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

Andrea Bracciali · Gianluigi Ferrari · Emilio Tuosto

A symbolic framework for multi-faceted security protocol analysis

Abstract Verification of software systems, and security protocol analysis as a particular case, requires frameworks that are *expressive*, so as to properly capture the relevant aspects of the system and its properties, *formal*, so as to be provably correct, and with a *computational counterpart*, so as to support the (semi-) automated certification of properties. Additionally, security protocols also present hidden assumptions about the context, specific subtleties due to the nature of the problem and sources of complexity that tend to make verification incomplete.
We introduce a verification framework that is expressive enough to capture a few relevant aspects of the problem, like symmetric and asymmetric cryptography and multi-session analysis, and to make assumptions explicit, e.g. the hypotheses about the initial sharing of secret keys among honest (and malicious) participants. It features a clear separation between the modeling of the protocol functioning and the properties it is expected to enforce, the former in terms of a calculus, the latter in terms of a logic. This framework is grounded on a formal theory that allows us to prove the correctness of the verification carried out within the fully-fledged model. It overcomes incompleteness by performing the analysis at a symbolic level of abstraction, which, moreover, transforms into executable verification tools.

Andrea Bracciali
Dipartimento di Informatica, University of Pisa
Largo B. Pontecorvo 3 - 56100 Pisa IT
Tel.: +39 050 2212743, Fax: +39 050 2212726
E-mail: braccia@di.unipi.it

Gianluigi Ferrari
Dipartimento di Informatica, University of Pisa
Largo B. Pontecorvo 3 - 56100 Pisa IT
Tel.: +39 050 2212766, Fax: +39 050 2212726
E-mail: giangi@di.unipi.it

Emilio Tuosto
Computer Science Department, University of Leicester
University Road, LE1 7RH Leicester UK
Tel.:+44 (0) 116 252 5392, Fax.:+44 (0) 116 252 3915
E-mail: et52@mcs.le.ac.uk

Keywords formal verification · security protocols · symbolic model checking · automated verification tools.

1 Introduction, motivations and related work

Security protocols are designed to *keep secret* pieces of relevant information within a context where some *actors*, called *principals*, communicate through *public channels*. Keeping secrets has a very broad sense, subsuming for instance confidentiality, non-modifiability or authenticity of the information itself, as well as the certification of the identity of the principals and their privileges. Principals, which might be software components or customers and their network services, perform a sequence of communications, often informally specified as a short list of point-to-point interactions, in order to enforce a desired security property. Since communication happens through public channels, it is not possible to guarantee that the exchanged messages are not accessed, manipulated or destroyed by third parties, called *intruders*. Security protocols based on cryptographic mechanisms such as encryption, digital signatures, and hashing have been widely adopted in distributed systems. Well-known examples include *Kerberos* authentication scheme for handling passwords and the *Secure Socket Layer* for internet communications. Other application domains such as *web services* are currently emerging.

The design and use of cryptographic protocols is error prone even if perfect cryptography is assumed. Despite their apparent simplicity, the analysis of security protocols is basically intractable [30,22,21,26,55]. For instance, since the intruder can intercept, analyse, and modify the messages exchanged among principals, *checking* the correctness of the protocol requires the analysis of the potentially infinitely many different behaviours of the intruder. This calls for verification models that are grounded on solid formal settings. Considerable success has already been achieved in the past years in approaching security protocol analysis and a variety of formal models and reasoning techniques have been developed to

specify and verify their properties, (see [20,42,53,51,32, 10,12,13,36,4] to cite a few). Many of these approaches have led to the development of automated verification toolkits. For instance, finite-state model checking techniques have been exploited in the design and implementation of automated tools [42,25,50]. The use of process calculi as formal specification languages for security protocols dates back to [42], where CSP has been successfully adopted for specifying and verifying the well known Needham-Schroeder public key protocol [52]. This result, achieved after about 20 years from the definition of the protocol, paradigmatically shows the needs for (automated) formal verification. In [34,33] properties are verified by checking bisimulations among protocols specified with a cryptographic version of CCS. To avoid state explosion, some approaches make some simplifying hypotheses on the underlying model. For instance, they assume a bound on the size of the messages synthesised by the intruder. Other techniques are based on *symbolic* models. These approaches address verification issues without making simplifying hypotheses on the formal model as far as the behaviour of the intruder is concerned, e.g., [5,3,13,4,61,18,15,1,38,17]. The advent of spi calculus [2] has made possible the convergence of two lines of research: symbolic analysis of security protocols [38] and symbolic semantics of process calculi [37, 16]. Symbolic techniques have been widely exploited in the last years for verification purposes. For instance, the state explosion problem of model checking can be faced by means of *abstractions*, so as to avoid using concrete representations of the state space in favour of smaller, abstract ones that preserve the property of interest [24]. Verification of cryptographic protocols has also adopted these ideas for reducing the state space, notably [13, 15] have adopted symbolic semantics for process algebras for trace analysis of security protocols, in [3] a symbolic reachability analysis has been described (improving ideas from [38]), and constraint differentiation [10] reduces the state space by removing some redundancies arising in symbolic state spaces. Hence, the verification of security protocols is being carried out on a substantial scientific basis, witnessed by the great number of successful research approaches to (semi-)automatic verification. However, correctness of security protocols depends on a fairly wide range of assumptions and factors upon which protocols rely on. To mention a few of these facets, consider the underlying cryptographic system, the intended sharing of secrets, the class of properties to be enforced, and the intruder model. The development of a soundly based framework, capable of *uniformly* handling the full range of issues that arise in the designing, specification and verification of security protocols is still an open and challenging research problem.

This paper presents a formal verification framework and the corresponding verification methodology with the aim of uniformly handling some of the above mentioned facets. More precisely, our approach features, in a coherent verification framework, (*i*) a clear separation of the specification of the protocol from the specification of its properties, (*ii*) the precise specification of the initial conditions of protocol executions, (*iii*) the automatic treatment of multi-sessions, (*iv*) the possibility to express properties abstractly by means of quantification over sessions. These features, together with the solid formal framework and the related supporting tools, lead to a versatile verification methodology.

We envisage a security protocol as an open system where principals may dynamically join and interact in multi-session runs of the protocol. Multi-session runs are modelled as the possibility of distinguished copies of the same principal, called *principal instances*, to participate in a set of concurrent executions of the protocol, called a *session*. The intruder, modelled as the environment, may "interfere" with all the communications taking place in the session. The interpretation of protocols as open systems is similar to the one in [45], where openness is represented by a non-completely specified and extensible context, the principals are expressed in the CCS calculus (equipped with cryptographic primitives) and properties are given in a suitable logic. Similarly, our formal framework consists of a *calculus* to describe the instances of principals and a *logical formalism* to express the security properties to be checked, clearly separating the descriptions of the behaviour and of its expected effects.

The calculus, called *cryptographic Interaction Patterns*, cIP for short, is a nominal calculus along the line of the spi-calculus [2]. The operational semantics of cIP formally models the Dolev-Yao intruder model. In Section 4.3 we discuss other models, which the cIP semantics can be smoothly adapted to.

The logical formalism, called $\mathcal{PL}$ logic, after Protocol Logic, predicates over messages exchanged by the principal instances and over the "presumed" identities of the instances. An original aspect of $\mathcal{PL}$ is that it can predicate over protocol sessions, by universally and existentially quantifying over principal instances. This accounts for formally defining a wide variety of security properties, like authentication that has been naturally defined in terms of relations amongst instances [44] (in Section 5.1 a more detailed account of the expressiveness of $\mathcal{PL}$ is given).

Formulas of $\mathcal{PL}$ are checked against the semantic model of cIP. More precisely, the infinite branching of cIP models is undertaken by a symbolic version of the semantics and $\mathcal{PL}$ formulas are checked on symbolic models. Symbolic verification has been proved sound and complete. Relying on these results, the methodology is properly supported by a fully-implemented model-checker, called $\mathcal{A}$SPASyA (Automatic Security Protocol Analysis via a SYmbolic model checking Approach).

The results illustrated in this paper revise and extend results firstly appeared in [61,18]. This paper mainly focuses on the theory underlying the framework. The

related verification methodology is briefly described in Section 10 and the reader interested in the practical experimentation is referred to [9,8]. The most relevant aspects of protocol verification that are dealt with and the features of the presented framework can be outlined as follows.

- *Explicit statement of the intended initial conditions.* Distinguishably, the cIP calculus provides linguistic mechanisms to explicitly state which keys and other relevant information can be acquired by any principal when it (dynamically) joins a protocol session. This mechanism is based on the introduction of a distinguished set of variables subject to late-binding mechanisms, called *open variables*. Open variables are the formal device exploited to represent the intended key sharing and other initial conditions. This can be done explicitly, left to exhaustive automated verification, or subject to constraints expressed as $\mathcal{PL}$ formulas, called *connection formulas*. These, by constraining open variables, and hence the set of admissible sessions, allow the analysis to be focused on the (initial) conditions of interests. For instance, one might rule out the case of a server accessed by more instances of the same principal by means of a suitable connection formula over the open variables modelling the server connections. In place of this versatile mechanism, most of the other approaches require the hand-coding of the intended key sharing for each (different) protocol formalisation or even lack of an explicit statement of it. Open variables and the join mechanism, together with the use of $\mathcal{PL}$ formulas as connection constraints, can be properly interpreted as a coordination mechanism for open systems. In this respect, they distinguish our approach from the one in [45], also based on a similar interpretation of protocols as open systems. Using *strand spaces* [60, 32], the approaches in [58,49] express properties in terms of connections between strands. A strand can be parameterised with variables and a trace is generated by finding a substitution for which an interaction graph exists. These approaches provide devices very similar to our join mechanism but they lack the possibility of constraining connections among principal instances. Finally, also the intruder power can be explicitly tuned by providing it with an initial knowledge about the protocol, e.g., a compromised key if one wants to test protocol robustness.
- *Multi-session verification.* Many known attacks are based on the exploitation of information coming from different executions of the same protocol. Sessions are explicitly modelled, in terms of both the distinguished linguistic primitives of cIP and the $\mathcal{PL}$ logic, with the quantification over principal instances. Although verification is bounded in the number of different instances in a session, this allows us to specify properties independently of the (fixed) number of instances to be tested, while other approaches, e.g. [15,62], require properties to be specified, if not hand-coded, for each different multi-run session under verification. Session handling also relies on open variables, which at the best of our knowledge, are a specific feature of cIP and there is no corresponding notion within other similar frameworks. A related approach, not featuring open variables, is in [27], and it will be discussed in Section 4.2.
- *Separation of concerns.* The formalisation of principals' behaviour is neatly separated from the properties it is intended to enforce. Methodologically, and similarly to the previous point, this appears as a relevant enhancement allowing orthogonal aspects to be dealt with separately. Indeed, this is one of the key features of our framework. Section 10 shows the flexibility of our verification methodology. That is evident when contrasted with others, e.g. in [62] security properties are "embedded" in the protocol specification and the verification phase just consists of invoking the verifier. Hence, it is necessary to modify the protocol representation in order to set different verification scenarios.
- *Adopted crypto-systems.* Symmetric and asymmetric crypto-systems are both supported in our framework. Some straightforward extensions to other cryptographic systems are discussed in Section 3, together with others that appears not viable, since they would make analysis incomplete.
- *Use of symbolic semantics.* Symbolic semantics has been adopted to deal with incompleteness issues. It is based on unification and represents a sound and complete approximation of the semantics of the framework presented in its full-fledged expressive power. The most similar use of symbolic semantics is perhaps the one proposed in [15] for generic crypto-systems. Our construction is more complex, paying the price for explicitly treating asymmetric cryptography and for having an abstraction that exactly preserves the concrete computations and avoids to construct symbolic messages that have no concrete counterpart. The symbolic semantics in [15] is simpler and exploits most general unifiers, but it is an (over-)approximation of the concrete model (from which the additional spurious behaviours can – and are – filtered out). Despite the similarities in the symbolic semantics, the two approaches are, however, quite different with respect to the analysis techniques. In [15] the analysis proceeds by checking correspondence assertions on the symbolic traces, while our framework introduces the $\mathcal{PL}$ logic. We argue that $\mathcal{PL}$ logic appears to more naturally express and model-check properties (a detailed comparison of $\mathcal{PL}$ logic and correspondence assertions is given in Section 5.1).

The presented model relies on several technical results that allow us to state the decidability of the model checking problem of our logical formalism, which justifies the verification methodology. A first decidability result

concerns the intruder model, namely the decidability of message derivation from a finite knowledge for asymmetric cryptography. This result (Theorem 1) complements an analogous result given in [25] for symmetric cryptography. The proof and the issues arising from asymmetric keys let us remark that asymmetric cryptography is not simply a direct extension of the symmetric case (and its complication has been sometimes underestimated, see Sections 3 and Section 6). Even if the result for the symmetric case is not spoiled when lifted to the asymmetric case, it seemed worth giving an explicit construction for the latter, which was to the best of our knowledge missing, even if the result is generally accepted and used. The proof we propose highlights how the similarity between message derivation and natural deduction exploited in the proof in [25] cannot be used anymore, basically because asymmetric decryption is not invertible, and a new ordering of message derivation rules is needed. Building on top of this result, we then prove soundness and completeness of symbolic semantics (Theorems 4 and 5) and decidability properties for the logic for concrete (Theorem 2) and symbolic semantics (Theorem 8).

After having specified protocol principals as cIP processes and the expected properties and $\mathcal{PL}$ formulas, the protocol can be automatically verified by means of the $\mathcal{A}$SPASyA symbolic model checker. From the methodological viewpoint, the separation of concerns amongst assumptions, specification and properties of protocols allows the definition of a verification methodology where different facets of security protocols are neatly grasped. The intruder power and the initial assumptions (as the sharing of keys and the way principal instances are connected one to another) can be suitably controlled by means of the explicit declaration of the intruder initial knowledge and suitable connection formulas. This is done without modifying neither the protocol or the property specification. Initial conditions clearly determine the space of the reachable states, and hence they can be used to control the portion of the state space to be explored (see Section 10). Finally, the automatic verification phase can be iterated easily modifying the conditions of the experiment, if needed according to the insights about the, maybe unexpected, behaviour of the protocol gained in previous iterations (details about the verification methodology are given in Section 10).

This paper is organised as follows. Security protocols and their models are briefly introduced before formally presenting the intruder model, cIP and the $\mathcal{PL}$ logic. Then, the symbolic approach adopted is illustrated, highlighting both its finiteness and correctness properties and how it supports the verification of $\mathcal{PL}$ formulas. A concrete example of verification, supported by $\mathcal{A}$SPASyA, is discussed and, finally, some concluding remarks are drawn.

2 Security protocols and their formalisation

We start by briefly reviewing security protocols and their basic ingredients so as to fix the context. The interested reader is referred to [59,46,23] for more comprehensive presentations. Moreover, we take the opportunity to point out some issues about the formal modelling of protocols, to discuss how these have been generally addressed in the literature and to explain the motivations that have led us to sometimes opt for different choices.

Cryptography. Cryptography is the process of hiding information by encoding the content of a message m, called *cleartext,* in a "secret" format, called *cryptogram.* Given a *cryptographic key* k, the message m is *encrypted* in the cryptogram $\{m\}_k$ and the original information can be retrieved only by *decrypting* it by using a suitable decryption key. Keys are supposed to be opportunely kept secret between communicating partners, called *principals*. Cryptographic algorithms, which are instead public, belong to two main classes: *symmetric algorithms* and *public-key algorithms*. In the case of symmetric algorithms the same key is used for encryption and decryption. Instead, in public-key algorithms a pair of distinct keys is associated to each principal. The *public key* is publicly available, while the *private key* is meant to be known only by the principal itself. The private key can decrypt messages encrypted with the public key and vice-versa. The secrecy of the private key guarantees that *i)* anyone can send a message encrypted with the public key, however, the message can be decrypted only by the intended receiver, and *ii)* anyone can decrypt a message encrypted with the private key by the principal, being therefore certain as to the identity of the sender. It is worth reminding that the way in which secret keys are off-line or dynamically assigned and shared is an essential step for determining the identity and the actual roles played by the principals during the execution of the protocol.

Protocols. Principals participate into a protocol by executing a finite number of possibly encrypted communications. More *instances* of the same principal can be interleaved together in a protocol *session*. We denote principals by $A, B, S, ...$, while A^- and A^+ are the private and public keys of a principal A, λ ranges over symmetric and asymmetric keys and λ^- is its complementary key (i.e. A^+ if $\lambda = A^-$, A^- if $\lambda = A^+$, and k if $\lambda = k$ is a symmetric key), m, n, p, q, o are messages or *nonces* (after "name once": fresh names, e.g. timestamps, used to mark messages as fresh) and (m, n) is the pair of messages m and n. A malicious principal that does not behave according to the protocol, called *intruder*, is denoted by I.

Traditionally, protocols have been expressed in an informal mix of natural language and ad hoc notation, called *narration* as:

$$(1)\ A \rightarrow B : \{n\}_k$$

"in the first communication, A sends the datum n encrypted by the key k to B". Actually, the precise meaning of the statement results to be more complex, since many relevant details are not explicitly described, like that A and B may be principal instances of interleaved protocol executions, they are supposed to *exclusively share* a common key k and the communication may be *intercepted and altered* by an intruder. The framework we are presenting aims at providing the right abstractions to naturally represent such kind of "hidden assumptions", so as to ease the error-prone modelling process. Similarly, a strong effort has been done to keep separate the different aspects of modelling.

The first problem to be addressed is to define a suitable formal representation of the protocol, starting from its informal description. Since a protocol consists of the interaction amongst principals, a quite natural choice is to enrich with cryptographic primitives the formalisms that have traditionally been used to model concurrency. Examples exist based, for instances, on Petri Nets and Process Algebras, other approaches use different techniques, like Strand Spaces [32]. Within Process Algebras, the spi-calculus [2], an extension of the $\pi-$calculus that accounts for cryptography, has emerged as an accepted standard. The above mentioned communication can be modelled in spi-calculus by means of a pair of input/output actions made by different processes:

$$A \triangleq \ldots\ \overline{c}\langle\{n\}_k\rangle\ \ldots$$
$$B \triangleq \ldots\ c(x)\ .\ \text{case } x \text{ of } \{y\}_k \text{ in } \ \ldots$$

where c is a *shared channel* between A and B. The message received by B in the input action $c(x)$ is assigned to x and then decrypted by using *pattern matching*: case x of $\{y\}_k$ in .... This action, when (the value of) x matches the "structure" of a cryptogram encrypted with k, instantiates y with the cleartext n for the continuation of the process.

We have followed the same approach, devising a formal framework that is based on a slightly modified calculus, adapted so as to more precisely express some features of interest:

- ▷ Focusing on communications within untrusted environments, we do not consider (private) channels: communications happens on a single public channel (not mentioned in the communication primitives) and can be protected only by cryptography. Indeed, in spi-calculus restriction over names is used mainly for keys or nonces, rather than channels, similarly to our framework where names, e.g. keys, are local. Also, in [13] channel names are used to determine the principal identities, while communications happen on a single public channel.
- ▷ In order to simplify the calculus, cryptography is embedded into communications: the structure of the expected messages and the decrypting keys used must be declared in input actions. The message can be exchanged and decrypted only if the sent cryptogram suitably matches the structure of the message declared in the input action. The notion of matching involves the correspondence of encrypting and decrypting keys.
- ▷ The open variables of the cIP calculus provide for a precise representation of the *initial* sharing of information amongst principals. Open variables must be instantiated before communication can happen and, by means of suitable instantiations, it is possible to formally specify the intended sharing of keys as well as to test specific cases of interest. However, if no condition is imposed on open variables, protocol verification can also be carried out against all the possible initial key sharing. It is important to remark that open variables are a mechanism to describe initial conditions, as the sharing of long term symmetric keys or public keys, which are typically managed off-line and often out of the protocol formalisation. Besides, each principal instance can locally declare and then communicate distinguished, i.e. fresh, local names like session keys and nonces.

In our calculus, the above fragments of processes are written as

$$A \triangleq (z)[\ldots\ out(\{n\}_z)\ \ldots]$$
$$B \triangleq (w)[\ldots\ in(\{y\}_w)\ \ldots]$$

where the open variables z and w are intended to be instantiated with the same key k. Only when $z = w = k$ the sent message $\{n\}_k$ matches the expected pattern $\{y\}_k$, the communication happens and y is instantiated with n, a message freshly generated by A, in the continuation of B. Note the lack of any explicit link from A to B in this communication.

