Relating Strand Spaces and Distributed Temporal Logic for Security Protocol Analysis

Carlos Caleiro, *CLC, Department of Mathematics, IST, Lisbon, Portugal. Email: ccal@math.ist.utl.pt*

Luca Viganò, *Department of Computer Science, ETH Zurich, Switzerland. Email: vigano@inf.ethz.ch*

David Basin, *Department of Computer Science, ETH Zurich, Switzerland. Email: basin@inf.ethz.ch*

Abstract

In previous work, we introduced a version of distributed temporal logic that is well-suited both for verifying security protocols and as a metalogic for reasoning about, and relating, different security protocol models. In this paper, we formally investigate the relationship between our approach and strand spaces, which is one of the most successful and widespread formalisms for analyzing security protocols. We define translations between models in our logic and strand-space models of security protocols, and we compare the results obtained with respect to the level of abstraction that is inherent in each of the formalisms. This allows us to clarify different aspects of strand spaces that are often left implicit, as well as pave the way to transfer results, techniques and tools across the two approaches.[1]

Keywords: Security protocols, protocol models, strand spaces, distributed temporal logic.

1 Introduction

Many security protocols have been proposed to help build secure distributed systems and many of them have later turned out to have subtle flaws. Due to the severity of this problem, a wide variety of different formalisms and associated tools have been proposed for, and applied to, the analysis of security protocols. These include *process algebras* [5, 18, 28, 33], *model-checking* and related techniques [1, 2, 12, 31, 34], special-purpose *epistemic logics* [11], and *inductive theorem proving* in higher-order logic [32].

This explosion in formalisms is natural in a young research area as researchers explore the problem domain in its many facets. But as the area matures, it becomes increasingly important to consolidate knowledge by clarifying the precise relationships between the formalisms, thereby developing a deeper understanding of them and their strengths and weaknesses. This consolidation phase is now underway, e.g. [3, 4, 10,

[1]This work was partially supported by FCT and EU FEDER via the Projects POCTI/2001/MAT/37239 FibLog and POCI/MAT/55796/2004 QuantLog of CLC, and by the FET Open Project IST-2001-39252 and the BBW Project 02.0431, "AVISPA: Automated Validation of Internet Security Protocols and Applications".

11, 15, 16, 20, 21, 25, 29].

One of the most successful and widespread formalisms for analyzing security protocols is that of *strand spaces* [24, 34, 36]. Given the naturalness of this formalism, it provides a good focal point for a consolidation effort, and a number of authors, e.g. [10, 15, 16, 25], have formalized the relationship of their protocol analysis approaches with strand spaces. In this paper, we contribute to this program by relating strand spaces to a distributed temporal logic for security protocol analysis.

In [7, 8], we introduced the *Distributed Temporal Protocol Logic* DTPL, a version of distributed temporal logic [19] that provides a language for formalizing both local and global properties of distributed communicating processes. We showed that DTPL can be effectively applied to security protocol analysis, for example formalizing security properties and reasoning about interleaved protocol executions. In addition, DTPL is well-suited as a metalogic. Its metalogical applications include rigorously formalizing and proving metalevel properties of different protocol models, establishing relationships between models, and showing the correctness of different modeling and simplification techniques used in building effective protocol-analysis tools.

We investigate here the relationship between our approach and strand spaces (which transitively yields a comparison with the approaches that have been related to strand spaces, mentioned above). Our main technical result is to define property-preserving translations between DTPL models and strand-space models. This is nontrivial as, despite the similarities between the two formalisms (for example, both are based on partially-ordered sets of events with labeling information), there are substantial differences. The differences concern the way the principals and the intruder executing a protocol are represented, the way communication is formalized, and the locality of information.

We show how to overcome these differences, defining a suitable equivalence between the two kinds of models, and formalizing back-and-forth translations that map strand-space models to equivalent DTPL models, and vice-versa. Both translations preserve the essential ingredients of protocol execution and thereby also security properties like secrecy and authentication. The translations also illuminate different implicit aspects of strand spaces and pave the way to transfer results, techniques and tools across the two formalisms that we consider in this paper, thereby gaining the advantages of both.

We proceed as follows. In Section 2, we summarize the main features of DTPL and strand spaces. Then, in Section 3, we investigate the relationship between the two approaches. We conclude in Section 4, discussing related and future work.

2 The two formalisms

We begin in this section by introducing a running example that allows us to illustrate the common features of DTPL and strand spaces. Afterwards we will summarize the characteristic features of each formalism.

2.1 Common features

Security protocols are typically described informally by short sequences of messages that are exchanged by principals in order to achieve particular security goals in open,

$$\begin{array}{llcll}
(\mathrm{step}_1) & a \rightarrow b & : & (n_1). & \{n_1; a\}_{K_b} \\
(\mathrm{step}_2) & b \rightarrow a & : & (n_2). & \{n_1; n_2\}_{K_a} \\
(\mathrm{step}_3) & a \rightarrow b & : & & \{n_2\}_{K_b}
\end{array}$$

FIG. 1. The (authentication part of the) Needham-Schroeder Public Key Protocol.

hostile environments. A popular example is the (authentication part of the) Needham-Schroeder Public Key Protocol NSPK [26], presented in Fig. 1. In this notation, a and b are variables of sort name, denoting the roles played in one execution of the protocol, and n_1 and n_2 are variables of sort nonce. The arrows represent communication from the sender to the receiver. The parenthesized nonces prefixing the first two messages signify that these nonces must be freshly generated before the message is sent. Moreover, it is assumed that public and private keys have been generated and appropriately distributed: K_a represents the public key of a, whose inverse key should be privately held by a.

As we remarked above, a large number of formal methods and tools have been developed to prove security protocols correct or to identify attacks on them. For instance, the NSPK protocol does not provide the mutual authentication property it was designed for, but rather suffers from a man-in-the-middle attack [26]. While the various protocol analysis methods are often based on different approaches and formalisms, they all share a common starting point, namely the description of the correct sequences of message exchanges that should be performed by each of the principals running the protocol.

Both DTPL and strand-space protocol models are based on partially-ordered sets of events with labeling information. We thus employ sequences (e.g. sequences of events or sequences of labels) to represent protocol executions and, as notation, we use $w = \langle w_1.w_2.w_3 \ldots \rangle$ to denote a (possibly infinite) sequence composed of the elements $w_1, w_2, w_3, \ldots$, and we write $|w|$ to denote its length. So, $\langle\rangle$ denotes the empty sequence and $|\langle\rangle| = 0$. We assume that $|w| = \infty$, if w is infinite. We write $w \centerdot w'$ to denote sequence concatenation, provided that the first sequence is finite.

Another key ingredient in both approaches is the flow of information between principals. We assume as given a context consisting of a finite set *Princ* of principal identifiers $A, B, C, \ldots$, and an indexed family $\mathit{Name} = \{\mathit{Name}_A\}_{A \in \mathit{Princ}}$ of pairwise disjoint finite sets of *names*, corresponding to the possible aliases used by each principal. We use primed notation to denote names, e.g. writing A' to denote a name used by principal A. By abuse of notation, we also use $\mathit{Name} = \bigcup_{A \in \mathit{Princ}} \mathit{Name}_A$. We also assume fixed two sets *Nonce* and *Key* of "numbers" that can be used as *nonces* and *keys*, respectively, and whose members we denote by N and K. We assume that several kinds of keys can coexist and that each key K has an inverse key K^{-1}. *Messages*, which we denote by M, are built inductively from *atomic messages* (names and numbers) using concatenation $\_;\_$ and encryption $\{\_\}_K$ under a key K. The set *Msg* of messages is thus defined by

$$\mathit{Msg} ::= \mathit{Name} \mid \mathit{Nonce} \mid \mathit{Key} \mid \mathit{Msg}; \mathit{Msg} \mid \{\mathit{Msg}\}_{\mathit{Key}} \,.$$

Note that we employ a sorted signature, with a sort for messages and subsorts for names, nonces, and keys.

2.2 The distributed temporal protocol logic DTPL

2.2.1 The logic.

The distributed temporal logic DTL [19] is a logic for reasoning about temporal properties of distributed systems from the local point of view of the system agents, which are assumed to execute sequentially and to interact by means of synchronous event sharing. Distribution is implicit, making it easier to state the properties of an entire system through the local properties of its agents and their interaction. In [7, 8], we have introduced the *distributed temporal protocol logic DTPL*, a version of DTL that can be used to reason about protocols and their properties, as well as about different protocol models. We now summarize the main features of DTPL.

DTPL supports the formal specification of and reasoning about systems built over a generic open network, where principals interact by exchanging messages through an insecure public channel, denoted by Ch. The local alphabet of each principal A consists of actions Act_A and state propositions $Prop_A$. Act_A includes

- $send(M, B')$: sending of the message M to B',
- $rec(M)$: reception of the message M,
- $spy(M)$: eavesdropping of the message M,
- $nonce(N)$: generation of the fresh nonce N, and
- $key(K)$: generation of the fresh key K,

and $Prop_A$ includes $knows(M)$, which represents A's knowledge of the message M (although note that we do not explore the epistemic properties of this knowledge).

For the channel, Ch, we do not require state propositions, i.e. $Prop_{Ch} = \emptyset$, whereas the actions Act_{Ch} include

- $in(M, A')$: arrival at the channel of the message M addressed to A',
- $out(M, A')$: delivery of the message M from the channel to principal A, and
- $leak$: leaking of messages.

Given these ingredients, the *global language* $\mathcal{L}$ is defined by the grammar

$$\mathcal{L} ::= @_i[\mathcal{L}_i] \mid \bot \mid \mathcal{L} \Rightarrow \mathcal{L}\,,$$

for $i \in Id$, where the *local languages* $\mathcal{L}_i$ are defined by

$$\mathcal{L}_i ::= Act_i \mid Prop_i \mid \bot \mid \mathcal{L}_i \Rightarrow \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{U}\, \mathcal{L}_i \mid \mathcal{L}_i \,\mathsf{S}\, \mathcal{L}_i \mid j{:}\mathcal{L}_j\,,$$

with $j \in Id$. Actions correspond to true statements about an agent when they have just occurred, whereas state propositions characterize the current local states of the agents. Locally for an agent, U and S are the temporal operators *until* and *since*. Note that the global formula $@_i[\varphi]$ means that φ holds at the current local state of agent i. A local formula $j{:}\varphi$ appearing inside a formula in $\mathcal{L}_i$ is called a *communication formula* and it means that agent i has just communicated with agent j for whom φ held.

