Under consideration for publication in Formal Aspects of Computing

Actor-network procedures: Modeling multi-factor authentication, device pairing, social interactions

Dusko Pavlovic[1] and Catherine Meadows[2]

[1]Royal Holloway University of London and Universiteit Twente; Email: dusko.pavlovic@rhul.ac.uk
[2]Naval Research Laboratory, Washington, DC, USA; Email: meadows@itd.nrl.navy.mil

Abstract. As computation spreads from computers to networks of computers, and migrates into cyberspace, it ceases to be globally programmable, but it remains programmable indirectly and partially: network computations cannot be controlled, but they can be steered by imposing local constraints on network nodes. The tasks of "programming" global behaviors through local constraints belong to the area of *security*. The "program particles" that assure that a system of local interactions leads towards some desired global goals are called *security protocols*. They are the software connectors of modern, world wide software systems.

As computation spreads beyond cyberspace, into physical and social spaces, new security tasks and problems arise. As computer networks are extended by nodes with physical sensors and controllers, including the humans, and interlaced with social networks, the engineering concepts and techniques of computer security blend with the social processes of security, that evolved since the dawn of mankind. These new connectors for computational and social software require a new "discipline of programming" of global behaviors through local constraints. Since the new discipline seems to be emerging from a combination of established models of security protocols with older methods of procedural programming, we use the name *procedures* for these new connectors, that generalize protocols.

In the present 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. On the way, we survey the guiding ideas of *Protocol Derivation Logic* (also PDL) that evolved through our work in security in last 10 years. Both formalisms are geared towards graphic reasoning and, ultimately, tool support. We illustrate their workings by analysing a popular form of two-factor authentication, and a multi-channel device pairing procedure, devised for this occasion.

1. Introduction

1.1. Motivation and background

In [55] we pondered about the "unreasonable ineffectiveness of security engineering", and suggested that one of the main causes was that the widely used methods for pervasive software design were low level. The

Correspondence and offprint requests to: D. Pavlovic and C. Meadows

age	*ancient times*	*middle ages*	*modern times*
platform	computer	operating system	network
applications	Quicksort, compilers	MS Word, Oracle	WWW, botnets
requirements	correctness, termination	liveness, safety	integrity, confidentiality
tools	programming languages	specification languages	scripting languages

Table 1. Paradigms of computation

high level methodologies to specify and design reusable software procedures were not lifted from traditional computer systems to modern network based computation. In Section IV of [55], we provided a sketch of a network computation model that might fill the gap, and be used as a high level tool to specify and analyze network procedures. But the sketch was very crude, and we did not even have space to provide any examples of network procedures. So the final part of that story remained rather obscure. We attempt to rectify that in the present paper.

1.2. Context

1.2.1. The ages of software

In the beginning, engineers built computers, and wrote programs to control computations. The platform of computation was the computer, and it was used to execute algorithms and calculations, allowing people to discover, e.g., fractals, and to invent compilers, that allowed them to write and execute more algorithms and more calculations more efficiently. Then the operating system became the platform of computation, and software was developed on top of it. The era of personal computing and enterprise software broke out. And then the Internet happened, followed by cellular networks, and wireless networks, and ad hoc networks, and mixed networks. Cyber space emerged as the distance-free space of instant, costless communication, where any pair of network nodes is directly connected. Nowadays software is developed to run in cyberspace. The Web is, strictly speaking, just a software system, albeit a formidable one. A botnet is also a software system. As social space blends with cyber space, many social (business, collaborative) processes can be usefully construed as software systems, that ran on social networks as hardware. Many social and computational processes become inextricable. Table 1 gives a crude picture of the paradigm shifts that led to this remarkable situation.

But as every person got connected to a computer, and every computer to a network, and every network to a network of networks, computation became interlaced with communication, and ceased to be programmable. The functioning of the Web and of web applications is not determined by the code in the same sense as in a traditional software system: after all, web applications do include the human users as a part of their runtime. The fusion of social and computational processes in cyber-social space leads to a new type of information processing, where the purposeful program executions at the network nodes are supplemented by spontaneous data-driven evolution of network links. While the network emerges as the new computer, data and metadata become inseparable, and new types of security problems arise.

1.2.2. The ages of software security

In early computer systems, security tasks mainly concerned sharing of the computing resources. In computer networks, security goals expanded to include information protection. Both computer security and information security essentially depend on a clear distinction between the secure areas, and the insecure areas, separated by a security perimeter. Security engineering caters for computer security and for information security by providing the tools to build the security perimeter. In cyber space, the secure areas are separated from the insecure areas by the "walls" of cryptography; and they are connected by the "gates" of cryptographic protocols.

