexing set identifiers to the sets they denote; we shall write $\Gamma[X \mapsto S]$ and $\Gamma(X)$ for updating and querying these mappings, respectively. The component $\mathcal{N}$ collects the free names of the process P.

The instantiation relation is defined as the smallest relation satisfying the rules in Table 9. The rule (Let) chooses a *finite* subset S' of the indexing set S and updates the environment accordingly. This is the only construct that chooses any subsets and this ensures that the *same subset* will be chosen throughout the meta-level process.

The rule (INew) generates a new restriction for each name in the indexing set bound to X; these names are potentially free names in the process obtained by instantiating the body of the restriction so the $\mathcal{N}$ component of the premise of the rule is extended to capture this. Furthermore, we write $\Gamma(\overline{X})$ for the set $\Gamma(X_1) \times \cdots \times \Gamma(X_k)$ when $\overline{X} = X_1 \cdots X_k$. If the indexing set is empty then the restriction instantiates simply as the process P obtained from L.

The indexed parallel composition construct instantiates a process for each element in the set bound to X as described by the rule (IPar). In each such process P all indices are substituted with different elements from the indexing set. Here, we write $P\langle i \mapsto a\rangle$ for the process that is as P except all indices i are substituted by a. If the indexing set is empty then the indexing parallel instantiates as $\mathsf{0}$.

The rules (MPar), (MAmb), and (MNew) simply require that the meta-level processes are instantiated while (Proc) says that any object-level part of a meta-level process immediately instantiates.

The rule (Atcr) allows $\bullet$ to be replaced by any process P that satisfies a number of conditions. First it is required that P does not have any free variables; this is quite natural given that the meta-level does not con-

tain any binding constructs for variables. Next P may have free names but they all have to come from $\mathcal{N}$, the set of names that are bound by the context of P; this turns out to account for all the names around because of the well-formedness conditions and our insistence on only dealing with systems that are fully closed. As for the names and variables created by P we shall fix their canonical identity as follows: all variables x have the same canonical variable $\lfloor x \rfloor = \mathsf{x}_\bullet$ and all names n^τ have the same canonical name reflecting their tag $\lfloor n^\tau \rfloor = \mathsf{n}^\tau_\bullet$.

5 Validating the Design

As already announced in the abstract, there are three factors governing the design of process calculi for analysing security protocols. The first two concern the faithful modelling of protocols and attack scenarios. In Section 3 we have given many examples to validate that our design indeed meets these requirements. More examples are given in Appendix A, where a number of protocols known from the literature are encoded in full.

The last test of the design of LySa$^{\text{NS}}$ is the extent to which it allows to exploit the strengths (and at the same time to limit the weaknesses) of our approach to static analysis. To do so, we outline an analysis of LySa$^{\text{NS}}$ and perform an in-depth analysis of how to use the features of the calculus in order to increase the precision of this analysis. The interested reader may enjoy the definition of the analysis in full in the electronic Appendix D.

5.1 Overview of the Analysis

Terms. A term t without any variables describes a single value; in the analysis we shall be interested in tracking only the canonical value (i.e. the value where all names have been replaced by their canonical representatives). If there are variables in a term t then it may describe a large set of values. To make this precise we need an abstract environment ξ that maps variables to sets of canonical values, and we then define the set $\xi[\![t]\!]$ of canonical values denoted by the term t. The definition amounts to a straightforward structural induction on t.

Patterns. When matching a term t to a pattern p we shall take the approach that we match each canonical value v in $\xi[\![t]\!]$ to the pattern p. This is performed by means of a judgement of the form

$$\xi \models v \rhd p : s$$

that will admit v as an element of the set s of successfully matched values whenever v matches the pattern. The definition is in the form of a flow logic and may be viewed as an inductive definition in the structure of the pattern p.

The Need for Two Stages. Looking carefully at the semantics for pattern matching in Table 5 and its use in the semantics of processes in Table 7 we observe that there are two stages in this process: first a set of candidate values are bound to variables in the substitution θ and secondly, if the substitution "survives until the end" (i.e. there is a judgement of the form $\emptyset \vdash t \rhd p : \theta$) the values are substituted into the process following the pattern matching. This means that there may be bindings of values to variables in θ that do not "survive until the end" and hence are not substituted into the process.

To deal with these two stages without incurring a major loss of precision we shall decide to use two abstract environments:

- σ records the set of canonical values that a variable may be bound to in some θ during the first stage of the process, and
- ρ records the set of canonical values that may be substituted for a variable in some process in the second stage of the process.

Both of these can be used in place of ξ in the function $\xi[\![t]\!]$ and judgement $\xi \models v \rhd p : s$ explained above. Intuitively we would expect that each $\rho(x)$ is a subset of $\sigma(x)$.

Example 8 Consider the pattern matching

$$x \,\mathsf{as}\, \mathsf{T}(_\%y, b)$$

and imagine that it is analysed in a context where

$$\rho(x) = \{\mathsf{T}(c, a), \mathsf{T}(d, a), \mathsf{T}(e, a)\}$$

Semantically, the pattern matching would not succeed because none of the tuples that may be bound to x match the entire pattern. Without a two stage approach, i.e. using ρ also for σ, the analysis would bind variables "on-the-fly" and show that $\rho(y) = \{c, d, e\}$. Thanks to the two stage approach, i.e. distinguishing between ρ and σ, we obtain $\sigma(y) = \{c, d, e\}$ but $\rho(y) = \emptyset$. $\square$

Processes. The judgement for analysing object-level processes then takes the form

$$(\rho, \sigma, \kappa) \models^n P$$

where we already introduced κ in Section 1 as the mechanism used to record the sequences of values communicated. Local communication within an ambient will be recorded in κ using the canonical name of the ambient to identify where the communication takes place while global communication is recorded with the special symbol $\updownarrow$. Formally, κ is a mapping from the canonical names of ambients and $\updownarrow$ to the set of sequences of canonical values communicated at that location. That is, $\kappa(\updownarrow)$ contains the messages sent on the global network while $\kappa(n)$ contains messages communicated inside an ambient with the canonical name n. The superscript n indicates the

canonical name of the immediately enclosing ambient of the process P and is used to decide in which ambient local communication occurs. Furthermore, the analysis takes care of the two stages of the analysis of matching with respect to σ and ρ as well as ensuring that a process is only analysed if indeed it is reachable.

The definition of $(\rho, \sigma, \kappa) \models^n P$ is in the form of a flow logic and may be viewed as an inductive definition in the structure of the object-level process P. Technically, the flow logic definition specifies when ρ, σ, and κ acceptably describes behaviour of the process P. Alternatively, one may think of judgements in an operational manner as something that given a process P returns a ρ, σ, and κ that describes how P may behave when placed inside the ambient n. Readers familiar with type systems may like to parallel these two points of view with whether your regard a typing judgement to be a specification of type checking or a means of type inference.

The judgement $(\rho, \sigma, \kappa) \models^n P$ is intended to yield true when the information in ρ, σ, and κ correctly describes the process P (placed in the context ρ) as well as the processes it may evolve to. Hence, we need to ensure that the definition of $(\rho, \sigma, \kappa) \models^n P$ (as given in the electronic Appendix D) correctly captures the formal semantics of Table 7. This amounts to formalising the intentions of Figure 3:

Theorem 1 (Subject reduction) *if* $(\rho, \sigma, \kappa) \models^n P$ *and* $P \to P'$ *then* $(\rho, \sigma, \kappa) \models^n P'$

The result is proved by induction on the inference $P \to P'$.

Meta-Level Processes. The judgement for meta-level processes takes the form

$$(\rho, \sigma, \kappa) \models^n_{\Gamma, \mathcal{N}} L$$

and has two subscripts not present in the case of object-level processes. One is the environment Γ that records the bindings of meta-level expressions of the form $\mathsf{let}\ X = S\ \mathsf{in}\ L$; more precisely it will map X to the set of canonical names in S. The other is the set of canonical names $\mathcal{N}$ that have been defined so far; it is augmented whenever we have a binding construct of the form $(\nu^T_{\overline{i \in X}}\ m_{\overline{bi}})$ or $(\nu^T m)$; this component is essential in order to formulate the Dolev-Yao condition for the places where the (otherwise closed) system may be open to attack.

As before, the definition of $(\rho, \sigma, \kappa) \models^n_{\Gamma, \mathcal{N}} L$ is in the form of a flow logic and may be viewed as an inductive definition in the structure of the meta-level process L. Semantic correctness means that the definition captures all processes that may arise as instantiations in the sense of Table 9. This amounts to formalising the intentions of Figure 2:

Theorem 2 *if* $(\rho, \sigma, \kappa) \models^n_{\Gamma, \mathcal{N}} L$ *and* $\Gamma', \mathcal{N}' \vdash L \Rrightarrow P$ *where* $\lfloor \Gamma'(X) \rfloor \subseteq \Gamma(X)$ *for all* X *and* $\lfloor \mathcal{N}' \rfloor \subseteq \mathcal{N}$ *then* $(\rho, \sigma, \kappa) \models^n P$

The result is proved by induction on the inference $\Gamma', \mathcal{N}' \vdash L \Rrightarrow P$. The conditions relating Γ and $\mathcal{N}$ to Γ' and $\mathcal{N}'$ state that the analysis uses canonical names while the semantics does not. The use of subset is necessary since instantiation may non-deterministically choose a subset of indices in the let-construct while the analysis has to consider the largest of these sets.

5.2 Rebinding Variables in Pattern Matching

In the remainder of this section we study how to exploit the annotations of our calculus in the interest of increasing the precision of the analysis. As a first example, we show how our carefully designed pattern matching mechanism may aid the analysis in this respect. Consider a process that tests whether the variable y contains a hashed value of the variable x and then sends the hashed value on the local network:

$$\mathsf{H}(x)\ \mathsf{as}\ y.\langle y \rangle^\circ.0$$

Now, let us assume that the analysis has already determined that the variable x may be bound to the value a and that y may be bound to either $\mathsf{H}(a)$ or $\mathsf{H}(b)$. Semantically, the matching can succeed only when y is bound to $\mathsf{H}(a)$ and only this value can be sent on the network. In the analysis, the information will be represented in the component ρ as:

$$\rho(x) = \{a\}$$
$$\rho(y) = \{\mathsf{H}(a), \mathsf{H}(b)\}$$

Consequently, the analysis determines that the pattern matching succeeds for the value $\mathsf{H}(a)$ and that the continuation, i.e. the process $\langle y \rangle^\circ.0$, should be analysed. Since no binding takes place in the pattern there are no changes to ρ, and in particular $\rho(y)$ is unchanged. Consequently, the analysis of the output would determine that all values in $\rho(y)$ may be sent on the local network, i.e. that both $\mathsf{H}(a)$ and $\mathsf{H}(b)$ may be sent.

