Actor-Network Procedures (Extended Abstract)

Dusko Pavlovic[1] and Catherine Meadows[2⋆]

[1] Royal Holloway, University of London, and Universiteit Twente, EWI/DIES,
dusko.pavlovic@rhul.ac.uk
[2] Naval Research Laboratory, Code 5543, Washington, DC 20375
catherine.meadows@nrl.navy.mil

Abstract. In this paper we propose *actor-networks* as a formal model of computation in heterogenous networks of computers, humans and their devices, where these new procedures run; and we introduce *Procedure Derivation Logic* (PDL) as a framework for reasoning about security in actor-networks, as an extension of our previous *Protocol Derivation Logic*. Both formalisms are geared towards graphic reasoning. We illustrate its workings by analysing a popular form of two-factor authentication.

1 Introduction

Over the last few years, almost without being aware of it, we have been seeing a marked change in our view computation and networking is. We are moving beyond networks of computers communication beyond channels constructed out of wires and routers to networks of human beings and various types of devices various types of devices communication over multiple channels: wired, wireless, cellular, as well as human-usable channels based on voice and vision.

This change has been particularly relevant to security, and to the development and analysis of security protocols. Over the past twenty years or so, there has been extensive, and often influential work on the development of formal methods for the analysis of security protocols. One of the secrets of the success of this work is that it has been based on a simple but powerful model, first introduced in the late thirty years ago by Dolev and Yao [14], in which abstract principals communicate across a network controlled by a hostile intruder. This model has made it possible to develop both model checkers for determining whether or not attacks are possible, and logical systems for determining what a principal can conclude as a result of participating in a protocol. However, this network model is harder to apply in a heterogeneous networks using multiple types of channels.

Our goal is to contribute towards a formal framework for for *reliable* and *practical* reasoning about security of computation and communication in networks.engineering. Towards this goal, we draw our formal models from the informal reasoning practices, and attempt to make them mathematically precise,

⋆ Supported by ONR

while trying to keep them as succinct and intuitive as possible. The main feature of our formalism is that it provides support for *diagrammatically* based security proofs, which we illustrate by examples in this paper. Our approach is intended to capture what we consider the two most salient concepts for security in this new paradigm for computing: the coalitions that are formed between humans and devices in order to enable secure computation and communication in these emerging networks, which we describe via the use of *actor-networks*, a concept borrowed from sociology, and the orchestration of different types of communication along different types of channels, which we describe by generalizing the notion of protocol to that of a *procedure*. These ideas are describe in more detail below.

Actor-networks. Networks have become an immensely popular model of computation across sciences, from physics and biology, to sociology and computer science [15, 33, 31]. Actor-networks [25] are a particularly influential paradigm in sociology, emphasizing and analyzing the ways in which the interactions between people and objects, as equal factors, drive social processes, in the sense that most people cannot fly without an airplane; but that most airplanes also cannot fly without people. Our goal in the present paper is to formalize and analyze some security processes in networks of people, computers, and the ever expanding range of devices and objects used for communication and networking, blurring many boundaries. The idea that people, computers, and objects are equal actors in such networks imposed itself on us, through the need for a usable formal model, even before we had heard of the sociological actor-network theory. After we heard of it, we took the liberty of adopting the name actor-network for a crucial component of our mathematical model, since it conveniently captures many relevant ideas. While the originators of actor-network theory never proposed a formal model, we believe that the tasks, methods and logics that we propose are not alien to the spirit of their theory. In fact, we contend that computation and society have pervaded each other to the point where computer science and social sciences already share their subject.

Procedures. In computer programs, frequently used sequences of operations are encapsulated into *procedures*, also called *routines*. A procedure can be called from any point in the program, and thus supports reuse of code.

In computer networks, frequently used sequences of operations are specified and implemented as *network protocols*, or as *cryptographic protocols*. So protocols are, in a sense, network procedures. Beyond computer networks, there are now hybrid networks, where besides computers with their end-to-end links, there may be diverse devices, with their heterogenous communication channels, cellular, short range etc. Online banking and other services are nowadays usually secured by two-factor and multi-factor authentication, combining passwords with smart cards, or cell phones. A vast area of *multi-channel* and *out-of-band* protocols opens up, together with the web service *choreographies* and *orchestrations*; and we have only scratched its surface. And then there are of course

also social networks, where people congregate with their phones, their cameras and their smiling faces, and overlay the wide spectrum of their social channels over computer networks and hybrid networks. Many sequences of frequently used operations within these mixed communication structures have evolved. This is what we call *actor-network procedures*.

Outline of the paper. The remainder of the paper is organized as followed. In Sec. 2, we give an overview of related work, in particular the work that has most contributed to our own. Sec. 3 introduces the formal model of actor-networks. Sec 4 explains how actor-networks compute, and introduces the formalisms to represent that computation, all the way to actor-network procedures. Sec. 5 presents Procedure Derivation Logic (PDL) as a method for reasoning about actor-network procedures. In Sec. 6 we provide the first case studies using PDL: we analyze the two-factor authentication in online banking. Sec. 7 contains a discussion of the results and the future work.

2 Related work

In social and computational networks, procedures come in many flavors, and have been studied from many angles. Besides cryptographic protocols, used to secure end-to-end networks, in hybrid networks we increasingly rely on multi-channel protocols [44], including device pairing [24]. In web services, standard procedures come in two flavors: choreographies and orchestrations [41]. There are, of course, also social protocols and social procedures, which were developed and studied first, although not formally modeled. As social networks are increasingly supported by electronic networks, and on the Web, social protocols and cryptographic protocols often blend together. Some researchers have suggested that the notion of protocol should be extended to study such combinations [6, 19, 23]. On the other side, the advent of ubiquitous computing has led to extensive, careful, but largely informal analyses of the problems, e.g., of device pairing, and of security interactions of using multiple channel types [44, 22, 32]. One family of device pairing proposals has been systematically analyzed in the computational model in [45, 34, 26, 27].

There is a substantial and extremely successful body of research on the formal specification and verification of security protocols. These can be roughly be divided into work in the cryptographic model, which directly formalizes cryptographic reasoning, and the symbolic model, which represents data and computations symbolically as terms and operators in a term algebra. The symbolic approaches, in turn, can be divided into two approaches. The first relies on model checking, which is used to implement exhaustive search for attacks. This often comes together with proofs that exploration of a certain finite search space without finding an attack guarantees security. See [4] for a history and survey of model checking security protocols. The second builds on logical systems that are used to derive what a participant in a protocol can conclude after the completion of a run. The earliest logical system for cryptographic protocol analysis was the

Burrows-Abadi-Needham logic [7]. A number of successful tools and logics followed. More recent work that has concentrated applying logical reconstructions includes the Cryptographic Protocol Shape Analyzer (CPSA) [13], the Protocol Composition Logic (described in more detail below), and our own Protocol Derivation Logic, which we use as the basis for logical framework developed in this paper.

It is well understood that model checking, by producing explicit attacks, can be very useful in aiding understanding of a protocol and where it has gone wrong. What seems to be less well appreciated is that the use of logical reconstructions can also give insight, but in a different way, since they give a clearer picture of the assumptions that are necessary for the security of a protocol, as well as the ways in which various pieces of the protocol contribute to its security. When combined with a diagrammatic reasoning, this can give considerable insight into the structure and applicability of a protocol. This is very important, since protocols are often reused and redesigned for different environments, with different security assumptions and types of communication channels. This approach of combining logical reconstruction with diagrammatic reasoning is the one we take in this paper.