- ▷ A last difference with mainstream calculi a-lá spi-calculus, is an explicit annotation of principals so as to distinguish their different instances participating to a *multi-session* execution of the protocol. A large number of known attacks is based on the intruder playing the part of the *man in the middle* [42,47], i.e. it exploits information gathered from one protocol execution into another one, greatly enhancing its competence, and, for instance, cheating about identities. Hence, a clear specification of how protocol executions can be interleaved and of the security properties expected after such concurrent runs are necessary, as recognised for instance in [20].

Our approach consists in annotating each principal, its data and variables, with a fresh index that makes its identity and communications unique over all the analysed multi-session protocol. An instance of A above, once its open variable have been instantiated, is e.g.,

$$A_3 \triangleq [\ldots\ out(\{n_3\}_k)\ \ldots]$$

Note that the symmetric key k is not a private key of A_3, and, hence has not been annotated. The choice of introducing this annotation is motivated by the relevance

of multi-session analysis (e.g. [47] shows that in principle is not possible to avoid considering general multi-sessions, given that analysis limited to a finite number of sessions may result incomplete). In general, the proposed formalisms for protocol analysis lack of an explicit abstraction to represent this kind of information, which is rather coded by hand in the model, for instance by naming all the different principals involved in the specific multiple session under analysis. As it will be clear from Section 5, the treatment of multi-sessions has an impact on the expressiveness of the logic used to characterise the properties of interest, and on the ease of the verification process. Indeed, one may wish to express properties independently of the specific number of instances under considerations, i.e. formulas that may be universally or existentially quantified over principal instances, like "there exists no principal instance whose secret keys can be acquired by the intruder".

The intruder. The verification model, other than the precise description of principal behaviour, must encompass a description of what an intruder is entitled to do in order to attempt to break the protocol. We follow a widely accepted model, namely the Dolev-Yao model [29]. The intruder enhances its knowledge about the protocol, hereafter indicated as κ, by interfering with virtually all the communications, with the only limit of the perfect functioning of the cryptography layer: it can attempt to violate the protocol in many different ways, but not guess cryptographic keys.

Security properties. Protocols are designed to enforce that a desired security property holds after their execution. Perhaps, the basic requirements to encrypted communications have been that nobody can read the secret message, i.e. *secrecy*, that nobody can corrupt or even destroy the message, i.e. *integrity*, that nobody can impersonate someone else, i.e. *authentication*. Nowadays, security protocols may also be devised in order to guarantee different kinds of properties, for instance *non-repudiation*: nobody can retract a commitment. Though their informal meaning is widespread understood, formalising security properties appears even more difficult than formalising protocols. This problem is however out of the scope of this paper. We provide a suitable logic that captures quite general properties, i.e. those that have traditionally been subject of extensive verification in the literature. The logic well fits in the framework, for instance featuring quantification over principal instances. Following our example, the *secrecy* of message n is simply expressed as

$$\forall i : A.\ \kappa \not\triangleright n_i,$$

that means "any message n_i, sent by any instance i of principal A, does not belong to, i.e. it cannot be derived from ($\not\triangleright$), the intruder knowledge κ". Other properties can be defined in terms of the messages sent, the relationships between sending and receiving instances, the knowledge κ and the way keys are shared. The verification of the properties, expressed as $\mathcal{PL}$ formulas, against the protocol behaviour, represented as cIP processes, is then fully supported by automated tools. Within this framework, reminding that each instance performs a finite number of actions and that a symbolic semantics has been adopted, we will prove that the problem of the existence of a reachable state in which a property does not hold, and hence the protocol is violated, is decidable once that the maximum number of participating principal instances has been fixed. Such results are based on extensions of known decidability results that have been then exploited within the multi-faceted framework we are presenting.

3 A decidable Intruder model

The behaviour that a malicious principal is entitled to perform in order to violate a protocol is defined as the *intruder model*. We adhere to the very general and accepted Dolev-Yao model [29]: the intruder acquires a knowledge κ by interfering with virtually all the communications in the protocol. It can copy, destroy, store and modify any message sent on the network and send any message derivable from κ, but cannot guess keys (perfect encryption hypothesis). Knowledge κ is a *finite subset* of the possible messages circulating in a protocol and it can grow as long as communications happen. We describe here the model (implemented in the verification environment $\mathcal{A}$SPASyA discussed in Section 10), outlining also its limitations and possible extensions. A protocol proved safe against this model gives reasonable guarantees when executed in a real environment, where intruders may typically be weaker in some aspects, e.g. they might not control the whole communication network. An extension of the model, according to which keys can be guessed up to a given probability, has been recently studied in [65], showing that with safe (long enough) keys the two models coincide.

The model has been defined for symmetric and asymmetric cryptography. The set of exchanged messages consists of a free algebra whose constants are basic messages, as names, nonces and keys, and the operators represent pairing and encryption. The intruder can generate an infinite number of messages by encrypting, decrypting, pairing and projecting those in its finite knowledge. Similarly to other proposals (e.g. [3,15,53,42]), we consider only atomic keys, while equational theories over messages are not allowed. Possible extensions to non-atomic keys, other cryptographic operators or equational theories will be discussed later on in this section. We first define the syntax of messages.

Definition 1 (Messages) The set M of *ground messages* is defined as:

$$\begin{array}{rcl} M & ::= & PN \mid K \mid PN^{+} \mid PN^{-} \mid NO \mid \\ & & \{M\}_{K} \mid \{M\}_{PN^{+}} \mid \{M\}_{PN^{-}} \mid (M, M) \end{array}$$

where PN, K, PN^+, PN^-, NO represent principal names, symmetric keys, public and private keys, and nonces, and $\{\_\}\_$ and $(\_,\_)$ are the encryption and pairing operators respectively.

The intruder can derive from a non-empty κ an *infinite* set of messages according to the next definition.

Definition 2 (Message derivation) A message m is *derivable* from a knowledge κ if and only if $\kappa \rhd m$ can be proved by the following rules

$$\frac{m \in \kappa}{\kappa \rhd m} \in \qquad \frac{\kappa \rhd m \quad \kappa \rhd n}{\kappa \rhd (m,n)} ()_i \qquad \frac{\kappa \rhd m \quad \kappa \rhd \lambda}{\kappa \rhd \{m\}_\lambda} \{\}_i$$

$$\frac{\kappa \rhd (m,n)}{\kappa \rhd m} ()_{e1} \qquad \frac{\kappa \rhd (m,n)}{\kappa \rhd n} ()_{e2} \qquad \frac{\kappa \rhd \{m\}_\lambda \quad \kappa \rhd \lambda^-}{\kappa \rhd m} \{\}_e$$

A message m can be *constructively derived* from a knowledge κ, $\kappa \rhd_i m$, if and only if $\kappa \rhd m$ can be proved by means of the constructor introduction rules only, namely $\in$, $()_i$ and $\{\}_i$.

We show, complementing a result by Clarke et al. [25] about symmetric cryptography, that message derivation from κ is decidable also for asymmetric cryptography. This result, although generally accepted and used, was, at the best of our knowledge, not formalised yet, and is needed to prove decidability of properties verification within our multi-session framework. Decidability of $\rhd$ is proved by reducing it to decidability of constructive derivations.

Proposition 1 ($\rhd_i$ is decidable) *Given a knowledge κ and a message $m \in M$, $\kappa \rhd_i m$ is decidable.*

Proof The constructive derivation of a message is monotone (and κ is a finite set). □

The decidability result in [25] relies on the similarity of message derivation with natural deduction. Indeed, in both the formal systems, each operator has a complementary constructive and destructive interpretation that are invertible (i.e. each operator has an introduction and an elimination inference rule). This allows us to finitely decompose all the known messages, and then use their atomic components to monotonically construct any derivable message from the knowledge (similarly to natural deduction proofs, where all the constructor elimination rules can be reordered at the beginning of the proof, which then proceeds monotonically). Since asymmetric destruction cannot be inverted, i.e. from $\{m\}_{A^+}$ and A^- it is possible to obtain m and A^-, but not vice-versa, the same proof cannot be used anymore. We show a different construction for message derivation that, after a finite set of possibly destructive steps, proceeds monotonically, and hence is decidable. It is based on a "remembering" transformation of κ, the e-limination function, that applies the elimination rules, taking care not to forget messages that cannot be reconstructed (for the sake of simplicity, $e(\_)$ is actually defined so as to, redundantly, remember all the messages).

Definition 3 Given a knowledge κ, its *explicit* form is $e(\kappa)$, where function $e(\_)$ takes a knowledge and returns a knowledge as follows:

$$e(\kappa) = \begin{cases} e(\kappa' \cup \{p,q\}) \cup \{m\} & m = (p,q) \in \kappa \\ e(\kappa' \cup \{n\}) \cup \{m\} & m = \{n\}_\lambda \in \kappa \ \wedge \ \lambda^- \in \kappa \\ \kappa & otherwise \end{cases}$$

where $\kappa' = \kappa \setminus \{m\}$.

Proposition 2 *$e(\_)$ is well defined.*

Proof The proof consists of showing that *the function always terminates* and *returns a uniquely determined set* (Appendix A.1). □

Finally, decidability of $\rhd$ can be proved by reducing derivation from κ to constructive derivation from $e(\kappa)$.

Theorem 1 ($\kappa \rhd m$ is decidable) *Let $m \in M$ be a message, and κ a knowledge, then*

$$\kappa \rhd m \Leftrightarrow e(\kappa) \rhd_i m,$$

(and hence, by Proposition 1, $\kappa \rhd m$ is decidable).

Proof By induction on the length of the proofs for $\rhd$ and $\rhd_i$, respectively (Appendix A.2). □

Example 1 Let be $\kappa = \{kb, \{(no,nn)\}_{A^+}, \{A^-\}_{kb}\}$. We prove that $\kappa \rhd A^-$ as follows:

$$\frac{\dfrac{\{A^-\}_{kb} \in \kappa}{\kappa \rhd \{A^-\}_{kb}} \in \quad \dfrac{kb \in \kappa}{\kappa \rhd kb} \in}{\kappa \rhd A^-} \{\}_e$$

Then we show that $\kappa \rhd \{no\}_{A^-}$

$$\frac{\dfrac{\dfrac{\{(no,nn)\}_{A^+} \in \kappa}{\kappa \rhd \{(no,nn)\}_{A^+}} \in \quad \cdots}{\kappa \rhd (no,nn)} \{\}_e}{\kappa \rhd no} ()_{e1} \qquad \cdots}{\kappa \rhd \{no\}_{A^-}} \{\}_i$$

where $\cdots$ stands for the derivation of A^- from κ.

Notice that, given the explicit form

$$e(\kappa) = \{no, nn, kb, A^-, (no,nn), \{(no,nn)\}_{A^+}, \{A^-\}_{kb}\},$$

$e(\kappa) \rhd_i \{no\}_{A^-}$ is proved by using rules $\in$ and $\{\}_i$.

Our intruder model relies upon some assumptions on the use of keys, mainly adopted to make it decidable. Even if a completely unrestricted use of keys may lead to incompleteness, it is worth discussing to what extent these assumptions can be relaxed. We will consider admitting non-atomic keys, since they appear in widespread protocols as SSL 3.0 [35], and equational theories over the algebra of messages. The impossibility of

constructing keys (e.g. $\lambda^- \in \kappa$ in Definition 3) allows message derivation steps to be suitably reordered as explained (see also the proof of Theorem 1). The use of non-atomic keys spoils this reordering strategy, where all the destructive steps precede the constructive ones. For instance, $\kappa \rhd m$, with $\kappa = \{\{m\}_{\overbrace{(p,\ldots,p)}^{n}}, p\}$, requires n constructive steps for deriving the key $(p, \ldots, p)$, before deriving m. A derivation strategy, which guarantees the reduction of the message complexity at each step, has been proved sound and complete in [49] for non-atomic symmetric keys, signatures, hashes, and a limited form of asymmetric cryptography. The integration of this result within our intruder model[1] appears straightforward, enabling the use of non-atomic symmetric keys. However, the extension to non-atomic asymmetric keys requires further work and is scope for future research.

Equational theories over the free algebra of messages could also have been considered, for instance to determine suitable decrypting keys, e.g. a key k fulfilling $k = \lambda^-$, for a given key λ. As recognised in [49], the treatment of cryptographic operators involving equational theories, like the commutative `xor` or Diffie-Hellman exponentiation, is more difficult: in general, decidability of message derivation may revert to the decidability of the equational theory. Hence, trivially, decidable theories could be admitted, possibly increasing the computational cost of the analysis, while undecidable theories would spoil the completeness of the framework. Finally, it is interesting to remark that there are attacks that the free-algebra models can fail to recognise, as shown in [48] for the case of explicit decryption operators. Indeed, in [48] it is shown how messages derivable with equational theories cannot be considered by pure free-algebra models. Informally speaking, let us indicate with $\}m\{_k$ the operation of "explicitly" decrypting a message m with the key k (note that m is not known to have a structure $\{\_\}\_$). Given $\kappa = \{n, \{m\}_k, k'\}$, in our model $\kappa \not\rhd m$. Let us suppose that the equality $\}m\{_{k'} = n$, indicating that n can be obtained by explicitly decrypting m with k', can be proved in a model extended with the above operator. Then, since $\kappa \rhd n$, it is possible to prove $\kappa \rhd \{n\}_{k'} = m$. Although a way to overcome the problem has been suggested in [48] for symmetric keys, the full understanding of the issue within our framework is scope for future work.

4 cIP: a Cryptographic calculus of Interacting Principals

The *cryptographic Interaction Pattern* calculus (cIP), introduced in [18,61], is a nominal calculus for modelling principal interaction within protocol sessions, like the spi-calculus [2]. Differently from the spi-calculus, cIP features asynchronous communications and does not have silent actions and (private) communication channels and uses names for cryptographic keys, nonces or principal names. The execution of a protocol is modelled by means of the execution of a session joined by principal instances according to a given sharing of keys and other relevant information.

4.1 Syntax

The cIP processes that represent protocol principals are defined as follows.

Definition 4 (Principal) A cIP *principal* P is defined as:

$$\begin{array}{l} P ::= PN \triangleq (X)[E] \\ E ::= 0 \mid \alpha.E \mid E|E \mid E + E \\ \alpha ::= in(M_V) \mid out(M_V) \end{array}$$

where $V = \{wa, wb, xa, xb, ya, yb, x, y, \ldots\}$ is a set of variables, $X \subset V$ is the (finite) set of *open variables*, E is the *behavioural expression* of the principal, and the set of *data* M_V is

$$M_V ::= PN \mid K \mid PN^+ \mid PN^- \mid NO \mid V \mid ?V \mid \{M_V\}_{M_V} \mid (M_V, M_V)$$

where PN, K, PN^+, PN^-, NO are as in Definition 1, and $?V$ denotes a *binding occurrence* of a variable (hereafter, a principal $A \triangleq (X)[E]$ can also be denoted as A or $(X)[E]$).

Open variables represent an explicit declaration of keys and names used by the principal. When principals join a session, these variables are instantiated so as to suitably share keys and other required names. The action $out(M_V)$ outputs M_V to the environment. The message in the action $in(M_V)$ is used to declare the structure of the expected input message. For instance, $in(\{?x\}_k)$ means that a cryptogram encrypted by the symmetric key k is expected to be received and decrypted. k must be know by the principal, for instance by means of the instantiation of an open variable. The communication and the decryption can happen only if the sent message matches M_V, according to Definition 7. If this is the case, the x in $in(\{?x\}_k)$ will be instantiated with the cleartext. A mechanism similar to scope extrusion in π-calculus can be used to further share keys, e.g. by receiving the message $\{k\}_{B^+}$, B can safely, if B^- is secret, acquire the key k. A behavioural expression may be the inaction 0 (sometimes omitted), a communication action α prefixed (".") to a behavioural expression, or a parallel ($||$) or non-deterministic ($+$) composition of behavioural expressions.

Variable scoping is ruled by two binders: *binding occurrences* and *open variables*. Binding occurrences can appear only inside input actions and, there, at most once. The scope of a binding occurrence, $?x$ say, within m in

[1] This could also be achieved in $\mathcal{A}$SPASyA by modularly updating the module for message generation.

$in(m).E$ covers the next (according to a left-to-right order) occurrences of x in m, and all the occurrences of x in E. The occurrences of variables within the scope of a binder are called *bound occurrences*, while those not in the scope of any binder are called *free occurrences*. A binding occurrence $?x$ with $x \in X$ binds all the free occurrences of x in its scope. Moreover, we require that variables occurring as keys in input actions cannot be binding occurrences (since decrypting keys must be known, and cannot be instantiated at the moment of decryption), and cannot be bound by binding occurrences within the same cryptogram, i.e. $?x$ cannot occur within m in $\{m\}_x$ (since a key received within an encrypted message cannot be used to decrypt the message itself). An open variable, $x \in X$ say, binds all the free occurrences of x in E. The set of bound and binding occurrences in an action α, a behavioural expression E, a principal $(X)[E]$, respectively, is indicated as $bv(\alpha)$, $bv(E)$, $bv((X)[E])$. The set of free occurrences of variables in E (resp., $(X)[E]$) is indicated as $fv(E)$ (resp.,$fv((X)[E])$. A principal $A \triangleq (X)[E]$ is *closed* if $fv((X)[E]) = \emptyset$. In the following, principals are closed, and all the binders are distinct (renamed if needed). Substitution works as expected, (if χ, σ are substitutions, $m\sigma$, $E\sigma$, $\kappa\sigma$ indicate the application of σ to a message, an expression or a knowledge, respectively, while $\chi\sigma$ indicates substitution composition).

Example 2 The Wide Mouthed Frog (WMF) protocol let A send a session key kab, i.e. a key valid for a limited temporal interval, to B through a trusted server S:

$$(1)\ A \to S : (A, \{(ta, B, kab)\}_{kas})$$
$$(2)\ S \to B : \{(ts, A, kab)\}_{kbs}$$

A sends S its name and a cryptogram containing the name of B, the session key kab and a nonce ta. Then, S sends B a cryptogram with the session key kab, the name of A, and a new nonce ts. The two messages have been encrypted by means of keys shared between A and S, and S and B, respectively. Nonces guarantee the freshness of the session key. A, B and S can be formalised in cIP as follows:

$$A \triangleq (q, xas)[out((A, \{(ta, q, kab)\}_{xas}))]$$
$$B \triangleq (zbs)[in(\{(?s, ?x, ?w)\}_{zbs})]$$
$$S \triangleq (u, ya, v, yb)[in((u, \{(?t, v, ?r)\}_{ya})).out(\{(ts, u, r)\}_{yb})]$$

Each principal is formalised as a cIP principal (Definition 4) where the behavioural expression is determined by the narration of WMF. For instance, (1) yields the first actions of A and S where S receives in the binding occurrences $?t$ and $?r$ respectively the nonce ta and the session key kab freshly generated by A (observe that $?t$ and $?r$ bind the occurrences t and r in the out action of S). The open variables of A are intended for the name of a principal with which A will share the session key, and for the key shared with S (notice that e.g., open variable u binds the two occurrences of u in the behavioural expression of S). This mechanism allows us to describe the behaviour of A in isolation, independently of the actual principal instances with which it will interact. Moreover, it will allow us to formalise properties that do not depend on the specific principal instance interacting with A. B only needs an open variable to share a key with S. S needs the open variables for the keys shared with A and B, and also, together with the keys, the names of the principals to which the keys are associated. This mechanism requires the association between keys and their owners to be explicit (hence restricting the set of messages S is ready to receive). ◇

4.2 Multi-session executions

The semantics of cIP is given as a semantics of *sessions*: sets of distinguished principal *instances* that execute the protocol one or more parallel and independent times. The distinguished choice of providing an abstraction for sessions within the framework allows us to precisely model multi-run protocol executions, whose importance for protocol verification has already been pointed out. Sessions are built by uniquely labelling the several instances of a principal, and stating how secret keys are shared between instances by assigning their open variables. For the purposes of verification, it is possible to automatically generate (a specific subset of) all the possibilities, or to provide off-line specific combinations of interest (Section 10).

Definition 5 (Instance and session) An *instance* $(X_i)[E_i]$ of a principal $A \triangleq (X)[E]$ is obtained by indexing all variables occurring in X or E and all names (except symmetric keys) occurring in E with an index i, viz. a positive natural number (hereafter, $(X_i)[E_i]$ can also be denoted as A_i).

A *session* $\{(X_{i_1})[E_{i_1}], ..., (X_{i_n})[E_{i_n}]\}$ is a (possibly empty) finite set of instances of principals.