The interpretation structures of $\mathcal{L}$ are suitably labeled distributed life-cycles, built upon a simplified form of Winskel's *event structures* [37]. A *distributed life-cycle* λ is a prime event structure without conflict, built from a discrete, linearly ordered, set of

$$\begin{tikzcd}
A & e_1 \arrow[r] & e_4 \arrow[r] \arrow[d, dash, dotted] & e_5 \arrow[rr] & & e_8 \arrow[r] \arrow[d, dash, dotted] & \cdots \\
B & e_2 \arrow[r] & e_4 \arrow[rr] \arrow[d, dash, dotted] & & e_7 \arrow[r] \arrow[d, dash, dotted] & e_8 \arrow[r] & \cdots \\
C & e_3 \arrow[r] & e_4 \arrow[r] & e_6 \arrow[r] & e_7 \arrow[r] & e_9 \arrow[r] & \cdots
\end{tikzcd}$$

FIG. 2. A distributed life-cycle for agents A, B and C.

$$\pi_A(\emptyset) \xrightarrow{\alpha_A(e_1)} \pi_A(\{e_1\}) \xrightarrow{\alpha_A(e_4)} \pi_A(\{e_1, e_4\}) \xrightarrow{\alpha_A(e_5)} \pi_A(\{e_1, e_4, e_5\}) \xrightarrow{\alpha_A(e_8)} \cdots$$

FIG. 3. The progress of agent A.

$$\begin{tikzcd}
 & \{e_1\} \arrow[r, dash] \arrow[rd, dash] & \{e_1, e_2\} \arrow[rd, dash] & & & \{e_1, e_2, e_3, e_4, e_5\} \\
\emptyset \arrow[ru, dash] \arrow[r, dash] \arrow[rd, dash] & \{e_2\} \arrow[ru, dash] \arrow[rd, dash] & \{e_1, e_3\} \arrow[r, dash] & \{e_1 e_2, e_3\} \arrow[r, dash] & \{e_1, e_2, e_3, e_4\} \arrow[ru, dash] \arrow[rd, dash] & \cdots \\
 & \{e_3\} \arrow[r, dash] \arrow[ru, dash] & \{e_2, e_3\} \arrow[ru, dash] & & & \{e_1, e_2, e_3, e_4, e_6\}
\end{tikzcd}$$

FIG. 4. The lattice of global configurations.

events Ev_i for each agent $i \in I$. Events can however be shared by several agents at communication points, as long as no causality loops are introduced. If we denote by $\rightarrow_i$ the *local successor relation* between the events in Ev_i, we obtain the overall event structure by taking $Ev = \bigcup_{i \in Id} Ev_i$ and the *global causality* relation by the reflexive and transitive closure $\rightarrow^*$ of $\rightarrow = \bigcup_{i \in Id} \rightarrow_i$. Collecting all the local events that have occurred up to a given point, we obtain the *local configuration* of an agent i: a finite set $\xi_i \subseteq Ev_i$ closed under local causality, i.e. if $e \rightarrow_i^* e'$ and $e' \in \xi_i$ then also $e \in \xi_i$. Each non-empty local configuration ξ_i is reached, by the occurrence of an event that we call $\mathit{last}(\xi_i)$, from the local configuration $\xi_i \setminus \{\mathit{last}(\xi_i)\}$. A *global configuration* is a finite set $\xi \subseteq Ev$ closed for global causality, i.e. if $e \rightarrow^* e'$ and $e' \in \xi$ then also $e \in \xi$. Clearly, every global configuration ξ includes the local configuration $\xi|_i = \xi \cap Ev_i$ of each agent i. Given $e \in Ev$, note that $e\!\downarrow = \{e' \in Ev \mid e' \rightarrow^* e\}$ is always a global configuration. An *interpretation structure* $\mu = \langle \lambda, \alpha, \pi \rangle$ of DTPL is a labeled distributed life-cycle, where labeling is local to each agent. For each $i \in I$, the function $\alpha_i : Ev_i \rightarrow Act_i$ associates a local action to each local event, and $\pi_i : \Xi_i \rightarrow \wp(Prop_i)$ associates a set of local state propositions to each local configuration.

Fig. 2 illustrates a distributed life-cycle, where each row comprises the local life-cycle of one agent. In particular, $Ev_A = \{e_1, e_4, e_5, e_8, \dots\}$ and $\rightarrow_A$ corresponds to the arrows in A's row. We can think of the occurrence of the event e_1 as leading agent A from its initial configuration $\emptyset$ to the configuration $\{e_1\}$, and then of the occurrence of the event e_4 as leading to configuration $\{e_1, e_4\}$, and so on. The state-transition sequence of agent A is displayed in Fig. 3. Shared events at communication points are highlighted by the dotted vertical lines. Note that the numbers annotating the events are there only for convenience since no global total order on events is imposed in general. Fig. 4 shows the corresponding lattice of global configurations.

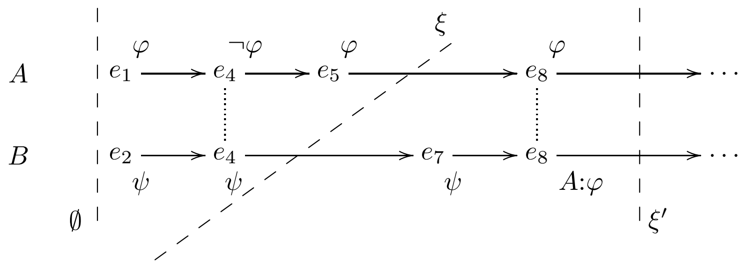


Fig. 5. Satisfaction of formulas.

We can then define the *global satisfaction relation* at a global configuration ξ of μ as

- $\mu, \xi \Vdash @_i[\varphi]$ if $\mu, \xi|_i \Vdash_i \varphi$,
- $\mu, \xi \nVdash \bot$,
- $\mu, \xi \Vdash \gamma \Rightarrow \delta$ if $\mu, \xi \nVdash \gamma$ or $\mu, \xi \Vdash \delta$,

where the *local satisfaction relations* at local configurations are defined by

- $\mu, \xi_i \Vdash_i act$ if $\xi_i \neq \emptyset$ and $\alpha_i(last(\xi_i)) = act$,
- $\mu, \xi_i \Vdash_i p$ if $p \in \pi_i(\xi_i)$,
- $\mu, \xi_i \nVdash_i \bot$,
- $\mu, \xi_i \Vdash_i \varphi \Rightarrow \psi$ if $\mu, \xi_i \nVdash_i \varphi$ or $\mu, \xi_i \Vdash_i \psi$,
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{U}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i \subsetneq \xi_i''$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i \subsetneq \xi_i' \subsetneq \xi_i''$,
- $\mu, \xi_i \Vdash_i \varphi \,\mathsf{S}\, \psi$ if there exists $\xi_i'' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i$ such that $\mu, \xi_i'' \Vdash_i \psi$, and $\mu, \xi_i' \Vdash_i \varphi$ for every $\xi_i' \in \Xi_i$ with $\xi_i'' \subsetneq \xi_i' \subsetneq \xi_i$, and
- $\mu, \xi_i \Vdash_i j{:}\varphi$ if $\xi_i \neq \emptyset$, $last(\xi_i) \in Ev_j$ and $\mu, (last(\xi_i) \downarrow)|_j \Vdash_j \varphi$.

Fig. 5 illustrates the satisfaction relation with respect to communication formulas. Clearly $\mu, \emptyset \Vdash @_B[\psi \,\mathsf{U}\, A{:}\varphi]$, because $\mu, \xi' \Vdash @_B[A{:}\varphi]$. Note however that $\mu, \xi \nVdash @_B[A{:}\varphi]$, although $\mu, \xi \Vdash @_A[\varphi]$.

We say that μ is a *model* of $\Gamma \subseteq \mathcal{L}$ if $\mu, \xi \Vdash \gamma$ for every global configuration ξ of μ and every $\gamma \in \Gamma$. Other standard operators are defined as abbreviations, e.g. connectives and also (local) temporal operators:

$\mathsf{X}\,\varphi$	$\equiv \bot \,\mathsf{U}\, \varphi$	next	$\dagger$	$\equiv \neg \mathsf{X} \top$	in the end
$\mathsf{Y}\,\varphi$	$\equiv \bot \,\mathsf{S}\, \varphi$	previous	$*$	$\equiv \neg \mathsf{Y} \top$	in the beginning
$\mathsf{F}\,\varphi$	$\equiv \top \,\mathsf{U}\, \varphi$	sometime in the future	$\mathsf{F}_\circ\,\varphi$	$\equiv \varphi \vee \mathsf{F}\,\varphi$	now or sometime in the future
$\mathsf{P}\,\varphi$	$\equiv \top \,\mathsf{S}\, \varphi$	sometime in the past	$\mathsf{P}_\circ\,\varphi$	$\equiv \varphi \vee \mathsf{P}\,\varphi$	now or sometime in the past
$\mathsf{G}\,\varphi$	$\equiv \neg \mathsf{F} \neg \varphi$	always in the future	$\mathsf{G}_\circ\,\varphi$	$\equiv \varphi \wedge \mathsf{G}\,\varphi$	now and always in the future
$\mathsf{H}\,\varphi$	$\equiv \neg \mathsf{P} \neg \varphi$	always in the past	$\mathsf{H}_\circ\,\varphi$	$\equiv \varphi \wedge \mathsf{H}\,\varphi$	now and always in the past

Rules for proving invariants by induction can also be formulated and proved in our logic in the standard way, see [6, 7, 8].

We can now introduce the specification of the communication network, where principals can send and receive messages at will, but always through the channel. If the principal A sends a message to B', then the message synchronously arrives at the channel, where it is stored for future delivery to B. If delivery ever happens, it must be synchronized with the corresponding receive action of B. However, A can only send M to B' if A knows both the name B' and how to produce the message M. In addition to their initial knowledge, principals gain knowledge from the messages they receive and from the fresh nonces and keys they generate. Dishonest principals may also spy on messages leaked by the channel and learn their content. We do not allow principals to explicitly divert messages, but we also do not guarantee that messages delivered to the channel are ever received.

To ensure that principals only learn new information through the messages they receive and the fresh data they generate, we specify that the *knows* propositions only hold where necessary. To this end, we follow the idea underlying Paulson's inductive model [32], in accordance with the usual assumption of *perfect cryptography* (that the only way to decrypt an encrypted message is to have the appropriate key). We restrict attention to those interpretation structures μ such that, for every principal A, the following condition holds for all messages M and non-empty local configurations ξ_A:

(K) $\mu, \xi_A \Vdash_A \mathit{knows}(M)$ iff $M \in \mathit{synth}(\mathit{analz}(\{M' \mid \mu, \xi_A \Vdash_A (\mathsf{Y}\,\mathit{knows}(M')) \vee \mathit{rec}(M') \vee \mathit{spy}(M') \vee \mathit{nonce}(M') \vee \mathit{key}(M')\}))$,

where *analz* and *synth* are analysis and synthesis functions that formalize how principals can compose messages by concatenation and encryption, and decompose messages by projection and decryption [32]. Formally, given a set S of messages:

- $\mathit{analz}(S)$ is the least set containing S such that
  - $M_1 \in \mathit{analz}(S)$ and $M_2 \in \mathit{analz}(S)$ if $M_1; M_2 \in \mathit{analz}(S)$, and
  - $M \in \mathit{analz}(S)$ if $\{M\}_K \in \mathit{analz}(S)$ and $K^{-1} \in \mathit{analz}(S)$,
- $\mathit{synth}(S)$ is the least set containing S such that
  - $M_1; M_2 \in \mathit{synth}(S)$ if $M_1 \in \mathit{synth}(S)$ and $M_2 \in \mathit{synth}(S)$, and
  - $\{M\}_K \in \mathit{synth}(S)$ if $M \in \mathit{synth}(S)$ and $K \in \mathit{synth}(S)$.