Alternatively, we may change the process and use the ability to rebind variables as part of pattern matching:

$$\mathsf{H}(x)\ \mathsf{as}\ y\%y'.\langle y' \rangle^\circ.0$$

Semantically, no change has taken place. However, since a variable binding is introduced, the analysis may give a more precise analysis result for this new process. If we analyse the process using the same ρ as before the analysis will once more determine that the pattern match succeeds for $\mathsf{H}(a)$. However, the variable y' only needs to be bound to this value:

$$\rho(y') = \{\mathsf{H}(a)\}$$

The analysis will then be able to determine that only $\mathsf{H}(a)$ can be sent on the network.

The problem described in this example came up when validating versions of the Otway-Rees protocol with hashing [25] (see also Appendix A) using the techniques of

[7]. Unfortunately, the more restrictive syntax of pattern matching incorporated in LySa does not allow rebinding of variables and therefore the problem remained unsolved in [7].

5.3 Arbitrarily Many Principals

The analysis of meta-level processes allows for analyses of systems with arbitrarily many principals and, thereby, overcomes a weakness of [7] where some representative number had to be chosen ad-hoc. The novel idea presented in this paper is that the meta-level may describe arbitrarily large system by allowing infinite sets to be used in the let-construct though these sets must have a finite canonical partitioning that is used by the analysis. The assignment of canonical values does not affect correctness of the analysis but is merely a mechanism for controlling its precision. Thus, the analysis result will be a valid estimate of what happens in systems of arbitrary size.

Example 9 As a simple example, we may stipulate that all elements of the infinite set of natural numbers $\mathbb{N}$ have the same canonical representative, e.g. $\lfloor \mathbb{N} \rfloor = \{\star\}$.

Next, consider a very simple "protocol" with arbitrarily many principals and where each I_i creates a new secret name s_i and sends it encrypted with its private key on the global ether. This is described by the meta-level process:

$$\begin{aligned} L \stackrel{\text{def}}{=} \ & \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ & (\nu^{\varepsilon}_{i \in X}\ I_i) \\ & (\nu^{\pm}_{i \in X}\ K_i) \\ & \|_{i \in X}\ \ I_i\,[\,(\nu^{\varepsilon} s_i)\,\langle \mathsf{P}_{K_i^+}(s_i)\rangle^{\updownarrow}\,] \end{aligned}$$

We analyse it using the meta-level analysis:

$$(\rho, \sigma, \kappa) \models^{\top}_{[\,],\emptyset} L$$

The analysis of the let-construct will update the (empty) environment with $[X \mapsto \{\star\}]$, since the canonical set of X is $\{\star\}$. Next, the analysis of the indexed parallel composition will analyses the principal I_i with i substituted for each of the (canonical) elements in $\{\star\}$. In turn, this gives that the network component is

$$\kappa(\updownarrow) = \{\mathsf{P}_{K_\star^+}(s_\star)\}$$

and it follows that terms with $\mathsf{P}_{K_\star^+}(s_\star)$ as their canonical representative may be sent on the global network.

It is important to note that this value is *a canonical representative* that not only describes messages like $\mathsf{P}_{K_3^+}(s_3)$ and $\mathsf{P}_{K_7^+}(s_7)$ but also messages like $\mathsf{P}_{K_3^+}(s_7)$ and $\mathsf{P}_{K_7^+}(s_3)$. Despite this over-approximation of the behaviour of processes the analysis result does provide valuable information about the system. For example, it makes it clear that no instance of L may send any s_i in clear on the network. □

5.4 Attack Scenarios

Our calculus is unique in admitting explicit syntax for the positions where a system may be open to attack. This gives a great variation in the classes of attack scenarios that can be considered and we shall illustrate this by means of two variations of Example 9.

Example 10 The traditional scenario is that the attacker is placed in parallel with the protocol:

$$\bullet \mid L$$

This means that the attacker has no initial knowledge of the identity of the principals nor of their keys. An analysis of this process takes the form

$$(\rho, \sigma, \kappa) \models^{\top}_{[\,],\emptyset} \bullet \mid L$$

and will result in a different result from before because the attacker can interact with the instances of L. For example, although $\kappa(\updownarrow)$ still contains $\mathsf{P}_{K_\star^+}(s_\star)$ it also contains elements that the attacker may send on the network. Nonetheless, the network component still does not contain $s_\star$ in clear and this means that the attacker is not able to break the secrecy properties of the protocol. □

Example 11 As a different scenario, we may place the attacker so that it has initial knowledge of the keys and the identity of the principals:

$$\begin{aligned} L' \stackrel{\text{def}}{=} \ & \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ & (\nu^{\varepsilon}_{i \in X}\ I_i) \\ & (\nu^{\pm}_{i \in X}\ K_i) \\ & \bullet\ | \\ & \|_{i \in X}\ \ I_i\,[\,(\nu^{\varepsilon} s_i)\,\langle \mathsf{P}_{K_i^+}(s_i)\rangle^{\updownarrow}\,] \end{aligned}$$

This time an analysis of L' will yield a network component that contains $s_\star$. It follows that we cannot guarantee that no s_i is sent in clear on the network. This is semantically the best we can hope for because the attacker has initial knowledge of the keys K_i^- and hence can decrypt the messages $\mathsf{P}_{K_i^+}(s_i)$, obtain the contents s_i and then send it in clear on the network. □

5.5 Variations of the Analysis

The analysis we have given in the electronic Appendix D is a relatively simple analysis. Our intention is that this analysis should only serve as a proof-of-concept that LySa$^{\text{NS}}$ indeed does fulfils our third design goal and easily gives a more precise analysis as presented earlier in this section. Below we briefly touch on some of the shortcomings of the analysis and suggest improvements that may be pursued in future work.

One disadvantage of the current analysis is that the communication box of an ambient is linked so closely to the name of the principal in which it resides. This means

that mere knowledge of that name will allow the attacker to influence the communication taking place in that communication box; e.g. by creating its own principal with that name and perform the communications there. It is possible to code the protocols in such a way that they are hardened against this problem, e.g. by distinguishing between their "real name" (which is kept secret) and their "official name" (which is made public). However, a better choice is to modify the analysis. A standard technique is to use labels on all ambients and to use the label as the "real name". This works because the labels cannot be manipulated by any process and, in particular, the attacker may only create ambients labelled with a special label, say $l_\bullet$.

Another disadvantage of the current analysis is that it analyses the rebinding construct, $p\%x$, in an independent attribute manner rather than a relational manner. This means that not all of the protocols encoded in Appendix A can be analysed to give results that are as precise as those in [7]. To obtain the same precision as in [7], one alternative is to change the encoding of the protocols in Appendix A to be more in the flavour of [7]. Another alternative is to develop an analysis that allows to deal with the rebinding construct, $p\%x$, in a relational manner, written $p\permil x$. We leave the development of this analysis as future work.

6 Perspectives

Let us reconsider the objectives of our development and show that they are met by the calculus designed here. We claim that it is quite natural to express security protocols in a precise and faithful manner. One reason is the careful consideration of the cryptographic primitives and the decomposition of cryptographic entities using different kinds of patterns. This not only gives a precise account of symmetric encryption but also of asymmetric encryption and digital signatures, capturing their different flavour, and hashing. Another is the meta-level processes that make it straightforward to analyse system scenarios in such a way that the results carry over to arbitrarily large systems falling within the scenario.

We also claim that our calculus is unique in the flexibility with which a variety of different attack scenarios can be described. This is due to the incorporation of explicit syntax, $\bullet$, for where the system is open to attack. It makes it possible to discuss at the syntactic level such fine distinctions as the ones between a malicious attacker and a dishonest principal. Also it makes it possible to study how systems should be "hardened" when the attack scenario becomes more demanding on the system, e.g. because the attacker is given more initial information. It would even be possible to add additional syntax for classes of attackers with more limited abilities, e.g. passive attackers, if necessary to describe the attack scenarios at hand.

Finally, we believe that our calculus goes a long way towards allowing the protocol designer to present the analysis in such a way that the strengths of the underlying analysis technology are best exploited. This should be viewed as a handle that assists the protocol designer in adapting the precision of the analysis without requiring deep knowledge of how to specify and implement analyses. This will not least be of interest for future developments where we may need to develop more powerful analyses as discussed in Section 5.5.

Surely much work needs to be done. The examples in the paper already indicate the ability to ensure secrecy properties and following the approach of [7] it should be straightforward to add annotations that will allow us to deal with origin and destination authenticity. As in [7] this may be seen as a kind of message authentication except that the property is non-injective [22] in the sense that it does not require a strict one-to-one correspondence between origin and destination of messages. We plan also to address non-injective session based authenticity whereas the stronger one-to-one correspondence properties are likely to require a more complex analysis; however, we do not foresee any need to make major changes in the calculus. Implementation of the analysis developed in the electronic Appendix D, using the techniques of [33, 7], remains to be done but is well beyond the scope of the present development.

Opening up for dynamically changing networks, in order to model the emerging wireless and mobile infrastructure, there may well be a need to extend the calculus with new constructs. We believe that the mobility capabilities of Mobile Ambients [10] could easily be adapted although we might find it more appropriate to use one of the many variations to be found in the literature (e.g. [10, 20, 34]). Some of these contain syntax for directly dealing with access control, whether discretionary or mandatory, and would show the way towards a calculus that treats network security and access control as equally important features of Global Computing.

Acknowledgements. Discussions with Pierpaolo Degano and Chiara Bodei on modelling and analysing security protocols have helped to form many of the ideas presented in this paper.