But as networks of computers and devices spread through physical and social spaces, the distinctions between the secure and the insecure areas become blurred. With network computation, the software-hardware distinction acquires a new meaning. In contrast with the purposefully built and programmed electronic computers, the new spontaneously evolving computer-as-a-network includes social networks as a part of its

age	*middle ages*	*modern times*	*postmodern times*
space	computer center	cyber space	cyber-social space
assets	computing resources	information	public and private resources
requirements	availability, authorization	integrity, confidentiality	trust, privacy
tools	locks, tokens, passwords	cryptography, protocols	mining and classification

Table 2. Paradigms of security

hardware, while social processes are becoming a part of its software. To follow these developments, computer science is endorsing themes and tools of social sciences [66, 26], while social sciences are increasingly concerned with computation [67, 7]. The formalism of *actor-networks* arises on this background.

1.3. Goals and ideas

Our goal is to contribute towards a formal framework for for *reliable* and *practical* reasoning about computation and communication in networks. Since network computation involves adversarial behaviors, security stands out as the central concern in network computation. But reliable reasoning about security, even in the familiar end-to-end networks, usually requires complicated models. Hence the tension between the requirements of reliability and of practicality: reliable reasoning requires a precise formal model, but formal models of security tend to be impractically complex. One approach is to mitigate this complexity through automated support. We have been studying this solution for many years [45, 47, 28, 3], and it has been broadly supported in the research community [2, 6, 5, 13]. Another approach is to try to decrease the complexity through model abstraction and refinement, and through search for convenient and intuitive notations. In the present work, we put more emphasis on this second approach, building upon our previous attempts [48, 11, 56] to extend to reasoning about security the incremental modeling methodologies, well established in software engineering. Towards this goal, we draw our formal models from the informal reasoning practices, and attempt to make them mathematically precise, 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. Although they cannot be directly automated, these proofs are as formal and as precise as the proofs in similar diagrammatic formalisms across mathematics, which also cannot be directly automated. After all, very few of the formal proofs presented in mathematics papers and textbooks are "formal enough" to be entered into a theorem prover. The hope is, however, that in the end, the two types of security models, those designed for software tools, and those designed for human consumption, will converge into a theory that will allow automating complex arguments, while resolving some complexities through insightful notations.

An important instrument of user-friendly mathematical formalisms is the "syntactic sugar", where we subsume a whole gamut of notational and graphical abbreviations, conventions and abuses. Although unsound informal reasoning can be a source of many troubles, and the pedagogical emphasis is usually placed squarely against it, sound informal reasoning can be a source of many insights. After all, the formal proofs and constructions are seldom born fully shaped and formalized, but begin their life as insights and ideas. A useful formalism supports such transformations from insights into formal proofs. The soundness of such transformations often depends on a natural selection of notational conventions and abuses. Not entirely unintentionally, our formalisms turn out to be rich in graphic and syntactic sugar, and in sound notational abuses. We begin by explaining some terminological abuses, starting from the first three words of the title.

1.3.1. "Actor-network"

Networks have become an immensely popular model of computation across sciences, from physics and biology, to sociology and computer science [22, 52, 50]. Actor-networks [39] 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.

It should be noted, though, that the goals of this work are completely different from the goals of sociology of actor-networks, and that our actor-network formalism deviates from the original ideas in a substantial way, even by being a formalism. We make no claims or attempts to faithfully interpret any of the actor-network authors; but we remain faithful to the spirit of their endeavor, since they all discourage orthodoxy.

1.3.2. "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. Conceptually, if not technically, protocol analysis can thus be viewed as an extension of the venerable science of program semantics, and of the methods of procedural programming, adapted for the purposes of network computation.

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*.

Conceptually, actor-network procedures extend program procedures from computers to networks; and they furthermore extend network protocols, and multi-channel protocols, and web service choreographies and orchestrations, into social networks.

Technically, actor-network procedures are, of course, immensely more complicated than their conceptual relatives in computer science, because humans use many types of physical resources, communicate through many parallel communication channels, with many levels of encoding interleaved over each other. If we ignore computers and devices, actor-network procedures already capture the main complexities of social life, as actor-network theorists are explaining. They are a sociologists' problem. In any case, they are completely out of reach for computer scientists' protocol models, symbolic, information theoretic, and computational, all designed for simple end-to-end communication. So a computationally minded reader may wonder why bother to bring them up here.

The reason is that the most frequent transactions that we engage with our computers and even among ourselves involve actor-network procedures. Online banking and shopping, as well as the checkout in the supermarket, travel search on the web, as well as the security line on the airport involve actor-network procedures. Time and again, it has been recognized and reconfirmed that most security breaches nowadays occur not through cryptanalysis, and not through buffer overflow, but through various forms of "social engineering" and channel interactions. Yet "social engineering" has remained a marginal note in technical research. An effort to change this may be foolhardy, but it leads to ideas and structures that seem interesting to explore — even independently of their utility.