Note that **(K)** implies that, in every model $\mu = \langle \lambda, \alpha, \pi \rangle$ of the specification, π is completely determined by λ and α, given $\pi_A(\emptyset)$ for each $A \in \mathit{Princ}$. This is equivalent to saying that the knowledge of each principal only depends on his initial knowledge and on the actions that have occurred. In fact, as shown in [6, 7, 8], **(K)** entails a number of useful properties about *knows*, characterizing how principals acquire knowledge. Namely, for each principal $A \in \mathit{Princ}$ we have that:

(K1) $@_A[\mathit{knows}(M_1; M_2) \Leftrightarrow (\mathit{knows}(M_1) \wedge \mathit{knows}(M_2))]$,

(K2) $@_A[(\mathit{knows}(M) \wedge \mathit{knows}(K)) \Rightarrow \mathit{knows}(\{M\}_K)]$,

(K3) $@_A[(\mathit{knows}(\{M\}_K) \wedge \mathit{knows}(K^{-1})) \Rightarrow \mathit{knows}(M)]$,

(K4) $@_A[\mathit{knows}(M) \Rightarrow \mathsf{G}_\circ\,\mathit{knows}(M)]$,

(K5) $@_A[\mathit{rec}(M) \Rightarrow \mathit{knows}(M)]$,

(K6) $@_A[\mathit{spy}(M) \Rightarrow \mathit{knows}(M)]$,

(K7) $@_A[\mathit{nonce}(N) \Rightarrow \mathit{knows}(N)]$, and

(K8) $@_A[\mathit{key}(K) \Rightarrow \mathit{knows}(K)]$.

The full specification of the communication network also comprises a number of axiom schemas that characterize the behavior of the channel and of each principal $A \in \mathit{Princ}$, as well as the way they can communicate:

(C1) $@_{\mathit{Ch}}[\mathit{in}(M, A') \Rightarrow \bigvee_{B \in \mathit{Princ}} B{:}\mathit{send}(M, A')]$,

(C2) $@_{\mathit{Ch}}[\mathit{out}(M, A') \Rightarrow \mathsf{P}\ \mathit{in}(M, A')]$,

(C3) $@_{\mathit{Ch}}[\mathit{out}(M, A') \Rightarrow A{:}\mathit{rec}(M)]$,

(P1) $@_A[\mathit{send}(M, B') \Rightarrow \mathsf{Y}(\mathit{knows}(M) \wedge \mathit{knows}(B'))]$,

(P2) $@_A[\mathit{send}(M, B') \Rightarrow \mathit{Ch}{:}\mathit{in}(M, B')]$,

(P3) $@_A[\mathit{rec}(M) \Rightarrow \mathit{Ch}{:}\bigvee_{A' \in \mathit{Name}_A} \mathit{out}(M, A')]$,

(P4) $@_A[\mathit{spy}(M) \Rightarrow \mathit{Ch}{:}(\mathit{leak} \wedge \mathsf{P} \bigvee_{B' \in \mathit{Name}} \mathit{in}(M, B'))]$,

(P5) $@_A[\bigwedge_{B \in \mathit{Princ} \setminus \{A\}} \neg B{:}\top]$,

(P6) $@_A[\mathit{nonce}(N) \Rightarrow \neg\, \mathit{Ch}{:}\top]$, and

(P7) $@_A[\mathit{key}(K) \Rightarrow \neg\, \mathit{Ch}{:}\top]$.

The channel axiom schemas **(C1–C3)** are straightforward. They state that a message addressed to A' only arrives at the channel if it is sent to A' by some principal B; that the channel only delivers a message to A' if such a message for A' has previously arrived; and that if the channel delivers a message to A', then A receives it. The principal axiom schemas are also simple. **(P1)** is a precondition for sending a message, stating that the sender must know both the message and the recipient's name beforehand. The next three formulas are related to interaction. **(P2)** and **(P3)** state that the sending and receiving of messages, respectively, must be shared with the corresponding arrival and delivery actions of the channel. **(P4)** guarantees that a spied message must have arrived at the channel, addressed to some recipient. The three final axiom schemas limit the possible interactions: **(P5)** guarantees that principals never communicate directly (only through the channel), **(P6)** and **(P7)** state that nonce and key generating actions are not communication actions.

To guarantee the freshness and uniqueness of the nonces and keys generated by each principal, in our network models we could also require the axiom schemas

(N1) $@_A[\mathit{nonce}(N) \Rightarrow \mathsf{Y} \neg\, \mathit{knows}(M_N)]$, and

(N2) $@_A[\mathit{nonce}(N)] \Rightarrow \bigwedge_{B \in \mathit{Princ} \setminus \{A\}} @_B[\neg\, \mathit{knows}(M_N)]$,

where M_N ranges over all the messages containing the nonce N. Together with (K7), **(N1)** and **(N2)** guarantee that every nonce is generated at most once, if at all, in each model, and always freshly (also taking into account the initial knowledge of all principals). Analogous formulas can be written regarding key generation. However, in the present work, we shall not consider them for the reasons explained at the end of Section 4.

Note that the specification given can, of course, be extended in many ways, e.g. by including other kinds of message constructors (such as for hashing), additional actions and state propositions, or additional channels with distinct accessibility and reliability properties. We consider some of these extensions in [6]. The above is however enough to abstractly formalize and reason about the properties of communication between principals executing security protocols.

2.2.2 Protocol modeling.

Formalizing in DTPL a protocol like NSPK as described in Fig. 1 involves defining the sequences of actions (*send*, *rec*, *nonce*, and *key*) taken by honest principals executing the protocol. Namely, for each role, we formalize the actions taken and the order in which they must be taken. In the case of the NSPK protocol, there are two roles: an initiator role *Init*, represented by a, and a responder role *Resp*, represented by b. Given distinct names A' and B', of principals A and B respectively, and nonces N_1 and N_2, the role instantiations should correspond to the execution, by principal A, of the following sequence $\text{run}_A^{Init}(A', B', N_1, N_2)$ of actions for the initiator role:

$$\langle \mathit{nonce}(N_1).\mathit{send}(\{N_1; A'\}_{K_{B'}}, B').\mathit{rec}(\{N_1; N_2\}_{K_{A'}}).\mathit{send}(\{N_2\}_{K_{B'}}, B')\rangle \,,$$

and similarly for the responder role.

In general, a protocol description like the one above may involve j name variables $a_1, \ldots, a_j$, corresponding to j distinct roles, m nonce variables $n_1, \ldots, n_m$, and p key variables $k_1, \ldots, k_p$, and consist of a sequence $\langle \text{step}_1 \ldots \text{step}_u \rangle$ of message exchange steps, each of the form

$$(\text{step}_q) \quad a_s \to a_r \ : \ (n_{q_{t_1}}, \ldots, n_{q_{t_\nu}}, k_{q_{v_1}}, \ldots, k_{q_{v_\kappa}}).\ M \,,$$

where $1 \leq q_{t_1}, \ldots, q_{t_\nu} \leq m$, $1 \leq q_{v_1}, \ldots, q_{v_\kappa} \leq p$, and M can include any of the name, nonce, and key variables. A *protocol instantiation* is a variable substitution σ such that each $\sigma(a_i) \in \mathit{Name}$, each $\sigma(n_i) \in \mathit{Nonce}$, each $\sigma(k_i) \in \mathit{Key}$, and σ is injective on name variables. We extend σ to messages, actions, sequences, and formulas in the natural way. Each instantiation prescribes a concrete sequence of actions to be executed by each of the participants in a run of the protocol: for each role i, if $\sigma(a_i) \in \mathit{Name}_A$ then we have the corresponding sequence $\text{run}_A^i(\sigma) = \sigma(\text{step}_1^i) \centerdot \cdots \centerdot \sigma(\text{step}_u^i)$ where

$$\text{step}_q^i = \begin{cases} \langle \mathit{nonce}(n_{q_{t_1}}) \ldots \mathit{nonce}(n_{q_{t_\nu}}).\mathit{key}(k_{q_{v_1}}) \ldots \mathit{key}(k_{q_{v_\kappa}}).\mathit{send}(M, a_r)\rangle & \text{if } i = s \,, \\ \langle \mathit{rec}(M)\rangle & \text{if } i = r \,, \\ \langle\rangle & \text{otherwise.} \end{cases}$$

In general, if we denote the set of all protocol instantiations by *Inst*, we can define the set Runs_A^i of all possible concrete runs of principal A in role i, and the set Runs_A of all of A's possible concrete runs in any of the j roles: $\mathit{Runs}_A^i = \bigcup_{\sigma \in \mathit{Inst}} \{\text{run}_A^i(\sigma) \mid \sigma(a_i) \in \mathit{Name}_A\}$ and $\mathit{Runs}_A = \bigcup_{i=1}^{j} \mathit{Runs}_A^i$.

Since it is enough to consider one dishonest principal, as we show in [8], we consider that Z is the identity of the intruder and $\mathit{Hon} = \mathit{Princ} \setminus \{Z\}$. While Name_Z, the set of names used by Z, can include many elements, we assume that $\mathit{Name}_A = \{A\}$, for every $A \in \mathit{Hon}$.

We define the *initial knowledge* of a principal $A \in \mathit{Princ}$ as the set of all messages built from a subset of all principal names plus a set $\mathcal{K}_A$, where $\mathcal{K}_A$ typically contains all public keys, all private keys held by A, and each symmetric key K_{Ax} shared by A with a principal x. A DTPL axiomatization of initial knowledge can be found in [6, 7, 8].

Models of a protocol are those network models where, furthermore, all honest principals strictly follow the rules of the protocol. That is, for every $A \in \mathit{Hon}$, if the local life-cycle of A is $e_1 \rightarrow_A e_2 \rightarrow_A e_3 \rightarrow_A \ldots$, then the corresponding (possibly infinite) sequence of actions $w(A) = \langle \alpha_A(e_1).\alpha_A(e_2).\alpha_A(e_3)\ldots\rangle$ must be an interleaving of prefixes of sequences in Runs_A, but using distinct fresh nonces in each of them.

For the NSPK protocol, this means that the life-cycle of each honest principal must be built by interleaving prefixes of sequences of the form $\mathrm{run}_A^{\mathit{Init}}(A, B', N_1, N_2)$ or $\mathrm{run}_A^{\mathit{Resp}}(B', A, N_1, N_2)$, where no two such initiator runs can have the same N_1, no two responder runs can have the same N_2, and the N_1 of an initiator run must be different from the N_2 of any responder run.

2.2.3 Security goals.

The aim of protocol analysis is to prove (or disprove) the correctness of a protocol with respect to the security goals that the protocol should achieve. Two of the most important goals concern the *secrecy* of data and the *authentication* of principals and message origination. There are many approaches to specifying these goals in the literature, depending in part on the underlying model used. However, the various approaches mostly agree on the general picture. Below, we just briefly sketch how such properties can be expressed in DTPL; further details can be found in [6, 7, 8]. More specifically, we show how below to formulate the required secrecy and authentication goals of protocols in the general case, illustrating them by means of the NSPK protocol.

Let us start with secrecy. We can formalize that the messages in a finite set S will remain shared secrets between the participants $A_1, \ldots, A_j$ after the complete execution of a protocol instantiation σ, with each $\sigma(a_i) \in \mathit{Name}_{A_i}$, by the formula

$$\bigwedge_{i=1}^{j} @_{A_i}[\mathsf{P}_\circ\ \mathrm{role}^i_{A_i}(\sigma)] \Rightarrow \bigwedge_{B \in \mathit{Princ} \setminus \{A_1, \ldots, A_j\}} \bigwedge_{M \in S} @_B[\neg\, \mathit{knows}(M)].$$

There are many possible notions of authentication (see, e.g., [27]). However, most authors agree that authentication expresses some kind of correspondence property. The typical authentication goal states that if an honest principal A completes his part of a run of a protocol in role i, with certain partners and data, then it must be the case that these partners have also been actively involved by sending to A the messages that A received. Given a protocol instantiation σ such that $\sigma(a_i) = A \in \mathit{Hon}$ and $\sigma(a_j) \in \mathit{Name}_B$, the property that A authenticates B in role j at step q of the protocol can be defined in our logic by the formula

$$@_A[\mathrm{role}^i_A(\sigma)] \Rightarrow @_B[\mathsf{P}_\circ\ \mathit{send}(\sigma(M), A)],$$

assuming that the protocol step_q requires that a_j send the message M to a_i.