References

1. M. Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 5(46):749–786, 1999.
2. M. Abadi and B. Blanchet. Analyzing security protocols with secrecy types and logic programs. In *Proceedings of the 29th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2002)*, pages 33–44. ACM Press, 2002.
3. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM SIGPLAN-SIGACT Symposium on Principles of*

Programming Languages (POPL 2001), pages 104–115. ACM Press, 2001.

4. M. Abadi and A. D. Gordon. A calculus for cryptographic protocols – The Spi calculus. *Information and Computation*, 148(1):1–70, 1999.
5. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW 2001)*, pages 82–96. IEEE Computer Society Press, 2001.
6. B. Blanchet. From secrecy to authenticity in security protocols. In *Static Analysis, 9th International Symposium (SAS 2002)*, volume 2477 of *Lecture Notes in Computer Science*, pages 342–359. Springer Verlag, 2002.
7. 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.
8. C. Bodei, P. Degano, F. Nielson, and H. Riis Nielson. Flow Logic for Dolev-Yao secrecy in cryptographic processes. *Future Generation Computer Systems*, 18(6):747–756, 2002.
9. M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Theoretical Aspects in Computer Science (TACS 2001)*, volume 2215 of *Lecture Notes in Computer Science*, pages 37–63. Springer Verlag, 2001.
10. L. Cardelli and A. D. Gordon. Mobile Ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
11. I. Cervesato. Data access specification and the most powerful symbolic attacker in MSR. In *Proceedings of the International Symposium on Software Security (ISSS 2002)*, volume 2609 of *Lecture Notes in Computer Science*, pages 384–416. Springer Verlag, 2003.
12. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. http://www-users.cs.york.ac.uk/~jac/papers/drareviewps.ps, 1997.
13. D. Dolev and A. C. Yao. On the security of public key protocols. In *22nd Annual Symposium on Foundations of Computer Science*, pages 350–357. IEEE, 1981.
14. A. Durante, R. Focardi, and R. Gorrieri. A compiler for analyzing cryptographic protocols using noninterference. *ACM Transactions on Software Engineering and Methodology*, 9(4):488–528, 2000.
15. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.
16. R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1995.
17. R. Focardi and R. Gorrieri. The compositional security checker: A tool for the verification of information flow security properties. *IEEE Transactions on Software Engineering*, 23(9), 1997.
18. C. Fournet and M. Abadi. Hiding names: Private authentication in the applied pi calculus. In *Proceedings of the International Symposium on Software Security (ISSS 2002)*, volume 2609 of *Lecture Notes in Computer Science*, pages 317–338. Springer Verlag, 2003.
19. A. D. Gordon and A. Jeffrey. Authenticity by Typing for Security Protocols. In *Proceedings of the 14th Computer Security Foundations Workshop (CSFW 2001)*, pages 145 –159. IEEE, 2001.
20. F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 352–364. ACM Press, 2000.
21. G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springer Verlag, 1996.
22. G. Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th Computer Security Foundations Workshop (CSFW 1997)*, pages 31–43. IEEE Computer Society Press, 1997.
23. G. Lowe. Casper: A compiler for the analysis of security protocols. *Journal of Computer Security*, 6(1):53–84, 1998.
24. G. Lowe. Towards a completeness result for model checking of security protocols. *Journal of Computer Security*, 7(2-3):89–146, 1999.
25. W. Mao and C. Boyd. Methodical use of cryptographic transformations in authentication protocols. *IEE Proceedings of Computers and Digital Techniques*, 142(4):272–278, 1995.
26. R. Milner. *Communication and Concurrency.* Prentice Hall Series in Computer Science. Prentice Hall, 1989.
27. R. Milner, J. Parrow, and D. Walker. A calculus of Mobile processes (I and II). *Information and Computation*, 100(1):1–77, 1992.
28. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
29. F. Nielson and H. Riis Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 1997)*, pages 332–345. ACM Press, 1997.
30. F. Nielson, H. Riis Nielson, and C. Hankin. *Principles of Program Analysis.* Springer Verlag, 1999.
31. F. Nielson, H. Riis Nielson, and R. R. Hansen. Validating firewalls using flow logics. *Theoretical Computer Science*, 283(2):381–418, 2002.
32. 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.
33. F. Nielson, H. Riis Nielson, and H. Seidl. Cryptographic analysis in cubic time. *Electronic Notes in Theoretical Computer Science*, 62, 2002.
34. H. Riis Nielson, F. Nielson, and M. Buchholtz. Security for mobility. In *Proceedings of FOSAD 2001*, Lecture Notes in Computer Science. Springer Verlag, To appear.
35. D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating Systems Review*, 21(1):8–10, 1987.
36. R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.
37. A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.
38. B. Schneier. *Applied Cryptography.* John Wiley & Sons, 1994.

A Example Protocols

Below we present a number of encodings of standard protocols as meta-level processes in LySa$^{\text{NS}}$. First we consider a simple protocol with just one message.

A.1 ISO Symmetric Key One-Pass Unilateral Authentication Protocol

The protocol is described in [12] by the narration

$$A \to B : m2, \mathsf{E}_{K_{AB}}(N_A, B, m1)$$

where $m1$ and $m2$ are messages, N_A is a nonce and it is assumed that the principals A and B already share the key K_{AB}.

The protocol may be used in a scenario where arbitrarily many principals may use it to communicate with one another any number of times:

$$\begin{array}{l} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ \quad \bullet \\ \mid\ (\nu^{\varepsilon}_{i\in X}\ I_i)\ (\nu^{\varepsilon}_{i\,j\in X\,X}\ K_{ij}) \\ \quad \|_{i\in X}\ I_i\,[\ \ \|_{j\in X}\,!\ \mathit{Init}(I_i, I_j) \\ \qquad\qquad \mid \|_{j\in X}\,!\ \mathit{Resp}(I_i, I_j)\,] \end{array}$$

Here, a new name I_i is generated for each principal along with a shared key K_{ij} for each pair of principals. At each principal, i.e. inside the ambient $I_i\,[\cdots]$, the protocol is run both as initiator and responder.

For the encoding of the initiator and the responder we shall follow [7] and include the source and destination addresses (A and B above) as the first two elements of the message. The initiator part of the protocol $\mathit{Init}(I_i, I_j)$ can be coded as follows:

$$\begin{array}{l} (\nu^{\varepsilon} N^A_{ij})\,(\nu^{\varepsilon} m1_{ij})\,(\nu^{\varepsilon} m2_{ij}) \\ \quad \langle I_i, I_j, m2_{ij}, \mathsf{E}_{K_{ij}}(n^A_{ij}, I_j, m1_{ij})\rangle^{\updownarrow}.\cdots \end{array}$$

The corresponding responder code $\mathit{Resp}(I_j, I_i)$ is then

$$(I_i, I_j, _\%x^2_{ij}, \mathsf{E}_{K_{ij}}(_\%x^A_{ij}, I_j, _\%x^1_{ij}))^{\updownarrow}.\cdots$$

and the variables x^1_{ij} and x^2_{ij} will eventually contain the messages $m1_{ij}$ and $m2_{ij}$. At completion of the protocol the principal may engage in other matters described by a process placed at the ellipses ($\cdots$).

A.2 ISO Symmetric Key One-Pass Unilateral Authentication Protocol using Cryptographic Check Functions

The above protocol also exists in a version where it uses *cryptographic check functions* [12]. A cryptographic check function is a key-dependent function $f_K(t)$ that returns a hash value for data t in a manner determined by the key K. In [12] the protocol is specified by:

$$A \to B : N_A, B, m2, f_{K_{AB}}(N_A, B, m1)$$

Here it is assumed that both A and B know the message *m1* before the protocol is run.

A standard technique for defining a cryptographic check function, $f_K(t)$ (see e.g. [38]) is to use a one-way hash function that gives a hash value for the combination of the key K and the argument t. In LySa$^{\text{NS}}$ we can model this as $\mathsf{H}(t, K)$.

The overall setting of the protocol is now slightly different from before as the message $m1$ is known to both principals before the protocol is initiated. We can code this as

$$\begin{array}{l} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ \quad \bullet \\ \mid\ (\nu^{\varepsilon}_{i\in X}\ I_i)\ (\nu^{\varepsilon}_{i\,j\in X\,X}\ K_{ij})\,(\nu^{\varepsilon}_{i\,j\in X\,X}\ m1_{ij}) \\ \quad \|_{i\in X}\ I_i\,[\ \ \|_{j\in X}\,!\ \mathit{Init}(I_i, I_j) \\ \qquad\qquad \mid \|_{j\in X}\,!\ \mathit{Resp}(I_i, I_j)\,] \end{array}$$

where the initiator $\mathit{Init}(I_i, I_j)$ and the responder $\mathit{Resp}(I_j, I_i)$ are specified by

$$\begin{array}{l} (\nu^{\varepsilon} N^A_{ij})\,(\nu^{\varepsilon} m2_{ij}) \\ \quad \langle I_i, I_j, N^A_{ij}, I_j, m2_{ij}, \mathsf{H}(N^A_{ij}, I_j, m1_{ij}, K_{ij})\rangle^{\updownarrow}.\cdots \end{array}$$

and

$$(I_i, I_j, _\%x^A_{ij}, I_i, _\%x^2_{ij}, \mathsf{H}(x^A_{ij}, I_j, m1_{ij}, K_{ij}))^{\updownarrow}.\cdots$$

respectively. To show the diversity of the calculus let us also specify a scenario where everyone, including the attacker, knows the messages $m1$ used by the principals:

$$\begin{array}{l} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ \quad (\nu^{\varepsilon}_{i\,j\in X\,X}\ m1_{ij}) \\ \quad \bullet \\ \mid\ (\nu^{\varepsilon}_{i\in X}\ I_i)\ (\nu^{\varepsilon}_{i\,j\in X\,X}\ K_{ij}) \\ \quad \|_{i\in X}\ I_i\,[\ \ \|_{j\in X}\,!\ \mathit{Init}(I_i, I_j) \\ \qquad\qquad \mid \|_{j\in X}\,!\ \mathit{Resp}(I_i, I_j)\,] \end{array}$$

The code for the initiator and responder is unchanged.