Indexes distinguish instances and their data. Only symmetric keys, which do not properly belong to a specific instance, are not indexed. It is assumed that the same symmetric key appears in different instances only after that it has been used to instantiate their open variables. We use $\__I$ to indicate indexed sets. Uniquely indexed principal instances can be joined to the session, as formalised by a *join* operation.

Definition 6 (Join) Let us consider

- a session $\mathcal{C} = \{(X_{i_1})[E_{i_1}], \ldots, (X_{i_{n-1}})[E_{i_{n-1}}]\}$,
- an instance A_h of $A \triangleq (Y)[F]$ for a fresh index h, and
- a partial mapping $\gamma : D \to K \cup PN_I^+ \cup PN_I^- \cup PN_I \cup NO_I$ (where $D = X_{i_1} \cup \ldots \cup X_{i_{n-1}} \cup Y_h$) from variables to (indexed) symmetric or asymmetric keys, principal names and nonces.

Then

$$join(A_h, \gamma, \mathcal{C}) = \bigcup_{j=1}^{n-1} \{(X_{i_j} \setminus D)[E_{i_j}\gamma]\} \cup \{(Y_h \setminus D)[F_h\gamma]\}$$

is the session where A_h *joins* $\mathcal{C}$ by means of γ.

Mapping γ is not required to be total since instances might share keys with other instances that join the session later.

Semantics of sessions is given by means of a labelled transition system inductively defined by a set of transition rules. We distinguish between transition rules that describe the semantics of principals (behavioural expressions), and transition rules that describes the interaction of instances within sessions.

The transition rules of behavioural expression semantics are illustrated in Figure 1 (instance labels have been omitted for simplicity). It consists of a subset of standard π-calculus rules for action execution, parallel ($||$) or choice ($+$) composition, and structurally equivalent terms (as usual, $\equiv$ is the smallest congruence induced by the commutative monoidal laws for $||$ and $+$, with 0 as the neutral element, and including α-conversion). There are no communication rules as intra-principal communication is not allowed. Notice that no rule for the cryptographic operators is present as cryptography has been embedded in communication. Actions appear as labels.

Interaction between principal instances is modelled by the transition rules of Figure 2, built on top of $\rightarrow$ and the definition of ($join$) operation. The states of the transition system are *configurations* $\langle \mathcal{C}, \chi, \kappa \rangle$, where $\mathcal{C}$ is a session, χ is the substitution for variables determined by communications and join executions, and κ is the intruder knowledge, storing the instances that have joined the session, their public keys and the data transmitted in the protocol. Communications are asynchronous (ideally, from and to the communication network) and all involve the intruder. Labels represent actions, as executed by the intruder.

Whenever a principal sends a (ground) message m, the message is recorded in κ and the principal instance evolves according to relation $\rightarrow$. Messages are required to be well-formed ($m \in M_I$), otherwise the transition cannot fire (rule (out)).

The case for a principal receiving a message (rule (in)) is more complex, since it encompasses cryptography. Input actions specify the structure d of the expected message, decrypting keys included. They are executed only if the intruder can derive a suitable message m from its current knowledge κ, in which case both communication and decryption happen. In fact, if a substitution σ exists such that $d\sigma \sim m$, i.e. m *matches* d once its variables are instantiated with σ, then the correct keys have been provided, and variables within cryptograms are assigned with cleartexts. The notion of matching is provided by the following definition.

Definition 7 (Matching) Let $m, m' \in M_I$ be two indexed messages. We say that m *and* m' *match* ($m \sim m'$) if, and only if, one of the following alternatives applies:

- $m = m' \ \wedge m, m' \in PN_I \cup K \cup PN_I^+ \cup PN_I^- \cup NO_I$,
- $m = (e, f) \ \wedge \ m' = (e', f') \ \wedge \ e \sim e' \ \wedge \ f \sim f'$,
- $m = \{e\}_\lambda \ \wedge \ m' = \{e'\}_{\lambda'} \ \wedge \ e \sim e' \ \wedge \ \lambda' = \lambda^-$.

An indexed datum $d \in M_{V_I}$ *matches* an indexed message $m \in M_I$ if, and only if, there exists a substitution σ such that $d\sigma \sim m$.

Atomic messages match when equal, pairs when their components match. Cryptograms $m = \{e\}_\lambda$ and $m' = \{e'\}_{\lambda'}$ match only if their contents e and e' match and they have been encrypted by complementary keys, λ and λ^-. This ensures that decryption is correctly modelled. For instance, following rule (in) (indexes have been omitted), if $d = \{?x\}_{A^-}$, i.e. a principal instance is waiting for a message to be decrypted with the private key of A, then any $\{e'\}_{A^+}$ such that $\kappa \rhd \{e'\}_{A^+}$, i.e. any message encrypted with A^+ that the intruder can derive, would fit. Indeed, $\sigma = [x \rightarrow e']$ is such that $d\sigma \sim e'$. Contextually, m is decrypted since the cleartext e' is assigned to x, σ is applied to the continuation of the principal instance, where x may occur, and recorded in χ, which keeps trace of the effects of the communications happened in the session. Finally, the principal instance evolves according to the transition relation $\rightarrow$.

A new principal instance can join the session according to rule ($join$): given an instance A_i and an assignment to its open variables γ, A_i is added to the session, γ is applied to the new session and recorded in χ, and A_i and A_i^+ (all the public key are assumed to be known) are added to κ. The application of γ to the session allows keys to be shared by principals. The transition is labelled with $j(A_i, \gamma)$. As explained, γ is not defined by the semantics, and in the practice of verification can exhaustively comprise all the possibilities, be constrained to fulfil some property, or determine a specific case of interest, like the intended key sharing. Observe that rule ($join$) in Figure 2 requires freshness of the index i in order to properly deal with sessions.

To the best of our knowledge, this is a distinguishing feature of cIP. In [15] multiple instances cannot dynamically join the execution context and must be specified by hand before executing the session. Indexed instances are also used in SPL [28,27], which is a variant of the asynchronous π-calculus. SPL has many similarities with cIP, for instance input prefixes embed a pattern matching mechanism such that variables are instantiated when a matching message is received. Both SPL and cIP use indexes in protocol sessions, however SPL lacks a notion of join and uses indexes, differently from cIP, to individuate the roles in charge of executing an action rather than the instances. Multi-sessions consist of copies of principals, namely, each instance is distinguished by other instances only when generating different events, e.g., when creating a new nonce. Differently, in cIP, principal instances

$$\frac{E \equiv E' \quad E' \xrightarrow{\alpha} F' \quad F' \equiv F}{E \xrightarrow{\alpha} F} \ (struct) \qquad \frac{}{\alpha \,.\, E \xrightarrow{\alpha} E} \ (pre)$$

$$\frac{E \xrightarrow{\alpha} E'}{E \,||\, F \xrightarrow{\alpha} E' \,||\, F} \ bv(\alpha) \cap fv(F) = \emptyset \ \ (par) \qquad \frac{E \xrightarrow{\alpha} E'}{E + F \xrightarrow{\alpha} E'} \ (sum)$$

Fig. 1 Principal semantics

$$\frac{E_j \overset{out(m)}{\rightarrow} E'_j \qquad m \in M_I}{\langle \{(X_j)[E_j]\} \cup \mathcal{C}, \chi, \kappa \rangle \overset{i(m)}{\mapsto} \langle \{(X_j)[E'_j]\} \cup \mathcal{C}, \chi, \kappa \cup \{m\} \rangle} \ (out)$$

$$\frac{E_i \overset{in(d)}{\rightarrow} E'_i \qquad \kappa \rhd m \qquad d\sigma \sim m}{\langle \{(X_i)[E_i]\} \cup \mathcal{C}, \chi, \kappa \rangle \overset{o(m)}{\mapsto} \langle \{(X_i)[E'_i\sigma]\} \cup \mathcal{C}, \chi\sigma, \kappa \rangle} \ (in)$$

$$\frac{\mathcal{C}' = join(A_i, \gamma, \mathcal{C}) \qquad A_i \triangleq (X_i)[E_i] \qquad i \text{ new}}{\langle \mathcal{C}, \chi, \kappa \rangle \overset{j(A_i,\gamma)}{\mapsto} \langle \mathcal{C}', \chi\gamma, \kappa \cup \{A_i, A_i^+\} \rangle} \ (join)$$

Fig. 2 Asynchronous session semantics

are distinguished by indexes freshly generated when the instances join the context. This has an impact on the verification techniques of the two approaches that are quite different. cIP exploits a symbolic model while SPL is equipped with an event-based semantics that accounts for the derivation of properties and proof principles. The event-based semantics of SPL establishes a tight connection between SPL and strand spaces [31], and the inductive proof methods for the verification of protocols e.g., [54].

4.3 On weakening the Dolev-Yao intruder

The intruder model of Dolev-Yao fits well with the verification of the most common security properties like secrecy, integrity, authentication or authenticity. However, it happens not to be perfectly suitable for other security properties. As highlighted in [56,57], the Dolev-Yao intruder is too powerful for verifying protocols against certain properties, hence weaker models of the execution environment are desirable. As an evidence, consider *fairness* [6] and *non-repudiation* [64] for e-commerce protocols (e.g., fair-exchange protocols). In this context, the underlying communication infrastructure is usually abstracted either as an unreliable channel (loss of messages), or as a resilient channel or as a synchronous network. We start sketching how to model unreliable and resilient communications in cIP through a smooth extension, while we do not consider the unrealistic case of a synchronous network [41]. Messages can be typed as *dischargeable* and *untouchable*. When a dischargeable message is sent, the intruder non deterministically decides if it is added or not to its knowledge. When a message must be generated for matching an untouchable input, exactly the message meant for the input will eventually be supplied by the intruder. In this case, type information will comprise the needed references to senders and receivers, according to their roles once joined in a session. This simple type system accounts for representing unreliable communications and resilient channels, and can be adapted to model exactly one of the two: by typing all messages as dischargeable, we get completely unreliable communications, while by typing all messages as untouchable we obtain resilient communications.

This extension would require small changes that affect the syntax of cIP (messages come equipped with type information), the rules of Figure 2 (as (in) and $(join)$) and the definition of $\rhd$ for message generation should be extended to deal with message typing. Overall, the rest of the verification framework is left unchanged; $\mathcal{PL}$ is not affected and the schema of the model checking algorithm remains unchanged as well. Remarkably, adapting our framework requires fairly limited changes exactly because of the separation of concerns discussed before.

However, although it seems that the execution context in terms of the communication infrastructure can be easily embedded in our framework, the verification of properties, like the one used for e-commerce protocols, still remains difficult to be carried out in the Dolev-Yao model. This is mainly due to both the strong power granted to the intruder, unrealistic in the discussed context (e.g. the control of all the communications), and to the properties itself, which often cannot be expressed as trace invariants (e.g., abuse-freeness for contract signing protocols), and can hardly be verified within environments that check trace invariants [57]. Our verification framework suffers this limitation and, once extended with suitable abstractions for communications as dis-

cussed above, it can be still profitably used for checking properties that can be reverted to trace invariants (e.g., fairness [57]). However, dealing with more general properties would require structural extensions to the model and the verification strategy that are far beyond the scope of this paper.

4.4 Traces

The traces of interest are those that originate from an empty session and reach a state that represents a session where all the principals have been reduced to 0 (such a session is indicated as $\mathcal{C}_\emptyset$). The knowledge κ in an initial state may be non empty in order to model the case in which the intruder may know some useful information about the protocol, for instance to test the protocol robustness when one of the keys is compromised.

Definition 8 (Traces) A state $\langle \mathcal{C}, \chi, \kappa \rangle$ is

- *initial* if and only if $\mathcal{C} = \emptyset$, and χ is the empty substitution;
- *final* if, and only if, for each $P \in \mathcal{C}$ $P = ()[E]$ and $E \equiv 0$.

A *trace* is a sequence

$$T \ = \ \Sigma_0.\alpha_1.\Sigma_1 \ldots \alpha_n.\Sigma_n$$

where Σ_0 is an initial state and $\Sigma_{i-1} \overset{\alpha_i}{\mapsto} \Sigma_i$ $(1 \leq i \leq n)$. T is *terminating* if Σ_n is a final state.

Example 3 A session of the WMF protocol can be obtained by populating an initial empty session with instances of A, S and B, as defined in Example 2. By applying rule $(join)$, we obtain the configuration $\Sigma_0 = \langle \{A_3, S_2, B_1\}, \gamma_0, \ \{A_3, A_3^+, S_2, S_2^+, B_1, B_1^+\} \rangle$, where

$$\begin{array}{rcl} A_3 & \triangleq & ()[out((A_3, \{(ta_3, B_1, kab_3)\}_{kas}))] \\ B_1 & \triangleq & ()[in(\{(?s_1, ?x_1, ?w_1)\}_{kbs})] \\ S_2 & \triangleq & ()[in((A_3, \{(?t_2, B_1, ?r_2)\}_{kas}).out(\{(ts_2, A_3, r_2)\}_{kbs})] \end{array}$$

and $\gamma_0 = \{zbs_1 \mapsto kbs, \ xas_3 \mapsto kas, \ ya_2 \mapsto kas, yb_2 \mapsto kbs, \ q_3, v_2 \mapsto B_1, \ u_2 \mapsto A_3\}$ represents the intended sharing of keys. Easily, we can obtain the terminating trace

$$\begin{array}{l} \Sigma_0.\ i((A_3, \{(ta_3, B_1, kab_3)\}_{kas})).\ \Sigma_1. \\ o((A_3, \{(ta_3, B_1, kab_3)\}_{kas})).\ \Sigma_2.\ i(\{(ts_2, A_3, kab_3)\}_{kbs}). \\ \Sigma_3.\ o(\{(ts_2, A_3, kab_3)\}_{kbs}).\ \langle \mathcal{C}_\emptyset, \gamma, \kappa \cup \kappa_0 \rangle \end{array}$$

where, in the final configuration $\langle \mathcal{C}_\emptyset, \gamma, \kappa \cup \kappa_0 \rangle$,

$$\begin{array}{l} \kappa = \{(A_3, \{(ta_3, (B_1, kab_3))\}_k), \{(ts_2, (A_3, kab_3))\}_k\} \\ \gamma = \gamma_0\{t_2, s_1 \mapsto ta_3, \ r_2, w_1 \mapsto kab_3, \ x_1 \mapsto A_3\}. \end{array}$$

The intruder has only recorded and forwarded the messages.

It is worth remarking that the presented semantics is *infinitely branching*, i.e. a state may have an infinite number of successor states, and this, although well defined mathematically, makes formal verification potentially incomplete and difficult to be automated. As already addressed in [25], reasons for this are:

The infinite generative power of the intruder. The premise of rule (in) can be fulfilled by any message derivable from κ that matches d. For instance, if d is a variable any of the infinitely many m that can be derived from a non empty κ fits. Similarly to other recent proposals in the literature, we have defined a *symbolic semantics* (Section 6) that, for each trace, reduces infinite branching due to messages derivable from κ into a finite number of symbolic transitions. These classes are tailored on the structure of the input message by means of a process similar to unification, but more complex because of the presence of asymmetric keys that do not "unify" with themselves, but with their complementary instances. Decidability results previously presented will be used to prove that the session semantics is represented by the symbolic one in a correct and complete way and that the associated logic is still decidable.

The unbounded number of principal instances. Even if it has been proved that there are properties that are not decidable by testing sessions with a finite number of principals instances (e.g., [47,30]), in practice verification has to be anyway limited to sessions with a bounded number of principal instances to avoid non-termination. However, most of the known attacks have been discovered by analysing sessions with few principal instances. Hence, devising frameworks that contribute to enhance and facilitate the treatment of multi-sessions, e.g. in the semantics of the calculus, in the expressiveness of the logic, and consequently in the supporting tools, has appeared to be anyway methodologically relevant.

5 The $\mathcal{PL}$ logic

The properties that the execution of a security protocol should enforce are expressed as formulas of $\mathcal{PL}$, the *Protocol logic* introduced in [18,61]. They refer to the messages exchanged in the protocol, the relations among the principals that exchange them, and the amount of the knowledge that the intruder can acquire interfering with the execution of the protocol. Coherently with the multi-session approach introduced, the logic features quantified variables ranging over instance indexes, so as to express (causal) relations over data and the instances in which they occur. Quantified formulas allow us to abstract from the actual number of instances participating in a given multi-session execution of the protocol.

This logic results expressive enough to cover most of the more commonly addressed properties. For instance, *integrity*, generalising the approach introduced in [2], can be modelled by a formula that expresses that "a datum does not differ from its expected value", *secrecy* as non-membership of the intruder knowledge, and *authen-*

tication by a formula that expresses relations between the values assumed by indexed variables and their instances. This last representation comprises what is generally expressed by means of *correspondence assertions* [63], which express causality relations over observed actions. Section 5.1 discusses relations with correspondence assertions.

The logic insists on a representation of the data, i.e. messages and variables, whose indexes may also be variables. Such variables $\{i, j, l, \dots\}$, called *index variables*, range over the set of indexes and the set of the so extended data is indicated as M_{V_I}. For instance, A_3^+ is the public key of instance A_3, while A_i^+ is the public key of an instance of A.

Definition 9 ($\mathcal{PL}$ – Syntax) Let be $m,\ xa_i \in M_{V_I}$, and $A \in PN$. The formulas ϕ and ψ of the logic $\mathcal{PL}$ (protocol logic) are defined as follows:

$$\phi, \psi ::= xa_i = m \mid \kappa \rhd m \mid \forall i : A.\ \phi \mid \exists i : A.\ \phi \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi$$

A formula can

- be an equality amongst a variable and a message (which in turn may contain variables). This allows data sent and received, i.e. once they have been assigned to variables, to be compared;
- represent the derivability of a message from the intruder knowledge: $\kappa \rhd m$ (for both the logical operator and the process of message derivation the same symbol $\rhd$ has been used);
- be universally and existentially quantified over the set of the indexes of principal instances. Such quantifiers, namely $\forall i : A$ and $\exists i : A$, are read respectively as "for all the instances i of A" and "exists an instance i of A". Quantifiers bind occurrences of index variables;
- be composed of the standard boolean operators. The derived formulas $\phi \rightarrow \psi$, $x_i \neq m$ and $\kappa \not\rhd m$ are read as usual as $\neg\phi \vee \psi$, $\neg(x_i = m)$ and $\neg(m \rhd \kappa)$, respectively.

Examples of quantified formulas are $\forall i : A.\ xa_i \neq ya_i$ stating that "for each instance of A, the two variables xa and ya never assume the same value", $\forall i : A.\ \forall j : A.\ xa_i \neq ya_j$ stating that "it never happens that any two instances of xa and ya assume the same value, whichever instance of A they may occur in", $\forall i : A.\ \kappa \not\rhd A_i^-$ stating that "the intruder does not know any of the private keys of the instances of A" and $\forall i : A.\ xa_i = k$ "all the instances of A associate to the variable xa the same symmetric key k". Formulas like $\forall i : A.\ xa_i...$ where xa does not occur in A are considered as erroneously specified properties. Other statements can be expressed by means of implications and others operators, like "given a server instance, if there is an instance of another principal which the server believes to be connected to, then they share a key and any other principal instance playing the role of B cannot share the same key" or "if an instance B_i receives a message from A_j, then A_j must have a reference to the receiver" which, adapted to the WMF protocol of Example 2, can be respectively formalised as $\forall i : S.\forall j : A.(u_i = A_j \rightarrow (xas_j = ya_i \wedge \forall h : B.xas_j \neq zbs_h))$ and $\forall i : B.\forall j : S.(zsb_i = S_j \rightarrow v_j = B_i)$.

The satisfaction of a formula depends on the point in the execution of a the protocol in which it is tested, so that the models of the logic are naturally defined in terms of χ and κ, i.e. the assignments due to communications and the intruder knowledge relative to a specific execution state. For the purposes of verification, we focus on final states, which surely contain enough information to instantiate the variables in the formula, according to the intuition that properties must be enforced by protocols as a consequence of their correct execution. Checking non final states could also be possible, as far as enough information has been collected (or a notion of model for non ground formulas is given). In general, this seems computationally expensive and unnecessary since, if a security property ϕ fails to hold in a state, it will not be recovered in states extending the trace to a final trace (instantiated variables of ϕ cannot be changed). Considering deadlock states is more delicate. They also can be checked without changing our framework (provided that they contain enough information), but we avoid it. Indeed, we conjecture that the intruder is always able to collect enough information so as to let any principal terminate, when principals are derived from protocols narration. Namely, all the pending input actions can eventually be matched by messages generated from intruder's knowledge.