Given a security goal γ, we call an *attack* on a protocol any protocol model μ and configuration ξ for which the formula expressing the goal does not hold, i.e. such that $\mu, \xi \not\Vdash \gamma$.

2.3 Strand spaces

2.3.1 The approach.

In this section, we summarize strand spaces, closely following [36]. A *strand* is a sequence of events that represents either the actions of a legitimate party (i.e. an honest principal) in a security protocol or the actions of an intruder. One speaks of *legitimate strands* in the former case and *intruder strands* in the latter.[2] A *strand space* is a collection of strands, equipped with a graph structure formalizing causal interaction: a *bundle* is a subspace of a strand space, representing communication between different strands.

More formally, let us consider a set $\mathcal{M}$ of messages that can be exchanged between principals in a protocol (the results that we present here are independent of the actual structure of the elements of $\mathcal{M}$). A *signed term* is a pair $\langle \Sigma, M \rangle$, with $\Sigma \in \{+, -\}$ and $M \in \mathcal{M}$. We write signed terms as $+M$ or $-M$, and we use the projections Π_1 and Π_2 to return the sign and the term part, respectively, of a signed term. Moreover, we denote the set of finite sequences of signed terms by $(\pm\mathcal{M})^*$, whose elements are of the form $\langle\langle \Sigma_1, M_1 \rangle \dots \langle \Sigma_n, M_n \rangle\rangle$. We will slightly abuse notation and refer to subterms of signed terms.

A *strand space* over $\mathcal{M}$ is a set $\mathcal{S}$ together with a trace mapping $tr : \mathcal{S} \to (\pm\mathcal{M})^*$, associating each strand in $\mathcal{S}$ with a sequence of signed terms. As is standard, we usually represent a strand space by its underlying set of strands $\mathcal{S}$.

Given a strand space $\mathcal{S}$, a *node* is a pair (s, i), where $s \in \mathcal{S}$ and i is an integer, $1 \le i \le |tr(s)|$. We denote the set of nodes of $\mathcal{S}$ by $\mathcal{N}_{\mathcal{S}}$ or simply by $\mathcal{N}$. Further, we say that the node (s, i) belongs to the strand s. It follows by definition that every node belongs to a unique strand. Given a node $n = (s, i)$, we define $str(n) = s$, $index(n) = i$, and $term(n) = tr(s)_i$, i.e. the i-th signed term in the trace of s.

Two nodes n_1 and n_2 can be connected by one of two different kinds of *edges*. If there is an edge $n_1 \to n_2$, then $term(n_1) = +M$ and $term(n_2) = -M$, for some $M \in \mathcal{M}$. This edge expresses that n_1 sends M, which is received by n_2, thereby recording a causal link between their respective strands. There is an edge $n_1 \Rightarrow n_2$ iff n_1 and n_2 are both on the same strand, $n_1 = (s, i)$, and $n_2 = (s, i+1)$. This edge expresses that n_1 is an immediate causal predecessor of n_2 on the strand s. We write $\Rightarrow^+$ for the transitive closure of $\Rightarrow$.

An unsigned term M *occurs in* $n \in \mathcal{N}$ iff M is a subterm of $term(n)$. If $\mathcal{U}$ is a set of unsigned terms, then the node $n \in \mathcal{N}$ is an *entry point* for $\mathcal{U}$ iff $term(n) = +M$ for some $M \in \mathcal{U}$, and $term(n') \notin \mathcal{U}$ whenever $n' \Rightarrow^+ n$. An unsigned term M *originates* on $n \in \mathcal{N}$ iff n is an entry point for the set $\mathcal{U} = \{M' \mid M \text{ is a subterm of } M'\}$, and M is *uniquely originating* iff it originates on a unique $n \in \mathcal{N}$. If a term M originates uniquely in a particular strand space, then it can play the role of a nonce or generated (session) key in that structure.

The set $\mathcal{N}$ of nodes together with the two sets of edges $n_1 \to n_2$ and $n_1 \Rightarrow n_2$ form a directed graph $\langle \mathcal{N}, (\to \cup \Rightarrow) \rangle$. A *bundle* is a finite subgraph of this graph for which we can regard the edges as expressing the causal dependencies between the nodes. Let $\mathcal{C} = \langle \mathcal{N}_{\mathcal{C}}, (\to_{\mathcal{C}} \cup \Rightarrow_{\mathcal{C}}) \rangle$, with $\to_{\mathcal{C}} \subseteq \to$, and let $\Rightarrow_{\mathcal{C}} \subseteq \Rightarrow$ be a subgraph of $\langle \mathcal{N}, (\to \cup \Rightarrow) \rangle$. Then $\mathcal{C}$ is a bundle if (1) $\mathcal{C}$ is finite; (2) if $n_2 \in \mathcal{N}_{\mathcal{C}}$ and $term(n_2)$ is

[2] In strands terminology, the intruder is called *penetrator* (and one speaks of *penetrator strands*), but in this paper we use only the name "intruder" for simplicity.

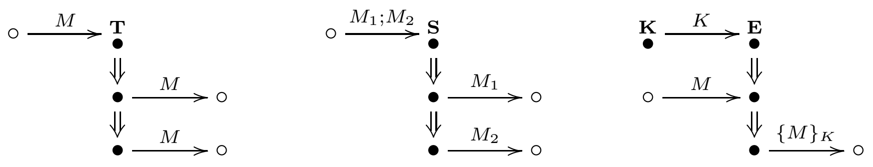


FIG. 6. Intruder traces **T**, **S**, and **K** and **E**.

negative, then there is a unique n_1 such that $n_1 \rightarrow_{\mathcal{C}} n_2$; (3) if $n_2 \in \mathcal{N}_{\mathcal{C}}$ and $n_1 \Rightarrow n_2$, then $n_1 \Rightarrow_{\mathcal{C}} n_2$; and (4) $\mathcal{C}$ is acyclic. Note that in conditions (2) and (3), it follows that $n_1 \in \mathcal{N}_{\mathcal{C}}$, because $\mathcal{C}$ is a graph.

In strand spaces, like in most other approaches to security protocol analysis, such as ours based on DTPL, it is customary to follow Dolev and Yao [17] and consider the model of an active intruder who controls the network but cannot break cryptography. In particular, the intruder can intercept messages and analyze and decrypt them if he possesses the corresponding keys for decryption, and he can generate messages from his knowledge and send them under any agent's name. Such an intruder is modeled in [36] as a principal whose capabilities are characterized by the set $\mathcal{K}_Z$ of keys initially known to him and a set of intruder strands that allow him to discard messages, compose messages, and apply cryptographic operations using the keys that become available to him.

An *intruder trace* is one of the following kinds:

- **M**(essage): $\langle +M \rangle$, for M an atomic message,
- **K**(ey): $\langle +K \rangle$, where $K \in \mathcal{K}_Z$,
- **F**(lushing): $\langle -M \rangle$,
- **T**(ee): $\langle -M. + M. + M \rangle$,
- **C**(oncatenation): $\langle -M_1. - M_2. + M_1; M_2 \rangle$,
- **E**(ncryption): $\langle -K. - M. + \{M\}_K \rangle$,
- **S**(eparation): $\langle -M_1; M_2. + M_1. + M_2 \rangle$,
- **D**(ecryption): $\langle -\{M\}_K. - K^{-1}. + M \rangle$.

Fig. 6 shows the diagrams that represent graphically the intruder traces **T**, **S**, and **K** and **E**, where the open circles $\circ$ show the points at which other diagrams can be connected. Here we have explicitly added the trace names, but we usually omit these to improve readability.

Note that it is possible to extend this collection of intruder traces to model additional capabilities of the intruder. Note also that, contrarily to [36], which we follow, more recent presentations of strand spaces, such as [24], do not consider **T** and **F** traces. Indeed, neither of these traces appears to be essential. We consider them here because they help in clarifying the relationship between strand spaces and DTPL, as explained in Section 3.

2.3.2 Protocol modeling.

An *infiltrated strand space* is a strand space that contains both legitimate strands and intruder strands, i.e. it is a pair $(\mathcal{S}, \mathcal{Z})$, where $\mathcal{S}$ is a strand space, $\mathcal{Z} \subseteq \mathcal{S}$, and $tr(z)$ is an intruder trace for all $z \in \mathcal{Z}$. A strand $s \in \mathcal{S}$ is an *intruder strand* if it belongs to $\mathcal{Z}$, and a node is an *intruder node* if it lies on an intruder strand. Otherwise we call it a *regular* (or honest) strand or node. A node n is an **M**-*node* (**K**-*node*, etc.), if it lies on an intruder strand with a trace of kind **M** (**K**, etc.). From now on, by strand space we will always mean an infiltrated strand space, and a bundle for a protocol will be a bundle that consists of the legitimate strands plus the intruder strands. Moreover, for simplicity, we will often identify an intruder strand with its corresponding intruder trace.

To fix the set of allowed legitimate strands, we depart from a given specification as before, yielding a set of concrete runs. To each such run, we also associate an honest strand. All these honest strands, together with the intruder strands, comprise our strand space of interest. Moreover, **K** and **M** intruder strands can only appear for the intruder's initial knowledge, or for uniquely originating data (no guessing). Of course, fresh data generation actions by honest principals have to be left implicit in strands and can only be derived through the notion of unique origination in the bundles including them. Essentially, we assign to $A \in \mathit{Hon}$ all strands

$$\mathit{run2str}(w) = \mathit{act2str}(\mathit{act}_1) \centerdot \cdots \centerdot \mathit{act2str}(\mathit{act}_n)$$

obtained from a run $w = \langle \mathit{act}_1 \ldots \mathit{act}_n \rangle$ of A as follows:

$$\mathit{act2str}(\mathit{act}) = \begin{cases} \langle +M \rangle & \text{if } \mathit{act} = \mathsf{send}(M, B'), \text{ for some } B' \in \mathit{Name}\,, \\ \langle -M \rangle & \text{if } \mathit{act} = \mathsf{rec}(M)\,, \\ \langle \rangle & \text{otherwise.} \end{cases}$$

There may, however, be several prefixes of a run w that are mapped to the same prefix of $\mathit{run2str}(w)$ since internal nonce and key generation actions are hidden from the resulting strand. In any case, we shall say that the i-th node in $\mathit{run2str}(w)$ corresponds to the j-th action of w if $j - i$ is the number of internal actions (nonce and key generation) occurring in $\langle \mathit{act}_1 \ldots \mathit{act}_{j-1} \rangle$.

Note also that all information concerning message destinations is absent in strands. Still, we believe that the concretization of this missing information is important, aside from being technically useful. Neglecting this information may lead to situations where strand models appear unnatural, and where the role effectively played by the intruder is not made completely explicit. Fig. 7 illustrates such a situation, where an honest $+M$ is directly connected to an honest $-M$, but the $+M$ corresponds to a run of the protocol containing $\mathsf{send}(M, C)$ and the agent owning the strand with $-M$ is not C, but rather B. In this case, there is an implicit interposition of the intruder, who does $\mathsf{spy}(M)$ and then $\mathsf{send}(M, B)$, which we can make explicit by using a **T** intruder strand (possibly along with an **F** strand to flush the additional M).