1.4. Related work

1.4.1. The expanding concept of a protocol

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 [64], including device pairing [37]. In web services, standard procedures

come in two flavors: choreographies and orchestrations [60]. 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 [9, 27, 35]. On the other side, the advent of ubiquitous computing has led to extensive, careful, but largely informal analyses of the problems of device pairing, and of security interactions of using multiple channel types [64, 34, 51]. One family of the device pairing proposals has been systematically analyzed in the computational model in [65, 53, 40, 41].

1.4.2. Protocol logics and graphics

There is a substantial and extremely successful body of research on the formal specification and verification of security protocols. As we have remarked, it is largely geared to supporting sound and efficient mechanisms for specification and verification, while considerably less attention has been paid to approaches that support the user's understanding of the structure of a protocol and how it contributes to its security. There have been some notable exceptions, however. In this section we describe the work in this direction that has contributed to our own efforts.

One of the most successful, and in our opinion most interesting formal methods for reasoning about security protocols are *strand spaces* [29]. Among its many salient features, the convenient diagrammatic protocol descriptions were an important reason for its 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 [25, 18, 16, 24, 17], 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 [58, 59]. 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. Unfortunately, with these extensions, the handy diagrammatic notation of the strand model got lost.

Protocol Derivation Logic (PDL) has been an ongoing effort [48, 11, 56, 3, 49, 57] 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.[1] It follows that the protocol executions and their logical annotations both actually describe the same structures, which can be viewed as partially ordered multisets [62], 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 [48, 11, 56, 3]. An application to a family of distance bounding protocols has been presented in [49]; and an extension supporting the probabilistic reasoning necessary for another such family has been proposed in [57]. In the present paper, we propose the broadest view of PDL so far — which should here be read as *Procedure* Derivation Logic. The underlying process model is enriched by a new network model, to support reasoning about network procedures. The logical annotations are extended accordingly — still geared towards the diagrammatic reasoning, which still seems like a reasonable strategy, since principals' reasoning towards security remains largely concerned with order of actions.

1.4.3. Computational soundness?

Security is an old social process, but it is a relatively new technical problem. As many other new problems, it often looks unreasonably complicated: a couple of lines of a protocol can conceal a subtle problem for many years. This is sometimes mentioned as the characterizing feature of the field of security. A direct

[1] E.g., in order to achieve mutual authentication, each participant of a run must be able to prove that their and their peers' actions in their conversation must have happened exactly in the order prescribed by the protocol: i.e., that the received messages were previously sent as claimed, and vice versa.

analysis leads to convoluted reasoning, often with an exponential explosion of the cases to be considered. As mentioned in Sec. 1.3, this naturally leads to the idea of automated reasoning [36], which we pursued through several different frameworks [45, 46, 3, 28]. As this idea came to be a widely accepted, it led to a convergence of research to a small number of standard formalisms, based on a sharp division between the symbolic and the computational models of security. This division originally started from the empiric observation, that motivated the seminal paper [1], that the extant security research was broadly based on two different models. Since the computational model is more precise, whereas the symbolic model is easier to use, the way to get the best of both worlds is to demonstrate that the proofs in the simpler model remain valid in the more precise model; in other worlds, that the symbolic model was *computationally sound*. This approach led to many important results and useful tools [8, 4, 14]. But the focus on computational soundness led some researchers to begin viewing all formal models of security as approximations of the standard computational model, forgetting that all models are approximations of some real processes. In the ensuing confusion, even the models involving the features that ostensibly go beyond the computational model (such as timed channels [57]) were required to demonstrate their computational soundness. This is, of course, a meaningless requirement, since a model can only be sound or unsound with respect to a model where it can be faithfully interpreted.

Our current effort is again of this type, as the actor-network model includes several features that preclude computational interpretations, and render the question of its computational soundness meaningless. One such feature is the fact that the actors needn't be standard computers: in this paper, we will see networks involving humans, with their free will; but they could also be, e.g. ants, drawing unusual computational powers from their pheromones [21]. Indeed, a framework that attempts to capture some social interactions, as announced in the title of this paper, can not be captured by a purely computational model, and thus can hardly be expected to be computationally sound. On the other hand, even if we accept to simulate all actors by Turing machines, including ants and humans, the resulting actor-network will still not boil down to the computational model, since the diverse communication channels, that can be specified in an actor-network, cannot be reduced to interactions of Turing machines. This is further discussed in Sec. 2.