Our formal model, as well as the basis for its diagrammatic support, is derived from the *strand space* model [20]. Among its many salient features, the convenient diagrammatic protocol descriptions of strand spaces has been an important reason for their wide acceptance and popularity. It is important to note that the strand space diagrams are not just an intuitive illustration, but that they are formal objects, corresponding to precisely defined components of the theory, while on the other hand closely resembling the informal "arrows-and-messages" protocol depictions, found in almost every research paper and on almost every white board where a protocol is discussed.

Protocol Composition Logic (PCL) was, at least in its early versions [18, 12, 10, 17, 11], an attempt to enrich the strand model with a variable binding and scoping mechanism, making it into a process calculus with a formal handle on data flows, which would thus allow attaching Floyd-Hoare-style annotations to protocol executions, along the lines of [39, 40]. This was necessary for incremental refinement of protocol specifications, and for truly compositional, and thus scalable protocol analyses, which were the ultimate goal of the project. However, less attention was paid to the purely diagrammatic contribution of the strand space model.

Protocol Derivation Logic (PDL) has been an ongoing effort [29, 8, 36, 2, 30, 37] towards a scalable, i.e. incremental protocol formalism, allowing composition and refinement like PCL, but equipped with an intuitive and succinct diagrammatic notation, like strand spaces. The belief that these two requirements can be reconciled is based on the observation that the reasoning of protocol participants is concerned mostly with the order of events in protocol executions.It follows that the protocol executions and their logical annotations both actually describe the same structures, which can be viewed as partially ordered multisets [43], and manipulated within the same diagrammatic language. This has been the guiding

idea of PDL. Several case studies of standard protocols, and the taxonomies of the corresponding protocol suites, have been presented in [29, 8, 36, 2]. An application to a family of distance bounding protocols has been presented in [30]; and an extension supporting the probabilistic reasoning necessary for another such family has been proposed in [37]. In the present paper, we propose the broadest view of PDL so far — which should here be read as *Procedure* Derivation Logic. Since cryptographic protocols are usually construed as the tools of computer security, we use the term *procedure* here to denote a frequently used pattern of operations in a modern network, which may include computers and software agents, but also humans, as well as various kinds of communication devices. Procedure Derivation Logic is thus our attempt to address the need for formal reasoning about the *pervasive* security problems, that arise at the interfaces of cyber space with physical and social spaces, as discussed in [38].

It should be mentioned that the mosaic of protocol logics, which we are thus attempting to expand beyond protocols, is, in a certain sense, conterbalanced by the more homogenous (if not entirely monolithic) world of computational modeling. Following the seminal work in [1], it has become customary to verify that every symbolic model, underlying a protocol logic, is sound when interpreted computationally. Such interpretations have led to many interesting results and useful tools [5, 3, 9]. It should be clear, however, that the standard computational model does not represent any of the physical or social features that we are trying to capture in procedures. Just like the model with the timed channels used in [37], the execution model used in this paper does not have a faithful computational interpretation. This renders the question of its computational soundness meaningless.

3 Actor-network model

We model network computation in terms of (1) computational agents, some of them controlled by various parties, others available as resources, and (2) communication channels between the agents, supporting different types of information flows.

3.1 Formalizing actor-networks

Definition 3.1 *An* actor-network *consists of the following sets: (1)* identities*, or* principals $\mathcal{J} = \{A, B, \ldots\}$*, (2)* nodes $\mathcal{N} = \{M, N, \ldots\}$*, (3)* configurations $\mathcal{P} = \{P, Q, \ldots\}$*, where a configuration can be a finite set of nodes, or a finite set of configurations; (4)* channels $\mathcal{C} = \{f, g, \ldots\}$*, and (5)* channel types $\Theta = \{\tau, \varsigma, \ldots\}$ *given with the structure* $\Theta \xleftarrow{\vartheta} \mathcal{C} \underset{\varrho}{\overset{\delta}{\rightrightarrows}} \mathcal{P} \overset{\copyright}{\rightharpoonup} \mathcal{J}$ *where the partial map* $\copyright : \mathcal{P} \rightharpoonup \mathcal{J}$ *tells which principals control which configurations, the pair of maps* $\delta, \varrho : \mathcal{C} \to \mathcal{P}$ *assign to each channel* f *an* entry δf *and an* exit ϱf*, and the map* $\vartheta : \mathcal{C} \to \Theta$ *assigns to each channel a type.*

An actor *is an element of a configuration.*

Notation. We denote by N_B a node N controlled by the principal $©N = B$. We write $g = (P \xrightarrow{\tau} N_B)$ for a channel g of type $\vartheta g = \tau$, with the entry $\delta g = P$, and with the exit $\varrho g = N$ controlled by $©N = B$. Since there is usually at most one channel of a given type between two given configurations, we usually omit the label g, and write just $P \xrightarrow{\tau} N_B$ to denote this channel.

3.2 Example: an actor-network for two factor authentication

To mitigate phishing attacks, some online banks have rolled out two factor authentication. This means that they do not just verify that the user knows a password, but also something else — which is the second authentication factor. This second factor often requires some additional network resources, besides the internet link between the customer and the bank. This is the first, quite familiar step beyond simple cyber networks.

Some banks authenticate that the user is in possession of her smart card. The underlying actor-network is on Fig. 1. The user *A*lice controls her computer C_A and her smart card S_A. She is also given a portable smart card reader R. She inserts the card in the reader to form the configuration Q. The reader is available to Alice, but any other reader would do as well. Configured into Q, the smart card and the reader verify that Alice knows the PIN, and then generates the login credentials, which Alice copies from R's screen to her computer C_A's keyboard, which forwards it to bank Bob's computer C_B. The details of the authentication procedure will be analyzed later.

In summary, the network thus consists of principals $\mathcal{J} = \{A, B\}$, $\mathcal{N} = \{I_A, C_A, S_A, R, C_B\}$; configurations $\mathcal{P} = \mathcal{N} \cup \{Q\}$, where $Q = \{S_A, R\}$, and the following six channels: (1 and 2) cyber channels $C_A \leftrightarrows C_B$ between *A*lice's and *B*ank's computers, (3) a visual channel $C_A \to I_A$ from *A*lice's computer to her human I_A, (4) a keyboard $I_A \to C_A$ from *A*lice's human to her computer, (5) a visual channel $R \to I_A$ from the smart card reader to *A*lice's human, and (6) a keyboard $I_A \to R$ from *A*lice's human to the card reader.

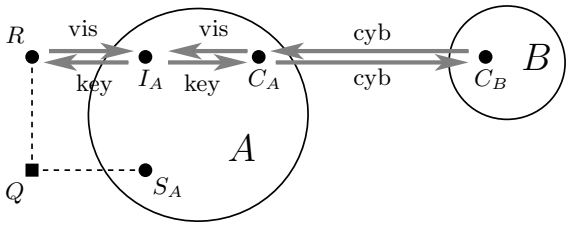

Fig. 1. A pervasive network: Online banking with a smart card reader

4 Actor-network processes

4.1 Computation and communication

Computation in a network consists of *events*, which are localized at nodes or configurations. An event that is controlled by a principal is an *action*.