Models of the logic are pairs $\langle \kappa, \chi \rangle$ and the relation $\kappa \models_\chi \phi$, read as "the set κ under the variable assignment χ is a model for the formula ϕ", defines the semantics of closed (w.r.t. index variables) formulas.

Definition 10 (Model for $\mathcal{PL}$ formulas) Let χ be a substitution from indexed variables to indexed messages, κ a knowledge and ϕ a closed formula of $\mathcal{PL}$. Then $\langle \kappa, \chi \rangle$ *is a model for* ϕ if $\kappa \models_\chi \phi$ can be proved by the following rules (where n stands for an instance index):

$$\frac{xa_n\chi = m\chi}{\kappa \models_\chi xa_n = m}\ (=) \qquad \frac{\kappa \rhd m\chi}{\kappa \models_\chi \kappa \rhd m}\ (\rhd)$$

$$\frac{\text{exists } n \text{ s.t. } A_n \in \kappa \quad \kappa \models_\chi \phi[n/i]}{\kappa \models_\chi \exists i : A.\ \phi}\ (\exists)$$

$$\frac{\text{forall } n \text{ s.t. } A_n \in \kappa \quad \kappa \models_\chi \phi[n/i]}{\kappa \models_\chi \forall i : A.\ \phi}\ (\forall)$$

$$\frac{\kappa \models_\chi \phi \quad \kappa \models_\chi \psi}{\kappa \models_\chi \phi \wedge \psi}\ (\wedge) \qquad \frac{\kappa \models_\chi \phi}{\kappa \models_\chi \phi \vee \psi}\ (\vee 1)$$

$$\frac{\kappa \models_\chi \psi}{\kappa \models_\chi \phi \vee \psi}\ (\vee 2) \qquad \frac{\kappa \not\models_\chi \phi}{\kappa \models_\chi \neg\phi}\ (\neg)$$

The definition of equality $xa_n = m$ and derivability $\kappa \rhd m$ are straightforward, $m\chi$ is expected to be ground, since under the assumptions made, at the end

of a protocol execution all the variables are expected to be instantiated, if not, it is reasonable to assume that the formula does not hold. Formulas are required to be closed with respect to index variables and quantifiers are eliminated according to the (finite) set of instances that have taken part into the protocol execution (recorded in the intruder knowledge, $A_n \in \kappa$). The quantifier also specifies the principal which instances belong to $(i : A)$. Quantifier elimination makes index variables to disappear from (closed) formulas. Rules for $\vee$, $\wedge$ and $\neg$ are standard. Our finitary interpretation of the logic allows the following result to be proved.

Theorem 2 (Decidability of $\models$) *Given a $\mathcal{PL}$ formula ϕ, a knowledge κ and an assignment χ, the statement $\kappa \models_\chi \phi$ is decidable*

Proof By structural induction on ϕ. For the basic cases, $=$ (syntactic equivalence) is trivially decidable, and $\rhd$ is decidable by Theorem 1.
The inductive step easily follows: the cases for $(\wedge)$, $(\vee 1)$ and $(\vee 2)$ are defined in terms of structurally simpler formulas, the cases for $(\exists)$ and $(\forall)$ are defined in terms of a finite set of structurally simpler formulas, and the case for $(\neg)$ holds by observing that $\kappa \models_\chi \phi$ is decidable by inductive hypothesis and hence also $\kappa \not\models_\chi \phi$ is decidable. □

Example 4 The natural property that the WMF protocol is expected to guarantee is the secrecy of the session key kab generated by A. This property can be expressed by a formula like

$$\forall i : A.\ \kappa \not\rhd kab_i.$$

As it can be discovered by performing an exhaustive analysis on the possible sharing of keys (Section 10), this property does not hold in the hypothesis that the intruder impersonates an instance of B, and A tries to exchange the session key with that instance. Technically, this is the case when I appears in the set of principals and it is allowed to share a key with S. In this case, the session would be $\{A_1, S_2\}$, where

$$A_1 \triangleq ()[out((A_1, \{(ta_1, \mathbf{I}, kab_1)\}_{kas}))]$$
$$S_2 \triangleq ()[in((A_1, \{(?t_2, \mathbf{I}, ?r_2)\}_k)).out(\{(ts_2, A_1, r_2)\}_{\mathbf{ksi}})]$$

with ksi the key shared by I and S, $\gamma = \{xas_1, ya_2 \mapsto kas,\ yb_2 \mapsto ksi,\ q_1, v_2 \mapsto I\}$ and the initial knowledge was $\kappa_0 = \{I, ksi\}$. A trace exists that yields, in a final state, $\kappa = \{(A_1, S_2, I, ksi, (A_1, \{(ta_1, I, kab_1)\}_{kas}), \{(ts_2, A, kab_1)\}_{ksi}\}$, and $\chi = \{xas_1, ya_2 \mapsto kas,\ yb_2 \mapsto ksi,\ q_1, v_2 \mapsto I,\ t_2 \mapsto ta_1,\ r_2 \mapsto\ kab_1\}$, such that

$$\frac{\{\ldots ksi, \ldots, \{(ts_1, A, kab_1)\}_{ksi}\} \rhd kab_1}{\dfrac{\kappa \models_\chi \kappa \rhd kab_1}{\kappa \models_\chi \forall i : A.\ \kappa \rhd kab_i.}\ (\forall)}\ (\rhd)$$

This "unintended but legal" attack can be easily ruled out by further specifying the desired property:

$$\forall i : A.\ (q_i \neq I \rightarrow \kappa \not\rhd kab_i)$$

"the key is secret, unless A discloses it directly to the intruder" (the open variable q_i of A is for its partner name).

The previous simple example underlines *i)* the subtleties and difficulties of precisely characterising the desired property and the assumptions about the execution context, *ii)* the utility of a framework equipped with suitable abstractions for modelling all the facets of the protocol and its context, *iii)* the expressiveness of the logic.

5.1 On the expressiveness of $\mathcal{PL}$

In order to present the expressiveness of $\mathcal{PL}$, we discuss how it relates to a widely accepted formalism for expressing security properties, namely *correspondence assertions* [63] (CA, for short). Moreover, we illustrate the relevance of one of the main features of our framework, namely *connection formulas*.

The basic ingredient for defining properties as CAs is to formalise principals as finite sequences of actions (basically as in cIP). A CA is a statement $\alpha \hookrightarrow \beta$ where α and β are *uniquely identifiable* actions within a principal[2].

As for cIP, a protocol execution consists of the interleaving of actions from a finite number of instances of principals. Then, it is necessary to distinguish between the actual executed actions since a protocol satisfies $\alpha \hookrightarrow \beta$ when, for all its execution traces, any occurrence of α is into a one-to-one correspondence with a previously appeared occurrence of β. There can be many way of distinguishing occurrences of actions executed by different instances of the same principal, for instance, in [63] indexes are used (as in our framework) while in [15] there is a combined use of names labelling actions and indexes.

In order to state security properties as CAs, input and output actions come together with specific actions (called *internal* actions) that do not require interactions with other principals. For instance, the CA

$$B.\mathbf{endResp}(A) \hookrightarrow A.\mathbf{beginInit}(B) \tag{1}$$

is an authentication property for the WMF protocol of Example 2. The intended meaning of (1) is "whenever an instance of B finishes the protocol supposing that an instance of A had initiated it, then the instance of A actually started the protocol intending to interact with the instance of B".

Notice that, in general, principal formalisation should be extended with internal actions (e.g., **endResp**(_) and

[2] Namely, α and β contain all the necessary information for uniquely determining in the formalisation of the protocol, the principal that can execute α or β.

beginInit(_)) in order to state (1) and, for instance using cIP, the principals of Example 2 should be modified by letting **beginInit**(q) and **endResp**(x) be the first and the last actions of A and B, respectively. In our framework, this is unnecessary because $\mathcal{PL}$ can express the same properties by exploiting quantification on instances and open variables. Indeed, $PL_{WMF} \triangleq \forall i : B.\forall j : A.(x_i = A_j \rightarrow q_j = B_i)$ is the $\mathcal{PL}$ formula corresponding[3] to (1). Also, this is important from a methodological point of view for two reasons. First, when using CAs, the formalisation of principals depends on the property to be checked, namely it must be "augmented" introducing actions related to the property (as above for A and B). On the contrary, cIP allows principals to be derived quite straightforwardly from the protocol narration and without considering the security property, while $\mathcal{PL}$ enables us to state security properties simply by considering cIP processes. Remarkably, this is due to the presence of open variables together with session quantifiers that allow us to express causal dependency as shown above in PL_{WMF}. Second, we argue that expressing security properties in terms of CAs might result a complex activity since they require to reason about execution traces in order to express causality conditions. Differently, in $\mathcal{PL}$ properties can be straightforwardly expressed by giving the expected connections between variables in principal specifications.

On the other hand, the definition of CAs might seem intuitive and simpler than the definition of $\mathcal{PL}$; also, it is *syntax independent*, i.e., based on the underling execution traces rather than on the variables occurring in the principals. Therefore, CAs can be used to give general secrecy properties [63], namely properties that do not depend on the protocol specification. However, the price to pay is breaking the orthogonality between the formalisation of principals and security properties because principals must conform to the formula expressing the property. Also, it is sometime necessary to introduce special actions (not present in any principal description) to enforce correspondences, which usually is error prone because it deviates from the narration of the protocol. For instance, the general secrecy of a message m is stated in [15] as $g(m) \hookrightarrow \bot$ where $\bot$ and g are special actions and "regular" principals run in parallel with a "guardian" process that is the equivalent of the cIP process $()[g(x).0]$. In $\mathcal{PL}$, a general notion of secrecy can be simply stated as a properly universally quantified formula of type $\kappa \rhd m$. Apparently, both $\mathcal{PL}$ and CAs can hardly be used to give general versions of security properties other than secrecy, albeit schemata formula can heuristically be individuated in $\mathcal{PL}$. For instance, authentication formulas frequently have the same pattern as PL_{WMF} above.

[3] Observe that, in $\mathcal{PL}$ the one-to-one correspondence is implicitly given by the fact that instances are distinguished by indexes, hence (indexed) variables (e.g., x_i or q_j) spot the (unique) instance they occur in.

Finally, it is worth pointing out the distinguished use of $\mathcal{PL}$ logic and open variables to constrain the possible connections among principals, by suitably restricting the set of names that can be used to instantiate a given open variable (e.g., as done in Example 4). Let us consider again the WMF server of Example 2 and suppose we want to check the protocol under the assumption that a principal cannot play at the same time the role of initiator and responder. This can be easily expressed with the connection formula $\forall i : S.u_i \neq v_i$. Basically, a connection formula is a $\mathcal{PL}$ formula not concerning variables used in binding occurrences. Namely, a connection formula involves only open variables and names. As clarified in Section 10, connection formulas are useful for avoiding that uninteresting executions of the protocols are checked. This is a further advantage of modelling different facets of security protocol verification using orthogonal concepts. Indeed, connections formulas are specified independently from the formulas stating security properties. Section 10 provides more examples of usage of connection formulas.

6 Symbolic semantics of cIP

The semantics of cIP given in Section 4 (hereafter referred to as concrete semantics) is adequate for formally characterising the possible traces of protocols. However, it is not suitable for verification purposes, the main drawback being infinite branching. Indeed, any input pattern might be matched by a possibly infinite number of messages derivable from the knowledge in the current configuration (see the final remarks of Section 4).

We undertake the infinite branching due to the choice of messages by defining a *symbolic semantics* where the actual instantiation of variables is delayed, in a sort of "lazy-evaluation". Roughly, a variable x is replaced by the *symbolic variable* $x[\kappa]$, which represents all the messages derivable from κ that can be assigned to x; even if no choice is committed for x, all possibilities are preserved by $x[\kappa]$. Hence, computations are now *symbolic traces*, namely traces where symbolic variables can appear. A (symbolic) trace represents a possibly infinite set of concrete traces, namely, the set of traces obtained by instantiating the symbolic variables with concrete messages derivable from the associated knowledge.

Hereafter, we use $\mathcal{N}$ for $PN \cup NO$ and $\mathcal{K}$ for $K \cup PN^+ \cup PN^-$ and, for the sake of the presentation, we ignore indexed messages and variables (constructions and theorem do not depend on instance indexes and, therefore, can adapt to indexed instances of principals).

Definition 11 (Symbolic variable and message) The set of *symbolic variables* is

$$\underline{V} \triangleq \{x[\kappa], \hat{x}[\kappa] \mid x \in V \wedge \kappa \in \wp_{\text{fin}}(\underline{M})\}$$

($\hat{x}[\kappa]$ is said a *marked symbolic variable*). We use $\mathtt{x}[\kappa]$ for denoting either $x[\kappa]$ or $\hat{x}[\kappa]$.

The set of *symbolic messages* $\underline{M}$ (ranged over by m) is generated by the following productions:

$$\underline{M} ::= \mathcal{N} \mid \mathcal{K} \mid ?V \mid \mathtt{x}[\kappa] \mid (\underline{M}, \underline{M}) \mid \{\underline{M}\}_{\underline{M}}$$

where $\mathtt{x}[\kappa] \in \underline{V}$.

Symbolic messages consist of (concrete) messages where symbolic variables may occur, analogously, a symbolic knowledge is a finite subset of $\underline{M}$, again indicated as κ. Notice that $\underline{M}$ and $\underline{V}$ are defined by mutual recursion. As in the concrete case, symbolic messages can be derived from a knowledge κ according to Definition 12, which is the symbolic counterpart of Definition 2. (In the following, symbolic messages appearing in *in* actions are called *data*.)

Definition 12 (Symbolic message derivation) Relation $\trianglerighteq \subseteq \wp_{\text{fin}}(\underline{M}) \times \underline{M}$ is the smallest relation satisfying the following rules:

$$\frac{m \in \kappa}{\kappa \trianglerighteq m} \qquad \frac{\kappa \trianglerighteq m \quad \kappa \trianglerighteq n}{\kappa \trianglerighteq (m,n)} \qquad \frac{\kappa \trianglerighteq m \quad \kappa \trianglerighteq \lambda}{\kappa \trianglerighteq \{m\}_\lambda}$$

$$\frac{\kappa \trianglerighteq (m,n)}{\kappa \trianglerighteq m} \qquad \frac{\kappa \trianglerighteq (m,n)}{\kappa \trianglerighteq n} \qquad \frac{\kappa \trianglerighteq \{m\}_\lambda \quad \kappa \trianglerighteq \lambda^-}{\kappa \trianglerighteq m}$$

$$\frac{\kappa \trianglerighteq m \text{ for all } m \in \kappa'}{\kappa \trianglerighteq \mathtt{x}[\kappa']}$$

We say that κ *derives* m (or m *is derived from* κ) if $\kappa \trianglerighteq m$ holds.

The first rules of Definition 12 recast those in Definition 2 for symbolic messages and knowledge. The last rule in Definition 12 allows the intruder to generate symbolic variables from κ. When the hypothesis holds, we say that κ *covers* κ' (or, equivalently, κ *covers* $\mathtt{x}[\kappa']$); similarly, κ covers $m \in \underline{M}$ if either κ covers all the symbolic variable occurring in m or $\kappa \trianglerighteq m$ whenever $m \in M$. Finally, κ covers a symbolic substitution σ if κ covers any symbolic variable in $\text{dom}(\sigma)$ and any message in $\text{cod}(\sigma)$.

The intended meaning of symbolic variables is evident when considering how they match each other, which is basically obtained by lifting matching from messages (Definition 7) to symbolic messages.

Definition 13 (Symbolic matching) Two messages $m, n \in \underline{M}$ *symbolically match* (written as $m \simeq n$) iff one of the following alternatives applies:

- $m = n \ \wedge \ m, n \in \mathcal{N} \cup \mathcal{K}$,
- $m = (p,q) \ \wedge \ n = (p',q') \ \wedge \ p \simeq p' \wedge \ q \simeq q'$,
- $m = \{m'\}_\lambda \ \wedge \ n = \{n'\}_{\lambda^-} \ \wedge \ m' \simeq n'$,
- $m = n = \hat{x}[\kappa]$.

Symbolic matching reduces to syntactic equality when names or keys are considered, two pairs symbolically match if their corresponding components do so and, if m symbolically matches m' then $\{m\}_\lambda$ and $\{m'\}_{\lambda^-}$ symbolically match. Matching of symbolic variables requires special care, indeed, it only holds between $\hat{x}[\kappa]$ and itself. Symbolic marked variables $\hat{x}[\kappa]$ have been introduced to overcome a specific problem arising when using asymmetric cryptography. For instance, consider $\kappa = \{A^+\}$, then $x[\kappa] \simeq x[\kappa]$ would not reflect matching for all the concrete messages derivable from κ, e.g., $\{A^+\}_{A^+} \not\sim \{A^+\}_{A^+}$. Therefore, in such cases, $\hat{x}[\kappa]$ will avoid considering those substitutions that cause this problem, according to the following definition.

Definition 14 (Symbolic substitution) A *symbolic substitution* is a partial function $\sigma : V \cup \underline{V} \to \underline{M}$ such that, whenever $\sigma : \mathtt{x}[\kappa] \mapsto m$, $\kappa \trianglerighteq m$ and, if $\mathtt{x}[\kappa]$ is marked, any sub-term of m is not a cryptogram encrypted with an asymmetric key.

A symbolic substitution σ is *valid for* $m \in \underline{M}$ *with respect to a behavioural expression* E if, and only if, $(x[\kappa])\sigma \in \mathcal{K} \cap \kappa$, for all $x[\kappa]$ appearing in E or in m as encryption key. We let ε denote the symbolic substitution with empty domain.

Valid symbolic substitutions guarantee that variables are kept consistent with their use in the protocol (a symbolic variable used as a key must be instantiated only to keys).

The symbolic semantics mimics the concrete one, its states are triples $\langle \mathcal{C}, \chi, \kappa \rangle$ where principals in $\mathcal{C}$ may exchange symbolic messages, χ is a symbolic substitution and $\kappa \in \wp_{\text{fin}}(\underline{M})$ is a (finite) set of symbolic messages.

Definition 15 (Symbolic session semantics) The *symbolic session semantics* is the smallest relation induced by the inference rules in Figure 3.

Rules (*out*) and (*join*) work as in the concrete semantics of cIP. The interesting rule is (*in*) where the chosen message m is derived from κ and must symbolically match d via a valid substitution σ that instantiates m by replacing each $\mathtt{x}[\kappa']$ appearing in key position within E with a key derivable from κ'. Basically, σ represents the constraints on symbolic variables of its domain that are propagated within the session, recorded in the bindings $\chi\sigma$ and used to update the knowledge. The choice of σ makes the (*in*) rule a source of non-determinism but, differently from the concrete semantics, the number of alternatives can be made finite, as shown in Section 7.

Remark 1 Symbolic semantics preserves a useful sort of "monotonicity" of sets κ. In fact, if κ covers any message in κ, then so does the knowledge in the target state of a transition between symbolic sessions. Indeed, either a message is generated from κ or it is an already covered message and added to κ. Since we will consider traces that start from states having concrete knowledge, hereafter we assume that any symbolic knowledge enjoys the cover property and this guarantees that composition of substitutions in subsequent states works as expected.

$$\frac{E \xrightarrow{out(m)} E'}{\langle\{(\tilde{X})[E]\} \cup \mathcal{C}, \chi, \kappa\rangle \overset{i(m)}{\rightsquigarrow} \langle\{(\tilde{X})[E']\} \cup \mathcal{C}, \chi, \kappa \cup m)\rangle} (\underline{out})$$

$$\frac{E \xrightarrow{in(d)} E' \qquad \kappa \trianglerighteq m \qquad \sigma \text{ valid symb. substit. for } m \text{ wrt } E \text{ s.t. } m\sigma \simeq d\sigma}{\langle\{(\tilde{X})[E]\} \cup \mathcal{C}, \chi, \kappa\rangle \overset{o(m\sigma)}{\rightsquigarrow} \langle\{(\tilde{X})[E'\sigma]\} \cup \mathcal{C}\sigma, \chi\sigma, \kappa\sigma\rangle} (\underline{in})$$

$$\frac{\mathcal{C}' = join(A_i, \gamma, \mathcal{C}) \qquad i \text{ new}}{\langle \mathcal{C}, \chi, \kappa\rangle \overset{j(A_i\ \gamma)}{\rightsquigarrow} \langle \mathcal{C}', \chi\gamma, \kappa \cup \{A_i, A_i^+\}\rangle} (\underline{join})$$

Fig. 3 Session symbolic semantics

Any session may be non-deterministically extended with new principals added by join transitions. As discussed in Section 4.2, this form of infinite branching is addressed[4] by considering a finite number of principal instances [50,38,13].