To overcome this problem, we assume in this paper that situations such as the one in Fig. 7 do not occur, or are replaced as illustrated in Fig. 8. For similar reasons, we also assume that bundles never contain more than one **M** or **K** node labeled with the same message. If such a message is needed more than once, we just replicate it using **T** strands, making it possible to maintain control on the origin of data, especially

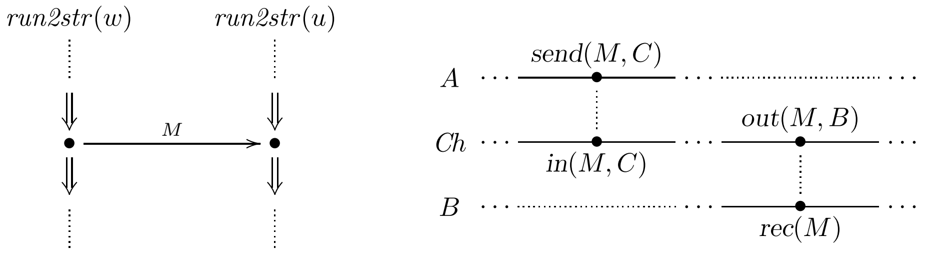


FIG. 7. Message destination is absent in strands: a problem.

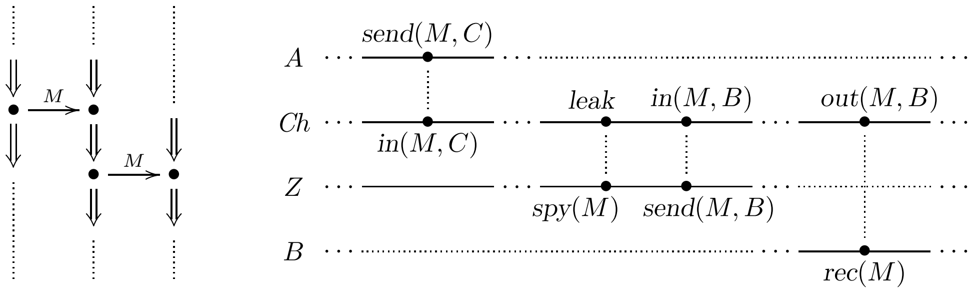


FIG. 8. Message destination is absent in strands: a solution of the problem.

for fresh data in **M** nodes, which can now be captured through the notion of unique origination.

Fig. 9 shows, on the left, the "infiltrated NSPK protocol", where the intruder Z performs a man-in-the-middle attack on two principals A and B who are executing the NSPK protocol. The open circles and dotted double arrows indicate the points at which the two "internal" steps of the intruder mesh with A and B's strands, and on the right of the figure, we show the details of the first step of the intruder, i.e. how he uses his private key K_Z^{-1} and B's public key K_B to transform A's message $\{N_1; A\}_{K_Z}$ into $\{N_1; A\}_{K_B}$. The second intruder step is similar.

For the sake of our comparison with DTPL, we also introduce the following notions. We say that $n = (s, i)$ is a *sending* or *receiving* node depending on the sign $\Pi_1(tr(s)_i)$ of the corresponding message. Furthermore, for a node n in a bundle, we say that:

- n is a *dangling* node if it is not connected via $\rightarrow$ (then n must be a sending node, in which case we also say that the corresponding unsigned message is dangling);
- n is an *interface* node if it is an intruder node that is dangling or connected via $\rightarrow$ to an honest node; and
- n is a *productive* node if it is an intruder node and causally precedes a sending interface node, or is itself one.

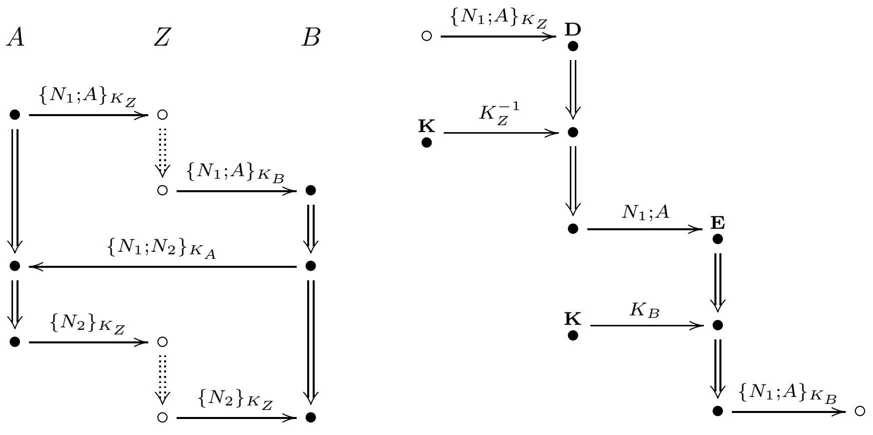


FIG. 9. Infiltrated NSPK and the intruder's first step.

2.3.3 Security goals.

Although a specific syntax has been proposed in [34] for expressing properties of bundles, we here follow [36] where both secrecy and authentication goals are formulated as metalevel assertions. No formal definitions are given, but the general shape of such properties is discussed. Given also the worked examples in [24, 36], we can come up with formulations that match the same situations that we have considered with DTPL.

In particular, the secrecy of the messages in a set S after the completion of a protocol instantiation σ by honest principals $A_1, \ldots, A_j$ can be formulated by requiring that if a protocol bundle contains the corresponding regular strands then none of the messages in S occurs in an intruder node. On the other hand, suppose the honest principal A has completed his role in a given protocol instantiation σ, where the honest principal B is supposed to have sent a message M to A. The authentication of B by A can now be formulated by requiring that if a protocol bundle contains the regular strand corresponding to A's execution of the protocol, then it must also contain B's strand, or at least a prefix of it with a node labeled with $+M$.

With respect to such security goals, it is worth mentioning that [24] contains a notion of *bundle equivalence*: two bundles on a given strand space are equivalent iff they contain exactly the same regular nodes. As explained there, both secrecy and authentication properties as expressed above are invariant under bundle equivalence.

3 Comparing DTPL and strand-space protocol models

3.1 The differences

To compare DTPL and strand-space protocol models, we must overcome several differences. We focus here on the three main ones and describe the solutions we have adopted. The first difference stems from the fact that although both approaches are based on partially-ordered sets of events with labeling information, they each have distinct starting points. In DTPL, both honest principals and the intruder have the same first-class status and their differences are made explicit only by different constraints on the way they can behave. In contrast, the strand-space approach focuses almost exclusively on the intruder (penetrator): while the smallest internal details of the way the intruder builds messages are explicitly represented in bundles, honest behavior is only represented in terms of interaction with the environment and all internal details are omitted. We may say that strand spaces provide an intruder view of protocols. In the extreme case, even protocol executability by honest participants must just be assumed. In contrast, DTPL models express whether a certain message can be assembled or not, but do not tell us how. To compare the two approaches, we shall have to choose, in each case, one of the potentially many ways to produce a message from the available information.

The second difference is that strand spaces abstract away the communication medium. In DTPL models, we have an explicit communication channel through which all message exchanges pass, and we can model the act of the intruder spying a message. In strands, no such medium exists. We can however view the strand space as also representing the intruder's view on the channel, according to its Dolev-Yao capabilities. Still, to make the approach meaningful, we must understand a dangling $+M$ node in a bundle as a resource that can be used by the intruder, rather than a message in transit. This resource-oriented interpretation underlies the usefulness of intruder strands of type **T** and **F**, although, as we remarked above, one can do without these strands. Hence, we shall assume that **T** and **F** strands are the way that bundles can model a channel that can loose or duplicate messages, like the channel we modeled in DTPL. According to this view, we will require $\rightarrow$ to be functional, i.e. no $+M$ node can be linked by $\rightarrow$ to more than a single $-M$ node. If several such arrows exist, then we can use **T** strands to replicate M as many times as necessary. Furthermore, **F** strands can be used to model the situation where a message is deliberately not spied by the intruder (the intruder node involved will not be productive).

Another related issue concerns the precise origin of messages. In bundles, the precise origin of a received or spied message is made explicit. In contrast, DTPL models express that particular messages can be received by the principals, but if a message has several possible origination points, then we shall have to choose one of them.

The third difference between the two approaches, as also noted by Halpern and Pucella in [25], is that the strand approach does not account for information about agents who may be involved in several distinct, possibly interleaved, executions of a protocol. Indeed, emphasizing the fact that honest principals should never reuse information from one protocol execution in another, each execution is treated as an independent honest strand. To overcome this situation, we have chosen to associate to each honest principal its own set of honest strands, according to the definition of *run2str* in the preceding section. Although we agree with Halpern and Pucella's argument about the need to make this information explicit if one is interested in reasoning about the evolution of each agent, we believe that our solution is cleaner.

Given these differences, it makes sense to compare a bundle $\mathcal{C}$ and a global configuration ξ of a DTPL model μ of a given protocol along the following lines.

Same honest behavior For every $A \in Hon$, $\alpha_A(\xi_A)$ is an interleaving of the run prefixes $u_1, \ldots, u_n$ if and only if the regular nodes in $\mathcal{C}$ correspond precisely to A's strand prefixes $run2str(u_1), \ldots, run2str(u_n)$.

Same intruder output If an event labeled with $send(M, A')$ appears in ξ_Z, then an intruder interface node labeled with $+M$ appears in $\mathcal{C}$, either dangling or connected by $\rightarrow$ to a node of A.

Same intruder knowledge For every message M, $\mu, \xi \Vdash @_Z[knows(M)]$ if and only if $M \in synth(analz(S))$, where S is the set of all (unsigned) messages occurring in dangling or productive nodes of $\mathcal{C}$.

Whenever these three conditions are fulfilled, we will say that the global configuration ξ of μ is *equivalent* to $\mathcal{C}$.

Note that the definition of same intruder output is asymmetric because a dangling interface node can be seen either as a message in transit or as an internal action of the intruder. The statement of same intruder knowledge reflects that the data in productive nodes is precisely the relevant data manipulated by the intruder, whereas dangling nodes represent the data that the intruder can spy.

3.2 From DTPL model configurations to bundles

Let μ be a DTPL model of a given protocol and ξ a global configuration of μ. Our goal is to build an equivalent bundle $\mathcal{C} = mc2b(\mu, \xi)$, in the sense defined above. As we have discussed, DTPL models do not contain explicit information on how the intruder produces messages from the available information. This is however essential for the translation at hand.

Let us say that a bundle is *M-producing* if it contains a dangling node labeled with $+M$. Further, given a set S of messages, let us call an *S-bundle* any bundle-like structure with only intruder strands, but excluding **M** and **K** intruder strands, where receiving nodes $-M$ can be dangling provided that $M \in S$.

The construction of bundles from DTPL models and configurations relies on the possibility of building S-bundles and integrating them in a structure with dangling $+M$ nodes for every $M \in S$. The next two lemmas show how to accomplish this.

Lemma 3.1
Let S be a set of messages. Then $M \in synth(analz(S))$ if and only if there exists an M-producing S-bundle.

Proof. (Sketch) Suppose that $M \in synth(analz(S))$. If $M \in S$, then just consider the empty S-bundle. Otherwise, just use the induction hypothesis and the **C**, **S**, **E**, or **D** intruder strands to build the appropriate S-bundle.