A.3 Needham-Schroeder symmetric key protocol

The classical protocol narration [28] is written as follows:

1. $A \to S :\ A, B, N_A$
2. $S \to A :\ \mathsf{E}_{K_A}(N_A, B, K, \mathsf{E}_{K_B}(K, A))$
3. $A \to B : \mathsf{E}_{K_B}(K, A)$
4. $B \to A : \mathsf{E}_K(N_B)$
5. $A \to B : \mathsf{E}_K(N_B{-}1)$

Here the two principals A and B are assumed to share master keys K_A and K_B with the server S. The protocol is initiated by A and S will then create the session key K and send it appropriately encrypted to A who forwards an encrypted message with the key to B. The protocol concludes by resolving a nonce challenge issued by B.

In LySA$^{\mathrm{NS}}$ the protocol can be written as follows:

$$\begin{array}{l} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ \quad \bullet \\ \mid\ (\nu^{\varepsilon}_{i\in X}\ I_i)\,(\nu^{\varepsilon}_{i\in X}\ K_i) \\ \qquad \|_{i\in X}\ I_i\,[\,\|_{j\in X}\,!\mathit{Init}(I_i,I_j)\mid !\mathit{Resp}(I_i)\,] \\ \mid\quad S\,[\,!\mathit{Server}\mid \mathit{Database}\,] \end{array}$$

where $\mathit{Init}(I_i, I_j)$ is the code for I_i initiating the protocol with I_j, $\mathit{Resp}(I_i)$ is the code for I_i when acting as a responder, and *Server* is the code for the server; as we shall see it will communicate locally with the component *Database*. The individual components are presented below; the rightmost numbers refer to the corresponding step in the protocol narration given above. For $\mathit{Init}(I_i, I_j)$ we have:

$$\begin{array}{ll} (\nu^{\varepsilon} N^A_{ij})\,\langle I_i, S, I_i, I_j, N^A_{ij}\rangle^{\updownarrow}. & (1) \\ \quad (S, I_i, \mathsf{E}_{K_i}(N^A_{ij}, I_j, _\%y^K_{ij}, _\%y_{ij}))^{\updownarrow}. & (2) \\ \qquad \langle I_i, I_j, y_{ij}\rangle^{\updownarrow}. & (3) \\ \qquad\quad (I_j, I_i, \mathsf{E}_{y^K_{ij}}(_\%z^B_{ij}))^{\updownarrow}. & (4) \\ \qquad\qquad \langle I_i, I_j, \mathsf{E}_{y^K_{ij}}(\mathsf{T}(z^B_{ij}, 1))\rangle^{\updownarrow}.\ \cdots & (5) \end{array}$$

The code $\mathit{Resp}(I_j)$ for the responder role is:

$$\begin{array}{ll} (_\%x^A_j, I_j, \mathsf{E}_{K_j}(_\%x^K_j, x^A_j))^{\updownarrow}. & (3) \\ \quad (\nu^{\varepsilon} N^B_j)\,\langle I_j, x^A_j, \mathsf{E}_{x^K_j}(N^B_j)\rangle^{\updownarrow}. & (4) \\ \qquad (x^A_j, I_j, \mathsf{E}_{x^K_j}(\mathsf{T}(N^B_j, 1)))^{\updownarrow}.\ \cdots & (5) \end{array}$$

The *Server* code is as follows:

$$\begin{array}{ll} (_\%x^A, S, x^A, _\%x^B, _\%x^N)^{\updownarrow}. & (1) \\ \quad (x^A, _\%x^{K_A})^{\circ}.\ (x^B, _\%x^{K_B})^{\circ}. & \\ \qquad (\nu^{\varepsilon} K) & \\ \qquad\quad \langle S, x^A, \mathsf{E}_{x^{K_A}}(x^N, x^B, K, \mathsf{E}_{x^{K_B}}(K, x^A))\rangle^{\updownarrow} & (2) \end{array}$$

In the second line the server consults a *Database* to obtain the master keys of the principals of the system; the code is as follows

$$\|_{i\in X}\,!\ \langle I_i, K_i\rangle^{\circ}$$

meaning that it constantly is willing to inform about the master keys for the various principals.

A.4 Needham-Schroeder asymmetric key protocol

There also exists a version of the Needham-Schroeder protocol relying on asymmetric key cryptography [28]. The protocol narration is as follows:

$$\begin{array}{l} 1.\ A \to S:\ A, B \\ 2.\ S \to A:\ \mathsf{S}_{K^-_S}(K^+_B, B) \\ 3.\ A \to B:\ \mathsf{P}_{K^+_B}(N_A, A) \\ 4.\ B \to S:\ B, A \\ 5.\ S \to B:\ \mathsf{S}_{K^-_S}(K^+_A, A) \\ 6.\ B \to A:\ \mathsf{P}_{K^+_A}(N_A, N_B) \\ 7.\ A \to B:\ \mathsf{P}_{K^+_B}(N_B) \end{array}$$

Both the principals A and B use the server S to obtain knowledge about each others public key (steps 1/2 and 4/5); the server signs its response with its private key thereby increasing the principals trust in the information. The remaining steps represent a nonce challenge relying on asymmetric key cryptography.

We shall use a scenario similar to the one above

$$\begin{array}{l} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ \quad \bullet \\ \mid\ (\nu^{\pm} K_S) \\ \qquad (\nu^{\varepsilon}_{i\in X}\ I_i)\,(\nu^{\pm}_{i\in X}\ K_i) \\ \qquad\quad \|_{i\in X}\ I_i\,[\,\|_{j\in X}\,!\mathit{Init}(I_i,I_j)\mid !\mathit{Resp}(I_i)\,] \\ \qquad \mid\qquad S\,[\,!\mathit{Server}\mid \mathit{Database}\,] \end{array}$$

so initially we make sure that the keys are in place (and they are not known to the attacker). As before each principal is ready to interact with any principal any number of times.

The definitions of $\mathit{Init}(I_i, I_j)$, $\mathit{Resp}(I_i)$, *Server* and *Database* reflect the details of the protocol and are specified below. For $\mathit{Init}(I_i, I_j)$ we have

$$\begin{array}{ll} \langle I_i, S, I_i, I_j\rangle^{\updownarrow}. & (1) \\ \quad (I_i, S, \mathsf{S}_{K^+_S}(_\%y^+_j, I_j))^{\updownarrow}. & (2) \\ \qquad (\nu^{\varepsilon} N^A_{ij})\,\langle I_i, I_j, \mathsf{P}_{y^+_j}(N^A_{ij}, I_i)\rangle^{\updownarrow}. & (3) \\ \qquad\quad (I_j, I_i, \mathsf{P}_{K^-_i}(N^A_{ij}, _\%v_{ij}))^{\updownarrow}. & (6) \\ \qquad\qquad \langle I_i, I_j, \mathsf{P}_{y^+_j}(v_{ij})\rangle^{\updownarrow}.\ \cdots & (7) \end{array}$$

so in step (2) the public key of the recipient is learnt in the variable y^+_j. For $\mathit{Resp}(I_j)$ we have

$$\begin{array}{ll} (_\%z^A_j, I_j, \mathsf{P}_{K^-_j}(_\%z^N_j, z^A_j))^{\updownarrow}. & (3) \\ \quad \langle I_j, S, I_j, z^A_j\rangle^{\updownarrow}. & (4) \\ \qquad (S, I_j, \mathsf{S}_{K^+_S}(_\%u^+_j, z^A_j))^{\updownarrow}. & (5) \\ \qquad\quad (\nu^{\varepsilon} N^B_j)\,\langle I_j, z^A_j, \mathsf{P}_{u^+_j}(z^N_j, N^B_j)\rangle^{\updownarrow}. & (6) \\ \qquad\qquad (z^A_j, I_j, \mathsf{P}_{K^-_j}(N^B_j))^{\updownarrow}.\ \cdots & (7) \end{array}$$

and here the public key of the initiator is learnt in the variable u^+_j. Finally, for *Server* and *Database* we have

$$\begin{array}{ll} (_\%x, S, x, _\%x')^{\updownarrow}. & (1/4) \\ \quad (x', _\%x^{K^+})^{\circ}. & \\ \qquad \langle S, x, \mathsf{S}_{K^-_S}(x^{K^+}, x')\rangle^{\updownarrow} & (2/5) \end{array}$$

and

$$\|_{i\in X}\,!\ \langle I_i, K^+_i\rangle^{\circ}$$

respectively.

A.5 Otway-Rees with hashing

In [25], Mao and Boyd describe how to encode protocols using cryptographic check functions rather than asymmetric key cryptography. One of the protocols that they

consider is the Otway-Rees protocol [35]:

$$\begin{array}{ll} 1.\ A \to B: & M, A, B, N_A \\ 2.\ B \to S: & M, A, B, N_A, N_B \\ 3.\ S \to B: & M, \mathsf{E}_{K_A}(K), \mathsf{H}(N_A, B, \mathsf{E}_{K_A}(K), K_A) \\ & \mathsf{E}_{K_B}(K), \mathsf{H}(N_B, A, \mathsf{E}_{K_B}(K), K_B) \\ 4.\ B \to A: & M, \mathsf{E}_{K_A}(K), \mathsf{H}(N_A, B, \mathsf{E}_{K_A}(K), K_A) \end{array}$$

Here M, N_A and N_B are nonces, K_A and K_B are master keys between the principals and the server and K is the session key created by the server. Again, the overall encoding takes the form:

$$\begin{array}{l} \mathsf{let}\ X = \mathbb{N}\ \mathsf{in} \\ \quad \bullet \\ \mid\ (\nu^{\varepsilon}_{i\in X}\ I_i)\,(\nu^{\varepsilon}_{i\in X}\ K_i) \\ \qquad \|_{i\in X}\ I_i\,[\,\|_{j\in X}\ !\mathit{Init}(I_i, I_j) \mid !\mathit{Resp}(I_i)\,] \\ \mid\quad S\,[\,!\mathit{Server} \mid \mathit{Database}\,] \end{array}$$

where the idea behind the abbreviations is as above. The initiator code $\mathit{Init}(I_i, I_j)$ is given by

$$\begin{array}{lr} (\nu^{\varepsilon} M_{ij})\,(\nu^{\varepsilon} N^A_{ij})\,\langle I_i, I_j, M_{ij}, I_i, I_j, N^A_{ij}\rangle^{\updownarrow}. & (1) \\ \quad (I_j, I_i, M_{ij}, \mathsf{E}_{K_i}(_\%u_{K_{ij}})\%u_{ij}, & \\ \qquad \mathsf{H}(N^A_{ij}, I_j, u_{ij}, K_i))^{\updownarrow}.\ \cdots & (4) \end{array}$$