7 Regaining finite branching

This section shows how cIP symbolic semantics can be finitely represented. Intuitively, in rule ($\underline{in}$) of Figure 3 only the "most general" messages are considered instead of all suitable ones. Basically, when $?x$ can be assigned with an infinite number of messages derivable from κ, the symbolic substitution simply records that x must take a message derivable from κ, rather than considering a substitution for each such messages. Example 5 shows our construction in a simple case.

Example 5 Let $d = \{?x\}_k$ and $\kappa = \{\{A\}_k, k\}$ be the messages to be used for deriving d after having assigned a (suitable) message to x. For instance, $\sigma : x \mapsto A$ is such a substitution because $\kappa \trianglerighteq d\sigma$. However, instead of substituting x with A, we consider the symbolic substitution $x \mapsto x[\kappa]$ which encompasses σ. Hence, we can simply consider the more general transition and safely discharge the other one in the symbolic transition system.

Most general matching messages and the corresponding substitutions are computed by means of the mutually recursive functions μ and ν defined below. The former constrains symbolic variables of the datum as weakly as possible, while the latter has the burden of checking whether can symbolic variables be instantiated with cryptograms and to mark them if necessary.

[4] This problem is dealt with differently in the static analysis approach, e.g., [12]. It does not consider actual computations but exploits static information of the protocol specification to detect attacks. This yields semi-decidable procedures for analysing protocols; indeed, either a protocol is found correct regardless the number of sessions or *suspected* attacks are reported. In fact, the outcome of the static analysis can only suggest possible attacks that might not take place within actual computations.

Definition 16 (Intruder output messages) Function $\mu : (\underline{M} \times \wp_{\text{fin}}(\underline{M})) \to \wp(\underline{M} \times [(V \cup \underline{V}) \to \underline{M}])$ is defined in Figure 4.

Function μ yields the set of pairs (m, σ), where $m \in \underline{M}$ and σ is a substitution, such that $\kappa \trianglerighteq m$ and $m\sigma \simeq d\sigma$ (Proposition 3 shows that $\mu(d, \kappa)$ is finite when κ is finite). The choice of a message m derivable from the current knowledge κ matching an input datum d, is driven by d itself. In the basic case $d = ?x$, the current κ is associated to x yielding the substitution $x \mapsto x[\kappa]$. If d is a name derivable from (actually, belongs to) κ then μ returns d together with the empty substitution. If d is a symbolic variable $\mathtt{x}[\kappa']$, then it must have been generated in a previous communication and hence κ covers κ'. However, according to Definition 13, it is necessary to constraint $\mathtt{x}[\kappa']$ to $\hat{x}[\kappa']$. If $d = (e, f)$ then first $\mu(e, \kappa)$ is computed; next, for each $(e', \sigma_e) \in \mu(e, \kappa)$, the substitution σ_e is propagated to f and κ and $\mu(f\sigma_e, \kappa\sigma_e)$ is computed; finally, the substitutions are composed (the order in which (e, f) is visited is imposed by the scoping rules of cIP). If $d = \{e\}_{\lambda^-}$ and $\lambda \in \kappa$, the problem reverts to producing the content of the cryptogram. Moreover, κ might contain cryptograms $\{e'\}_\lambda$ (regardless if $\lambda^- \in \kappa$ or not). In this case, the problem is not to construct m and σ such that $m\sigma \simeq e\sigma$, but to check whether $e'\sigma \simeq e\sigma$ for a suitable σ. This is the task of function ν (Definition 17). When none of the previous cases applies, the function μ returns the empty set of substitutions corresponding to the absence of matching messages derivable from κ.

Function μ relies on ν to verify if $\{e\}_\lambda$ and $\{e'\}_{\lambda^-}$ can match under an appropriate substitution.

Definition 17 (Checking substitution) Given two sets of messages κ_1 and κ_2, let be $\kappa_1 \sqcap \kappa_2 = \{m \in \kappa_1 \cup \kappa_2 : \kappa_1 \trianglerighteq m \wedge \kappa_2 \trianglerighteq m\}$. If $m \in \underline{M}$ is a message and $d \in \underline{M}$ is a datum, then function $\nu : (\underline{M} \times \underline{M}) \to [(V \cup \underline{V}) \to \underline{M}]$ is defined in Figure 5.

Most of the cases in Figure 5 are straightforward, hence we comment only the non trivial ones. If $m = x[\kappa]$ and $d = y[\kappa']$ are symbolic variables, then they are constrained to the symbolic variable $\hat{x}[\overline{\kappa}]$ such that $\overline{\kappa} \trianglerighteq m$ and

$$\mu(d,\kappa) = \begin{cases} \{(x,\ x \mapsto x[\kappa])\}, & d = ?x \\ \{(d,\varepsilon)\}, & d \in \mathcal{N} \cup \mathcal{K}\ \wedge\ \kappa \trianglerighteq d \\ \{(x[\kappa'], \mathbf{x}[\kappa'] \mapsto \hat{x}[\kappa'])\}, & d = \mathbf{x}[\kappa']\ \wedge\ \kappa \text{ covers } \kappa' \\ \left\{ ((e',f'), \sigma_e\sigma_f) : \begin{array}{l}(e',\sigma_e) \in \mu(e,\kappa)\ \wedge \\ (f',\sigma_f) \in \mu(f\sigma_e, \kappa\sigma_e)\end{array} \right\}, & d = (e,f) \\ \left\{ (\{e'\}_\lambda, \sigma_e) : \begin{array}{l}(\kappa \trianglerighteq \lambda \wedge (e',\sigma_e) \in \mu(e,\kappa))\ \vee \\ (\{e'\}_\lambda \in \kappa \wedge \sigma_e \in \nu(e',e))\end{array} \right\}, & d = \{e\}_{\lambda^-} \\ \emptyset, & otherwise \end{cases}$$

Fig. 4 Definition of μ

$$\nu(m,d) = \begin{cases} \{x \mapsto m\}, & d = ?x \\ \{\varepsilon\}, & d = m \in \mathcal{N} \cup \mathcal{K} \\ \left\{ \sigma_e\sigma_f : \begin{array}{l}\sigma_e \in \nu(e,e')\ \wedge \\ \sigma_f \in \nu(f\sigma_e, f'\sigma_e)\end{array} \right\} & m = (e,f)\ \wedge\ d = (e',f') \\ \nu(e,e'), & m = \{e\}_\lambda\ \wedge\ d = \{e'\}_{\lambda^-} \\ \{x[\kappa], y[\kappa'] \mapsto \hat{x}[\overline{\kappa}], \hat{x}[\overline{\kappa}]\}, & m = x[\kappa]\ \wedge\ d = y[\kappa']\ \wedge\ \bar{\kappa} = \kappa \sqcap \kappa' \neq \emptyset \\ \{(y[\kappa] \mapsto m')\sigma : (m',\sigma) \in \mu(m,\kappa)\}, & m \in \underline{M} \setminus \underline{V}\ \wedge\ d = y[\kappa] \\ \{(x[\kappa] \mapsto n)\sigma : (n,\sigma) \in \mu(\kappa,d)\}, & m = x[\kappa]\ \wedge\ d \in \underline{M} \setminus \underline{V} \\ \emptyset, & \text{otherwise} \end{cases}$$

Fig. 5 Definition of ν

$\overline{\kappa} \trianglerighteq d$. We remark that operation $\sqcap$ allows to constraint $x[\kappa]$ and $y[\kappa]'$ to larger set of messages $\kappa \sqcap \kappa'$ that covers the messages derivable from both κ and κ' (indeed, by induction on the structure of m, it can be proved that $\kappa \trianglerighteq m \wedge \kappa' \trianglerighteq m \iff \kappa \sqcap \kappa' \trianglerighteq m$). If $d = y[\kappa]'$ and m is not a symbolic variable, it is necessary to check if κ can generate a message that matches m. This is obviously done by $\mu(\kappa, m)$ (similarly for $m = x[\kappa]$ and $d \in \underline{M} \setminus \underline{V}$).

The following proposition states that μ terminates on finite set of messages.

Proposition 3 *If $\kappa \subseteq \underline{M}$ is finite, then $\mu(d,\kappa)$ is finite, for any datum d.*

Proof The proof trivially follows by induction on the structure of d considering that (i) the base cases always yield finite sets and (ii) recursive calls in Figure 4 are done on terms having a decreasing complexity and preserve finiteness of knowledges. □

Example 6 Consider the set of symbolic messages $\kappa = \{k,\ \{no\}_{B^-},\ no,\ \{y[A^+, \{A^+\}_k]\}_{C^+}\}$. If $d = z[\kappa_1]$ with $\kappa_1 = \{\{no\}_k, \{no\}_{B^-}\}$, then $\kappa \trianglerighteq z[\kappa_1]$ and $\mu(d,\kappa) = (d, z[\kappa_1] \mapsto \hat{z}[\kappa_1])$.

If $d = \{w[k, A^+]\}_{C^-}$, is a cryptogram containing a symbolic variable and $\{y[A^+, \{A^+\}_k]\}_{C^+} \in \kappa$, then $\mu(d,\kappa)$ invokes ν which computes $\overline{\kappa} = \{A^+, \{A^+\}_k\}$. Finally, the symbolic variables are mapped to $\hat{w}[\overline{\kappa}]$.

(Notice also that Example 6 shows that $\sqcap \neq \cap$.)

Function μ takes into account *all* the possible cases, hence the symbolic semantics faithfully represent all the evolutions derivable according to the concrete semantics.

Proposition 4 *Let d be a datum and $\kappa \in \wp(\underline{M})$ covers d. If $(m,\sigma) \in \mu(d,\kappa)$ then $d\sigma \simeq m\sigma$ and $\kappa \trianglerighteq m$.*

The proof is given in Appendix B.

Theorem 3 *Let d be a datum, $m \in M$ a concrete message and $\kappa \in \wp_{fin}(\underline{M})$ such that $\kappa \trianglerighteq m$. For any symbolic substitution σ covered by κ and $m\sigma \simeq d\sigma$, there are a substitution ρ and $(\underline{m}, \underline{\sigma}) \in \mu(d,\kappa)$ such that $\mathrm{dom}(\rho) \subseteq \mathrm{dom}(\sigma)$, $\underline{\sigma}\rho = \sigma$ and $\underline{m}\sigma \simeq m$.*

Proof By induction on the structure of d (we show the details of the base cases since the others easily follow by the inductive hypothesis).

If $d = ?x$ then, by definition, $\mu(d,\kappa) = \{(x, x \mapsto x[\kappa])\}$. Any σ covered by κ such that $m\sigma \simeq x\sigma$ must associate to x either a symbolic variable $x[\kappa']$ or a message m' derivable from κ'. In either case, κ covers $x\sigma$ by hypothesis, hence σ can only be a further instantiation of $x \mapsto x[\kappa]$.

If $d \in \mathcal{N} \cup \mathcal{K}$ then $m\sigma \simeq d\sigma = d$, hence $\kappa \trianglerighteq d$ and $\mu(d,\kappa) = \{(d,\varepsilon)\}$ and either $m = d$ or $\sigma : m \mapsto d$; if $m = d$ then $\sigma = \varepsilon$, while, in the other case $\varepsilon\sigma = \sigma$.

If $d = x[\kappa']$ then $\mu(d,\kappa) = \{(x[\kappa'], x[\kappa'] \mapsto \hat{x}[\kappa'])\}$ and $d\sigma$ is either a symbolic variable or a message. In the former case, it must be a marked variable $\hat{y}[\kappa']$ which can be obtained by further instantiations of $x[\kappa'] \mapsto \hat{x}[\kappa']$ while in the latter case $\sigma : x[\kappa] \mapsto m'$ whose domain contains $x[\kappa']$ and $\kappa \trianglerighteq m'$. □

Theorem 3 states that any "unifier" of d and m derivable from κ can be obtained by further instantiating a substitution computed by μ. If $(\underline{m}, \underline{\sigma}) \in \mu(d,\kappa)$, then $\underline{\sigma}$ is "minimal" (with respect to κ). Namely, any other substitution σ such that $m\sigma \simeq d\sigma$, can be decomposed as $\underline{\sigma}\rho$ for a suitable substitution ρ. Intuitively, μ determines "the most general matching" substitution among m and d and this is basically achieved by constraining the symbolic variables as weakly as possible. Indeed, μ takes into account any pair (m,σ) satisfying the hypothesis of the rule ($\underline{in}$) in Figure 3. Messages matching a datum d either have a structure similar to d or, at some level, they are replaced by symbolic variables. Since μ always introduces symbolic variables that cover any other variable that can be generated by κ, we can use such messages for considering all the possible symbolic ($\underline{in}$) transitions.

Our construction has many similarities with the corresponding construction presented in [15]. For instance, the knowledge in a state of a symbolic trace of cIP corresponds to the *basis* of a *frame* in [15], while our μ and ν correspond to a message reduction relation relying on a most general unifier. We remark that, in order to state decidability of symbolic frames, the image finiteness of basis is required in [15], while in our construction this hypothesis is not required. Moreover, the use of unification in [15] introduces symbolic traces that do not correspond to any concrete computation, hence further work is required to eliminate those spurious symbolic computations. On the contrary, Theorem 3 is also relevant to relate symbolic and concrete traces, in the sense that any instantiation of a symbolic trace yields a concrete one.

8 Relating concrete and symbolic semantics

This section shows how the symbolic semantics of cIP faithfully represents its concrete semantics. More precisely, we give a clear correspondence between the traces of the symbolic and concrete semantics. *Symbolic traces* can be defined as done for concrete traces, but, of course, with respect to the symbolic semantics.

Definition 18 (Symbolic trace) A *symbolic trace* is a sequence

$$\underline{T} \;=\; \Sigma_0.\alpha_1.\Sigma_1 \ldots .\alpha_n.\Sigma_n$$

where Σ_0 is an initial state and $\Sigma_{i-1} \overset{\alpha_i}{\rightsquigarrow} \Sigma_i$ $(1 \leq i \leq n)$. If Σ_n is a final state, $\underline{T}$ is a *terminating* trace. For any $i \in \{1,\ldots,n\}$, we say that $\underline{T}$ *reaches* Σ_i and write $\underline{T} \searrow \Sigma$ (and similarly for concrete traces).

Symbolic traces can be instantiated to concrete ones through suitable substitutions that replace all symbolic variables $x[\kappa]$ with concrete messages that can be derived from κ.

Definition 19 (Concretising substitution) A substitution $\rho : \underline{V} \to M$ is a *concretising substitution for* χ if, for any $x[\kappa]$ occurring in messages in $cod(\chi)$, $\kappa\rho \subseteq M$ and $\kappa\rho \rhd (x[\kappa])\rho$.

If $\underline{T}$ is a symbolic trace whose last state has bindings χ and ρ is a concretising substitution for χ, then we say that ρ is a *concretising substitution for* $\underline{T}$.

Given a state $\Sigma = \langle \mathcal{C}, \chi, \kappa \rangle$ and a substitution ρ, we write $\Sigma\rho$ to denote the component-wise application of ρ to Σ, namely $\Sigma = \langle \mathcal{C}\rho, \chi\rho, \kappa\rho \rangle$. Similarly, given a trace $\underline{T}$, $\underline{T}\rho$ is obtained by applying ρ to each state and each label in $\underline{T}$.

Symbolic semantics is coherent with respect to the concrete one. Indeed, any symbolic trace can be concretised to a non-symbolic trace as stated by the following theorem.

Theorem 4 (Correctness of symbolic semantics) *If $\underline{T}$ is a symbolic trace, for all ρ concretising substitutions for $\underline{T}$, $\underline{T}\rho$ is a trace in the concrete semantics.*

The proof is by induction on the length of $\underline{T}$ and is reported in Appendix B.

Basically, Theorem 4 states that symbolic semantics is correct with respect to the concrete one. Namely, as long as symbolic traces are properly instantiated, concrete traces are obtained. The following theorem states the completeness of symbolic semantics with respect to the concrete one.

Theorem 5 (Completeness of symbolic semantics) *If T is a concrete trace starting from $\langle \emptyset, \varepsilon, \kappa_0 \rangle$ then there are*

1. *a symbolic trace $\underline{T} \searrow \langle \mathcal{C}', \chi', \kappa_1 \rangle$ from $\langle \emptyset, \varepsilon, \kappa_0 \rangle$ and*
2. *a concretising substitution ρ for χ'*

such that $\underline{T}\rho = T$.

The proof is by induction on n and case analysis (see Appendix B).

9 Symbolic models

Symbolic traces contain all the necessary information for detecting flaws (if any) in a protocol and retrieving the corresponding attacks of the Dolev-Yao intruder.

By constructing symbolic traces, it is possible to give a finite representation of the bindings and of the knowledge that the intruder acquires during the execution of protocols.

Definition 20 (Symbolic model and attack) Let $\kappa \in \wp_{\text{fin}}(\underline{M})$ and χ be a symbolic substitution. The pair $\langle \kappa, \chi \rangle$ is a *symbolic model for a closed formula* ϕ (written $\kappa \models_\chi \phi$) if, and only if, a concretising substitution ρ for χ exists such that $\kappa\rho \models_{\chi\rho} \phi$ holds.

A *symbolic attack for* ϕ is a symbolic terminating trace $\underline{T}$ reaching a state $\langle \mathcal{C}, \chi, \kappa \rangle$ such that $\kappa \models_\chi \neg\phi$.

Given a protocol and a property ϕ, a symbolic attack is a symbolic trace that, after "concretisation" has taken place, gives rise to a trace of the concrete semantics such that bindings and knowledge of the last state in the trace are model for $\neg\phi$.

There is a strong relation between concrete and symbolic models: the next two theorems prove that they are actually equivalent.

Theorem 6 (Correctness of $\models$) *Given a symbolic trace $\underline{T}$ and a $\mathcal{PL}$-formula ϕ such that $\underline{T} \searrow \langle \mathcal{C}, \chi, \kappa \rangle$ and $\kappa \models_\chi \phi$ then there is a concretisation T of $\underline{T}$ reaching a state $\langle \mathcal{C}', \chi', \kappa' \rangle$ such that $\kappa' \models_{\chi'} \phi$.*

Proof By Definition 20, there is a concretising substitution ρ for χ such that $\kappa\rho \models_{\chi\rho} \phi$. Substitution ρ is a concretising substitution of $\underline{T}$ as well because covering is preserved by symbolic semantics and the symbolic variables in κ and $\mathcal{C}$ are in $\text{dom}(\chi)$, hence, $\underline{T}\rho \searrow \langle \mathcal{C}\rho, \chi\rho, \kappa\rho \rangle$. □

Theorem 7 (Completeness of $\models$) *Let ϕ be a $\mathcal{PL}$-formula, and T a concrete trace such that $T \searrow \langle \mathcal{C}, \chi, \kappa \rangle$ and $\kappa \models_\chi \phi$. If T is a concretisation of $\underline{T}$, then there is a symbolic configuration $\langle \mathcal{C}', \chi', \kappa' \rangle$ such that $\underline{T} \searrow \langle \mathcal{C}', \chi', \kappa' \rangle$ and $\kappa' \models_{\chi'} \phi$.*

Proof By Definition 20, we must prove that there is a concretising substitution ρ for χ' such that $\kappa'\rho \models_{\chi'\rho} \phi$. This is trivially given by the concretising substitution ρ such that $\underline{T}\rho = T$. □

Instead of seeking concretising substitutions of symbolic traces, symbolic models allow us to check a formula ϕ without resorting to concrete models. Concretisation of a symbolic substitution can indeed be driven by *phi* without affecting satisfiability. Impossibility of such a concretisation corresponds to unsatisfiability. Intuitively,

- ϕ is transformed in an equivalent formula where
  - quantifiers have been eliminated (as usual by replacing $\forall$ with conjunctions and $\exists$ with disjunctions),
  - $\neg$ is pushed as far as possible inside the formula and, finally,
  - the result is transformed in a disjunction of conjuncts.
- Disjuncts further constrain symbolic variables

The procedure sketched above can also yield the undefined substitution or a contradictory set of inequalities meaning that κ and χ are not a symbolic model for ϕ.

It is useful to represent $\mathcal{PL}$ formulas in *disjunctive normal form* $\bigvee_{i\in 1,\ldots,u}(\psi_{i,1} \wedge \ldots \wedge \psi_{i,j_i})$ where $\psi_{i,j}$'s are negative or positive atoms (i.e., formulas of the form $x[\kappa] = m$ or $\kappa \rhd m$). Of course, it is straightforward to transform any $\mathcal{PL}$ formula into an equivalent disjunctive normal form (see Appendix B.3).