Communication in a network consists of *information flows* along the channels. Each flow corresponds to a pair of events: (1) a *write* event at the entry of the channel, and (2) a *read* event at the exit of the channel. There are two kinds of flows: (1) *messages*, which consist of a *send action* at the entry of the channel, and a *receive coaction* at the exit, and (2) *sources*, which consist of an *sample action* at the exit, and a *emit coaction* at the entry.

Besides transferring information from one configuration to another, the flows also synchronize the events that take place at different localities, because: every receive coaction must be preceded by a corresponding send action, and every sample action must be preceded by a corresponding emit coaction.

If a source has not been emitted to anywhere, then there is nothing to sample, and no sampling of that source can occur. If a message has not been sent, then the corresponding receive event cannot occur. So when I receive a message, then I know that it must have been sent previously by someone; and when I sample a source, then I know that someone must have emited to this source. That is how I draw conclusions about non-local events from the observations of my own local actions. This is formalized in Sec. 5.2.

4.2 Formalizing data as terms

Each flow carries some data, which contain information. As is standard in the symbolic protocol model, we represent this as terms in an algebra. Recall that an algebraic theory is a pair (O, E), where O is a set of finitary operations (given as symbols with arities), and E a set of well-formed equations (i.e. where each operation has a correct number of arguments) [21].

Definition 4.1 *An algebraic theory* $\mathbb{T} = (O, E)$ *is called a* data theory *if* O *includes a binary pairing* $(-,-)$ *operation, and the unary operations* π_1 *and* π_2 *such that* E *contains the equations* $\pi_1(u, v) = u$, $\pi_2(u, v) = v$, *and* $((x, y), z) = (x, (y, z))$. *A* data algebra *is a polynomial extension* $\mathcal{T}[\mathcal{X}]$ *of a* $\mathbb{T}$*-algebra* $\mathcal{T}$.

Function notation. When no confusion seems likely, we elide the function applications to concatenation, and write $f.x$ instead of $f(x)$.When no confusion is likely, we even elide the dot from the concatenation and simply write fx instead of $f.x$, or $f(x)$.

Random values are represented by indeterminates. A polynomial extension $\mathcal{T}[\mathcal{X}]$ is the free $\mathbb{T}$-algebra generated by adjoining a set of *indeterminates* $\mathcal{X}$ to a $\mathbb{T}$-algebra $\mathcal{T}$ [21, §8]. The elements $x, y, z \ldots$ of $\mathcal{X}$ are used to represent nonces and other randomly generated values.

Easy subterms. We assume that every data algebra comes equipped with the *easy subterm relation* $\sqsubseteq$. The idea is that that $s \sqsubseteq t$ implies that s is a subterm

of t such that every principal who knows t also knows s. In other words, the views Γ_A are lower closed under $\sqsubseteq$, as explained in [36]. This is in contrast with hard subterms, which cannot be extracted: e.g., the plaintext m and the key k are hard subterms of the encryption $E.k.m$.

4.3 Formalizing events and processes

In this section we define processes, the events that processes engage in, and the ordering of events within a process.

An event or action is generally written in the form $a[t]$ where a is the event identifier, and t is the term on which the event may depend. When an event does not depend on data, the term t is taken to be a fixed constant $t = \checkmark$, and we often abbreviate $a[\checkmark]$ to a.

The most important events for our analyses are the action-coaction couples send-receive, and sample-emit, for which we introduce special notations: send $\langle\cdot\, t\,\cdot\rangle$, receive $(\cdot\, t\,\cdot)$, and emit $\langle: t :\rangle$, sample $(: t :)$. Generically, we write $\langle\, t\,\rangle$ for a write action, which can be either $\langle\cdot\, t\,\cdot\rangle$ or $\langle: t :\rangle$, and $(\, t\,)$ for a read action, which can be either $(\cdot\, t\,\cdot)$ or $(: t :)$. Another often used action is $\nu[x]$ for the generation of a random value. It could also be implemented as sampling a source of randomness represented as a devoted node. In addition, the nodes are capable of performing various local operations, which are specified in the definition of the procedure. For actions, such as $\langle\cdot\, t\,\cdot\rangle$ and $(: t :)$, the configuration P must be controlled, i.e. the partial function $\copyright : \mathcal{N} \rightharpoonup \mathcal{J}$ must have a definite value $\copyright P$.

Definition 4.2 *A* process $\mathcal{F}$ *is a partially ordered multiset of localized events, i.e. a mapping*

$$\mathcal{F} \;=\; \langle \mathcal{F}_{\mathbb{E}}, \mathcal{F}_{\mathcal{P}} \rangle : \mathbb{F} \to \mathbb{E} \times \mathcal{P}$$

where $(\mathbb{F}, \to)$ *is a well-founded partial order, representing the structure time,* $\mathbb{E}$ *is a family of events, and* $(\mathcal{P}, \subseteq)$ *the partial order of configurations, and they satisfy the requirements that*

(a) if $\mathcal{F}_{\mathbb{E}}\phi$ *is an action, then* $\copyright(\mathcal{F}_{\mathcal{P}}\phi)$ *is well defined, and*
(b) if $\phi \to \psi$ *in* $\mathbb{F}$ *then* $\mathcal{F}_{\mathcal{P}}\phi \subseteq \mathcal{F}_{\mathcal{P}}\psi$ *or* $\mathcal{F}_{\mathcal{P}}\phi \supseteq \mathcal{F}_{\mathcal{P}}\psi$ *in* $\mathcal{P}$.

Notation: The points in time are denoted by events. By abuse of notation, we usually write $a[t]_P$ for $\phi \in \mathcal{F}$ where $\mathcal{F}_{\mathbb{E}}\phi = a[t]$ and $\mathcal{F}_{\mathcal{P}} = P$.

(a) if an action takes place at a configuration P, then P is controlled, i.e. $\copyright P$ must be well defined, and
(b) if $a[t]_P \to b[s]_Q$ then $P \subseteq Q$ or $P \supseteq Q$.

Definition 4.3 *We say that the term* t originates *at the point* $\phi \in \mathcal{F}$ *if* ϕ *is the earliest write of a term containing* t. *Formally,* ϕ *thus satisfies* $\mathcal{F}_{\mathbb{E}}\phi = \langle\, s\,\rangle$ *where* $t \sqsubseteq s$, *and* $\mathcal{F}_{\mathbb{E}}\xi = \langle\, s\,\rangle \wedge t \sqsubseteq s \Longrightarrow \phi \to \xi$ *holds for all events* ξ.

Notation: Origination. We extend the notational conventions described above by denoting by $\sqrt{\langle\langle\, t\,\rangle\rangle_P}$ the event ϕ where the term t originates. The configuration P is the *originator* of t.

4.4 Formalizing flows, runs and procedures

We now extend our discussion to the definition of communication between processes, and extend our ordering to events occurring within a procedure as well as individual processes.

We begin by defining a more general version of channel between two configurations, called a flow channel. A flow channel exists between any two configurations if a channel exists between any two nodes on the configuration trees. It is called a flow channel because the information passed along the channel flows upwards to the configuration as a whole. It is defined formally below.