Undoubtedly, the fact that our model cannot be proven computationally sound, or even given a computational interpretation, can be interpreted as the evidence that we are modeling what cannot be modeled; whereas the fact that our proofs cannot be automated can be viewed as the evidence that they are not completely formal. Both interpretations are true, for some suitable meanings of the words "formal", and "model". Instead of arguing whether these meanings are reasonable or not, we present our formal model. We contend that our actor-network based proofs in Procedure Derivation Logic are as rigorous as the proofs in any of the standard mathematical formalisms; and hopefully somewhat insightful for the reader. In particular, our diagrammatic proofs can be viewed as a formal method similar to *diagram chasing* in category theory [42, 54], from which we drew inspiration.

Outline of the paper

Sec. 2 introduces the formal model of actor-networks. Sec 3 explains how actor-networks compute, and introduces the formalisms to represent that computation, all the way to actor-network procedures. Sec. 4 presents Procedure Derivation Logic (PDL) as a method for reasoning about actor-network procedures. In Sec. 5 we provide the first case studies using PDL: we analyze the two-factor authentication in online banking, and a device pairing procedure combining physical and biometric channels. Sec. 6 contains a discussion of the results and the future work.

2. Actor-network model

2.1. A computer is a network is a computer

The standard model of computation is a Turing machine. It uses a tape as a storage medium. Sometimes additional tapes are used to represent the input and the output interfaces. Probabilistic Turning machines also read some random strings from a devoted tape, whereas oracle Turing machines communicate with the oracle through an additional tape. Last but not least, interactive computation is often modeled using several

Turing machines that interact with each other on joint tapes. Such joint tapes are actually the *communication channels* between the Turing machines.

Interactive Turing machines can be viewed as a computational network, with the machines as nodes, and the joint tapes as links between them. However, pervasive networks that compute in the world around us nowadays include a wide variety of computational agents at their nodes, including humans, and the various devices with different computational powers. Luckily, an abstract view of the nodes suffices for most analyses. We usually just need to specify that the state, i.e. the variables where the node can store its data; and we postulate which computations can be effectively performed by the node, and which computations are unfeasible.

The channels between the nodes can be viewed as a generalization of the joint tapes, shared by Turing machines in their interactions. But while the joint tapes trivially pass information from one machine to the other, nontrivial channels perform nontrivial data transformations. E.g., a pair of nodes (which may or may not be Turing machines) can be connected by a noisy binary channel, flipping each bit with a certain probability while passing it from one node to the other. Another pair of computational agents can be connected by a cyber channel, controlled by an adaptive attacker, who can change and modify the data flows at will. In symbolic protocol analysis, such attackers are usually specified as simple state machines. In cryptography, they are usually modeled as probabilistic polynomial-time Turing machines. In network models, it is convenient and intuitive to view them as processes encapsulated in channels.

In summary, the simplest model of network computation consists of

- computational agents, some of them controlled by various parties, others available as resources; and
- communication channels between the agents, supporting different types of information flows.

2.2. Idea of actor-networks

Actor-networks depict social processes as computations, and computation as a social process. An example of an actor-network is a configuration consisting of a musician and her instrument. Their intended interaction is the music. The process and the result are highly structured, and the network representation helps with the analysis. The network first of all displays the symmetry of this interaction: the musician cannot play without the instrument, and the instrument cannot play without the musician. The fact that some musician's instrument may be a part of her body (e.g., her voice), and that some instrument's musician may be a computer makes the picture only more interesting. A smart card and a smart card reader form another network of this type. But this network is complicated by the need for someone to key in the pin of the card. The network grows still further if the card reader is connected to a bank, and perhaps dispenses money.

When networks involve heterogenous nodes, and heterogenous communication channels, then the diverse computational resources lead to different computational powers. This is where network computation essentially deviates from machine computation, where according to Church's Thesis, all the different machines have the same computational powers. In a network, one computer may be equipped with a camera and may provide a visual channel to a remote user, whereas another computer may be equipped with sensors and controllers, allowing it to stabilize the flight of an aircraft. A smart card can perform some cryptographic operations when inserted in a reader, and other ones in the contactless mode. A musical instrument can produce one type of music with one musician, and something completely different with somebody else. The musician also depends on the instrument. Furthermore, after they are configured with their instruments, the musicians may further configure themselves into a higher-order configuration: an orchestra. And the orchestra can also be viewed as an actor within the configuration of an opera performance...

Such configurations are what we call *actor-networks*. A computational agent who participates in a configuration is an *actor*, in the sense that she plays a particular *role* assigned to it by a particular network *procedure*. As computational networks spread and diversify, it is becoming increasingly important, and increasingly difficult, to assure that procedures provide the desired actor and network behaviors. Towards this goal, we formalize the above intuitions about actor-networks, and build a framework for reasoning about their procedures.