The responder part $\mathit{Resp}(I_j)$ is:

$$\begin{array}{lr} (_\%x^A, I_j, x^M_j, x^A, I_j, x^N_j)^{\updownarrow}. & (1) \\ \quad (\nu^{\varepsilon} N^B_j)\,\langle I_j, S, x^M_j, x^A_j, I_j, x^N_j, N^B_j\rangle^{\updownarrow}. & (2) \\ \qquad (S, I_j, x^M_j, _\%z_j, _\%z'_j, & \\ \qquad\quad \mathsf{E}_{K_j}(_\%z^K_j)\%z''_j, \mathsf{H}(N^B_j, x^A_j, z''_j, K_j))^{\updownarrow}. & (3) \\ \qquad \langle I_j, x^A_j, x^M_j, z_j, z'_j\rangle^{\updownarrow}.\ \cdots & (4) \end{array}$$

Finally, the server part uses a database similar to the one for the Needham-Schroeder symmetric key protocol and is coded as follows:

$$\begin{array}{lr} (_\%y^B, S, _\%y^M, _\%y^A, y^B, _\%y^N, _\%y^{N'})^{\updownarrow} & (2) \\ \quad (y^A, _\%y^{K_A})^{\circ}.\ (y^B, _\%y^{K_B})^{\circ}. & \\ \qquad (\nu^{\varepsilon} K)\,\langle S, y^B, y^M, & \\ \qquad\quad \mathsf{E}_{y^{K_A}}(K), \mathsf{H}(y^N, y^B, \mathsf{E}_{y^{K_A}}(K), y^{K_A}), & \\ \qquad\quad \mathsf{E}_{y^{K_B}}(K), \mathsf{H}(y^{N'}, y^A, \mathsf{E}_{y^{K_B}}(K), y^{K_B})\rangle^{\updownarrow} & (3) \end{array}$$

B Well-formedness of Meta-level Processes

Any well-formed meta-level process must be fully closed meaning that there are no free names and no free variables. Note that since the attacker is part of the meta-level process, this does not mean that the overall meta-level process represents a closed system; it just means that the protocol designer specifically tells where the system is open for attack and that the system is closed otherwise. The aim of the well-formedness condition is to ensure that names, variable, indices, and indexing set identifiers only are used after they have been introduced.

Example 12 The meta-level process

$$\begin{array}{l} \mathsf{let}\ X = \{a\}\ \mathsf{in} \\ \mathsf{let}\ X' = \{a, b\}\ \mathsf{in} \\ \quad (\nu^{\varepsilon}_{i\in X}\ I_i)\ \|_{i'\in X'}\ I_{i'}\,[\,0\,] \end{array}$$

is *not well-formed* since (by taking X to be $\{a\}$ and X' to be $\{b\}$) it instantiates to the process $(\nu^{\varepsilon} I_a)\, I_b\,[\,0\,]$ where I_b is free. It does not matter that the meta-level process also instantiates to $(\nu^{\varepsilon} I_a)\, I_a\,[\,0\,]$ where the name I_a is indeed restricted, since we require that *all* instances must be closed. On the other hand, the meta-level process

$$\begin{array}{l} \mathsf{let}\ X = \{a, b\}\ \mathsf{in} \\ \quad (\nu^{\varepsilon}_{i\in X}\ I_i)\ \|_{i'\in X}\ I_{i'}\,[\,0\,] \end{array}$$

is *well-formed* since the same indexing set identifier is used both in restriction and in parallel composition and this ensures that $I_{i'}$ is bound in all instances.

The overall idea in the formalisation of the well-formedness requirement is to collect the sets of names, variables, etc. that have been introduced and then to check that *applied occurrences* of these entities are indeed included in these sets. For this, we define a judgement in Table 13 of the form

$$N, V \vdash^{\Delta}_{\delta} L$$

where N is a set of names, V is a set of variables, and Δ is a set of indexing identifiers. All these sets have been previously defined and may be used in L.

The indices that have been previously introduced are represented by δ as a mapping of indices to indexing set identifiers. As usual, we write $[\,]$ for the empty mapping, $\delta[i \mapsto X]$ for update, and $\delta(i)$ for querying a mapping. We shall frequently want to check whether all indices in a sequence $\bar{i} = i_1 \cdots i_j$ have been define i.e. whether they are all in the domain of δ. For this, we introduce the predicate $\bar{i} \trianglelefteq \delta$ that holds whenever $\{i_1, \cdots, i_k\} \subseteq \mathrm{dom}(\delta)$. Checks of the correct use of indices on names and variables are only done at *defining occurrences.*

If the variable x_i, for example, is in a defining position and the index i has been introduced then V is updated to include x_i. This updated set is then consulted whenever an applied occurrence of a variable is found and, in particular, the variable x_i will be allowed.

For names, we update the set N in a similar fashion whenever a name is defined by the ordinary restriction operator. When the name m_i is introduced using the indexed restriction operator with $i \in X$ we shall include m_X in N to signify that m has been introduced for all indices in X. Correspondingly, we define a set-membership operator that respects the declarations in δ: we let $n \in_{\delta} N$ mean that

(1) n is of the form $m_{i_1 \cdots i_k}$, and
(2) there is a unique $m_{j_1 \cdots j_l}$ in N,
(3) and that $k = l$ and $\wedge^k_{o=1}(j_o = i_o \vee j_o = \delta(i_o))$

Well-formedness of terms are given as the judgement	Well-formedness of patterns are given as the judgement
$N, V \vdash_\delta t$	$N, V \vdash_\delta p : V'$
that holds if $\mathsf{fn}(t) \subseteq_\delta N$ and $\mathsf{fv}(t) \subseteq V$.	that holds if $\mathsf{fn}(p) \subseteq_\delta N$, $\mathsf{fv}(p) \subseteq V$, $\mathsf{dv}(p) \subseteq V'$, and for all $x_{\bar{i}} \in V'$ it holds that $\bar{i} \trianglelefteq \delta$ and that no $x_{\bar{j}}$ is in V.

Table 11 Well-formedness of terms and patterns; $N, V \vdash_\delta t$ and $N, V \vdash_\delta p : V'$, respectively.

$$\frac{N, V \vdash_\delta P_1 \qquad N, V \vdash_\delta P_2}{N, V \vdash_\delta P_1 \mid P_2} \qquad \frac{N, V \vdash_\delta P}{N, V \vdash_\delta !P} \qquad N, V \vdash_\delta \mathsf{0} \qquad \frac{n \in_\delta N \qquad N, V \vdash_\delta P}{N, V \vdash_\delta n\,[\,P\,]}$$

$$\frac{N \cup \{m_{\bar{i}}^\tau \mid \tau \in T\}, V \vdash_\delta P}{N, V \vdash_\delta (\nu^T m_{\bar{i}})\, P} \ \text{if } \bar{i} \trianglelefteq \delta \text{ and } \forall \delta' : \forall \tau \in T : m_{\delta'}^\tau \notin N \qquad \frac{N, V \vdash_\delta t \qquad N, V \vdash_\delta p : V' \qquad N, V \cup V' \vdash_\delta P}{N, V \vdash_\delta t \,\mathsf{as}\, p.P}$$

$$\frac{N, V \vdash_\delta t_1 \quad \cdots \quad N, V \vdash_\delta t_k \quad N, V \vdash_\delta P}{N, V \vdash_\delta \langle t_1, \cdots, t_k \rangle^\eta.P}$$

$$\frac{N, V \vdash_\delta p_1 : V_1 \quad N, V \cup V_1 \vdash_\delta p_2 : V_2 \quad \cdots \quad N, V \cup V_1 \cup \cdots \cup V_{k-1} \vdash_\delta p_k : V_k \quad N, V \cup V_1 \cup \cdots \cup V_k \vdash_\delta P}{N, V \vdash_\delta (p_1, \cdots, p_k)^\eta.P}$$

Table 12 Well-formedness of processes; $N, V \vdash_\delta P$.

$$\frac{N, V \vdash_\delta^{\Delta \cup \{X\}} L}{N, V \vdash_\delta^\Delta \mathsf{let}\ X = S \ \mathsf{in}\ L} \ \text{if } \lfloor S \rfloor \text{ is finite.} \qquad \frac{N, V \vdash_{\delta[i \mapsto X]}^\Delta L}{N, V \vdash_\delta^\Delta \|_{i \in X}\ L} \ \text{if } X \in \Delta$$

$$\frac{N \cup \{m_{\bar{j}\bar{X}}^\tau \mid \tau \in T\}, V \vdash_\delta^\Delta L}{N, V \vdash_\delta^\Delta (\nu_{\bar{i} \in \bar{X}}^T\, m_{\bar{j}\bar{i}})\, L} \ \text{if } \overline{X} \in \Delta^* \text{ and } \bar{j} \trianglelefteq \bar{\delta} \text{ and } \forall \delta' : \forall \tau \in T : m_{\delta'}^\tau \notin N$$

$$\frac{N, V \vdash_\delta^\Delta L_1 \qquad N, V \vdash_\delta^\Delta L_2}{N, V \vdash_\delta^\Delta L_1 \mid L_2} \qquad \frac{N \cup \{m_{\bar{i}}^\tau \mid \tau \in T\}, V \vdash_\delta^\Delta L}{N, V \vdash_\delta^\Delta (\nu^T m_{\bar{i}})\, L} \ \text{if } \bar{i} \trianglelefteq \delta \text{ and } \forall \delta' : \forall \tau \in T : m_{\delta'}^\tau \notin N$$

$$\frac{n \in_\delta N \qquad N, V \vdash_\delta^\Delta L}{N, V \vdash_\delta^\Delta n\,[\,L\,]} \qquad \frac{N, V \vdash_\delta P}{N, V \vdash_\delta^\Delta P}$$

Table 13 Well-formedness of meta-level processes; $N, V \vdash_\delta^\Delta L$.