Example 7 Quantifier elimination from $\phi \triangleq \forall i : A.\exists j : B.(x_i = y_j \wedge \kappa \rhd y_i)$ in $\kappa = \{A_1, B_2, A_3, B_4\}$, yields the normal form formula

$$((x_1 = y_2 \wedge \kappa \rhd y_2) \vee (x_1 = y_4 \wedge \kappa \rhd y_4)) \wedge \\ ((x_3 = y_2 \wedge \kappa \rhd y_2) \vee (x_3 = y_4 \wedge \kappa \rhd y_4))$$

from which, applying the De Morgan laws, we obtain normal forms for ϕ, e.g.,

$$(x_1 = y_2 \wedge \kappa \rhd y_2 \wedge x_3 = y_2 \wedge \kappa \rhd y_2) \vee \\ (x_1 = y_2 \wedge \kappa \rhd y_2 \wedge x_3 = y_4 \wedge \kappa \rhd y_4) \vee \\ (x_1 = y_4 \wedge \kappa \rhd y_4 \wedge x_3 = y_2 \wedge \kappa \rhd y_2) \vee \\ (x_4 = y_4 \wedge \kappa \rhd y_4 \wedge x_3 = y_4 \wedge \kappa \rhd y_4)$$

Given a conjunction of positive atoms ϕ, we consider ϕ to be a set of atoms instead of a conjunct of atoms, since it is a more suitable representation for the algorithm that computes the *refinement substitution* from ϕ.

Definition 21 (Constraining equalities) Given $\kappa \in \wp_{\text{fin}}(\underline{M})$ and a symbolic substitution χ, $\Psi_{\kappa,\chi}$ is the *refinement substitution under κ and χ* and is defined in Figure 6 where m^{-1} is the message obtained from replacing any encryption key λ in m with its inverse λ^-.

Given a set of atoms ϕ, $\Psi_{\kappa,\chi}$ chooses an atom from ϕ and determines a substitution that, if defined (i.e., different from $\perp$), is propagated on κ and χ in the recursive invocations of Ψ. If the returned substitution is $\perp$ then $\Psi_{\kappa,\chi}$ returns $\perp$ too. Basically, Ψ checks, for each atom, whether symbolic variables can be replaced by messages to either unify right-hand- and left-hand-side of equalities, or determine the values that make a message belong to the intruder knowledge. Figure 6 defines $\Psi_{\kappa,\chi}(\phi)$ by induction on the set ϕ. If $\delta_1 = \delta_2$ is an equality in ϕ, then three cases are possible when we apply χ to the equality: (*i*) two symbolic variables are equated, (*ii*) a variable and a message are equated, (*ii*) the equation is a tautology on names. In case (*i*), first it is checked whether the two symbolic variables can be refined with a non-empty common knowledge ($\bar{\kappa} \neq \emptyset$) and, if so, the variables are refined and Ψ continues on the remaining atoms. In case (*ii*), Ψ checks if κ_1 can generate the message and propagates the computed substitution σ. Finally, in case (*iii*), tautologies are simply removed. The

$$\Psi_{\kappa,\chi}(\phi \cup \{\delta_1 = \delta_2\}) = \begin{cases} \Psi_{\kappa\sigma,\chi\sigma}(\phi'), & \delta_1\chi = x[\kappa_1] \ \wedge \ \delta_2\chi = y[\kappa_2] \ \wedge \ \bar{\kappa} = \kappa_1 \sqcap \kappa_2 \neq \emptyset \ \wedge \\ & \sigma = x[\kappa_1], y[\kappa_2] \mapsto x[\bar{\kappa}], x[\bar{\kappa}] \\ \Psi_{\kappa\sigma,\chi\sigma}(\phi'), & \delta_1\chi = x[\kappa_1] \ \wedge \ \delta_2\chi \in \underline{M} \setminus \underline{V} \ \wedge \ x \text{ not occur in } \delta_2\chi \ \wedge \\ & \sigma' \text{ in } \mu((\delta_2\chi)^{-1}, \kappa_1) \ \wedge \ \sigma = \sigma'[x[\kappa_1] \mapsto \delta_2\chi\sigma'] \\ \Psi_{\kappa,\chi}(\phi'), & \delta_1 = \delta_2 \in N \cup K \end{cases}$$

$$\Psi_{\kappa,\chi}(\phi \cup \{\kappa \rhd \delta\}) = \Psi_{\kappa\sigma,\chi\sigma}(\phi'), \quad \sigma' \text{ in } \mu((\delta\chi)^{-1}, \kappa) \ \wedge \ \sigma = \sigma'[x[\kappa_1] \mapsto \delta\chi\sigma']$$

$$\Psi_{\kappa,\chi}(\emptyset) = \chi$$

$$\Psi_{\kappa,\chi}(\phi) = \bot, \text{ otherwise}$$

Fig. 6 Constraint refinement function

atom $\kappa \rhd \delta$ is exploited to check if $\kappa \trianglerighteq \delta$ and yields the constraints on symbolic variables computed by μ. When ϕ is empty, $\Psi_{\kappa,\chi}$ terminates returning χ. All the other possibilities correspond to unsatisfiability of atoms, hence Ψ returns $\bot$. Trivially, Ψ terminates on finite κ and ϕ because μ does so.

Once a substitution has been refined exploiting positive atoms, negative ones test whether the refined model satisfies the whole conjunction. When all negative atoms are satisfied we have found a model that can be concretised to obtain a non-symbolic model of the initial conjunction. Moreover, this test is simpler than the refinement mechanism of positive atoms because we must only consider three cases: $x[\kappa] \neq m$, $x[\kappa] \neq y[\kappa']$, $\kappa \not\rhd m$. By observing that a non-empty set of messages can generate infinite messages, we can immediately derive that the first case is always true, and the second is true when x differs from y ($x[\kappa] \neq x[\kappa]$ is a contradiction). The last case means that $\kappa \not\trianglerighteq m$ must be checked.

As stated by Theorem 8, Ψ yields an effective procedure to decide the validity of a formula in a symbolic model.

Theorem 8 (Decidability of $\models$) *Let ϕ be a $\mathcal{PL}$-formula, $\kappa \in \wp_{fin}(\underline{M})$ and χ a symbolic substitution.*

$$\kappa \models_\chi \phi \iff \Psi_{\kappa,\chi}(\phi) \neq \bot$$

(the proof is reported in Appendix B).

10 Verification in practice

Our formal framework has been turned into a verification environment called $\mathcal{A}$SPASyA (Automatic Security Protocol Analysis via a SYmbolic model checking Approach, [8]). The verification with $\mathcal{A}$SPASyA consists of a quite human-interactive process where the expertise of the analyst is exploited to formalise the protocol and to point the verification process at the properties of interest. Then, a fully automated phase is exploited to explore and analyze the possible protocol execution. The analyst is allowed to vary the intruder knowledge, the portion of the state space to be explored, and the specification of the implicit assumptions, without modifying neither the protocol nor the property specification. This appears to be relevant as soon as one realises that, given the complexity of the problem, verification may become an iterative process pointing at precisely setting the desired experiment. We briefly illustrate the use of $\mathcal{A}$SPASyA showing how the choices done for the theory are reflected on the practice of semi-automated verification. This is illustrated by means of a simple case study, where an "atypical" attack is found by means of a verification session that well illustrates the versatility of the tool, and by another attack that has been discovered by others using different verification techniques, illustrating the generality of the tool. The purpose of this section is not to present the tool and its experimental use into detail, but rather discuss its methodological relevance. A more precise description of $\mathcal{A}$SPASyA is in [9]. The reader interested in the practical experimentation is referred to [8,7], where the tool and the relative documentation can be downloaded, together with reports on the verification of some well-known protocols, as the Needham-Schroeder, KSL, Yahalom, Denning-Sacco key distribution, Beller-Yacobi, Bilateral key exchange and Carlsen protocol.

$\mathcal{A}$SPASyA is written in Ocaml and uses a depth-first strategy when exploring the state space. Though no heuristic has been used, $\mathcal{A}$SPASyA further abstracts cIP symbolic semantics for optimising the state space (e.g., a simple type system permits us not to concretely expand some states but to abstractly represent them by typing some variables). The state space is maintained as compact as possible by means of connection formulas that are checked whenever new instances join a protocol session so that "undesired" configurations are ruled out. Finally, $\mathcal{A}$SPASyA can operate under several modalities, more precisely, it can halt as soon as one attack is found (as generally model checkers do) or it can search the whole state space. Though not technically relevant, this is rather useful during the analysis because formulas and set of messages used in the verification can be more easily "tuned". A more complete description of $\mathcal{A}$SPASyA technical aspects can be found in [7] (Chapter 5) and in [9].

10.1 The verification process

The verification process consists of four steps:

1. formalisation of both the protocol narration in cIP and the security property as a $\mathcal{PL}$ formula;
2. specification of the $\mathcal{PL}$ connection formula that yields the constraints on the open variables to be fullfilled by the join operation;
3. definition of the intruder initial knowledge;
4. the automatic phase of the verification starts. According to the results returned by $\mathcal{A}$SPASyA steps 2 and 3 can be iterated, in order to tune the connection conditions and the initial knowledge.

Step 1 is clearly the most dependent on the analyst expertise, since the translation form the protocol narration into cIP cannot be automated. The practice can give some rule of thumbs, e.g. the initiator usually needs an open variable to be connected to the responder, no open variables should be needed to manage partner identity when this is acquired by means of communications, open variables are generally needed to share keys with servers, etc. The difficulty of exactly expressing a property as a $\mathcal{PL}$ formula has also been underlined. Sometimes formulas can be incrementally refined by observing the results of the analysis until the intended property has been precisely characterised.

Step 2 allows the verification to be restricted to given initial conditions, in particular about the sharing of keys. More precisely, $\mathcal{A}$SPASyA builds all the possible sessions[5] and, among them, those for which the connection formula does not hold are discharged, and only the others are considered for the verification of the property. This is a distinguished feature that facilitates the precise statement of the initial context by ruling out the uninteresting cases so that the accuracy of the verification process is improved.

Step 3 sets the initial power of the intruder. This can be used to let the intruder know some secrets (e.g. compromised keys), for instance to test the robustness of the protocol, and to let the intruder know something about past interactions between principals, e.g. cryptograms exchanged in previous sessions. This is particularly useful, for instance, in finding *replay attacks* where the intruder exploits messages appeared in previous sessions.

10.2 Experiments: an example

We revisit the analysis of the KSL protocol [39], a simplification of Kerberos [40], originally reported in [9]. The protocol aims at the repeated authentication between A and B and consists of two phases.

[5] These are the sessions with a number of principal instances lower than the fixed maximal number of instances and connected in all possible ways.

In the first phase (not reported here and verified as safe in [9]) A and B exchange a session key k^{ab} (generated by a trusted server S) and a *ticket* $\{Tb, A, k^{ab}\}_{k^{bb}}$ containing the expiring time Tb, the name of A and the session key. Only B, that has issued the ticket for A, knows k^{bb}. At the end of the first phase the following facts hold: (i) A and B are aware to have reciprocally interacted together, (ii) A and B share a key and the ticket issued by B in the current session, (iii) the ticket and other exchanged data are known by the intruder and (iv) the key is valid in virtue of the ticket issued by B.

In the second phase, until the ticket is valid, A uses k^{ab} to re-authenticate itself to B without the help of S:

$$\begin{array}{ll}(2.1) & A \rightarrow B : ma, \{Tb, A, k^{ab}\}_{k^{bb}} \\ (2.2) & B \rightarrow A : mb, \{ma\}_{k^{ab}} \\ (2.3) & A \rightarrow B : \{mb\}_{k^{ab}}\end{array}$$

First, A sends a nonce ma and the ticket to B that, provided that ticket is valid, accepts it as referring to A and sends mb together with the cryptogram $\{ma\}_{k^{ab}}$ to A, proving it knows k^{ab}. In the last message, A proves to B that it also knows k^{ab}. In cIP, A and B can be represented as follows:

$$\begin{array}{l} A \triangleq (b, sk, tk)[\\ \quad out(ma, \{b, A, sk\}_{tk}).in(?xmb, \{ma\}_{sk}).out(\{xmb\}_{sk})] \end{array}$$

$$\begin{array}{l} B \triangleq (a, sk, tk)[\\ \quad in(?xma, \{B, a, sk\}_{tk}).out(mb, \{xma\}_{sk}).in(\{mb\}_{sk})] \end{array}$$

where, b and a are used to reconstruct the ticket shared by A and B (facts (i-ii)). Here, time is not modelled (i.e., fact (iv) is assumed) and the time-stamp generated by B is simply substituted by the name of B itself (as something that B can recognise). Fact (iii) is formalised with the initial knowledge κ_0 that contains the names of the principals that executed the first phase together with $\{B_1, A_3, sk_1\}_{tk_1}$, the ticket that they exchanged. The modular setting of the initial conditions provided by $\mathcal{A}$SPASyA, as the knowledge in this case, helps in verifying separate parts or phases of a protocols.

Authentication is based on the mutually exchanged (and decrypted) nonces, and formalised as follows:

$$\begin{array}{l} \bar{\psi}_{KSL} \triangleq \ \forall l : B.\ \forall j : A. \\ \qquad (b_j = B_l \wedge a_l = A_j \rightarrow xma_l = ma_j \wedge xmb_j = mb_l) \end{array}$$

any pair of "partners" B_l and A_j sharing a ticket (and hence being properly connected, $b_j = B_l \wedge a_l = A_j$) eventually exchange nonces ma_j and mb_l.

The results of a series of experiment where the number of instances, the intruder's knowledge and connection formulas are varied are reported in Table 1. Each row corresponds to different knowledge and join conditions, while columns correspond to the case of sessions with two, three or four instances, respectively. Each of these columns reports the number of explored states, the elapsed time (overall, in the format min:sec.hundredth,

	2 Instances			3 Instances			4 Instances		
Join/Know.	States	Time (S-J-V)	Attacks	States	Time (S-J-V)	Attacks	States	Time (S-J-V)	Attacks
$true, \kappa_0$	104	0.83 (1-1-0)	0	3918	2.89 (2-1-1)	48	132996	4:05.63 (1:33-1:37-0:58)	2126
$true, \bar{\kappa}_0$	104	0.83 (1-1-0)	0	6261	4.52 (3-1-1)	84	—	—	—
$\bar{\phi}1_{KSL}, \kappa_0$	101	0.87 (1-1-0)	0	3162	2.93 (2-2-1)	32	89278	10:43.35 (0:55-8:57-0:54)	1000
$\bar{\phi}2_{KSL}, \kappa_0$	71	0.81 (1-1-0)	0	2553	2.68 (1-2-1)	32	72718	10:25.68 (0:44-8:52-0:53)	1000
$\bar{\phi}3_{KSL}, \kappa_0$	62	0.82 (0-1-0)	0	1228	92.29 (1-92-1)	2	...	...	...

Table 1 Attack report for KSL repeated authentication part

and an approximation of the time spent in searching the space (S), joining the principals (J), and verifying the security property (V)).

The first row shows, as expected, the exponential growth of the problem in the number of the principal considered. Here no connection formula has been applied. The second row shows the case of an extended initial knowledge $\bar{\kappa}_0 = \kappa_0 \cup \{\{B_2, A_3, sk_2\}_{tk_2}\}$, i.e., κ_0 augmented with a ticket generated by B_2 for A_3. Illustrating for a naive case the relevance of initial conditions, this change doubles the number of attacks, and increases correspondingly the execution times (the case for four principals has not been tested).

Correctly, in the case of two instances, no attack is found. Attacks are found for three and four instances (thus stressing the relevance of multi-session analysis techniques). One of the attacks reported by $\mathcal{A}$SPASyA for the case of three instances is the following:

(1) $A_3 \rightarrow I : ma_3, \{B_2, A_3, \mathbf{ks}\}_{kb2}$
(2) $I \rightarrow B_2 : ma_3, \{B_2, A_3, \mathbf{ks}\}_{kb2}$
(3) $B_2 \rightarrow I : \mathbf{mb_2}, \{ma_3\}_{ks}$
(**4**) $\mathbf{I \rightarrow B_1 : mb_2, \{B_1, A_3, ks\}_{kb1}}$
(**5**) $\mathbf{B_1 \rightarrow I : mb_1, \{mb_2\}_{ks}}$
(6) $I \rightarrow B_2 : \{\mathbf{mb_2}\}_{\mathbf{ks}}$
(7) $I \rightarrow A_3 : mb_1, \{ma_3\}_{ks}$
(8) $A_3 \rightarrow I : \{mb_1\}_{ks}$
(**9**) $\mathbf{I \rightarrow B_1 : \{mb_1\}_{ks}}$

I is able to authenticate itself to B_1 as A_3, even if A_3 has never intended to communicate with B_1 ((4), (5) and (9)). Noteworthy, the attack is possible because two instances of B (i.e., B_1 and B_2) use the same session key. Remarkably, this possibility has not been forbidden (it was actually assumed in [43], where the attack has been reported while analysing the robustness of KSL). In our framework, once spotted, this case could be easily ruled out from subsequent experiments by means of a suitable connection formula. Moreover, communications between A_3 and B_2 are also not correct. Indeed, in (6), B_2 receives mb_2 instead of ma_3 and in (7), A_3 receives mb_1 (generated by B_1) instead of mb_2 that B_2. This, amongst the rest, clearly violates $\bar{\psi}_{KSL}$, since $xmb_3 = mb_1$ (from (7), xmb_3 is the variable of A_3), while, being A_3 and B_2 partners, it should have been $xmb_3 = mb_2$.

The use of connection formulas is shown in the last rows of Table 1. Three formulas, beyond *true*, have been used in the set of experiments presented here. We remark that the connections formulas are independent from the checked security protocols and do not require any change to the cIP principals.

The first connection formula restricts the analysis to the cases in which A and B properly share the same third party certificate:

$$\bar{\phi}1_{KSL} \triangleq \forall i : A.\ \forall j : B.\ (tk_i = tk_j \rightarrow a_j = A_i \wedge b_i = B_j \wedge sk_i = sk_j)$$

for any A_i and B_j sharing a ticket encrypted by means of the same key ($tk_i = tk_j$) the content of the ticket must be the one generated by B_j, session key included ($sk_i = sk_j$, see facts (i-ii) and (iv) above). This allows for a reduction of the number of the explored states and of the found attacks, significant for the case of four instances (third row). On the other hand, more time is required by the join operation (most of the time consumed for the case of four instances), which needs to verify all the possible assignments for open variables. A second formula is used to further restrict the cases of interest by adding the requirement that at least one A and one B instances are present in the tested sessions:

$$\bar{\phi}2_{KSL} \triangleq \bar{\phi}1_{KSL} \wedge (\exists o : A.\ true) \wedge (\exists o : B.\ true)$$

that slightly improves time and space consumption with respect to the previous test. Inspecting the found attacks, it is easy to realise that many of them are due to the fact that two different instances of A are given the same session key sk. This unintended possibility is ruled out by the third connection formula:

$$\bar{\phi}3_{KSL} \triangleq \bar{\phi}2_{KSL} \wedge (\forall j : B.\ tk_j \neq sk_j) \wedge (\forall i : A.\ \forall j : A.\ (sk_i = sk_j \rightarrow (\exists l : B.\ a_l = A_i \wedge a_l = A_j))$$

that states that the session key and the key used by B for encrypting the ticket must be different ($tk_j \neq sk_j$) and that two instances of A can use the same session key only if they are both connected to the same instance of B, namely they are the same instance (the open variable a_l of B can be instantiated only with one instance of A). The use of this formula allows for a neat characterisation of the protocol flaw, now detected by means of only two attacks reported (one of which discussed above), which allow the problem to be easily understood. On the other hand, this requires a significantly longer verification time, contained in few tens of seconds for the minimal case detecting the attack and amounting to several hours for the case of four instances (not reported in the table, ...).

Summarizing, the experiments reported show how connection formulas can be used to tune the analysis towards the cases of interest. Indeed, many of the attacks ruled out by the use of connection formulas consists of special cases, or cases of scarce interests, e.g. two instances of B using the same session key and the same key for encrypting the tickets (which is unrealistic). A careful discipline of the use of connection formulas, as well as the optimisation of the implementation of formula verification, not attempted in the current implementation of $\mathcal{A}$SPASyA, might further improve the performance of the tool (in other experiments, see [8,9], the successful use of connection formulas has helped in improving performances of one order of magnitude). This is scope for future work.