For the converse, we proceed by cases, with **M** and **K** excluded. **F** strands are irrelevant since they do not produce new messages, as well as **T** strands because they only duplicate already existing messages. We are then left with **C**, **S**, **E**, or **D**, which are precisely covered by analysis and synthesis. ∎

Fig. 10 shows two possible S-bundles producing $M_1; M_2; M_3$ from $M_1; M_2$ and $M_2; M_3$. Note that there are many other possibilities for producing such a message. Indeed, it is always the case that messages can be produced in several different ways. Fig. 11 shows also two possible S-bundles producing $M; M$ from $\{M; M\}_K$ and K^{-1},

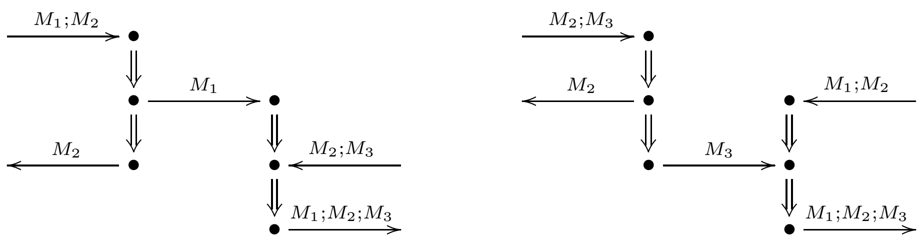


FIG. 10. Two ways of producing $M_1; M_2; M_3$ from $M_1; M_2$ and $M_2; M_3$.

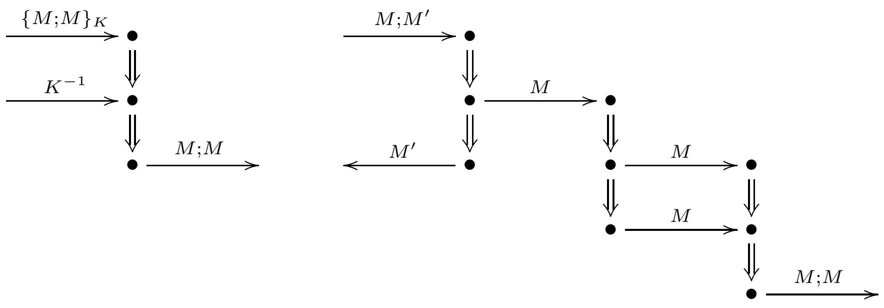


FIG. 11: Two possible bundles producing $M; M$ from $\{M; M\}_K$ and K^{-1}, and $M; M'$.

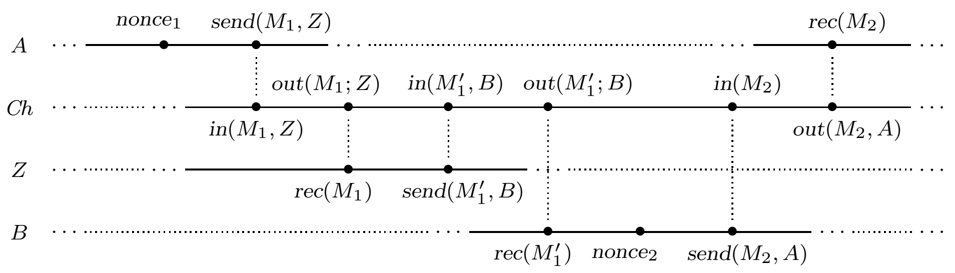
and from $M; M'$. Still, as shown in [24], one can always choose a bundle without redundancies.

LEMMA 3.2
Let S be the set of all dangling messages in a bundle $\mathcal{C}$, and M a message. Then, every M-producing S-bundle can be used to extend $\mathcal{C}$ into a bundle $\mathcal{C}'$ whose set of dangling messages contains $S \cup \{M\}$.

PROOF. (Sketch) Given an M-producing S-bundle, build $\mathcal{C}'$ by gluing it to $\mathcal{C}$ through a number of intruder **T** strands to allow messages to be used multiple times. In general, for each dangling occurrence of $-M'$ in the S-bundle, use a **T** strand $\langle -M'. +M'. + M'\rangle$, connecting the $-M'$ to any dangling $+M'$, the first $+M'$ to the $-M'$ of the S-bundle, and leaving the second $+M'$ dangling for future use. ∎

The general construction of $mc2b(\mu, \xi)$ is then the following:

1. Collect the prefixes $run2str(u)$ of all honest strands corresponding to the prefixes u of the honest runs used to build ξ_A, for each $A \in$ *Hon*.
2. Add a **K** intruder node for each key $K \in \mathcal{K}_Z$, where $knows(K) \in \pi_Z(\emptyset)$.
3. Add an **M** intruder node for each name A', where $knows(A) \in \pi_Z(\emptyset)$.
4. Following the order of ξ_Z, for each event labeled with



Here $nonce_1 = nonce(N_1)$, $M_1 = \{N_1; A\}_{K_Z}$, $M'_1 = \{N_1; A\}_{K_B}$, $nonce_2 = nonce(N_2)$, and $M_2 = \{N_1; N_2\}_{K_A}$.

FIG. 12: Part of the DTPL model of the man-in-the-middle attack on the NSPK protocol.

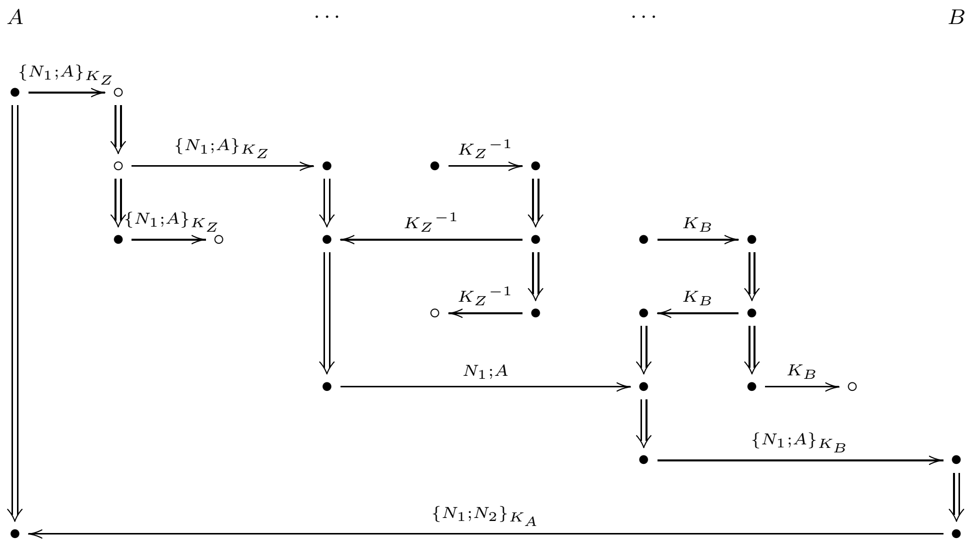


FIG. 13: Part of a bundle obtained from Fig. 12 (the unnamed strands are intruder strands).

(a) $send(M, A')$: enlarge the structure with an S-bundle producing M, according to Lemmas 3.1 and 3.2, where S is the set of all messages currently dangling in the structure (note that this is always possible without introducing cycles);

(b) $rec(M)$ or $spy(M)$: ignore for now;

(c) $nonce(N)$ or $key(K)$: add an **M** intruder node $+N$ or $+K$, respectively.

5. Introduce an edge $\rightarrow$ to each honest $-M$ from a compatible existing $+M$ previously duplicated by a **T** strand (again, this can always be done without cycles).

6. Finally, flush the dangling $+M$ that occur in honest nodes, or in $\mathbf{T}$ nodes that only depend causally on honest nodes, for all messages M that have not been received or spied by the intruder in ξ_Z.

As an illustration of the overall construction of bundles from DTPL models, consider the configuration depicted in Fig. 12, which corresponds to a part of the man-in-the-middle attack on the NSPK protocol. Fig. 13 illustrates part of a possible bundle obtained by the construction, assuming that $\mathcal{K}_Z = \{K_Z^{-1}, K_B\}$. (Details on our analysis of NSPK and the corrected version NSL [26] can be found in [6, 7, 8]).

We thus have the following result:

Theorem 3.3
mc2b(μ, ξ) is a bundle and is equivalent to the configuration ξ of μ.

The fact that *mc2b*(μ, ξ) is a bundle is due to the successive integration of S-bundles in step 4, together with step 5, which removes dangling receives. The absence of cycles can always be guaranteed if in steps 4 and 5 we choose resources corresponding to previously occurring messages (e.g. with respect to the channel ordering in μ). This can be explicitly achieved by tagging all the nodes of the bundle according to the order of ξ_Z as follows: in step 1, tag each honest node with the order (1, 2, 3, ...) of the corresponding *in*/*out* event; in steps 2, 3, and 4(c), tag all nodes with 0; in step 4(a), tag all nodes of the S-bundle with the order of the corresponding *in*(M, A') event, and all nodes of the needed $\mathbf{T}$ strands with the same tag as their linking node. In this way, steps 4 and 5 can be accomplished by connecting a $-M$ tagged i to a $+M$ tagged j with $j < i$. This is always possible given the specification of DTPL models, namely the channel axiom schemas **(C1–3)**.

Note that steps 4(a) and 5 of our construction involve choices and hence this construction can also be seen as defining a collection of bundles. In step 4(a) we must choose both an adequate S-bundle and how to glue it to the rest of the structure, and there may be several choices for both. Step 5 includes choosing connections between sending and receiving nodes, and again there may be different possible choices of a sending node for each receiving node. These choices reflect the amount of indetermination that we can find in DTPL models, with respect to bundles. The internal behavior of the intruder is left implicit, and the direct correspondence between sent and received messages is not present in DTPL models. Still, any bundle *mc2b*(μ, ξ) built in this way ends up with dangling nodes corresponding precisely to the data actually held by the intruder. Most of these nodes should however be interpreted as internal resources of the intruder, rather than messages that he actually sent.

The first step of the construction, together with the fact that all other steps can only introduce intruder strands, guarantees that the bundle *mc2b*(μ, ξ) has the same honest behavior as the configuration ξ of μ. Step 4(a) guarantees that they also have the same intruder output, since steps 5 and 6 do not invalidate it. The same intruder knowledge is guaranteed by the fact that the set S of dangling messages of $\mathcal{C}$ contains all the initial knowledge of Z (steps 2 and 3), plus all the messages Z received or spied (not flushed in the last step), plus all the fresh data Z generated (step 4(c)), possibly along with additional messages that could already be produced from these.