Remark. The hierarchical structure of our actor-network formalism is alien to the spirit and the letter of original actor-network idea from sociology [39]. But it is essential for the goals of our logical analyses, which

are different from the goals of sociological analyses. It may be of interest to explore the relation between the two sets of goals.

2.3. Formalizing actor-networks

Definition 2.1. An *actor-network* consists of the following sets:

- *identities*, or *principals* $\mathcal{J} = \{A, B, \ldots\}$,
- *nodes* $\mathcal{N} = \{M, N, \ldots\}$,
- *configurations* $\mathcal{P} = \{P, Q, \ldots\}$, where a configuration can be
  - a finite set of nodes, or
  - a finite set of configurations;
- *channels* $\mathcal{C} = \{f, g, \ldots\}$, and
- *channel types* $\Theta = \{\tau, \varsigma, \ldots\}$

given with the following structure:

$$\Theta \xleftarrow{\vartheta} \mathcal{C} \underset{\varrho}{\overset{\delta}{\rightrightarrows}} \mathcal{P} \overset{©}{\rightharpoonup} \mathcal{J}$$

where

- the partial map $© : \mathcal{P} \rightharpoonup \mathcal{J}$ 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 $\vartheta : \mathcal{C} \to \Theta$ assigs to each channel a type.

An *actor* is an element of a configuration.

Actors formally. By the above definition, a configuration is thus a tree whose leaves are annotated by network nodes. A configuration is an actor when viewed as an element of another configuration. In other words, an actor is formally a maximal subtree of a configuration tree. It is useful to distinguish the trees that come together to form another tree because this is what they do to perform some action together. For instance, a hand, a pencil and a piece of paper come together as actors to record a thought. A hand itself is a configuration of fingers, which come together to hold the pencil. The pencil is a configuration of graphite and wood. A door, a lock and a key come together to enforce someone's authority over a space. The lock is a configuration of its metal components. The writing configuration and the locking configuration come together to put a novel in a drawer. And all of it is just trees.

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.

2.4. Examples of networks

2.4.1. Cyber networks

Cyber networks are built following the *"end-to-end"* architecture [63]. In our formalism, a cyber network is characterized by the fact that

- $\mathcal{C} = \{\text{cyb}\}$, i.e. there is just one channel type, which we call *cyber* channel. This is are the insecure channel over which cryptographic protocols are usually run.
- $\mathcal{P} = \mathcal{N}$, i.e. the only configurations are the nodes.
- There is a channel $M \to N$ for every pair $M, N \in \mathcal{N}$.

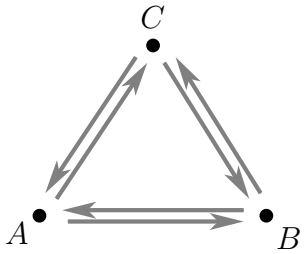

Fig. 1. Cyberspace as completely connected

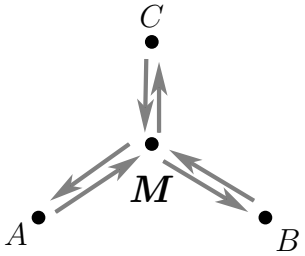

Fig. 2. Cyberspace as completely adversarial

- All communication is done by broadcast from the sender to all nodes in the network. The recipient does not observe the sender directly (although, of course, the sender can identify, or misidentify herself in the message). If a principal controls several nodes, it makes no difference which node he uses to send a message. Without loss of generality, we can thus assume that each principal controls exactly one node, i.e. that $\mathcal{N} = \mathcal{J}$.

The actor-network structure of a cyber network is thus degenerate, and boils down to a single type $\mathcal{P} = \mathcal{N} = \mathcal{J}$, since the only configurations are the nodes, and the nodes are in one-to-one correspondence with the principals. That is why in crypto protocol analysis, we usually just specify how many different principals should play different roles. The fact that any two principals, *viz* network nodes, are directly connected by a completely insecure cyber channel is assumed tacitly. The cyber network with three nodes/principals is presented on Fig 1.

The fact that the cyber channels are insecure can be captured in the model of a cyber network by assuming that all traffic is routed through the attacker, i.e. that *A*lice, *B*ob and *C*arol are all linked with each other through *M*allory. In a sense, the attacker Mallory is the embodyment of cyberspace. This architecture is presented on Fig. 2.

2.4.2. An actor-network for two factor authentication

To mitigate phishing attacks, most 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.