For example, checking well-formedness of

$$(\nu_{i \in X}^\varepsilon\ I_i)\ \|_{i' \in X}\ I_{i'}\,[\,\mathsf{0}\,]$$

will firstly update N with I_X, secondly update δ with $[i' \mapsto X]$, and finally check that $I_{i'} \in_\delta N$ at the applied occurrence of $I_{i'}$ in the ambient construct. The check succeeds since (1) $I_{i'}$ has the right form, (2) correspondingly I_X is in N, and (3) X is equal to $\delta(i')$. Furthermore, we will write $\subseteq_\delta$ for its point-wise extension.

The well-formedness condition ensures not only that names and variables are introduced before they are used but also that base names and variables are not redefined. This is ensured by the side-conditions $\forall \delta' : \forall \tau \in T : m_{\delta'}^\tau \notin N$ to the rules for restriction in Table 12 and Table 13 .

The definition of well-formedness for the object-level processes uses a corresponding judgement $N, V \vdash_\delta P$ defined in Table 12. It works in a similar fashion except that the component Δ is not needed since object-level processes do not use indexing set identifiers. The definition of $N, V \vdash_\delta P$ relies on well-formedness judgements for terms and patterns as defined in Table 11.

Accompanying this paper are appendices C and D to be published in electronic form.

C Free Names and Free Variables

Free names and free variables are given by the functions fn and fv, respectively. They are defined structurally on the syntax of terms, patterns, and object-level processes. The definition does not distinguish between patterns, constructive patterns and signature patterns but regards them as instances of the same syntactic category.

The definition of free names is straightforward and is given in Table 14. The definition of free variables in Table 15 and has to ensure left-to-right scoping of variables defined in patterns as discussed in the main text. For this, the definition uses the function $\mathsf{dv}(p)$ that collects the variables that occur in a defining position in the pattern p.

D Control Flow Analysis

Recall from the main text that the analysis is based on canonical names and canonical variables both of which are stable under α-conversion. The canonical operator, $\lfloor\cdot\rfloor$, defined on variables and names is extended homomorphically to all syntactic categories and it is extended in a point-wise manner to sets of entities from these categories. We write $\lfloor\mathsf{Name}\rfloor$ for the set $\{\lfloor n\rfloor \mid n \in \mathsf{Name}\}$ and similarly $\lfloor\mathsf{Val}\rfloor$ for $\{\lfloor v\rfloor \mid v \in \mathsf{Val}\}$.

Patterns. The analysis will work with abstract environments of the functionality:

$$\xi, \sigma, \rho : \lfloor\mathsf{Var}\rfloor \to \mathcal{P}(\lfloor\mathsf{Val}\rfloor)$$

The clauses for the analysis of pattern matching have the form

$$\xi \models v \rhd p : s$$

and express the matching of the value $v \in \lfloor\mathsf{Val}\rfloor$ against the pattern $p \in \mathsf{Pat}$; the set $s \subseteq \lfloor\mathsf{Val}\rfloor$ is an over-approximation of the set of values that matches p. Hence s will contain v if the match of v against p is successful; otherwise it is not necessarily the case. The judgements are defined in Table 16 and briefly explained below.

First consider the judgement for matching a value v against the pattern n. Here it is required that if v matches the canonical name of n then v must also be in s. If v does not equal $\lfloor n\rfloor$ then there are no requirements on s.

The clause for matching a value v and binding of a variable in the pattern $p\%x$ first checks whether v matches p in the analysis. The successful matches are reported in s' and hence the elements of s' will not only be potential values bound to x but they are also potential successful matches of $p\%x$. Clearly, we do not want to require that the pattern matching must succeed for *all* s'; hence s' is existentially quantified in the manner of [29].

The analysis of matching of the value v against one of the composite patterns all have the same form. First they require that v has a particular form where each subterm matches the corresponding subpattern. Only if all these matches succeed then it is required that v is in the set s of successful matches.

Terms. The analysis of pattern matching is concerned with a single value but in general the analysis has to be concerned with terms. Therefore we shall need a way of finding the set $\xi[\![t]\!] \subseteq \lfloor\mathsf{Val}\rfloor$ of values that a term may denote for a given abstract environment ξ. The definition is given in Table 17 and amounts to a straightforward structural induction. Note that if t has no free variables then $\xi[\![t]\!] = \{\lfloor t\rfloor\}$.

Object-level processes. The analysis of object-level processes uses a network component of the functionality

$$\kappa : (\lfloor\mathsf{Name}\rfloor \cup \{\updownarrow\}) \to \mathcal{P}(\lfloor\mathsf{Val}\rfloor^*)$$

where $\updownarrow$ is an element not in $\lfloor\mathsf{Name}\rfloor$. Intuitively, $\kappa(n)$ describes the communication that takes place inside all ambients with the canonical name n while $\kappa(\updownarrow)$ describes the communication on the global network. Thus, local communication within several ambients with the same canonical names will be represented by the same component in the analysis. Similarly, all communication on the global ether is recorded in $\kappa(\updownarrow)$ though semantically there may be different "global" networks at different levels in the ambient hierarchy. In both cases this may result in over-approximations.

The analysis is defined by the judgements in Table 18 of the form:

$$(\rho, \sigma, \kappa) \models^n P$$

Here, $n \in (\lfloor\mathsf{Name}\rfloor \cup \{\top\})$ is the name of the ambient where the process P is located and $\top$ is an element not in $\lfloor\mathsf{Name}\rfloor$ indicating that P is at the top level. As explained in Section 5 the analysis operates with two abstract environments: σ contains the potential bindings for which the pattern matching will succeed whereas ρ is a refined version of σ eliminating some of the non-successful bindings.

The clauses of Table 18 are defined in a structural way with output, input and matching being the interesting ones. First consider the clause for output. The sets $\rho[\![t_i]\!]$ contain the potential values of the terms t_i that may be sent on the ether and the clause simply checks that they are present in κ for the appropriate entry and for this we use the auxiliary function $c(\eta, n)$ defined by

$$c(\eta, n) = \begin{cases} \lfloor n\rfloor & \mathsf{if}\ \eta = \circ \\ \updownarrow & \mathsf{if}\ \eta = \updownarrow \end{cases}$$

The clause for input begins by demanding that all information in ρ also have to be present in σ and that all potential values to be input by the process are present in the κ component. For each combination we first use the σ component to approximate the potential successful

Processes		
$\mathsf{fn}(P_1 \mid P_2)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(P_1) \cup \mathsf{fn}(P_2)$
$\mathsf{fn}(!P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(P)$
$\mathsf{fn}(0)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fn}((\nu^T m)\, P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(P) \setminus \{m^\tau \mid \tau \in T\}$
$\mathsf{fn}(n\,[\,P\,])$	$\stackrel{\text{def}}{=}$	$\{n\} \cup \mathsf{fn}(P)$
$\mathsf{fn}(t\,\mathsf{as}\,p.P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t) \cup \mathsf{fn}(p) \cup \mathsf{fn}(P)$
$\mathsf{fn}(\langle t_1, \cdots, t_k\rangle^\eta.P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t_1) \cup \cdots \cup \mathsf{fn}(t_k) \cup \mathsf{fn}(P)$
$\mathsf{fn}((p_1, \cdots, p_k)^\eta.P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p_1) \cup \cdots \cup \mathsf{fn}(p_k) \cup \mathsf{fn}(P)$

Terms		
$\mathsf{fn}(n)$	$\stackrel{\text{def}}{=}$	$\{n\}$
$\mathsf{fn}(x)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fn}(\mathsf{T}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t_1) \cup \cdots \cup \mathsf{fn}(t_k)$
$\mathsf{fn}(\mathsf{H}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t_1) \cup \cdots \cup \mathsf{fn}(t_k)$
$\mathsf{fn}(\mathsf{E}_{t_0}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t_0) \cup \cdots \cup \mathsf{fn}(t_k)$
$\mathsf{fn}(\mathsf{P}_{t_0}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t_0) \cup \cdots \cup \mathsf{fn}(t_k)$
$\mathsf{fn}(\mathsf{S}_{t_0}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(t_0) \cup \cdots \cup \mathsf{fn}(t_k)$

Patterns		
$\mathsf{fn}(n)$	$\stackrel{\text{def}}{=}$	$\{n\}$
$\mathsf{fn}(x)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fn}(_)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fn}(p\%x)$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p)$
$\mathsf{fn}(\mathsf{T}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p_1) \cup \cdots \cup \mathsf{fn}(p_k)$
$\mathsf{fn}(\mathsf{H}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p_1) \cup \cdots \cup \mathsf{fn}(p_k)$
$\mathsf{fn}(\mathsf{E}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p_0) \cup \cdots \cup \mathsf{fn}(p_k)$
$\mathsf{fn}(\mathsf{P}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p_0) \cup \cdots \cup \mathsf{fn}(p_k)$
$\mathsf{fn}(\mathsf{S}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fn}(p_0) \cup \cdots \cup \mathsf{fn}(p_k)$

Table 14 Free names; $\mathsf{fn}(P)$, $\mathsf{fn}(t)$, $\mathsf{fn}(p)$.