3.3 From bundles to DTPL model configurations

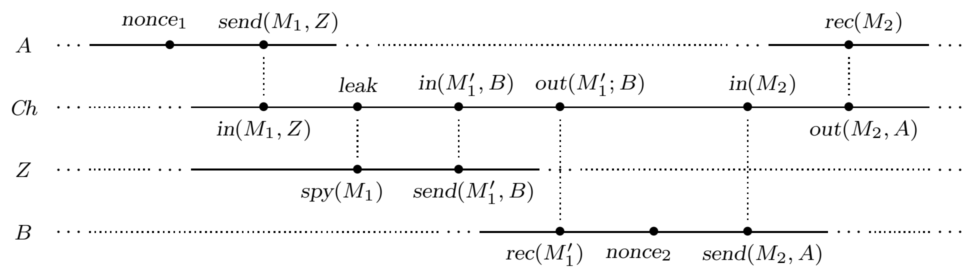
Let $\mathcal{C}$ be a bundle over the infiltrated strand space of a protocol. Our goal is to define a translation $b2mc(\mathcal{C})$ that builds a DTPL model μ of the given protocol that has a global configuration ξ equivalent to $\mathcal{C}$. The general construction of $(\mu, \xi) = b2mc(\mathcal{C})$ is as follows:

1. Extend the bundle to a well-founded total order $\leq$.
2. Collect in a set U_A the prefixes u of honest runs corresponding to the prefixes $run2str(u)$ of each of the honest strands of A occurring in $\mathcal{C}$, for each $A \in Hon$.
3. For each $A \in Hon$:
 (a) let $Ev_A = \{\langle u, i\rangle \mid u \in U_A, 1 \leq i \leq |u|\}$,
 (b) choose $\rightarrow_A$ such that
  - if $\langle u, i\rangle, \langle u, j\rangle \in U_A$ and $i < j$ then $\langle u, i\rangle \rightarrow_A^* \langle u, j\rangle$, and
  - if two nodes n, n' in $\mathcal{C}$ such that $n \leq n'$ correspond to $\langle u, i\rangle, \langle u', i'\rangle \in U_A$, respectively, then $\langle u, i\rangle \rightarrow_A^* \langle u', i'\rangle$,

 (c) define $\alpha_A(\langle u, i\rangle) = u_i$, and
 (d) define $\pi_A(\emptyset) = \{knows(M) \mid M \in synth(analz(S_A))\}$, where S_A includes all principal names and public keys, as well as A's private keys.
4. For the intruder:
 (a) fix the subset D of the set of all dangling intruder nodes of $\mathcal{C}$ that are to be considered as sending actions,
 (b) let Ev_Z contain an event e_N or e_K for each nonce N or key K uniquely originating in a productive intruder **M** node (we call these nodes $o(N)$ or $o(K)$, respectively), and additionally an event corresponding to every node n of $\mathcal{C}$ that is either a receiving interface node that is productive, a dangling honest node, or a sending interface node that is not dangling or is in D,
 (c) choose $\rightarrow_Z$ such that
  - if $n \leq n'$ then $n \rightarrow_Z^* n'$, and
  - if $o(X) \leq n$ then $e_X \rightarrow_Z^* n$,

 (d) define
  - $\alpha_Z(e_N) = nonce(N)$,
  - $\alpha_Z(e_K) = key(K)$,
  - $\alpha_Z(n) = spy(M)$, if n is a productive receiving or dangling honest node whose unsigned label is M,
  - $\alpha_Z(n) = send(M, A')$, if n is a sending interface node labeled with $+M$ and such that $A' \in Name_A$ if n is linked by $\rightarrow$ to an A strand,

  and
 (e) define $\pi_Z(\emptyset) = \{knows(M) \mid M \in synth(analz(S_Z))\}$, where S_Z includes all principal names and keys from $\mathcal{K}_Z$ that occur in productive intruder nodes.
5. For the channel:
 (a) let Ev_{Ch} consist of all honest events labeled with $send/rec$ actions plus all events of Z labeled with $send/spy$ actions,
 (b) define $\rightarrow_{Ch}$ to be compatible with $\leq$, by taking the node corresponding to each honest $\langle u, i\rangle$ instead of $\langle u, i\rangle$ itself, and such that $\langle u, i\rangle \rightarrow_{Ch}^* n$ if $\langle u, i\rangle \in Ev_A$ with label $send(M, B')$ corresponds to a dangling node n,
 (c) define



Here $nonce_1 = nonce(N_1)$, $M_1 = \{N_1; A\}_{K_Z}$, $M_1' = \{N_1; A\}_{K_B}$, $nonce_2 = nonce(N_2)$, and $M_2 = \{N_1; N_2\}_{K_A}$.

FIG. 14. Part of the DTPL model and configuration obtained from Fig. 9.

- $\alpha_{Ch}(e) = in(M, A')$, if e is already labeled $send(M, A')$,
- $\alpha_{Ch}(e) = out(M, A)$, if $e \in Ev_A$ is already labeled $rec(M)$, and
- $\alpha_{Ch}(e) = leak$ if $e \in Ev_Z$ is already labeled $spy(M)$ for some message M, and

(d) $\pi_{Ch}(\emptyset) = \emptyset$.

To illustrate the construction of DTPL models and configurations from protocol bundles, we shall consider as a starting point the bundle shown in Fig. 9. For simplicity, we exclude the last nodes of both A's and B's strands, as well as the (unspecified) second step of the intruder. Part of the resulting DTPL model and configuration are given in Fig. 14.

The following result holds independently of the choices made in the construction:

THEOREM 3.4
Let $(\mu, \xi) = b2mc(\mathcal{C})$. μ is a DTPL model and ξ is a configuration of μ equivalent to $\mathcal{C}$.

First, note that linearizing the whole bundle order in step 1 is necessary to guarantee that the local orders subsequently chosen for each principal and the channel are compatible with each other. In this way, we can choose each local $\rightarrow_A$ in step 3 to be compatible with $\leq$ by just inserting the internal actions *nonce*/*key* in a suitable order with respect to the definition of the runs introduced in step 2. The two conditions on $\rightarrow_A$ stated in step 3(b) guarantee precisely this: the first ensures that the order of actions in each run is kept, and the second that the total order $\leq$ is not violated. A similar situation happens in step 4. Step 5 is slightly different due to the fact that a dangling honest node labeled with $+M$ may correspond to an intruder *spy* event in addition to an the honest *send* event. In the case of a *spy* event, we just introduce the corresponding channel event *leak* sometime after the corresponding *in* event, compatibly with $\leq$. Overall, these conditions guarantee that we have a distributed life-cycle. The fact that it meets the specification is straightforward, once we assume that the protocol is executable.

Given $(\mu, \xi) = b2mc(\mathcal{C})$, the global configuration ξ required is precisely the set of all events of μ. Note however that, as in the previous translation, the construction defines not just a DTPL model, but rather a collection of models. The possible choices now reflect the information that is absent in bundles but needs to be made explicit in DTPL models. Namely, the total order chosen in step 1 affects the whole construction. But there is also a choice of the relative ordering of *nonce*/*key* actions, both for honest principals and for the intruder, in steps 3(b) and 4(c). In the channel, we also have a choice of how to order spying actions corresponding to dangling honest nodes at step 5(b). Moreover, we also have a choice in step 4(d) in the case that the node is not connected. This highlights the fact that the strand-space approach does not make explicit any information about the destination of a sent message. Step 4(a), in turn, fixes a classification of dangling intruder nodes as sending actions or just internal activity.

The property of honest behavior follows straightforwardly from steps 2 and 3. The same intruder output is guaranteed by steps 4(a) and 4(d). The same intruder knowledge can be shown by noting that at ξ of $b2mc(\mathcal{C})$ the intruder has precisely all messages that he can build using his initial information, plus the messages he gained through spying, and the nonces and keys he generated. The result then follows by observing that all this information must be gathered in dangling or productive nodes of $\mathcal{C}$, according to steps 4(e), 4(d), and again 4(d), respectively. It is clear that further information at productive nodes must be built from the information gathered in preceding interface receiving nodes, or $\mathbf{M}/\mathbf{K}$ nodes with uniquely originating data.

3.4 Analysis of the translations

Above we have shown that it is possible to start from a bundle model of a given protocol and build a collection of equivalent DTPL models and configurations, and vice-versa. We can analyze additional properties of the two translations, characterizing, in particular, what happens when we do a “full loop”. That is, what is the relationship between a given bundle $\mathcal{C}$ and $\mathcal{C}' = mc2b(b2mc(\mathcal{C}))$, and between a given DTPL model and configuration, μ and ξ, and $(\mu', \xi') = b2mc(mc2b(\mu, \xi))$?

Let us start with the first question. In general, independently of the choices made during each of the constructions, the equivalences proved guarantee that both $\mathcal{C}$ and $\mathcal{C}'$ have exactly the same honest (prefixes of) strands. This in turn guarantees that they are equivalent bundles, in the sense introduced in [24], and that the two bundles satisfy precisely the same security goals. Of course, due to the different possibilities for producing messages and obtaining information, the activity of the intruder may be quite different. However, in general, the equivalences also guarantee that the overall knowledge of the intruder, given the data he manipulated or has available in dangling nodes, is precisely the same in the two bundles. Intruder output may however increase from $\mathcal{C}$ to $\mathcal{C}'$ since the translations will create dangling nodes for all the resources available to him.

Concerning the second question, let us analyze the equivalences guaranteed by the translations. In general, μ' will consist of an interleaving of prefixes of exactly the same honest runs that are interleaved in μ. However, the choices that occur along the translations may lead to a different interleaving, or even to considering in μ' shorter prefixes for certain runs than those considered in μ. The latter reflects the fact that honest *nonce*/*key* actions are not represented in bundles, unless through the future use of the generated data. Among the possible choices, however, it is possible to obtain

μ' with exactly the same prefixes and interleaved in the same way. In this case, it is an easy exercise to check that all the DTPL formulas local to honest principals that are needed to express security goals will hold equally at μ and at μ'. With respect to the intruder, μ' features an intruder who will be more eager in spying messages (and thus no longer bothers receiving them), and whose sending activity can vary, with respect to those messages that are never received by any principal. Again, however, it is possible to choose to send precisely the same messages. In any case, the intruder of μ' is guaranteed to possess precisely the same information at ξ' that was held by the intruder of μ at ξ. Thus, ξ and ξ' are also equivalent with respect to all the DTPL intruder formulas that may appear in a security goal. This allows us to conclude that security goals expressed in DTPL are preserved and reflected by the translations.

Although a detailed analysis of different security goals and how they can be expressed in either formalism is out of the scope of this paper, we note here that the DTPL approach has the advantage of using an expressive formal language. Moreover, the formulation of security goals in the strand-space approach suffers from a bias towards the strict separation of honest principals, on the one hand, and the intruder, on the other. Secrecy, for instance, requires that certain data cannot be known by the intruder. In DTPL, however, the formulation of secrecy guarantees that the very same data cannot be known by any principal that has not participated in the execution of the protocol, which can of course include honest principals. In the end, the protocols that guarantee any of the two formulations are exactly the same. For this purpose, if a secret item can fall in the hands of an honest principal who did not participate in the execution of the protocol, then it can also fall in the hands of the intruder, if the intruder acts honestly and replaces that honest agent in another execution. Still, if we stick to the DTPL formulation, we can at least recognize as an attack on the secrecy property the model where the wrong honest principal got the secret instead of the intruder. We cannot do this using the intruder-centered strand-space formalism.

The case of authentication is even more interesting since the principal being authenticated must be honest in the strand-space formalism, but not in DTPL. Certainly it is unrealistic to assume that honest principals have any a priori knowledge about who the intruder is, and moreover no one can prevent the intruder from engaging in whatever protocol execution he decides to start, or is solicited to respond to. Therefore, the DTPL formulation of authentication is strictly stronger than the one used in the strand-space formalism. Indeed, it is possible that an attack on a DTPL authentication property happens in a model where an honest principal has run the protocol with the intruder without knowing it and fails to authenticate him, for instance because the intruder tricked someone else into producing relevant authentication messages. Such a model will not be an attack on the property if we consider the strand-space formalism, although it might constitute an attack on another property since the intruder may have tricked another principal. In any case, if we exclude this possibility also in the formulation of authentication in DTPL, then the two coincide.

4 Related work and concluding remarks

We have given back and forth translations between bundles over the strand space of a given protocol and configurations of the corresponding DTPL models. Both translations preserve the essential ingredients of protocol execution, despite the substantial differences between the two approaches. We have shown too that both translations preserve and reflect security goals like secrecy and authentication.