In the simplest case, the bank authenticates the browser used to access the service, by leaving a persistent cookie. The server often also records some data about user's computer and network location. The user only notices this when she tries to access the bank from another location, or using another browser: she is then asked to go through a round of "mother's maiden name" type of challenge questions. A more interesting type of second factor are the single-use Transaction Authentication Numbers (TANs) that the server may generate. Initially, they were be predistributed on paper. Nowadays they are often sent to user's mobile phone in an SMS message when login is initiated. The user is thus authenticated as the owner of her mobile phone. The other way around, the server is authenticated to be in possession of user's phone number, which eliminates the general phishing attacks.

Some banks authenticate that the user is in possession of her smart card. The underlying actor-network is on Fig. 3. 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 $\mathcal{J} = \{A, B\}$,
- nodes $\mathcal{N} = \{I_A, C_A, S_A, R, C_B\}$;
- configurations $\mathcal{P} = \mathcal{N} \cup \{Q\}$, where $Q = \{S_A, R\}$,

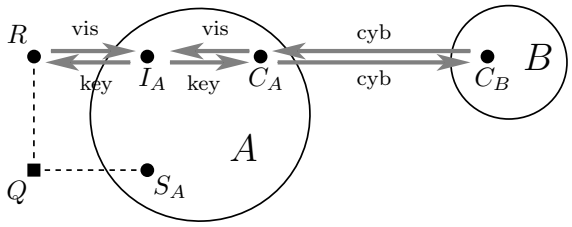

Fig. 3. A pervasive network: Online banking with a smart card reader

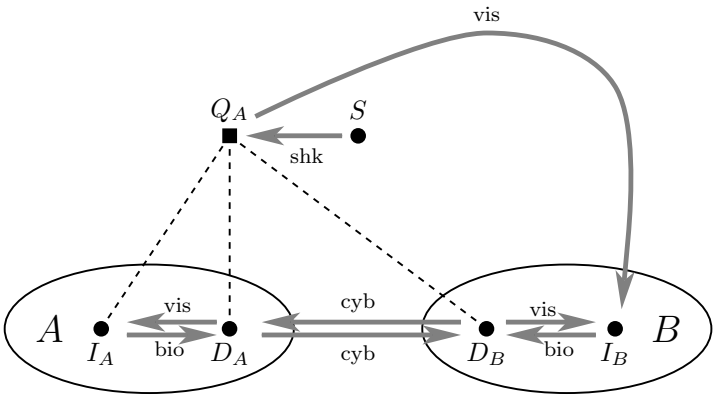

Fig. 4. Actor-network for device handshake

- and the following six channels
  - cyber channels $C_A \leftrightarrows C_B$ between *A*lice's and *B*ank's computers,
  - visual channel $C_A \to I_A$ from *A*lice's computer to her human I_A,
  - keyboard $I_A \to C_A$ from *A*lice's human to her computer,
  - visual channel $R \to I_A$ from the smart card reader to *A*lice's human,
  - keyboard $I_A \to R$ from *A*lice's human to the card reader.

2.4.3. An actor-network for device handshake

Suppose that Alice and Bob have some hand held devices and that they want to pair them, i.e. set up a secure cryptographic channel, without any previous encounters or infrastructure. There is a whole industry of methods to do this. We describe a method inspired by [43] and [10]. Alice's and Bob's devices D_A and D_B are equipped with accelerometers. If a device with an accelerometer is shaken, then the accelerometer can be used as a source of randomness. If two devices are shaken together, their accelerometers will generate roughly similar random strings. A shared random string can be extracted by the techniques described in [43], or using fuzzy extractors [19]. We simply assume that a jointly samplable source is given, and denote id by the node S.

We now describe an actor-network supporting a *device handshake procedure*, where the usual device pairing task is strengthened by the requirement that the secret shared by the devices D_A and D_B is also bound to the identities of their human owners I_A and I_B. Fig. 4 shows an actor-network supporting, in a sense, a half of this task: it will allow Alice to shake two devices together, to extract the shared secret; and *moreover* Alice's device D_A will biometrically verify that it is being shaken by Alice's human I_A, and not by someone else. If the device D_A signals whether this verification succeeds in a way visible to Bob's human I_B, then Bob knows that the key extracted into his device D_B is shared with D_A, *and* bound to the identity of I_A. Alice's human I_A can obtain similar assurances in an analogous round of the same procedure. The network for both rounds is depicted on Fig. 5.