Processes		
$\mathsf{fv}(P_1 \mid P_2)$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(P_1) \cup \mathsf{fv}(P_2)$
$\mathsf{fv}(!P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(P)$
$\mathsf{fv}(0)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fv}((\nu^T m)\, P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(P)$
$\mathsf{fv}(n\,[\,P\,])$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(P)$
$\mathsf{fv}(t\,\mathsf{as}\,p.P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t) \cup \mathsf{fv}(p) \cup (\mathsf{fv}(P) \setminus \mathsf{dv}(p))$
$\mathsf{fv}(\langle t_1, \cdots, t_k\rangle^\eta.P)$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t_1) \cup \cdots \cup \mathsf{fv}(t_k) \cup \mathsf{fv}(P)$
$\mathsf{fv}((p_1, \cdots, p_k)^\eta.P)$	$\stackrel{\text{def}}{=}$	$\cup_{i=1}^{k}(\mathsf{fv}(p_i) \setminus \cup_{j=1}^{i-1}\mathsf{dv}(p_j))$ $\cup\, (\mathsf{fv}(P) \setminus (\cup_{i=1}^{k}\mathsf{dv}(p_i))$

Terms		
$\mathsf{fv}(n)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fv}(x)$	$\stackrel{\text{def}}{=}$	$\{x\}$
$\mathsf{fv}(\mathsf{T}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t_1) \cup \cdots \cup \mathsf{fv}(t_k)$
$\mathsf{fv}(\mathsf{H}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t_1) \cup \cdots \cup \mathsf{fv}(t_k)$
$\mathsf{fv}(\mathsf{E}_{t_0}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t_0) \cup \cdots \cup \mathsf{fv}(t_k)$
$\mathsf{fv}(\mathsf{P}_{t_0}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t_0) \cup \cdots \cup \mathsf{fv}(t_k)$
$\mathsf{fv}(\mathsf{S}_{t_0}(t_1, \cdots, t_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(t_0) \cup \cdots \cup \mathsf{fv}(t_k)$

Patterns		
$\mathsf{fv}(n)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fv}(x)$	$\stackrel{\text{def}}{=}$	$\{x\}$
$\mathsf{fv}(_)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{fv}(p\%x)$	$\stackrel{\text{def}}{=}$	$\mathsf{fv}(p)$
$\mathsf{fv}(\mathsf{T}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\cup_{i=1}^{k}(\mathsf{fv}(p_i) \setminus \cup_{j=1}^{i-1}\mathsf{dv}(p_j))$
$\mathsf{fv}(\mathsf{H}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\cup_{i=1}^{k}(\mathsf{fv}(p_i) \setminus \cup_{j=1}^{i-1}\mathsf{dv}(p_j))$
$\mathsf{fv}(\mathsf{E}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\cup_{i=0}^{k}(\mathsf{fv}(p_i) \setminus \cup_{j=0}^{i-1}\mathsf{dv}(p_j))$
$\mathsf{fv}(\mathsf{P}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\cup_{i=0}^{k}(\mathsf{fv}(p_i) \setminus \cup_{j=0}^{i-1}\mathsf{dv}(p_j))$
$\mathsf{fv}(\mathsf{S}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\cup_{i=0}^{k}(\mathsf{fv}(p_i) \setminus \cup_{j=0}^{i-1}\mathsf{dv}(p_j))$

Defined variables		
$\mathsf{dv}(n)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{dv}(x)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{dv}(_)$	$\stackrel{\text{def}}{=}$	$\emptyset$
$\mathsf{dv}(p\%x)$	$\stackrel{\text{def}}{=}$	$\mathsf{dv}(p) \cup \{x\}$
$\mathsf{dv}(\mathsf{T}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{dv}(p_1) \cup \cdots \cup \mathsf{dv}(p_k)$
$\mathsf{dv}(\mathsf{H}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{dv}(p_1) \cup \cdots \cup \mathsf{dv}(p_k)$
$\mathsf{dv}(\mathsf{E}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{dv}(p_0) \cup \cdots \cup \mathsf{dv}(p_k)$
$\mathsf{dv}(\mathsf{P}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{dv}(p_0) \cup \cdots \cup \mathsf{dv}(p_k)$
$\mathsf{dv}(\mathsf{S}_{p_0}(p_1, \cdots, p_k))$	$\stackrel{\text{def}}{=}$	$\mathsf{dv}(p_0) \cup \cdots \cup \mathsf{dv}(p_k)$

Table 15 Free variables; $\mathsf{fv}(P)$, $\mathsf{fv}(t)$, $\mathsf{fv}(p)$. Note that patterns have left-to-right scoping of variables.

matches (in s_1) and subsequently we use the ρ component to pinpoint a subset s_2 of s_1. Only if that set is non-empty we require that the continuation P should be analysed.

The clause for pattern matching is analogous to the one for input. So first we state a condition forcing information from ρ also to be present in σ. We then consult all the potential values $\rho[\![t]\!]$ of the term t and check, using σ, whether the match succeeds and we then refine the set s_1 of successful matches using the abstract environment ρ. Only if the resulting set s_2 is non-empty we require that the continuation should be analysed.

Meta-level processes. A meta-level process is analysed with judgements of the form

$$(\rho, \sigma, \kappa) \models^{n}_{\Gamma,\mathcal{N}} L$$

$$
\begin{array}{lll}
\xi \models v \rhd n : s & \text{iff} & v = \lfloor n \rfloor \Rightarrow v \in s \\
\xi \models v \rhd x : s & \text{iff} & v \in \xi(\lfloor x \rfloor) \Rightarrow v \in s \\
\xi \models v \rhd _ : s & \text{iff} & v \in s \\
\xi \models v \rhd p\%x : s & \text{iff} & \exists s' : \xi \models v \rhd p : s' \wedge s' \subseteq \xi(x) \wedge s' \subseteq s \\
\xi \models v \rhd \mathsf{T}(p_1, \cdots, p_k) : s & \text{iff} & v = \mathsf{T}(v_1, \cdots, v_k) \Rightarrow \\
& & \quad \exists s_1 : \xi \models v_1 \rhd p_1 : s_1 \wedge (v_1 \in s_1 \Rightarrow \\
& & \quad\quad \exists s_2 : \xi \models v_2 \rhd p_2 : s_2 \wedge (v_2 \in s_2 \Rightarrow \\
& & \quad\quad \cdots \exists s_k : \xi \models v_k \rhd p_k : s_k \wedge (v_k \in s_k \Rightarrow \\
& & \quad\quad\quad \mathsf{T}(v_1, \cdots, v_k) \in s) \cdots)) \\
\xi \models v \rhd \mathsf{H}(p_1, \cdots, p_k) : s & \text{iff} & v = \mathsf{H}(v_1, \cdots, v_k) \Rightarrow \\
& & \quad \exists s_1 : \xi \models v_1 \rhd p_1 : s_1 \wedge (v_1 \in s_1 \Rightarrow \\
& & \quad\quad \exists s_2 : \xi \models v_2 \rhd p_2 : s_2 \wedge (v_2 \in s_2 \Rightarrow \\
& & \quad\quad \cdots \exists s_k : \xi \models v_k \rhd p_k : s_k \wedge (v_k \in s_k \Rightarrow \\
& & \quad\quad\quad \mathsf{H}(v_1, \cdots, v_k) \in s) \cdots)) \\
\xi \models v \rhd \mathsf{E}_{p_0}(p_1, \cdots, p_k) : s & \text{iff} & v = \mathsf{E}_n(v_1, \cdots, v_k) \Rightarrow \\
& & \quad \exists s_0 : \xi \models n \rhd p_0 : s_0 \wedge (n \in s_0 \Rightarrow \\
& & \quad\quad \exists s_1 : \xi \models v_1 \rhd p_1 : s_1 \wedge (v_1 \in s_1 \Rightarrow \\
& & \quad\quad \cdots \exists s_k : \xi \models v_k \rhd p_k : s_k \wedge (v_k \in s_k \Rightarrow \\
& & \quad\quad\quad \mathsf{E}_n(v_1, \cdots, v_k) \in s) \cdots)) \\
\xi \models v \rhd \mathsf{P}_{p_0}(p_1, \cdots, p_k) : s & \text{iff} & v = \mathsf{P}_{m^+}(v_1, \cdots, v_k) \Rightarrow \\
& & \quad \exists s_0 : \xi \models m^- \rhd p_0 : s_0 \wedge (m^- \in s_0 \Rightarrow \\
& & \quad\quad \exists s_1 : \xi \models v_1 \rhd p_1 : s_1 \wedge (v_1 \in s_1 \Rightarrow \\
& & \quad\quad \cdots \exists s_k : \xi \models v_k \rhd p_k : s_k \wedge (v_k \in s_k \Rightarrow \\
& & \quad\quad\quad \mathsf{P}_{m^+}(v_1, \cdots, v_k) \in s) \cdots)) \\
\xi \models v \rhd \mathsf{S}_{p_0}(p_1, \cdots, p_k) : s & \text{iff} & v = \mathsf{S}_{m^-}(v_1, \cdots, v_k) \Rightarrow \\
& & \quad \exists s_0 : \xi \models m^+ \rhd p_0 : s_0 \wedge (m^+ \in s_0 \Rightarrow \\
& & \quad\quad \exists s_1 : \xi \models v_1 \rhd p_1 : s_1 \wedge (v_1 \in s_1 \Rightarrow \\
& & \quad\quad \cdots \exists s_k : \xi \models v_k \rhd p_k : s_k \wedge (v_k \in s_k \Rightarrow \\
& & \quad\quad\quad \mathsf{S}_{m^-}(v_1, \cdots, v_k) \in s) \cdots))
\end{array}
$$

Table 16 Analysis of pattern matching; $\xi \models v \rhd p : s$.

$$
\begin{array}{rcl}
\xi[\![n]\!] & \stackrel{\text{def}}{=} & \{\lfloor n \rfloor\} \\
\xi[\![x]\!] & \stackrel{\text{def}}{=} & \xi(\lfloor x \rfloor) \\
\xi[\![\mathsf{T}(t_1, \cdots, t_k)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{T}(v_1, \cdots, v_k) \mid v_1 \in \xi[\![t_1]\!], \cdots, v_k \in \xi[\![t_k]\!]\} \\
\xi[\![\mathsf{H}(t_1, \cdots, t_k)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{H}(v_1, \cdots, v_k) \mid v_1 \in \xi[\![t_1]\!], \cdots, v_k \in \xi[\![t_k]\!]\} \\
\xi[\![\mathsf{E}_{t_0}(t_1, \cdots, t_k)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{E}_{v_0}(v_1, \cdots, v_k) \mid v_0 \in \xi[\![t_0]\!], \cdots, v_k \in \xi[\![t_k]\!]\} \\
\xi[\![\mathsf{P}_{t_0}(t_1, \cdots, t_k)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{P}_{v_0}(v_1, \cdots, v_k) \mid v_0 \in \xi[\![t_0]\!], \cdots, v_k \in \xi[\![t_k]\!]\} \\
\xi[\![\mathsf{E}_{t_0}(t_1, \cdots, t_k)]\!] & \stackrel{\text{def}}{=} & \{\mathsf{E}_{v_0}(v_1, \cdots, v_k) \mid v_0 \in \xi[\![t_0]\!], \cdots, v_k \in \xi[\![t_k]\!]\}
\end{array}
$$

Table 17 Evaluating a term given the environment ξ; $\xi[\![t]\!]$.

where as before $n \in \lfloor \mathsf{Name} \rfloor \cup \{\top\}$ is the enclosing ambient or the special element $\top$ and also the components ρ, σ and κ are as above. The component Γ is a mapping of indexing set identifiers (i.e. X's) to sets of canonical indices whereas the component $\mathcal{N} \subseteq \lfloor \mathsf{Name} \rfloor$ contains the set of (canonical) names that are allowed within L. The judgements for the analysis of the meta-level processes are defined in Table 19 and we shall comment on the clauses below.