Definition 4.4 *For configurations $P, Q \in \mathcal{P}$, a* flow channel $P \xrightarrow{\tau} Q$ *can be either (1) a channel $P \xrightarrow{\tau} Q$, (2) a flow channel $P \xrightarrow{\tau} Q'$, where $Q' \in Q$, (3) a flow channel $P' \xrightarrow{\tau} Q$, where $P' \in P$, or (4) a flow channel $P' \xrightarrow{\tau} Q'$, where $P' \in P$ and $Q' \in Q$.*

A flow $a[t]_P \xrightarrow{\tau} b[s]_Q$ *is given by a flow channel $P \xrightarrow{\tau} Q$, and an interaction pair $a[t], b[s]$, i.e. a pair where either $a[t] = \langle \cdot\, t \,\cdot \rangle$ and $b[s] = (\cdot\, s \,\cdot)$, or $a[t] = \langle : t : \rangle$, and if $b[s] = (: s :)$.*

A flow $a[t]_P \xrightarrow{\tau} b[s]_Q$ is complete *if $s = t$.*

Definition 4.5 *Let $\mathcal{F}$ be a process. A* run*, or* execution $\mathcal{E}^{\mathcal{F}}$ *of $\mathcal{F}$ is an assignment for each coaction $b[s]_Q$ of a unique flow $a[t]_P \xrightarrow{\tau} b[s]_Q$, which is required to be* sound*, in the sense that $b[s]_Q \neq a[t]_P$ in $\mathcal{F}$.*

A run is complete *if all of the flows that it assigns are complete: the terms that are received are just those that were sent, and the inspections find just those terms that were submitted.*

A run is a pomset extending its process. Setting $a[t]_P \to b[s]_Q$ whenever there is a flow $a[t]_P \xrightarrow{\tau} b[s]_Q$ of some type τ makes a run $\mathcal{E}^{\mathcal{F}}$ into an extension of the ordering of the process $\mathcal{E}$, as a partially ordered multiset. The pomset $\mathcal{E}^{\mathcal{F}}$ does not have to satisfy condition (b) of Def. 4.2 any more. Indeed, the whole point of running a process is to extend in $\mathcal{E}^{\mathcal{F}}$ the internal synchronizations, given by the ordering of $\mathcal{F}$, with the additional external synchronizations.

Definition 4.6 *A* network procedure $\mathcal{L}$ *is a pair $\mathcal{L} = \langle \mathcal{F}_{\mathcal{L}}, E_{\mathcal{L}} \rangle$ where $\mathcal{F}_{\mathcal{L}}$ is a process, and $E_{\mathcal{L}} = \{\mathcal{E}_1^{\mathcal{F}_{\mathcal{L}}}, \mathcal{E}_2^{\mathcal{F}_{\mathcal{L}}}, \mathcal{E}_3^{\mathcal{F}_{\mathcal{L}}} \ldots\}$ is a set of runs of $\mathcal{F}_{\mathcal{L}}$. The elements of $E_{\mathcal{L}}$ are called* secure *runs. All other runs are* insecure*. A procedure is said to be secure if every insecure run can be detected by a given logical derivation from the observations of a specified set of participants.*

Graphic presentations of procedures. To specify a procedure $\mathcal{L}$, we draw a picture of the pomset $\mathcal{F} = \mathcal{F}_{\mathcal{L}}$, and then each of its extensions $\mathcal{E} = \mathcal{E}_i^{\mathcal{F}_{\mathcal{L}}}$. Because of condition (b) of Def.4.2, the events comparable within the ordering of a process $\mathcal{F}$ must happen within a maximal configuration. Therefore, if the diagram of the partially ordered multiset $\mathcal{F}$ is drawn together with the underlying network, then each component of the comparable events can all be depicted under the corresponding configuration. We can thus draw the network above the process,

and place the events occurring at each configuration along the imaginary vertical lines flowing, say, downwards from it, like in Fig. 3. The additional ordering, imposed when in a run $\mathcal{E}$ the messages get sent and the facts get observed, usually run across, from configuration to configuration. This ordering can thus be drawn along the imaginary horizontal lines between the events, or parallel with the channels of the network. Such message flows can also be seen in Fig. 3. The dashed lines represent the data sharing within a configuration.

4.5 Examples of procedures

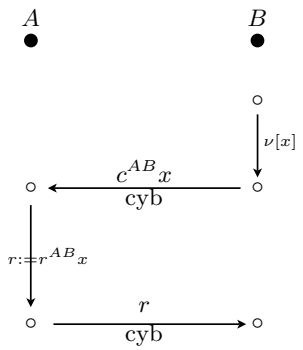

Fig. 2. Challenge-Response (CR) protocol template

Challenge response authentication protocols. We begin a familiar special case of a procedure: a protocol. A large family of challenge-response authentication protocols is subsumed under the template depicted on Fig. 2. Bob wants to make sure that Alice is online. It is assumed that Alice and Bob share a secret k^{AB}, which allows them to define functions c^{AB} and r^{AB} such that

- $r^{AB}x$ can be computed from $c^{AB}x$ using s^{AB}, but
- $r^{AB}x$ can*not* be computed from $c^{AB}x$ alone, without s^{AB}.

So Bob generates a fresh value x, sends the challenge $c^{AB}x$, and if he receives the response $r^{AB}x$ back, he knows that Alice must have been online, because she must have originated the response. The idea behind this template has been discussed, e.g., in [29, 8, 36, 37]. The template instantiates the concrete protocol components by refining the abstract functions c^{AB} and r^{AB} to concrete implementations, which satisfy the above requirements: e.g., c^{AB} may be the encryption by Alice's public key, and r^{AB} may be the encryption by Bob's public key, perhaps with Alice's identity.

Two-factor authentication procedure. Next we describe the first nontrivial procedure, over the actor-network described in Sec. 3.2. It can be viewed as an extension of the simple challenge-response authentication. There, Bob authenticates Alice using her knowledge of a secret s^{AB}, which they both know. Here Bob authenticates that that knows a secret p^A that Bob does not know, and that she has a security token S_A, in this case a smart card. The secret and the smart card are the "two factors". This is the idea of the procedure standardized under the name *Chip Authentication Programme (CAP)*, analyzed in [16]. The desired run of the challenge-response option of this procedure is depicted on Fig. 3.

We assume that, prior to the displayed run, Alice the customer identified herself to Bob the bank, and requested to be authenticated. Bob's computer C_B then extracts a secret s^{AB} that he shares with Alice. This time, though, the shared secret is too long for Alice's human I_A to memorize, so it is is stored in the smart card S_A. Just like in CR protocol above, Bob issues a challenge, such that the response can only be formed using the secret. So Bob in fact authenticates the smart card S_A. He entrusts the smart card S_A with authenticating Alice's human I_A. This is done using the secret p^A shared by I_A and S_A. The secret is stored in both nodes. To form the response to Bob's challenge, Alice forms the configuration Q by inserting her card S_A into the reader R. The configuration Q requests that I_A enters the secret PIN (Personal Identification Number) p^A before it forms the response for Bob. There is no challenge from Q to I_A, and thus no freshness guarantees in this authentication: anyone who sees I_A's response can replay it at any time. Indeed, the human I_A cannot be expected to perform computations to authenticate herself: most of us have trouble even submitting just the static PIN. The solution is thus to have the card-reader configuration Q computes the response, which Alice relays it to Bob. The old PIN authentication is left to just convince Q that Alice's human I_A is there: Q tests p^A, sent through the keybord channel from I_A to the reader R, coincides with $\overline{p}^A$ stored in the card S_A, and then generates a keyed hash $Hs^{AB}x$ using the shared secret s^{AB} and the challenge x. This hash is displayed for Alice on the card reader R as the response r, which Alice then sends to her computer C_A by the keyboard channel, and further to C_B by the cyber channel.