Formally, in the actor-network for one round, depicted on Fig. 4, Alice controls the configuration Q_A,

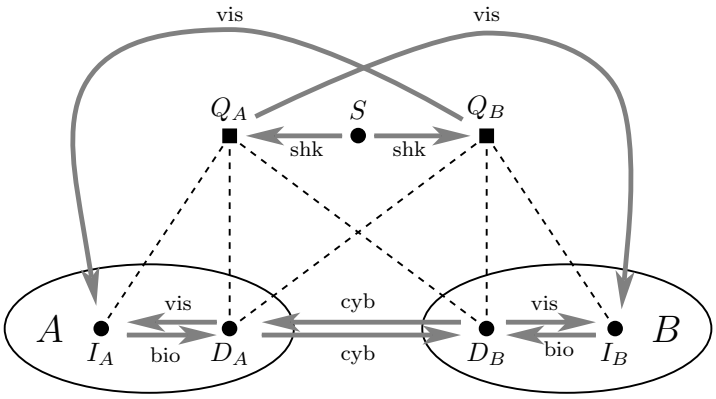

Fig. 5. Actor-network for two-round device handshake

which consists of her device D_A, Bob's device D_B, and Alice's hand I_A, holding the two devices together. Alice's action of shaking the devices is represented as Q_A's action of sampling a source of randomness S along a devoted channel. The accelerometers are abstracted away, and reduced to the node S. The fuzzy extractors are abstracted away and reduced to the fact that the randomness conveyed to Q_A is distributed to both of the actors D_A and D_B participating in it. The details of the procedure are analyzed later. Here we just summarize that the actor-network for device handshake consists of the following data:

- two principals in $\mathcal{J}$, *A*lice and *B*ob,
- five nodes in $\mathcal{N}$:
  - D_A and D_B are Alice's and Bob's devices
  - I_A and I_B are Alice's and Bob's human identities,
  - S is a source of randomness;
- one configuration in $\mathcal{P}$
  - Q_A is the configuration where Alice holds her and Bob's devices in her hand: it consists of I_A, D_A and D_B;
- eight channels in $\mathcal{C}$:
  - two cyber channels $D_A \leftrightarrows D_B$ between *A*lice's and *B*ob's devices;
  - one biometric channel:
    - $I_A \to D_A$ from Alice's hand to her device
  - three visual channels:
    - $D_A \to I_A$ from Alice's device to her eyes,
    - $D_B \to I_B$ from Bob's device to his eyes,
    - $Q_A \to I_B$ from Alice's hand, holding both her and Bob's device to Bob's eyes;
  - one physical channel:
    - $S \to Q_A$ from the source of randomness to Alice's configuration, which conveys the randomness to D_A and D_B.

To assure that each principal contributes to the randomness, the above procedure can be repeated with Bob shaking. The actor-network where both Alice and Bob see each other shaking both devices is depicted on Fig. 5

2.5. Metaphysics of security

2.5.1. Knowing, Having and Being

It is often repeated that security is based on:

- something you *know*: digital keys, passwords, and other secrets,
- something you *have*: physical keys and locks, smart cards, tamper-resistant devices, or
- something you *are*: biometric features, such as fingerprints, eye irises, and other unmodifiable properties of your body; or the capabilities that you cannot convey to others, such as your handwriting.

An identity can thus be determined by its secrets, its tokens, and its features. In our model, this is captured by three levels of control that a principal may have over its nodes:

- *secrets:* what you know can be copied and sent to others,
- *tokens:* what you have cannot be copied, but can be given away, whereas
- *features:* what you are cannot be copied, or given away.

Comments. The common end-to-end security goals are usually realized by means of cryptographic software, and the principals prove their identities by their secrets. In cyber networks, *a principal can be identified with the list of secrets that she knows*. If Alice and Bob share all their secrets, then there is no way to distinguish them by the challenges that can be issued on the standard completely insecure network channels[2]. For all purposes, they must be considered as the same principal.

In pervasive networks, on the other hand, security is also supported by physical security tokens and hardware. Formally, this is where the network model becomes nontrivial: security tokens correspond to network nodes which may be controlled by one principal at one point in time, and by another one at another point.

Finally, security features correspond to network nodes which are controlled by a single principal, and cannot be relinquished. Such nodes correspond to biometric properties. Their counterpart are the nodes that represent *biometric devices*. When all is well, a biometric channel thus has a biometric property at its entry, and a biometric device capable of observing this property at the exit.

2.5.2. What is an identity?

In a cyber network, anyone who knows all my secrets can impersonate me, and the standard models thus assume that an identity is a set of secrets.[3] In a pervasive network, even if someone knows all my secrets, we can still be distinguished as long as only one of us has my smart card, or my fingerprints, or if only one of us is standing at the door. In the actor-network model, besides the secrets, there are also the various tokens and features that a principal may control. An identity is thus a set of actors, which may include tokens and features. Formally, the set of principals $\mathcal{J}$ can be represented

- in a cyber network along the injection

  $$\Gamma : \mathcal{J} \hookrightarrow \wp\mathbb{T}$$

  which assigns to each identity the set of terms that she knows [56]; and
- in an actor-network, along the injection

  $$\begin{aligned} ® \ : \ & \mathcal{J} \hookrightarrow \wp\mathcal{C} \\ & A \mapsto \{P \in \mathcal{C} \mid ©P = A\} \end{aligned}$$

  which assigns to each identity the set of configurations that she controls.