Our results show that DTPL and strand-space models are compatible, although they offer different views of protocol executions. As previously remarked, other authors have also formalized the relationship of their protocol analysis approaches with strand spaces (which transitively yields an indirect comparison with our DTPL models). We consider here [10, 15, 16, 25], which are the most prominent of these works. The differences between our work and these lie in the comparison methodology used and the results obtained.

Both [10] and [25] consider variants of the strand-space model, while [16] proposes various extensions. Moreover, both Halpern and Pucella [25] and Crazzolara and Winskel [15, 16] compare their formalisms with strand spaces as a model of general distributed systems as opposed to a formalism for security protocol analysis. For such comparisons, intruder strands play no essential role and are basically neglected. In contrast, our comparison is focused around security protocol analysis, where the role of the intruder is central.

The key issue in [25] in comparing multi-agent systems and strand spaces is how agents are modeled. The view taken there is that multi-agent systems should capture additional information about the knowledge and belief of the agents. This in turn requires a clear notion of an agent participating in a protocol interaction, which multi-agent systems provide as each agent has a state that is shared across all the interactions that the agent performs. In contrast, strand spaces provide no such notion. Halpern and Pucella provide translations from strand spaces to *strand systems*, a subclass of multi-agent systems that they define to capture the intuitions underlying strand spaces. The translations are parameterized by the choice of agents in the strand space (via an assignment from strands to agents). In our comparison, a similar situation arises. We have guided the choice of agents in the strand, however, based on the underlying protocol, as explained in Section 3.1.

In [10], Cervesato et al. compare strand spaces with multiset rewriting with existential quantification, which provides a precise way of specifying protocols with a bounded initialization phase but allowing arbitrarily many instances of each protocol role. A number of modifications are made in each setting to obtain a meaningful equivalence between the models. The authors extend the strand formalism with a means of incrementally constructing bundles in order to emulate an execution of a protocol with parametric strands. They omit the initialization part of the multiset rewriting setting, which formalizes the choice of initial data, such as shared public or private keys, and which has no counterpart in the strand-space setting. The correspondence between the modified formalisms directly relates the intruder theory from the multiset rewriting formalism to the intruder strands. In contrast to our work, their comparison has an operational flavor, where the notion of step that they define in their models simulates the one they define on bundles (similar one-step transitions are also considered in [25]). We have not adopted this operational view, but we could achieve something similar using the notion of an *immediately successive configuration*, which would relate any configuration to all those obtained by adding one further event.

In [15], Crazzolara and Winskel introduce the security protocol language SPL and study its semantics. They remark that events and their causal dependencies underlie both strand spaces and the inductive method, but that neither of these approaches builds up the events of a protocol in a compositional way and hence there is an informal jump from the protocol to its model. By broadening the models to certain kinds of Petri nets (a restricted form of contextual nets), they give a compositional, event-based semantics for a simple, but expressive, language for describing protocols. The net semantics is formally related to a transition semantics, strand spaces and inductive rules, as well as trace languages. Event structures are dealt with in more detail in [16], where extensions of strand spaces are given that are designed to address compositionality issues.

Overall, the approach of [16] is probably the closest to our own work, even though it relies on a notion of bundle equivalence that is too strict for our purposes as we mentioned before. Nevertheless, we anticipate that we can profit from some of their ideas. For example, strand spaces with a built-in notion of conflict (also mentioned in [25]) can be adapted for DTPL models once we consider the full-fledged extraction of protocol models from Alice&Bob-style specifications [6, 7, 8, 9]. Strand spaces with conflict would also enable one to correctly model freshness of data in strand spaces, by excluding from bundles the simultaneous occurrence of honest strands that use intersecting sets of "fresh" nonces or keys. In this way, the properties of the translations would still be valid for DTPL models with additional axioms for modeling freshness (cf. Section 2.2). Moreover, [16] also introduces a compositionality mechanism that relies on the notion of an *open bundle*, which is essentially the same as our notion of an S-bundle. In fact, we have recently begun investigating the application of DTPL for reasoning about protocol composition.

We have also started defining a deduction system for DTPL. The system will capitalize on the combined nature of the logic, in the sense that the resulting global deduction system will glue together the different linear temporal deduction systems that are local to each agent. Such a deduction system will complement the semantic-based analysis that can be carried out both in strand spaces and in DTPL models.

Further work in progress is the application of DTPL to investigate general metatheoretic properties of underlying protocol models and model simplification techniques. Although these results are independent of our investigation of the interplay between DTPL and strand spaces, they also point to trade-offs in the formalisms. For example, in [8], we have used DTPL to prove a general lemma about *secret data* that is similar to the protocol-independent secrecy results of [14, 30], which capitalize on the notion of honest ideals on strand spaces introduced in [35]. We also believe that it will be interesting to use DTPL in the context of the authentication tests proposed in [22, 24], or the protocol compositionality results stemming from the disjoint encryption theorems of [22, 23]. The combination of results, techniques and tools from these and other approaches will play an important role in the consolidation of the security protocol analysis discipline.

Clearly, on the one hand, results like the normal forms for intruder strands in protocol bundles obtained in [24] will not have a DTPL counterpart. On the other hand, DTPL does allow one to reason about the number of intruders or the role of the communication channel; for instance, in [8] we also prove the soundness and completeness, with respect to typical security goals, of two further model-simplification techniques: *one intruder is enough*, in the lines of [13], and the *predatory-intruder*, a bound on the behavior of the intruder that goes in the direction of the trace models used in practice, e.g. [32]. Similar kinds of analysis are either impossible or quite artificial in the strand-space approach.

To conclude, we observe that while similar model-simplification techniques, mutatis mutandis, have already been shown for other protocol analysis formalisms, DTPL provides a means for proving them in a general and uniform way, within the same formalism, and provides a basis for further general investigations. In fact, our formalization has also allowed us to clarify aspects of these simplification properties (e.g. concerning principals' identities and the way security properties are established) that are often neglected or cannot be specified in strand spaces and other approaches.

References

[1] Alessandro Armando, Luca Compagna, and Pierre Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In Keijiro Araki, Stefania Gnesi, and Dion Mandrioli, editors, *Proceedings of the 12th International Symposium of Formal Methods Europe (FME'03)*, LNCS 2805, pages 875–893. Springer-Verlag, 2003.

[2] David Basin, Sebastian Mödersheim, and Luca Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[3] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis. In Roberto Gorrieri, editor, *Proceedings of WITS'03*, pages 21–31, 2003.

[4] Stefano Bistarelli, Iliano Cervesato, Gabriele Lenzini, and Fabio Martinelli. Relating Multiset Rewriting and Process Algebras for Security Protocol Analysis. *Journal of Computer Security*, 13(1):3–47, 2005.

[5] Chiara Bodei, Pierpaolo Degano, Riccardo Focardi, and Corrado Priami. Primitives for authentication in process algebras. *Theoretical Computer Science*, 283(2), 2002.

[6] Carlos Caleiro, Luca Viganò, and David Basin. Distributed Temporal Logic for Security Protocol Analysis. In preparation.

[7] Carlos Caleiro, Luca Viganò, and David Basin. Towards a metalogic for security protocol analysis. In Walter A. Carnielli, F. Miguel Dionísio, and Paulo Mateus, editors, *Proceedings of the Workshop on the Combination of Logics: Theory and Applications (Comblog'04)*, pages 187–196. Instituto Superior Técnico, Lisbon, 2004.

[8] Carlos Caleiro, Luca Viganò, and David Basin. Metareasoning about Security Protocols using Distributed Temporal Logic. *Electronic Notes in Theoretical Computer Science 125(1):67–89 (Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis, ARSPA 2004)*, 2005.

[9] Carlos Caleiro, Luca Viganò, and David Basin. Deconstructing Alice and Bob. *Electronic Notes in Theoretical Computer Science 135(1):3–22 (Proceedings of the Workshop on Automated Reasoning for Security Protocol Analysis, ARSPA 2005)*, 2005.

[10] Iliano Cervesato, Nancy A. Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis. In *Proceedings of ISSS 2002*, LNCS 2609, pages 356–383. Springer-Verlag, 2003.

[11] Iliano Cervesato and Paul F. Syverson. The logic of authentication protocols. In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design – Tutorial Lectures*, LNCS 2171, pages 63–136. Springer-Verlag, 2001.

[12] Yannick Chevalier and Laurent Vigneron. Automated Unbounded Verification of Security Protocols. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.

[13] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In Pierpaolo Degano, editor, *Proceedings of the 12th European Symposium on Programming (ESOP'2003)*, LNCS 2618, pages 99–113. Springer-Verlag, 2003.

[14] Véronique Cortier, Jonathan K. Millen, and Harald Rueß. Proving secrecy is easy enough. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, 2001.

[15] Federico Crazzolara and Glynn Winskel. Events in security protocols. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 96–105. ACM Press, 2001.

[16] Federico Crazzolara and Glynn Winskel. Composing strand spaces. In Manindra Agrawal and Anil Seth, editors, *Proceedings of the 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FST TCS 2002)*, LNCS 2556, pages 97–108. Springer-Verlag, 2002.

[17] Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[18] Ben Donovan, Paul Norris, and Gavin Lowe. Analyzing a library of security protocols using Casper and FDR. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[19] Hans-Dieter Ehrich and Carlos Caleiro. Specifying communication in distributed information systems. *Acta Informatica*, 36:591–616, 2000.

[20] Riccardo Focardi and Roberto Gorrieri. Classification of Security Properties (Part I: Information Flow). In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design – Tutorial Lectures*, LNCS 2171, pages 331–396. Springer-Verlag, 2001.

[21] Riccardo Focardi, Roberto Gorrieri, and Fabio Martinelli. Classification of Security Properties (Part II: Network Security). In Riccardo Focardi and Roberto Gorrieri, editors, *Foundations of Security Analysis and Design II – Tutorial Lectures*, LNCS 2946, pages 139–185. Springer-Verlag, 2004.

[22] Joshua D. Guttman. Authentication tests and disjoint encryption: A design method for security protocols. *Journal of Computer Security*, 12(3/4):409–434, 2004.

[23] Joshua D. Guttman and F. Javier Thayer Fábrega. Protocol independence through disjoint encryption. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW'00)*. IEEE Computer Society Press, 2000.

[24] Joshua D. Guttman and F. Javier Thayer Fábrega. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[25] Joseph Y. Halpern and Riccardo Pucella. On the relationship between strand spaces and multi-agent systems. *ACM Transactions on Information and System Security*, 6(1):43–70, 2003.

[26] Gavin Lowe. Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In Tiziana Margaria and Bernhard Steffen, editors, *Proceedings of the Second International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[27] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*. IEEE Computer Society Press, 1997.

[28] Gavin Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998.

[29] Catherine Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, 2003.

[30] Jonathan K. Millen and Harald Rueß. Protocol-independent secrecy. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2000.

[31] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS'01)*, pages 166–175. ACM Press, 2001.

[32] Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.

[33] Peter Ryan, Steve Schneider, Michael Goldsmith, Gavin Lowe, and Bill Roscoe. *Modelling and Analysis of Security Protocols*. Addison Wesley, 2000.

[34] Dawn Song, Sergey Berezin, and Adrian Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

[35] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Honest ideals on strand spaces. In *Proceedings of the 11th IEEE Computer Security Foundations Workshop (CSFW'98)*, pages 66–78. IEEE Computer Society Press, 1998.

[36] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.

[37] Glynn Winskel. Event structures. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency*, LNCS 255, pages 325–392. Springer-Verlag, 1987.

Received submission date