5 Procedure Derivation Logic

5.1 The language of PDL

A statement of PDL is in the form $A : \Phi$, where $A \in \mathcal{J}$ is a principal, and Φ is a predicate asserted by A. The predicate Φ is formed by applying logical connectives to the atomic predicates, which can be (1) $a[t]_P$ — meaning "the event $a[t]_P$ happened", or (2) $a[t]_P \to b[s]_Q$ — meaning "the event $a[t]_P$ happened before $b[s]_Q$".

5.2 Communication axioms

The statements of PDL describe the events that happen in a run of a process, and their order. The basic PDL statements are its axioms, which we describe

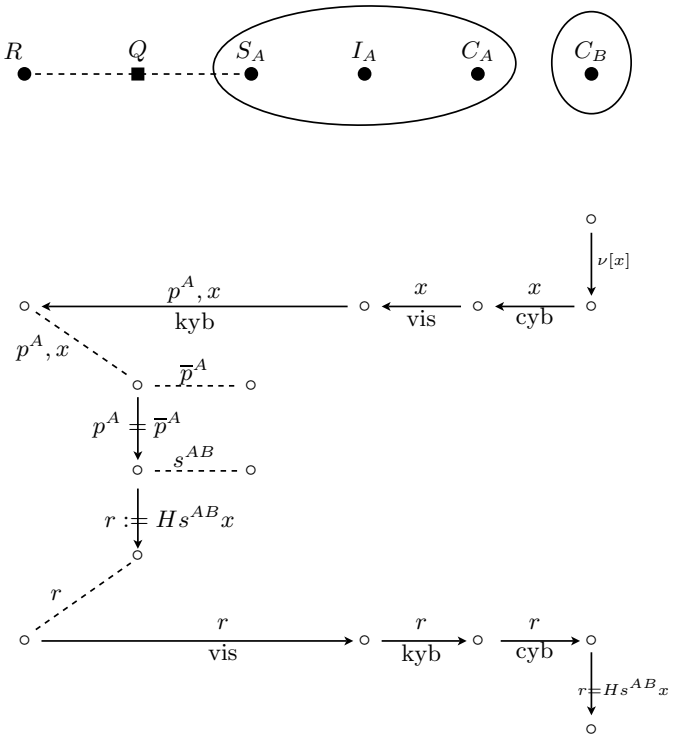

Fig. 3. Chip Authentication Program (CAP) procedure

next. They are taken to be valid in all runs of all processes. The other valid statements are derived from them.

Origination. The origination axioms say that any message that is received must have been sent, and that any source that is sampleed must have been emitted to. This has been explained early in Sec.4. More precisely, any principal that controls a configuration P where a message is received knows that it must have been sent by someone, no later than it was received; and similarly for a source that is sampleed. Formally

$$\copyright P \ : \ (\cdot\, t\, \cdot)_P \Longrightarrow \ \exists X.\ \langle \cdot\, t\, \cdot \rangle_X \to (\cdot\, t\, \cdot)_P \qquad \textsf{(orig.m)}$$
$$\copyright P \ : \ (:\, t\, :)_P \Longrightarrow \ \exists X.\ \langle : t :\rangle_X \to (:\, t\, :)_P \qquad \textsf{(orig.s)}$$

Freshness. In Sec. 4.2 we explained the idea of modeling random values as the indeterminates in polynomial algebras of messages. The freshness axiom extends this idea to processes, by requiring that each indeterminate x must be (1) *freshly generated* by an action $\nu[x]$ before it is used anywhere, and (2) that it can only be used elsewhere after it has passed in a message or a source. which formally becomes

$$\copyright P \ : \ a[t.x]_P \Longrightarrow \exists X.\ \nu[x]_X \to a[t.x]_P \qquad \textsf{(fresh.1)}$$
$$\copyright P \ : \ \neg\nu[x]_P \wedge a[t.x]_P \Longrightarrow \exists X.\ \big(\nu[x]_X \to \langle\langle \cdot\ x\ \cdot\rangle\rangle_X \to ((\cdot\ x\ \cdot))_P \to a[t.x]_P\big)$$
$$\vee\ \big(\nu[x]_X \to \langle\langle :\, x\, :\rangle\rangle_X \to ((:\, x\, :))_P \to a[t.x]_P\big) \qquad \textsf{(fresh.2)}$$

where, using the easy subterm order $\sqsubseteq$ from Sec. 4.2, $\langle\langle \cdot\ x\ \cdot\rangle\rangle_X$ abbreviates $\exists t.\ x \sqsubseteq t\ \wedge\ \langle \cdot\, t\, \cdot\rangle_X$, $((\cdot\ x\ \cdot))_X$ abbreviates $\exists t.\ x \sqsubseteq t\ \wedge\ (\cdot\, t\, \cdot)_X$, etc.

5.3 Authentication axioms

In our model, there are two forms of authentication: interactions along authentic channels, and challenge-response authentication.

Interactions along authentic channels. An authentic channel allows at least one of the participants to observe not only the events on their own end of the channel, but also on the other end. So there are four types of authentic channels, supporting the following assertions:

$$\copyright P \ : \ \langle \cdot\, t\, \cdot\rangle_P \to (\cdot\, t\, \cdot)_Q \ \ \textsf{(auch.m.1)} \qquad \copyright P \ : \ \langle :\, t\, :\rangle_P \to (:\, t\, :)_Q \ \ \textsf{(auch.p.1)}$$
$$\copyright Q \ : \ \langle \cdot\, t\, \cdot\rangle_P \to (\cdot\, t\, \cdot)_Q \ \ \textsf{(auch.m.2)} \qquad \copyright Q \ : \ \langle :\, t\, :\rangle_P \to (:\, t\, :)_Q \ \ \textsf{(auch.p.2)}$$

Channels that satisfy auch.m.1 or auch.p.1 are called *write*-authentic; channels that satisfy auch.m.2 or auch.p.2 are called *read*-authentic. Here are some examples from each family:

– (auch.m.1): A keyboard channel guarantees to the sender that the device at which she is typing is receiving the message
– (auch.m.2): A visual channel used for sending a message allows the receiver to see the sender.
– (auch.p.1): When my fingerprints are taken, I observe that they are taken, and can see who is taking them.
– (auch.p.2): Moreover, the person taking my fingerprints also observes that they are taking my fingerprints.

Besides these assertions about the order of events, some authentic channels support other assertions. They are usually application specific, and we impose them as procedure specific axioms.