10.3 Another attack report

As a further example of usage of our methodology, we show an analysis done on the Beller-Yacobi MSR protocol showing an attack firstly discovered by means of static analysis techniques in [11]. The protocol is an improved version of the Beller-Yacobi protocol designed to let a mobile base B exchange a session key with a control point A. When B enters a new cell asks A for its public key A^+ (not modelled here). The phase our analysis focus on starts with A sending B its identity together with its public key and a certificate issued by a shared trusted server S (not modelled here). Then, B generates the session key k and sends it encrypted with A^+. The informal narration of the protocol is:

$$\begin{array}{l}(1)\ A \rightarrow B : A, \{A\}_{S^-}, A^+ \\ (2)\ B \rightarrow A : \{k\}_{A^+} \\ (3)\ B \rightarrow A : \{B, \{B\}_{S^-}\}_k\end{array}$$

In message (1), the certificate issued by a trusted third party S confirms the identity of A to B which, in turn, (message (2)) replies with a session key k encrypted with A^+ (received in message (1)). In message (3) B sends the certificate of its own identity encrypted with session key k. The cIP specification of principals A and B is:

$$\begin{array}{rcl}A & \triangleq & (b, sa)[out(A, \{A\}_{sa}, A^+).in(\{?r\}_{A^-}).in(\{b, \{b\}_{sa}\}_r)] \\ B & \triangleq & (sb)[in(?a, \{a\}_{sb}, ?ak).out(\{k\}_{ak}).out(\{B, \{B\}_{sb}\}_k)]\end{array}$$

In our framework, interactions with the trusted server S need not to be explicitly modelled. Indeed, open variables sa and sb are used to let A and B share a key representing S.

The desired security property is represented by the $\mathcal{PL}$ formula ψ

$$\begin{array}{rcl}\psi & \triangleq & \forall i : B.((\kappa \rhd k_i \rightarrow a_i = I) \wedge \\ & & (\forall j : A.(b_j = B_i \rightarrow (r_j = k_i \wedge a_i = A_j \wedge \kappa \not\rhd k_i)))\ )\end{array}$$

that states that the intruder I should know the session key k of an instance B only if I initiated the protocol with B, while, if an instance A intends to interact with B then session key k should be shared between A and B without the intruder being able to acquire it. A connection formula is needed to restrict the analysis to the cases in which A and B share the same third party certificate. This is formalised by the $\mathcal{PL}$ formula ϕ

$$\begin{array}{rcl}\phi & \triangleq & \forall i : A.(\ (\exists j : B.b_i = B_j \wedge sa_i = sb_j) \vee \\ & & (\exists l : A.b_i = A_l \wedge sa_i = sb_l) \vee (b_i = I)\ )\end{array}$$

which states that the server used by any instance of A is either used by some other instance of regular principals A or B, or else by I.

Performing a search with $\mathcal{A}$SPASyA yields the aforementioned flaw, as presented in the following automatically generated table, where the behaviour of the intruder has been reported.

Violated Constraints:	**Open Variables:**
$b_2 = B_1$,	$b_2 \rightarrow B_1$
$a_1 \neq A_2$	$s_2 \rightarrow s_1 \rightarrow ks$
Knowledge:	**Model:**
$\{B_1, \{B_1\}_{ks}\}_{k_1}, \{k_1\}_{A_2^+}, \{A_2\}_{ks}, A_2, A_2^+,$	$a_1(\kappa) \rightarrow A_3$
$B_1, B_1^+, \{A_3\}_{ks}, A_3, I_0, I_0^+, I_0^-$	$ak_1(\kappa) \rightarrow A_2^+$
Intruder:	
$(1) A_2 \rightarrow I : A_2, \{A_2\}_{ks}, A_2^+$	
$(2) I \rightarrow B_1 : A_3, \{A_3\}_{ks}, A_2^+$	
$(3) B_1 \rightarrow I : \{k_1\}_{A_2^+}$	
$(4) B_1 \rightarrow I : \{B_1, \{B_1\}_{ks}\}_{k_1}$	
$(5) I \rightarrow A_2 : \{k_1\}_{A_2^+}$	
$(6) I \rightarrow A_2 : \{B_1, \{B_1\}_{ks}\}_{k_1}$	

As shown, the intruder is able to let B_1 "believe" he is running the protocol with A_3 (viz. $a_1(\kappa) \rightarrow A_3$) but authenticates himself to A_2 (viz. $b_2 \rightarrow B_1$) which falsifies the second conjunct in ϕ.

11 Concluding remarks

In this paper we have presented a framework where several aspects of security protocols and their verification are considered uniformly. The framework is mainly focused on problematic issues of specification and verification of protocols. In particular, a great attention has been reserved to the separation of concerns between the formal definition of protocols and the formal definition of properties. On the one hand, we propose the cIP calculus as a calculus of sessions, namely a calculus with explicit linguistic mechanisms for representing multi-sessions and sharing of secrets among principals; on the other hand, the specification of security properties is delegated to the companion logic $\mathcal{PL}$, that, on top of cIP principals, can express the security properties of interest. One of the specific merits of our framework lies exactly on the separation between cIP and $\mathcal{PL}$. This allows many parameters of the verification phase to be properly and *independently* tuned. For instance, once the protocol has been specified, it can be checked under different capabilities of the intruder by varying its initial knowledge or, also, open variables can be used to enforce assumptions

that at a first sight have been neglected. We remark that this is a distinguished feature of our framework, indeed other similar proposals (e.g., [14,62,38]) require a rather complex process for repeating verification on a protocol with different parameters and, often, this requires the protocol specification itself to be modified.

The $\mathcal{A}$SPASyA toolkit provides a model checker for $\mathcal{PL}$ formulas that are checked on cIP transition systems of protocol specifications, fully and properly supporting the verification framework proposed. Another important feature of $\mathcal{A}$SPASyA is the possibility of using connection formulas for experimenting with the protocol under specific assumptions on the secret sharing. This feature adds an extra distinguished parameter to the verification, strengthening the control over the dimension of the searched state space.

The presented framework is grounded on a theory that proves the decidability of the model checking algorithm implemented by $\mathcal{A}$SPASyA. The theory underlying cIP and $\mathcal{PL}$ exploits a symbolic semantics in order to overcome the infinite branching problem arising from the infinite set of possible messages that the Dolev-Yao intruder can build for principals. Possible extensions and relaxations of the intruder model have been discussed. We have shown that the messages derivation from a finite intruder knowledge is decidabible also in presence of asymmetric cryptography, and then that the session semantics is equivalent to a symbolic semantics that faithfully preserves the traces. Indeed, the defined symbolic semantics is a precise abstraction yielding a finite state space (when the number of session is bound). Finally, the symbolic model checking technique is proved sound and complete with respect to the *naive* model checking of $\mathcal{PL}$ formulas in the concrete case.

As future work we intend to adapt our methodology to other asymmetric cryptographic mechanisms (for instance, hashing functions), extend it with abstractions for time and timestamps, and embed non-atomic key treatment, as outlined in the paper. We conjecture that the theory (and hence the whole framework) fairly scales to other asymmetric crypto-system because we have not used specific features of public-key cryptography. Finally, we would like to extend the open variables and the constrained join mechanism, now based on connection formulas, to the more general case of open system verification, where open variables represent resources, and the join is constrained by formulas aimed at guaranteeing more general composition properties, as initially suggested in [19].

A Proofs for the framework

A.1 Proof of Proposition 2

Proof The proof consists of two parts:

1. *The function always terminates.* Either $e(\_)$ is recursively called on a set having a strictly smaller number of parenthesis and brackets, or it returns the set itself. Trivially, the number of parenthesis and brackets can not become negative. It follows that recursion terminates.
2. *The function is deterministic.* The function has a non-deterministic definition, since the conditions $m = (p, q) \in \kappa$ and $m = \{n\}_\lambda \in \kappa \ \wedge \ \lambda \in \kappa$ are not mutually exclusive. It must be shown that, despite the possible choices for the recursive calls, $e(\_)$ returns a unique value. It is necessary to prove that, given a knowledge κ

$$\exists\ \kappa_1, \kappa_2.\ \kappa_1 = e(\kappa)\ \wedge\ \kappa_2 = e(\kappa)\ \Rightarrow\ \kappa_1 = \kappa_2$$

Let us assume, by absurdum and without loss of generality, that $\exists\ m.\ m \in \kappa_1 \wedge m \notin \kappa_2$. The proof proceeds by induction on the definition of $e(\_)$. Assuming that recursive calls of $e(\_)$ over smaller knowledges are well defined, it is possible to show that $e(\_)$ is well defined. In the rest of the proof the observation that if $m \in \kappa$, then $m \in e(\kappa)$ will be used (in fact no rule of the definition of $e(\_)$ eliminates an item of κ). The proof is given by cases on the construction of $\kappa_1 = e(\kappa)$.

(a)

$$\kappa_1 = e(\kappa) = \kappa \tag{2}$$

The other rules can not applied and hence the only outcome of $e(\kappa)$ is κ itself.

(b)

$$\kappa_1 = e(\kappa) = e(\kappa \setminus o \cup p \cup q) \cup o \tag{3}$$

Analogously, κ_2 can be constructed from κ in two different ways.

i. $\kappa_2 = e(\kappa) = e(\kappa \setminus o' \cup p' \cup q') \cup o'$
Obviously $o \neq o'$, otherwise, applying the inductive hypothesis, $\kappa_1 = \kappa_2$. Since $m \in \kappa_1$, from (3), either $m = o$ or $m \in e(\kappa \setminus o \cup p \cup q)$.
In the first case, by the side conditions of rule (3), $m \in \kappa$, and hence $m \in \kappa \setminus o' \cup p' \cup q'$ and finally $m \in e(\kappa \setminus o' \cup p' \cup q') \subset \kappa_2$. Absurdum.
On the other hand, assuming $m \neq o \in e(\kappa \setminus o \cup p \cup q)$, from rule (3), it follows that $m \in e(\kappa) \setminus o$. Recalling that $m \neq o'$, it also follows that $m \in e(\kappa) \setminus o' \subset \kappa_2$. Absurdum.

ii. $\kappa_2 = e(\kappa) = e(\kappa \setminus \{n\}_\lambda \cup n) \cup \{n\}_\lambda$
Necessarily $m \neq \{n\}_\lambda$, otherwise it would belongs to κ_2. Let us suppose again that $m = o$. Then $m \in \kappa$, $m \in \kappa \setminus \{n\}_\lambda \cup n$, $m \in e(\kappa \setminus \{n\}_\lambda \cup n)$, and finally $m \in \kappa_2 \setminus \{n\}_\lambda \subset \kappa_2$. Absurdum.
On the other hand, assuming that $m \neq o \in e(\kappa \setminus o \cup p \cup q)$, from rule (3), it follows that $m \in e(\kappa) \setminus o$. Recalling that $m \neq \{n\}_\lambda$, it also follows that $m \in e(\kappa) \setminus \{n\}_\lambda \subset \kappa_2$. Absurdum.

(c)

$$\kappa_1 = e(\kappa) = e(\kappa \setminus \{n\}_\lambda \cup n) \cup \{n\}_\lambda \tag{4}$$

Again, the two possible cases of the construction of κ_2 must be considered.

i. $\kappa_2 = e(\kappa) = e(\kappa \setminus o \cup p \cup q) \cup o$
First, we observe that $m \neq o$, otherwise it would belong to κ_2. Since $m \in \kappa_1$, from (4), either $m = \{n\}_\lambda$ or $m \in e(\kappa \setminus \{n\}_\lambda \cup n)$.
In the first case, necessarily, $m \neq o$. By the side conditions of rule (4), $m \in \kappa$, and hence $m \in \kappa \setminus o \cup p \cup q$ and finally $m \in e(\kappa \setminus o \cup p \cup q) \subset \kappa_2$. Absurdum.
On the other hand, assuming $m \neq \{n\}_\lambda \in e(\kappa \setminus \{n\}_\lambda \cup n)$, from rule (4), it follows that $m \in e(\kappa) \setminus \{n\}_\lambda$. Recalling that $m \neq o$, it finally follows that $m \in e(\kappa) \setminus o \subset \kappa_2$. Absurdum.

ii. $\kappa_2 = e(\kappa) = e(\kappa \setminus \{n'\}_{\lambda'} \cup n') \cup \{n'\}_{\lambda'}$
In this last case, it is straightforward to observe that $\{n\}_\lambda \neq \{n'\}_{\lambda'}$, otherwise, by applying the inductive hypothesis, $\kappa_1 = \kappa_2$ holds. Moreover, by the definition of κ_2, necessarily $m \neq \{n'\}_{\lambda'}$
Let us suppose that $m = \{n\}_\lambda$, then, by the side conditions of rule (4), it holds that $m \in \kappa$, and $m \in e(\kappa \setminus \{n'\}_{\lambda'} \cup n') \subset \kappa_2$. Absurdum.
On the other hand, if $m \neq \{n\}_\lambda \in e(\kappa \setminus \{n\}_\lambda \cup n)$, by rule (4), $m \in e(\kappa) \setminus \{n\}_\lambda$. Recalling that $m \neq \{n'\}_{\lambda'}$, it finally follows that $m \in e(\kappa) \setminus \{n'\}_{\lambda'} \subset \kappa_2$. Absurdum.

□

A.2 Proof of Theorem 1

The proof is based on the following lemmas.

Lemma 1 *Let $m \in M$ be a message, and κ a knowledge, then $m \in \kappa \Rightarrow m \in e(\kappa)$.*

Proof By definition of $e(\_)$ (no element is never deleted). □

Lemma 2 *Let $m \in M$ be a message, and κ a knowledge, then $m \in e(\kappa) \Rightarrow \kappa \rhd m$.*

Proof (by structural induction on e). The set $e(\kappa)$ can have been obtained by applying one of the three rules of the definition of the function $e(\_)$.

$e(\kappa) = \kappa$
In this (base) case, $m \in \kappa$ and hence, by applying rule $\in$, $\kappa \rhd m$.
$e(\kappa) = e(\kappa \setminus o \cup p \cup q) \cup o$
The side conditions for this case imply that $o = (p, q) \in \kappa$. Either $m = o \in \kappa$, but then $\kappa \rhd m$ (by rule $\in$), or $m \in e(\kappa \setminus o \cup p \cup q)$. By the inductive hypothesis, it follows that $\kappa \setminus o \cup p \cup q \rhd m$, and equivalently $\kappa \cup p \cup q \rhd m$. Since $o = (p, q) \in \kappa$, by applying $()_{e1}$ and $()_{e2}$, it holds that $\kappa \rhd p$ and $\kappa \rhd q$. Trivially (reasoning on the proofs):

$$\kappa \cup p \cup q \rhd m \ \wedge \ \kappa \rhd p \ \wedge \ \kappa \rhd q \Rightarrow \kappa \rhd m$$

$e(\kappa) = e(\kappa \setminus o \cup n) \cup o$
Analogously, $o = \{n\}_\lambda \in \kappa$ and $\kappa \rhd_i \lambda$ (and, obviously, $\kappa \rhd \lambda$). If $m = o \in \kappa$, then $\kappa \rhd m$. Otherwise, $m \in e(\kappa \setminus o \cup n)$, and then, by inductive hypothesis, $\kappa \cup n \rhd m$. Since $o \in \kappa$, $\kappa \rhd n$ by $\{\}_e$. Trivially:

$$\kappa \cup n \rhd m \ \wedge \ \kappa \rhd n \Rightarrow \kappa \rhd m$$

□

The proof for the **Theorem 1** follows.

Proof $\kappa \rhd \mathbf{m} \Rightarrow \mathbf{e}(\kappa) \rhd_\mathbf{i} \mathbf{m}$: by induction on i, the length of the proof for $\kappa \rhd m$).
Base case: the only proof of length one is

$$\frac{m \in \kappa}{\kappa \rhd m} \ \in,$$

and then, by Lemma 1, $m \in e(\kappa)$, and hence $e(\kappa) \rhd_i m$.
Inductive step: message m is derived by a proof which can terminate by applying one of the remaining five rules.

If $m = (p, q)$ is obtained by applying the rule

$$\frac{\kappa \rhd p \quad \kappa \rhd q}{\kappa \rhd (p, q)} \ ()_i,$$

then, by inductive hypothesis $e(\kappa) \rhd_i p$ and $e(\kappa) \rhd_i q$, and hence

$$\frac{e(\kappa) \rhd_i p \quad e(\kappa) \rhd_i q}{e(\kappa) \rhd_i (p, q)} \ ()_i.$$

If $m = \{n\}_\lambda$ is obtained by applying the rule

$$\frac{\kappa \rhd n \quad \kappa \rhd \lambda}{\kappa \rhd \{n\}\lambda} \ \{\}_i,$$

then, by inductive hypothesis $e(\kappa) \rhd_i n$ and $e(\kappa) \rhd_i \lambda$, and hence

$$\frac{e(\kappa) \rhd_i n \quad e(\kappa) \rhd_i \lambda}{e(\kappa) \rhd_i \{n\}_\lambda} \ \{\}_i.$$

If $m = p$ is obtained by applying the rule

$$\frac{\kappa \rhd (p, q)}{\kappa \rhd p} \ ()_{e1},$$

then, by inductive hypothesis $e(\kappa) \rhd_i (p, q)$. The proof for this last judgment can be concluded in two ways. A case occurs when applying the rule

$$\frac{e(\kappa) \rhd_i p \quad e(\kappa) \rhd_i q}{e(\kappa) \rhd_i (p, q)} \ ()_i,$$

which implies that $e(\kappa) \rhd_i p$. The other case occurs when the rule

$$\frac{(p, q) \in e(\kappa)}{e(\kappa) \rhd_i (p, q)} \ \in$$

is applied. If $(p, q) \in e(\kappa)$, from the definition of $e(\_)$, it follows that also $p \in e(\kappa)$ (if not the recursive construction of $e(\_)$ could not have terminated). It follows that $e(\kappa) \rhd_i p$.
The case for $()_{e2}$ is symmetric to the previous one.
The last case is analogous to the previous two. If $m = n$ is obtained by applying the rule

$$\frac{\kappa \rhd \{n\}_\lambda \quad \kappa \rhd \lambda^-}{\kappa \rhd n} \ \{\}_e,$$

then, by inductive hypothesis $e(\kappa) \rhd_i \{n\}_\lambda$ and $e(\kappa) \rhd_i \lambda$. The proof for $e(\kappa) \rhd_i \{n\}_\lambda$ can be concluded in two ways. A case occurs when applying the rule

$$\frac{e(\kappa) \rhd_i n \quad e(\kappa) \rhd_i \lambda}{e(\kappa) \rhd_i \{n\}_\lambda} \ \{\}_i,$$

which implies that $e(\kappa) \rhd_i n$. The other case occurs when the rule

$$\frac{\{n\}_\lambda \in e(\kappa)}{e(\kappa) \rhd_i \{n\}_\lambda} \ \in$$

is applied. If $\{n\}_\lambda \in e(\kappa)$, from the definition of $e(\_)$, it follows that also $n \in e(\kappa)$ (if not the recursive construction of $e(\_)$ could not have terminated). It follows that $e(\kappa) \rhd_i n$.

$\mathbf{e}(\kappa) \rhd_\mathbf{i} \mathbf{m} \Rightarrow \kappa \rhd \mathbf{m}$: by induction on i, the length of the proof for $e(\kappa) \rhd_i m$).
Base case: the only proof of length one is

$$\frac{m \in e(\kappa)}{e(\kappa) \rhd_i m} \ \in,$$

and then, by Lemma 2, $\kappa \rhd m$.
Inductive step: two rules can be applied to derive $e(\kappa) \rhd_i m$ in more than one step, namely $()_i$ and $\{\}_i$. In the first case $m = (p, q)$, and

$$\frac{\kappa \rhd_i p \quad \kappa \rhd_i q}{\kappa \rhd_i (p,q)}\ ()_i,$$

then, by inductive hypothesis $\kappa \rhd p$ and $\kappa \rhd q$, and hence

$$\frac{\kappa \rhd p \quad \kappa \rhd q}{\kappa \rhd (p,q)}\ ()_i.$$

Analogously, if $m = \{n\}_\lambda$ and $\{\}_i$ is used, then

$$\frac{e(\kappa) \rhd_i n \quad e(\kappa) \rhd_i \lambda}{e(\kappa) \rhd_i \{n\}\lambda}\ \{\}_i.$$

By inductive hypothesis $\kappa \rhd n$ and $\kappa \rhd \lambda$, and hence

$$\frac{\kappa \rhd n \quad \kappa \rhd \lambda}{\kappa \rhd \{n\}_\lambda}\ ()_i.$$

□

B Effectiveness of symbolic constructions

The technical proofs of Sections 7 and 8 are collected in this appendix.