Most of the clauses amount to a straightforward structural induction on the form of the processes. In the case of the let-construct the environment Γ is updated so that X is mapped to the *canonical* set of indices associated with S whereas the $\mathcal{N}$ component is unchanged. In the case of the indexing restriction construct the Γ component is unchanged but the $\mathcal{N}$ component is modified to reflect the new names introduced by the construct. The same holds for the classical restriction construct.

The analysis of the indexing parallel construct demands that the meta-level process can be analysed for each of the finitely many canonical indices of relevance; an appropriate number of copies of the process is then constructed and analysed. The ordinary parallel composition of two processes is analysable if both components are.

$$
\begin{array}{lll}
(\rho,\sigma,\kappa) \models^n P_1 \mid P_2 & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n P_1 \wedge (\rho,\sigma,\kappa) \models^n P_2 \\
(\rho,\sigma,\kappa) \models^n !P & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n P \\
(\rho,\sigma,\kappa) \models^n \mathsf{0} & \mathsf{iff} & \mathsf{true} \\
(\rho,\sigma,\kappa) \models^n (\nu^T m)\, P & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n P \\
(\rho,\sigma,\kappa) \models^n n'\,[\,P\,] & \mathsf{iff} & (\rho,\sigma,\kappa) \models^{\lfloor n' \rfloor} P \\
(\rho,\sigma,\kappa) \models^n \langle t_1,\cdots,t_k\rangle^\eta.P & \mathsf{iff} & \forall v_1 \in \rho[\![t_1]\!],\cdots,v_k \in \rho[\![t_k]\!] : \\
& & \quad v_1\cdots v_k \in \kappa(c(\eta,n)) \wedge \\
& & (\rho,\sigma,\kappa) \models^n P \\
\\
(\rho,\sigma,\kappa) \models^n (p_1,\cdots,p_k)^\eta.P & \mathsf{iff} & \exists s_1,s_2 : (\\
& & \forall x : \rho(x) \subseteq \sigma(x) \wedge \\
& & \quad \forall v_1\cdots v_k \in \kappa(c(\eta,n)) : \\
& & \qquad \sigma \models \mathsf{T}(v_1,\cdots,v_k) \rhd \mathsf{T}(p_1,\cdots,p_k) : s_1 \wedge \\
& & \qquad \forall v \in s_1 : \rho \models v \rhd \mathsf{T}(p_1,\cdots,p_k) : s_2 \wedge \\
& & \qquad\quad s_2 \neq \emptyset \Rightarrow (\rho,\sigma,\kappa) \models^n P) \\
\\
(\rho,\sigma,\kappa) \models^n t\,\mathsf{as}\,p.P & \mathsf{iff} & \exists s_1,s_2 : (\\
& & \forall x : \rho(x) \subseteq \sigma(x) \wedge \\
& & \quad \forall v_1 \in \rho[\![t]\!] : \sigma \models v_1 \rhd p : s_1 \wedge \\
& & \quad \forall v_2 \in s_1 : \rho \models v_2 \rhd p : s_2 \wedge \\
& & \qquad s_2 \neq \emptyset \Rightarrow (\rho,\sigma,\kappa) \models^n P)
\end{array}
$$

Table 18 Analysis of object-level processes; $(\rho,\sigma,\kappa) \models^n P$.

$$
\begin{array}{lll}
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} \mathsf{let}\ X = S\ \mathsf{in}\ L & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n_{\Gamma[X \mapsto \lfloor S \rfloor],\mathcal{N}} L \\
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} (\nu^T_{i \in \overline{X}}\ m_{\overline{bi}})\, L & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N} \cup \{\lfloor m^\tau_{\overline{ba}} \rfloor \mid \tau \in T \wedge \overline{a} \in \Gamma(\overline{X})\}} L \\
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} \|_{i \in X}\ L & \mathsf{iff} & \wedge_{a \in \Gamma(X)} (\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} L\langle i \mapsto a\rangle \\
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} L_1 \mid L_2 & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} L_1 \wedge (\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} L_2 \\
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} (\nu^T m)\, L & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N} \cup \{\lfloor m^\tau \rfloor \mid \tau \in T\}} L \\
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} n'\,[\,L\,] & \mathsf{iff} & (\rho,\sigma,\kappa) \models^{\lfloor n' \rfloor}_{\Gamma,\mathcal{N}} L \\
(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} P & \mathsf{iff} & (\rho,\sigma,\kappa) \models^n P
\end{array}
$$

Table 19 Analysis of meta-level processes; $(\rho,\sigma,\kappa) \models^n_{\Gamma,\mathcal{N}} L$.

Finally, the analysis of the ambient process requires that the encapsulated process is analysable but in a modified context. The final clause of Table 19 simply exploits the analysis of the object-level processes.

The Attacker. In the specification of the protocol the presence of the attacker is indicated by $\bullet$ and, hence, we are going to specify a number of clauses that at the level of the analysis describes what the attacker might do. We shall follow the Dolev-Yao approach [13] and specify the different abilities of the attacker separately. We also investigated this approach in [7] where we proved that this formulation corresponds to the notion of a "hardest" attacker [31] i.e. it corresponds to the combination of analysis result for all processes that may take the place of the attacker.

The clause defining the analysis of the attacker is given in Table 20 and consists of seven components. Conceptually, these Dolev-Yao conditions are written as if an attacker stores its knowledge in private variables and uses this knowledge to send messages, generate new knowledge, etc. These variables will all be represented by a single canonical variable $\mathsf{x}_\bullet$ and, thus, $\rho(\mathsf{x}_\bullet)$ contains the complete knowledge of the attacker seen from the point of view of the analysis.

The first two components express that the attacker has knowledge of all the free names (as specified by $\mathcal{N}$) and that it can construct new names provided they have one of three canonical names $\{\mathsf{n}^\varepsilon_\bullet, \mathsf{n}^+_\bullet, \mathsf{n}^-_\bullet\}$.

The next four clauses express that the attacker can construct new composite values from his knowledge, that he can destruct composite values that he may happen to know (provided that he also has knowledge of the required keys), he can eavesdrop on all communication (including local communication if he knows the ambient names) and finally that he can manufacture messages to be sent on these networks as well. Note that these four conditions are written to hold for all natural numbers k. However, when a particular processes is analysed, this may be limited to be the largest arity used in process that is under attack (and is larger than some fixed

$(\rho, \sigma, \kappa) \models^{n}_{\Gamma,\mathcal{N}} \bullet$ iff $\mathcal{N} \subseteq \rho(\mathsf{x}_\bullet) \wedge$	knows bound names where it is placed
$\{\mathsf{n}^{\varepsilon}_\bullet, \mathsf{n}^{+}_\bullet, \mathsf{n}^{-}_\bullet\} \subseteq \rho(\mathsf{x}_\bullet) \wedge$	create own names
$\forall k \geq 0 : \forall v_0 \in \rho(\mathsf{x}_\bullet), \cdots, v_k \in \rho(\mathsf{x}_\bullet) :$ $\mathsf{T}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \wedge$ $\mathsf{H}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \wedge$ $\mathsf{E}_{v_0}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \wedge$ $\mathsf{P}_{v_0}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \wedge$ $\mathsf{S}_{v_0}(v_1, \cdots, v_k) \in \rho(\mathsf{x}_\bullet) \wedge$	construct composite values
$\forall v \in \rho(\mathsf{x}_\bullet) :$ $\forall k \geq 0 : \forall v_0, \cdots, v_k :$ $(v = \mathsf{T}(v_1, \cdots, v_k) \vee$ $(v = \mathsf{E}_{v_0}(v_1, \cdots, v_k) \wedge v_0 \in \rho(\mathsf{x}_\bullet)) \vee$ $(v = \mathsf{P}_{m^+}(v_1, \cdots, v_k) \wedge m^- \in \rho(\mathsf{x}_\bullet)) \vee$ $v = \mathsf{S}_{v_0}(v_1, \cdots, v_k))$ $\Rightarrow \wedge_{i=1}^{k} v_i \in \rho(\mathsf{x}_\bullet)) \wedge$	destruct (i.e. match and learn) composite values
$\forall c \in \{\updownarrow, \mathsf{n}^{\varepsilon}_\bullet, \mathsf{n}^{+}_\bullet, \mathsf{n}^{-}_\bullet\} \cup \mathcal{N} :$ $\forall k \geq 1 : \forall v_1 \cdots v_k \in \kappa(c) :$ $\wedge_{i=1}^{k} v_i \in \rho(\mathsf{x}_\bullet) \wedge$	input – globally, and locally in all ambients it may create
$\forall c \in \{\updownarrow, \mathsf{n}^{\varepsilon}_\bullet, \mathsf{n}^{+}_\bullet, \mathsf{n}^{-}_\bullet\} \cup \mathcal{N} :$ $\forall k \geq 1 : \forall v_1 \in \rho(\mathsf{x}_\bullet), \cdots, v_k \in \rho(\mathsf{x}_\bullet) :$ $v_1 \cdots v_k \in \kappa(c) \wedge$	output
$\rho(\mathsf{x}_\bullet) \subseteq \sigma(\mathsf{x}_\bullet)$	

Table 20 Analysis of the attacker i.e. of an arbitrary process fulfilling the conditions of the instantiation relation in Table 9.

minimum) without restraining the attackers ability to influence the process in question.

The final condition simply imposes the condition also enforced in the other clauses, namely that information in the more precise ρ component of the analysis also is present in the σ component.