Challenge-response authentication. The challenge-response axiom is in the form

$$ⓒP \ : \ \mathsf{Local}_P \Longrightarrow \ \mathsf{Global}_{PQ} \qquad (\mathsf{cr})$$

where, using the notation from Sec. 5.2

$$\begin{aligned} \mathsf{Local}_P &= \nu[x]_P \to \left\langle \cdot\, c^{PQ} x\, \cdot \right\rangle_P \qquad\qquad \to \qquad\qquad \left(\cdot\, r^{PQ} x\, \cdot\right)_P \\ \mathsf{Global}_{PQ} &= \nu[x]_P \to \left\langle \cdot\, c^{PQ} x\, \cdot \right\rangle_P \to \left(\left(\cdot\ c^{PQ} x\ \cdot\right)\right)_Q \to \sqrt{\langle\langle \cdot\ r^{PQ} x\ \cdot \rangle\rangle_Q} \to \left(\cdot\, r^{PQ} x\, \cdot\right)_P \end{aligned}$$

Translated into words, (cr) says that the owner ⓒP of the configuration P knows that (1) if he generates a fresh x, sends the challenge $c^{PQ}x$, and receives the response $r^{PQ}x$, then (2) Q must have received a message containing $c^{PQ}x$ after he sent it, and then she must have sent a message containing $r^{PQ}x$ before he received it.

Using (cr), from certain observations of the local events at P, the principal ⓒP can thus draw the conclusions about certain non-local events at Q, which he cannot directly observe. Fig. 4 shows depicts this reasoning diagrammatically.

Remark. The (cr) axiom, and the corresponding protocol template, displayed on Fig. 9, has been one of the crucial tools of the Protocol Derivation Logic, all the way since [29, 8], through to [37].

6 Examples of reasoning in PDL

6.1 On the diagrammatic method

In its diagrammatic form depicted on Fig. 9, axiom (cr) says that the verifier P, observing the local path on the left, can derive the path around the non-local actions on the right. This pattern of reasoning resembles the categorical practice of *diagram chasing* [28, 35]. Categorical diagrams are succinct encodings of lengthy sequences of equations. Just like the two sides of the implication in (cr) correspond to two paths around Fig. 9, the two sides of an equation are represented in a categorical diagram as two paths around a face of that diagram. The

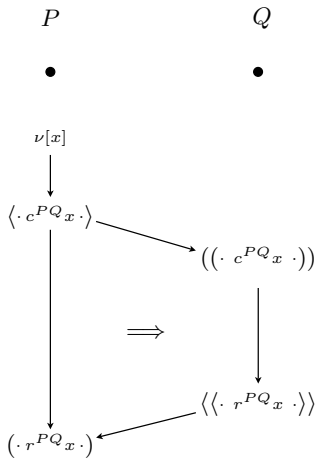

Fig. 4. The graphic view of (cr) axiom

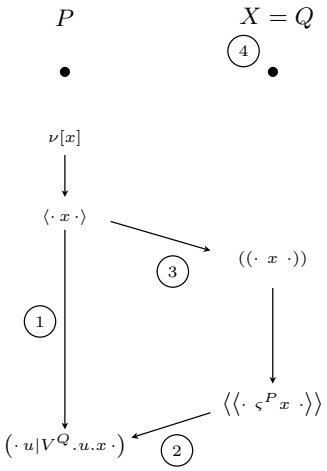

Fig. 5. Challenge-response using signatures

components of the terms in the equations correspond to the individual arrows in the paths. The equations can be formally reconstructed from the diagrams. Moreover, the diagrams can be formally combined into new proofs. The algebraic structures are thus formally transformed into geometric patterns. After some practice, the geometric intuitions begin to guide algebraic constructions in the formal language of diagrams. We apply a similar strategy to PDL.

6.2 Cryptographic (single-factor) authentication

We begin with a very simple example of diagrammatic reasoning, present already in [29].

Theorem 6.1 *The functions $c^{PQ}x = x$ and $r^{PQ}x = \varsigma^Q x$ implement* (cr)*, provided that the abstract signature function ς satisfies the following axioms:*

(a) $\varsigma^Q u = \varsigma^Q v \Longrightarrow u = v$*, i.e., ς^Q is injective,*
(b) $\sqrt{\langle\langle \varsigma^Q t \rangle\rangle_X} \Longrightarrow X = Q$*, i.e., $\varsigma^Q t$ must originate from Q,*
(c) $V^Q.u.t \iff u = \varsigma^Q t$*, i.e., the predicate V^Q is satisfied just for the pairs u, t where $u = \varsigma^Q t$,*

and that these axioms are known to the principal Bob = ©P.

Proof. To prove the claim, we chase the diagram on Fig. 10. The numbered arrows arise from the following steps:

1. Bob = ©P observes $\nu[x]_P \to \langle \cdot\, x \,\cdot \rangle_P \to \big(\cdot\, r | V^Q r x \,\cdot\big)$, i.e. after sending a fresh value x, he receives a response u which passes the verification $V^Q r x$.
2. Using the axioms (c) and (orig.m), he concludes that there is some X such that $\big\langle \cdot\, V^Q x \,\cdot \big\rangle_X \to \big(\cdot\, r | V^Q r x \,\cdot\big)_P$.

3. Using (fresh.2) he further derives that for the same X holds $\langle \cdot\, x \,\cdot \rangle_X \rightarrow ((\cdot\ x\ \cdot))_X \rightarrow \langle \cdot\, V^Q x \,\cdot \rangle_X$.
4. Using (a) and (b), Bob concludes that $V^Q x$ must have originated from Q.

□

6.3 Pervasive (two-factor) authentication

Next we describe how Bob the bank authenticates Alice the customer in the CAP procedure.

Theorem 6.2 *The procedure on Fig. 3 implements authentication, i.e. satisfies (cr), provided that the following assumptions are true, and known to Bob:*

(a) $Hu = Hv \Longrightarrow u = v$*, i.e., H is injective;*
(b) $\sqrt{\langle\langle\, s^{AB} \,\rangle\rangle_X} \Longrightarrow X = S_A \vee X = C_B$*, i.e., s^{AB} must originate from S_A or C_B;*
(c) $\sqrt{\langle\langle\, p^A \,\rangle\rangle_X} \Longrightarrow X = I_A \vee X = S_A$*, i.e., p^A must originate from I_A or S_A;*
(d) $\langle \cdot\, Hs^{AB} x \,\cdot \rangle_Q \implies \Big(\big(\cdot\, p^A, x \,\cdot\big)_Q \rightarrow \langle \cdot\, Hs^{AB} x \,\cdot \rangle_Q \Big) \wedge p^A = \overline{p}^A$ *, i.e., S_A and R are honest.*

Proof. Prior to the displayed execution, Alice is assumed to have sent to Bob her identity, and a request to be authenticated. Following this request, Bob's computer C_B has extracted the secret s^{AB} from a store, which he will use to verify that S_A has generated the response.