B.1 Proof of Theorems 4 and 5

Theorem 4 relies on Proposition 4 which on turn relies on the following two auxiliary lemmas.

Lemma 3 *Let $\kappa \in \wp(\underline{M})$ and σ a symbolic substitution such that κ covers messages in $\mathrm{dom}(\sigma) \cup \mathrm{cod}(\sigma)$. If $\kappa\sigma \trianglerighteq m$ then $\kappa \trianglerighteq m$.*

Proof By hypotheses any $x[\kappa_1]$ is replaced in κ with a message m' that can be generated by κ. Indeed, it must be the case that $\kappa_1 \trianglerighteq m'$, hence, $\kappa \trianglerighteq m'$ because κ covers κ_1. Hence, $\kappa\sigma$ cannot built m using sub-messages that cannot be built by κ. □

Lemma 4 *Let $\kappa \in \wp(\underline{M})$ and d a datum such that κ covers d. For all $(m, \sigma) \in \mu(d, \kappa)$, κ covers m and κ covers any message in $\mathrm{dom}(\sigma) \cup \mathrm{cod}(\sigma)$.*

Proof By inspecting Figure 4 and Table 5 it is easy to see that all symbolic variables introduced while computing μ are obtained by κ or by a set κ_1 already occurring in d. By hypothesis, κ_1 and d are covered by κ. □

Proposition 5 (Proposition 4) *Let d be a datum and $\kappa \in \wp(\underline{M})$ covers d. If $(m, \sigma) \in \mu(d, \kappa)$ then $d\sigma \simeq m\sigma$ and $\kappa \trianglerighteq m$.*

Proof The proof proceeds by induction on the structure of d.

By definition of substitution and $\trianglerighteq$, the proposition holds in the first three cases (the base cases).

The computation of $\mu((e, f), \kappa)$ proceeds by first computing $\mu(e, \kappa)$ and then $\mu(f\sigma_e, \kappa\sigma_e)$, where σ_e is a substitution resulting from $\mu(e, \kappa)$ (recall that binding variables have at most one occurrence and bind all the remaining occurrences). If $\mu(e, \kappa) = \emptyset$ then a matching message for d cannot be derived, while, if $(e', \sigma_e) \in \mu(e, \kappa)$ then $e'\sigma_e \simeq e\sigma_e \wedge \kappa \trianglerighteq e'$ (by inductive hypothesis). With a similar searsoning we deduce $f'\sigma_f \simeq f\sigma_e\sigma_f \wedge \kappa\sigma_e \trianglerighteq f'$ (or $\mu(f\sigma_e, \kappa\sigma_e) = \emptyset$). Observing that $\kappa\sigma_e\sigma_f \trianglerighteq (e', f')$ and κ covers $\sigma_e\sigma_f$, by Lemma 3 and by Lemma 4

$$d\sigma_e\sigma_f \simeq ((e\sigma_e\sigma_f), (f\sigma_e\sigma_f)) \simeq$$

$$\simeq (((e'\sigma_e)\sigma_f), ((f'\sigma_e)\sigma_f)) \simeq (e', f')(\sigma_e\sigma_f),$$

Let d be $\{e\}_{\lambda^-}$; we distinguish two cases.

1. If $\kappa \trianglerighteq \lambda$, then the matching substitution for e is returned and the induction hypothesis guarantees the proposition.
2. However, cryptograms $\{e'\}_\lambda \in \kappa$ must be considered. Indeed, ν returns (if exist) the symbolic substitutions such that e' matches e. Basically, ν works as μ but avoids manipulating its second argument, hence, by induction, for any $\sigma_e \in \nu(e, e')$, $e\sigma_e \simeq e'\sigma_e$.

This concludes the proof. □

Finally, we can prove Theorems 4 and 5.

Theorem 9 (Theorem 4) *If $\underline{T}$ is a symbolic trace, for all ρ concretising substitutions for $\underline{T}$, $\underline{T}\rho$ is a trace in the concrete semantics.*

Proof Assuming $\underline{T} = \Sigma_0.\alpha_1.\ldots.\alpha_n.\Sigma_n$, the proof easily follows by induction on n.

If $n = 0$ then Σ_0 is a concrete state by definition and $\Sigma_0\rho = \Sigma_0$, for any concretising substitution ρ.

Assume that the theorem holds on traces $\underline{T}$ of length n, and consider $\underline{T}\alpha_{n+1}\Sigma_{n+1}$ where $\Sigma_{n+1} = \langle \mathcal{C}', \chi', \kappa_1 \rangle$. By construction, χ' either extends χ with substitutions of new symbolic variable or refines variable already in χ. Hence, any concretising substitution for χ' is a concretising substitution for χ because of Remark 1 and covering is preserved by refinement. We proceed by case analysis on α_{n+1} (hereafter, $\langle \mathcal{C}, \chi, \kappa \rangle$ represents the last state of $\underline{T}$).

If $\alpha_{n+1} = j\ A_i\ \gamma$ then $\Sigma_{n+1}\rho = \langle join(A_i, \gamma, \mathcal{C}\rho), \chi\rho\gamma, \kappa\rho\gamma \rangle$, since γ does not introduce symbolic variables, $\gamma\rho = \rho\gamma$. Clearly, $\Sigma_n\rho \mapsto \Sigma_{n+1}\rho$ can be obtained by applying the $(join)$ rule.

If $\alpha_{n+1} = o\ m\sigma$, a principal in $\mathcal{C}$ performs an action $in(d)$ and σ is valid substitution. Without loss of generality (for Proposition 4), there i $(m, \sigma') \in \mu(d, \kappa)$ such that σ is obtained by concretising some symbolic variable in σ' to keys. This implies that $m\sigma\rho$ is deducible from $\kappa\rho$ and matches $d\rho$. Finally, $\mathcal{C}\rho$ contains the concretisation of the principal performing the $in(d)$ action, therefore, we can apply rule (in).

(The case $\alpha_{n+1} = i\ m$ is dealt as done for $\alpha_{n+1} = o\ m\sigma$). □

Theorem 10 (Theorem 5) *If $T = \langle \emptyset, \varepsilon, \kappa_0 \rangle \mapsto \ldots \mapsto \langle \mathcal{C}_n, \chi_n, \kappa_n \rangle$ is a concrete trace then there are*

1. *a symbolic trace $\underline{T} \searrow \langle \mathcal{C}', \chi', \kappa_1 \rangle$ from $\langle \emptyset, \varepsilon, \kappa_0 \rangle$*
2. *and a concretising substitution ρ for χ'*

such that $\underline{T}\rho = T$.

Proof The proof follows the same pattern of the proof of Theorem 4. The cases of rules $(join)$ and (out) are straightforward because they essentially restrict the possible concretising substitutions along the trace.

The interesting cases are the transitions obtained by applying rule (in) and we give a proof by contraddiction. Let assume that it is not possible to mimic a concrete input action where a message matching the input datum is derived from the intruder knowledge. By inductive hypothesis, such knowledge can be obtained by concretising a corresponding symbolic one. Hence, such concretisation would also generate the message chosen in the transition by Theorem 3. □

B.2 Proof of Theorem 8

Theorem 11 (Decidability of $\models$) *Let ϕ be a $\mathcal{PL}$-formula, $\kappa \in \wp_{fin}(\underline{M})$ and χ a symbolic substitution.*

$$\kappa \models_\chi \phi \iff \Psi_{\kappa,\chi}(\phi) \neq \bot$$

Proof By Proposition 6, we can assume that

$$\phi \equiv \bigvee_{i \in 1,\ldots,u} (\psi_{i,1} \wedge \ldots \wedge \psi_{i,j_i})$$

where the $\psi_{i,j}$ are atoms.

($\Leftarrow$) Let $\sigma \in \Psi_{\kappa,\chi}(\phi)$ and assume, *per absurdum*, that a concretising substitution of ρ of χ such that $\kappa\sigma\rho \models_{\chi\sigma\rho} \phi$ does not exist. Namely, for any ρ concretising substitution of χ and for any $i \in \{1,\ldots,n\}$, there is $h \in \{1,\ldots,j_i\}$ such that $\Psi_{i,h}$ does not hold in $\langle \kappa\sigma\rho, \chi\sigma\rho\rangle$. By case analysis on the form of $\phi_{i,h}$:
- if $\phi_{i,h}$ is $x[\kappa_1] = y[\kappa_2]$, a message m such that $\kappa_1 \rhd m \wedge \kappa_2 \rhd m$ does not exists; this contraddicts the hypothesis $\kappa_1 \sqcap \kappa_2 \neq \emptyset$ required for $\phi_{\kappa,\chi}(\phi) \neq \bot$.
- If $\phi_{i,h}$ is $x[\kappa'] = m$, then $\kappa' \not\rhd m$ (otherwise $\phi_{i,h}$ would be satisfiable in $\langle \kappa, \chi\rangle$); hence, $\mu(m^{-1}, \kappa') = \emptyset$ and then $\phi_{i,h} = \bot$ which contraddicts the hypothesis.
- If $\phi_{i,h}$ is $\kappa \rhd m$, then the thesis follows reasoning as in the previous case.

($\Rightarrow$) Let ρ such that $\kappa\rho \models_{\chi\rho} \phi$ and, by contriddiction, assume $\Psi_{\kappa,\chi}(\phi) = \bot$. Then, for $i \in \{1,\ldots,n\}$, ψ_{i,j_i} are all satisfied in $\langle \kappa\rho, \chi\rho\rangle$, but there is $h \in \{1,\ldots,j_i\}$ such that $\Psi_{\kappa,\chi}(\psi_{i,h}) = \bot$. As before, we proceed by case analysis on $\phi_{i,h}$.
- if $\phi_{i,h}$ is $x[\kappa_1] = y[\kappa_2]$ then ρ maps both $x[\kappa_1]$ and $y[\kappa_2]$ to a message m, i.e., $(x[\kappa_1])\rho = (y[\kappa_2])\rho = m$. By Definition 19, $\kappa_1 \rhd m \wedge \kappa_2 \rhd m$ which contraddicts $\kappa_1 \sqcap \kappa_2 = \emptyset$ necessary for $\phi_{\kappa,\chi}(\phi) \neq \bot$.
- If $\phi_{i,h}$ is $x[\kappa'] = m$, similarly to the previous case $\kappa' \rhd m\rho$, but it must be $\mu(m^{-1}, \kappa') = \emptyset$ which by Proposition 4 implies $\kappa' \not\rhd m$ which yields a contraddiction.
- If $\phi_{i,h}$ is $\kappa \rhd m$, the thesis follows reasoning as in the previous case.

□

B.3 Normal forms

Proposition 6 *For any $\mathcal{PL}$ formula ϕ, any $\kappa \in \wp_{fin}(M)$ and substitution any χ, there exists a normal form ψ such that, $\kappa \models_\chi \phi \iff \kappa \models_\chi \psi$.*

Proof By induction on the structure of ϕ. □

By Proposition 6, model checking $\mathcal{PL}$ formulas can be reduced to model check formulas in normal form. A further simplification is to consider only atomic conjuncts because it suffices to find a model for one of the disjunct to obtain a model for the whole formula. First, positive atoms can be exploited for determining a substitution (if any) that "refines" the symbolic variables, then negative atoms are used to establish inequalities that must hold in the models of the formula.

Acknowledgements The authors wish to thank the anonymous referees for their helpful suggestions.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL '01. Proceedings of the 28th ACM SIGPLAN-SIGACT on Principles of programming languages, 2001*, ACM SIGPLAN Notices, 2001. ACM Press.
2. M. Abadi and A. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999.
3. R. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In C. Palamidessi, editor, *International Conference in Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 380–394. Springer-Verlag, 2000.
4. R. Amadio, D. Lugiez, and V. Vanackère. On the symbolic reduction of processes with cryptographic functions. *Theoretical Computer Science*, 290(1):695–740, 2003.
5. R. Amadio and S. Prasad. The game of the name in cryptographic tables. In P. S. Thiagarajan and R. Yap, editors, *Advances in Computing Science - ASIAN'99*, volume 1742 of *Lecture Notes in Computer Science*, pages 15–26. Springer-Verlag, 1999.
6. N. Asokan. *Fairness in Electronic Commerce*. PhD thesis, University of Waterloo, 1998.
7. G. Baldi. *Security Protocols Verification by Means of Symbolic Model Checking* Ms.Thesis, University of Pisa, Available at [8].
8. G. Baldi, A. Bracciali, G. Ferrari, and E. Tuosto. ASPASyA: Automated tool for Security Protocols Analysis based on a Symbolic Approach. Available at `http://www.cs.le.ac.uk/people/et52/aspasya/aspasya.html`.
9. G. Baldi, A. Bracciali, G. Ferrari, and E. Tuosto. A Coordination-based Methodology for Security Protocol Verification. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *International Workshop on Security Issues with Petri Nets and other Computational Models*, volume 121 of *Electronic Notes in Theoretical Computer Science*, pages 23–46, 2005. Elsevier.
10. D. Basin, S. Mödersheim, and L. Viganò. Constraint differentiation: a new reduction technique for constraint-based analysis of security protocols. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 335–344, 2003. ACM Press.
11. C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. Riis Nielson. Control Flow Analysis Can Find New Flaws Too. In *Workshop on Issues on the Theory of Security (WITS'04)*, *Electronic Notes in Theoretical Computer Science*, Elsevier, 2004.
12. C. Bodei, P. Degano, F. Nielson, and H. Nielson. Static analysis for the π-calculus with applications to security. *Information and Computation*, 168: 68-92, 2001.
13. M. Boreale. Symbolic trace analysis of cryptographic protocols. In F. Orejas, P. Spirakis, and J. van Leeuwen, editors, *Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
14. M. Boreale and M. Buscemi. A Framework for the Analysis of Security Protocols. In L. Brim, P. Jančar, M. Křetinský, and A. Kučera, editors, *International Conference in Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 483–498. Springer-Verlag, 2002.
15. M. Boreale and M. Buscemi. A method for symbolic analysis of security protocols. *Theoretical Computer Science*, 338(1-3):393–425, 2005.
16. M. Boreale and R. De Nicola. A Symbolic Semantics for the π-calculus. *Information and Computation*, 126(1):34–52, 1996.

17. J. Borgström, S. Briais, and U. Nestmann. Symbolic Bisimulation in the Spi Calculus. In P. Gardner and N. Yoshida, editors, *International Conference in Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 161–176. Springer-Verlag, 2004.
18. A. Bracciali. *Behavioural Patterns and Software Composition*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2003.
19. A. Bracciali, A. Brogi, G. Ferrari, and E. Tuosto. Security and Dynamic Compositions of Open Systems. In H. Arabnia, editor, *Conference on Parallel and Distributed Processing Techniques and Applications*, volume 3, pages 1372–1377, 2002. CSREA Press.
20. M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.
21. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. Deciding the security of protocols with Diffie-Hellman exponentiation and products in exponents. volume 2914 of *Lecture Notes in Computer Science*, pages 124–135. Springer-Verlag, 2003.
22. Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP decision procedure for protocol insecurity with XOR. In *Annual Symposium on Logic in Computer Science*, pages 261–270. IEEE Computer Society, 2003.
23. J. Clark and J. Jacob. A survey of authentication protocols 1.0. Technical report, University of York, 1997.
24. E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5):1512–1542, 1994.
25. E. Clarke, S. Jha, and W. Marrero. Using State Space Exploration and a Natural Deduction Style Message Derivation Engine to Verify Security Protocols. In *IFIP Working Conference on Programming Concepts and Methods (PROCOMET)*, 1998.
26. H. Comon, V. Cortier, and J. Mitchell. Tree automata with one memory, set constraints, and ping-pong protocols. In F. Orejas, P. Spirakis, and J. van Leeuwen, editors, *Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 682–693. Springer-Verlag, 2001.
27. F. Crazzolara. *Language, Semantics, and Methods for Security Protocols*. PhD thesis, BRICS, May 2003.
28. F. Crazzolara and G. Winskel. Events in security protocols. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 96–105, 2001. ACM Press.
29. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.
30. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Workshop on Formal Methods and Security Protocols, Part of the Federated Logic Conference*, 1999.
31. J. Fabrega, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? In *RSP: 19th IEEE Computer Society Symposium on Research in Security and Privacy*, 1998.
32. J. Fabrega, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2,3):181–230, January 1999.
33. R. Focardi and R. Gorrieri. A Classification of Security Properties. *Journal of Computer Security*, 3(1), 1995.
34. R. Focardi and R. Gorrieri. The Compositional Security Checker: A tool for the verification of information flow security properties. *IEEE Computer Society*, 23(9):550–571, 1997.
35. A. Freier, P. Karlton and P. Kocher. The SSL protocol version 3.0, 1996. Available at `http://home.netscape.com/eng/ssl3`.
36. A. D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *J. Comput. Secur.*, 11(4):451–519, 2004.
37. M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995.
38. A. Huima. Efficient finite-state analysis of security protocols. In *Formal methods and security protocols*, FLOC Workshop, 1999. INRIA.
39. A. Kehne, J. Schönwälder, and H. Langendörfer. Multiple authentications with a nonce-based protocol using generalized timestamps. In *Proc. ICCC '92*, 1992.
40. J. Kohl and B. Neuman. The kerberos network authentication service (version 5). Internet Request for Comment RFC-1510, 1993.
41. S. Kremer, O. Markowitch, and J. Zhou. An intensive survey of non-repudiation protocols. *Computer Communications*, 25(17):1606–1621, 2002.
42. 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, pages 147–166. Springer-Verlag, 1996.
43. G. Lowe. Some New Attacks upon Security Protocols. In *Proceedings of 9th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 1996.
44. G. Lowe. A hierarchy of authentication specifications. In *Computer Security Foundation Workshop*. IEEE Computer Society, 1997.
45. F. Martinelli. Analysis of security protocols as open systems. *Theoretical Computer Science*, 209(1):1057–1106, 2003.
46. A. Menzies, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
47. J. K. Millen. A necessarily parallel attack. In N. Heintze and E. Clarke, editors, *Workshop on Formal Methods and Security Protocols, Part of the Federated Logic Conference*, 1999.
48. J.K. Millen. On the freedom of decryption. *Information Processing Letters*, 86:329-333, 2003.
49. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 166–175, 2001.
50. J. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Computer Security Foundation Workshop*, pages 141–151. IEEE Computer Society, 1997.
51. J. Mitchell, V. Shmatikov, and U. Stern. Finite-state analysis of SSL 3.0. In *Proceedings of the 7th USENIX Security Symposium (SECURITY-98)*, pages 201–216. Usenix Association.
52. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, 1978.
53. L. Paulson. Proving properties of security protocols by induction. In *Computer Security Foundation Workshop*. IEEE Computer Society, 1997.
54. L. Paulson. *The inductive approach to verifying cryptographic protocols*. Technical report; no. 443. 4006797499. University of Cambridge, Computer Laboratory, 1998.
55. V. Shmatikov. Decidable analysis of cryptographic protocols with products and modular exponentiation. volume 2986 of *Lecture Notes in Computer Science*, pages 355–369. Springer-Verlag, 2004.
56. V. Shmatikov and J. Mitchell. Analysis of a fair exchange protocol. In *Symposium on Network and Distributed Systems Security (NDSS 2000)*, pages 119–128, 2000. Internet Society.
57. V. Shmatikov and J. Mitchell. Finite-state analysis of two contract signing protocols. *Theoretical Computer Science, special issue on Theoretical Foundations of Security Analysis and Design*, 283(2):419–450, 2002.

58. D. Song, S. Berezin, and A. Perrig. Athena, a novel approach to efficient automatic security protocol analysis. *Computer Security*, 9(1,2):47–74, 2001.
59. D. Stinson. *Cryptography: Theory and practice*. CRC Press, 1995.
60. J. Thayer, J. Herzog, and J. Guttman. Honest ideals on strand spaces. In *Computer Security Foundation Workshop*. IEEE Computer Society, 1998.
61. E. Tuosto. *Non-Functional Aspects of Wide Area Network Programming*. PhD thesis, Dipartimento di Informatica, Università di Pisa, 2003.
62. V. Vanackére. The TRUST protocol analyser. Automatic and Efficient Verification of Cryptographic Protocols. In *VERIFY02*, 2002.
63. T. Woo and S. Lam. A semantic model for authentication protocols. In *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.
64. J. Zhou. *Non-repudiation*. PhD thesis, University of London, 1996.
65. R. Zunino and P. Degano. Weakening the perfect encryption assumption in Dolev-Yao adversaries. *Theoretical Computer Science*, 340(1):154–178, 2005.