To prove the claim, we chase the diagram on Fig. 6. The enumerated steps in the diagram chase correspond to the following steps in Bob's reasoning:

1. Bob observes $\nu[x]_{C_B} \rightarrow \langle \cdot\, x \,\cdot \rangle_{C_B} \rightarrow \big(\cdot\, Hs^{AB} x \,\cdot\big)_{C_B}$.
2. Using (orig.m) he concludes that there is some X such that $\langle \cdot\, Hs_A x \,\cdot \rangle_X \rightarrow \big(\cdot\, Hs^{AB} x \,\cdot\big)_{C_B}$.
3. Using (fresh.2) he further derives that for the same X holds $\langle \cdot\, x \,\cdot \rangle_{C_B} \rightarrow ((\cdot\ x\ \cdot))_X \rightarrow \langle \cdot\, Hs^{AB} x \,\cdot \rangle_X$.
4. By (a) and (b), from the observation that he did not use s^{AB}, Bob concludes that $Hs^{AB} x$ must have originated in a configuration Q containing S_A.
5. By (c), $\big\langle\langle \cdot\ p^A\ \cdot \rangle\big\rangle_{I_A} \rightarrow \big((\cdot\ p^A\ \cdot)\big)_Q \rightarrow \big(p^A = \overline{p}^A\big) \rightarrow \big(Hs^{AB} x\big)$, where the last action abbreviates $\big(r := Hs^{AB} x\big)$, and we write out r as $Hs^{AB} x$ in the rest of the diagram.
6. Since Q had to also receive x before computing the response in $\big(\cdot\, p^A, x \,\cdot\big)_R \rightarrow \big(Hs^{AB} x\big)$ follows by (d). So $\big((\cdot\ p^A\ \cdot)\big)_Q$ from 5 is $\big(\cdot\, p^A, x \,\cdot\big)_R$.
7. By (orig-m), there is Y with $\langle \cdot\, p^A, x \,\cdot \rangle_Y \rightarrow \big(\cdot\, p^A, x \,\cdot\big)_R$. By (e), $\big\langle\langle \cdot\ p^A\ \cdot \rangle\big\rangle_{I_A}$ from 5 must be $\langle \cdot\, p^A, x \,\cdot \rangle_{I_A}$.
8. The fresh value x has thus been sent to Q by I_A. It follows that in 2 and 3 above must be $X = I_A$.

9. Since A controls S_A and I_A, and $S_A \in Q$ generated the response $Hs^{AB}x$, only I_A could have sampled $Hs^{AB}x$ along the visual channel.
10. Since A controls I_A and C_A, only I_A could have sent $Hs^{AB}x$ to C_A along the keyboard channel.

These logical steps suffice to assure Bob that if he observes the local flow on the right in Fig. 6, then the non-local flow along the external boundary, all the way to the left side of the diagram and back, must have taken place. Comparing this diagrammatic conclusion with the pattern of (cr) on Fig. 9, we see that Bob has proven an instance of authentication.

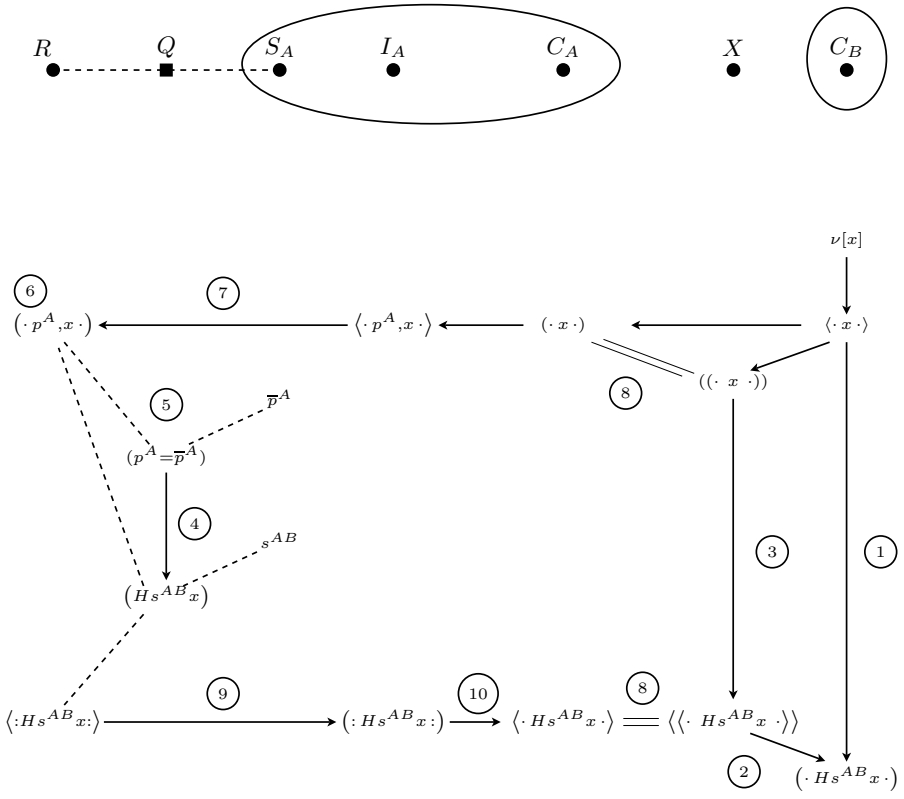

Fig. 6. B's reasoning in CAP

□

We have provided a security proof of the CAP protocol discussed in [16]. However, the discussion in that paper is devoted to pointing out the security

risks inherent in relying upon that very protocol. How can this happen? As it turns out, we can reproduce the situations that the authors in [16] warn against by relaxing the assumptions that our proof relies upon. This is one advantage of combining logical reconstruction with explicit specifications of the configurations and channels involved. We describe this in more detail below.

Relaxing the assumption that Alice is honest: In this case $\sqrt{\langle\langle\, p^A \,\rangle\rangle_X}$ fails to hold, because if Alice is not honest she could turn PIN over to a third party. This is discussed in [16] in terms of the card being stolen and Alice being intimidated into revealing her PIN.

Relaxing the assumption that R is honest: in this case $\sqrt{\langle\langle\, p^A \,\rangle\rangle_X}$ fails to hold again, because S_A or R could reveal the PIN to a third party. This is discussed in [16] in two places. First, the reader could inadvertently reveal the PIN because the keys used to enter it become visibly worn. Secondly, for practical reasons users are often required to use untrusted readers provided by third parties, which could steal the PIN.

Relaxing the assumption that Alice controls C_A: if C_A has been infiltrated by malware then Alice no longer controls it. As the reader can verify, this has absolutely no effect on the proof of security of the CAP and PIN protocol in isolation. Indeed, C_A could be replaced by a cyber channel where ever it is used without affecting the protocol's security. The problem is when the hash computed by the smart card is used to authenticate a bank transaction. The most straightforward way of doing this is for the hash to be passed to C_A, which computes, for example, a Message Authentication Code on the transaction. If C_A is not controlled by Alice, it could substitute a different transaction.

7 Conclusion

We have presented a logical framework for reasoning about security of protocols that make use of a heterogeneous mixture of humans, devices, and channels. We have shown how different properties of channels and configurations can be expressed and reasoned about within this framework. A key feature of this framework is that it supports explicit reasoning about both the structure of a protocol and the contributions made by its various components, using a combination of diagrammatic and logical methods. Because of this, we believe that our approach can be particularly useful in giving a more rigorous foundation for white-board discussions, in which protocols are usually displayed graphically. By annotating the diagram with the proof using the methods demonstrated in this paper, formal reasoning could be brought to bear at the very earliest states of the design process.

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. of Cryptology*, 15(2):103–127, 2002.

2. Matthias Anlauff, Dusko Pavlovic, Richard Waldinger, and Stephen Westfold. Proving authentication properties in the Protocol Derivation Assistant. In Pierpaolo Degano, Ralph Küsters, and Luca Vigano, editors, *Proceedings of FCS-ARSPA 2006*. ACM, 2006.
3. Gilles Barthe, Daniel Hedin, Santiago Zanella Béguelin, Benjamin Grégoire, and Sylvain Heraud. A machine-checked formalization of sigma-protocols. In *CSF*, pages 246–260. IEEE Computer Society, 2010.
4. David Basin, Cas Cremers, and Catherine Meadows. Model checking security protocols. In Edmund Clarke, Tom Henzinger, and Helmut Veith, editors, *Handbook of Model Checking*. Springer, 2011. To appear. Also available at `http://people.inf.ethz.ch/cremersc/publications/index.html`.
5. Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.
6. Matt Blaze. Toward a broader view of security protocols. In Bruce Christianson, Bruno Crispo, James A. Malcolm, and Michael Roe, editors, *Security Protocols Workshop*, volume 3957 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2004.
7. Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, February 1990.
8. Iliano Cervesato, Catherine Meadows, and Dusko Pavlovic. An encapsulated authentication logic for reasoning about key distribution protocols. In Joshua Guttman, editor, *Proceedings of CSFW 2005*, pages 48–61. IEEE, 2005.
9. Véronique Cortier, Steve Kremer, and Bogdan Warinschi. A survey of symbolic methods in computational analysis of cryptographic systems. *J. Autom. Reasoning*, 46(3-4):225–259, 2011.
10. Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. Secure protocol composition. *E. Notes in Theor. Comp. Sci.*, pages 87–114, 2003.
11. Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. A derivation system and compositional logic for security protocols. *J. of Comp. Security*, 13:423–482, 2005.
12. Anupam Datta, Ante Derek, John C. Mitchell, and Dusko Pavlovic. A derivation system for security protocols and its logical formalization. In Dennis Volpano, editor, *Proceedings of CSFW 2003*, pages 109–125. IEEE, 2003.
13. Shaddin F. Doghmi, Joshua D. Guttman, and F. Javier Thayer. Searching for shapes in cryptographic protocols. In Orna Grumberg and Michael Huth, editors, *TACAS*, volume 4424 of *Lecture Notes in Computer Science*, pages 523–537. Springer, 2007.
14. Danny Dolev and Andrew C. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.
15. S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.
16. Saar Drimer, Steven J. Murdoch, and Ross J. Anderson. Optimised to fail: Card readers for online banking. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 184–200. Springer, 2009.
17. Nancy Durgin, John Mitchell, and Dusko Pavlovic. A compositional logic for proving security properties of protocols. *J. of Comp. Security*, 11(4):677–721, 2004.
18. Nancy Durgin, John C. Mitchell, and Dusko Pavlovic. A compositional logic for protocol correctness. In Steve Schneider, editor, *Proceedings of CSFW 2001*, pages 241–255. IEEE, 2001.

19. Carl Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399, October 2007.
20. Javier Thayer Fabrega, Jonathan Herzog, and Joshua Guttman. Strand spaces: What makes a security protocol correct? *Journal of Computer Security*, 7:191–230, 1999.
21. George A. Gratzer. *Universal Algebra*. Van Nostrand Princeton, N.J.,, 1968.
22. Jaap-Henk Hoepman. Ephemeral pairing on anonymous networks. In Dieter Hutter and Markus Ullmann, editors, *SPC*, volume 3450 of *Lecture Notes in Computer Science*, pages 101–116. Springer, 2005.
23. Chris Karlof, J. D. Tygar, and David Wagner. Conditioned-safe ceremonies and a user study of an application to web authentication. In *Proceedings of the 5th Symposium on Usable Privacy and Security*, SOUPS '09, pages 38:1–38:1, New York, NY, USA, 2009. ACM.
24. Arun Kumar, Nitesh Saxena, Gene Tsudik, and Ersin Uzun. A comparative study of secure device pairing methods. *Pervasive Mob. Comput.*, 5:734–749, December 2009.
25. Bruno Latour. *Reassembling the Social: An Introduction to Actor-Network Theory*. Oxford University Press, 2005.
26. Sven Laur and Kaisa Nyberg. Efficient mutual data authentication using manually authenticated strings. In David Pointcheval, Yi Mu, and Kefei Chen, editors, *CANS*, volume 4301 of *Lecture Notes in Computer Science*, pages 90–107. Springer, 2006.
27. Sven Laur and Sylvain Pasini. User-aided data authentication. *IJSN*, 4(1/2):69–86, 2009.
28. Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.
29. Catherine Meadows and Dusko Pavlovic. Deriving, attacking and defending the GDOI protocol. In Peter Ryan, Pierangela Samarati, Dieter Gollmann, and Refik Molva, editors, *Proceedings of ESORICS 2004*, volume 3193 of *Lecture Notes in Computer Science*, pages 53–72. Springer Verlag, 2004.
30. Catherine Meadows, Radha Poovendran, Dusko Pavlovic, LiWu Chang, and Paul Syverson. Distance bounding protocols: authentication logic analysis and collusion attacks. In R. Poovendran, C. Wang, and S. Roy, editors, *Secure Localization and Time Synchronization in Wireless Ad Hoc and Sensor Networks*. Springer Verlag, 2006.
31. Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010.
32. Long Hoang Nguyen and Andrew William Roscoe. Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey. *Journal of Computer Security*, 2011. to appear.
33. Bernhard O. Palsson. *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press, 2006.
34. Sylvain Pasini and Serge Vaudenay. Sas-based authenticated key agreement. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 395–409. Springer, 2006.
35. Dusko Pavlovic. Maps II: Chasing diagrams in categorical proof theory. *J. of the IGPL*, 4(2):1–36, 1996.
36. Dusko Pavlovic and Catherine Meadows. Deriving secrecy properties in key establishment protocols. In Dieter Gollmann and Andrei Sabelfeld, editors, *Proceedings of ESORICS 2006*, volume 4189 of *Lecture Notes in Computer Science*. Springer Verlag, 2006.

37. Dusko Pavlovic and Catherine Meadows. Bayesian authentication: Quantifying security of the Hancke-Kuhn protocol. *E. Notes in Theor. Comp. Sci.*, 265:97 – 122, 2010.
38. Dusko Pavlovic and Catherine Meadows. Deriving ephemeral authentication using channel axioms. In Bruce Christianson, editor, *Proceedings of the Cambridge Workshop on Security Protocols 2009*, Lecture Notes in Computer Science. Springer Verlag, 2010. to appear.
39. Dusko Pavlovic and Douglas R. Smith. Composition and refinement of behavioral specifications. In *Automated Software Engineering 2001. The Sixteenth International Conference on Automated Software Engineering*. IEEE, 2001.
40. Dusko Pavlovic and Douglas R. Smith. Guarded transitions in evolving specifications. In H. Kirchner and C. Ringeissen, editors, *Proceedings of AMAST 2002*, volume 2422 of *Lecture Notes in Computer Science*, pages 411–425. Springer Verlag, 2002.
41. Chris Peltz. Web services orchestration and choreography. *Computer*, 36:46–52, October 2003.
42. Wolter Pieters. Representing humans in system security models: An actor-network approach. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 2(1):75–92, 2011.
43. Vaughan Pratt. Modelling concurrency with partial orders. *Internat. J. Parallel Programming*, 15:33–71, 1987.
44. Frank Stajano, Ford-Long Wong, and Bruce Christianson. Multichannel protocols to prevent relay attacks. In Radu Sion, editor, *Financial Cryptography*, volume 6052 of *Lecture Notes in Computer Science*, pages 4–19. Springer, 2010.
45. Serge Vaudenay. Secure communications over insecure channels based on short authenticated strings. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 309–326. Springer, 2005.