[2] Here we assume that they share the secrets with each other dynamically: the secrets it will immediately be shared with the other. This implies that they also observe the events on the same set of network nodes.

[3] In reality, even in an end-to-end network, two principals with the same set of secrets but, say, different computational powers, can be distinguished by timing their responses. Or they may be distinguished by their histories, since since may have derived their secrets from different initial data, as explained in [55, Sec. IV.D.1]. The standard models, however, abstract away all that.

For simplicity, we assume here that the storage containing the terms from $\Gamma(A)$ is subsumed among the nodes $®A$. More about this in Sec. 3.2.

Why do we use the words "principal" and "identity" as synonyms? For the benefit of the readers with a background in protocol analysis, let us emphasize that *principals do not perform any actions* in the present model. The principals control their actors, and the actors perform the actions, and play roles. A principal determines if and when its actors perform an action, and can thus coordinate the order in which the actions of her actors will be executed. But without the actors, a principal cannot execute any actions. That is why the alternative term *"identity"* may be preferred over *"principal"*. On the other hand, for the special case of protocols, the concept still boils down to the familiar idea of principals, who play their roles in protocols, etc. So we retain that term as well.

Metaphysics of actors and principals. Intuitively, the relation between a principal and her actors in an actor-network can be construed in terms of the *mind-body duality*: the principal is the mind, and the actors are some parts of the body, that the mind can use to observe the world and to act in it. The data received or sampled by some actors through the suitable channels are directly available to the principal: e.g., the principal may control a camera, and observe the visual signal that the camera receives. On the other hand, some other actors may not convey their data to the principal that controls them: the camera may have no cable, or not enough light. The body has its limitations. Which type of information each actor conveys to its principal must be specified by the procedure specific axioms. Such specifications determine the semantics of each model and the intent of each actor-network procedure.

2.5.3. What are the channel types?

Some of the channel types that we shall study are:

- cyber channels: each node broadcasts to all nodes; there is no notion of distance; the recipient cannot observe the sender[4]
- visual channel: the events at all nodes within some distance are observed; the observed nodes may or may not observe that they are observed;
- binary channel: streams bits from one node to another, flipping them with some given probability.

The binary channel is one of the basic concepts of information theory, capturing a simple notion of random noise. Intuitively, the cyber attacker can be viewed as *"adaptive noise"*, disturbing the integrity of the messages.

3. Actor-network processes

3.1. 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:

- a *write* event at the entry of the channel, and
- a *read* event at the exit of the channel.

There are two kinds of flows:

- *messages*, which consist of a *send action* at the entry of the channel, and a *receive coaction* at the exit; and
- *sources*, which consist of an *sample action* at the exit, and a *emit coaction* at the entry.

[4] We can assume that the sender always includes her identity into the message, within some standard format such as email. But such source claims can be easily spoofed in cyber space.

		events (actions•)	
		write	read
flows	message	send•	receive
	source	emit	sample•

Table 3. Flows and events

The information flows and the corresponding events are summarized in Table 3. The black dots mark the actions. A consistent action-coaction pair is called an *interaction*: i.e., an interaction consists of a send action and a receive coaction, or of a emit coaction and a sample action. We presently consider only these two types of interactions. Both the receive coactions and the emit coactions are construed as passive events: neither the principal who receives a message, nor the one who emits from a source controls when this happens. Of course, in reality a principal may, e.g. actively refuse to receive a message, or to emit emit a source, etc. But our goal is to enable simple analyses, and we leave these details outside the scope of the general model.

A computational process that is localized at a node proceeds as in the traditional models of computation. The node can be thought of as a state machine (e.g. a Turing machine), and the computational events change its state. An event at a configuration P may changes the states of any of the actors $N \in P$. In the actual analyses, the state changes often need to be traced, but we did not encounter an example where an actual state machine would need to be specified. The most abstract models that capture the relevant features usually support the simplest analyses, hiding the implementation details.

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 sampleion 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. 4.3.1.

Intuitions and ideas. A configuration can be thought of as a mechanism, assembled from separate components that may be owned and controlled by different principals. Another view is that a configuration is like a team, composed of the players that came to act together towards some goal, but will separate and go their own ways when they are done. The usual (physical) handshake, confirming a social contact, can be viewed as a configuration. A couple dancing together is a configuration. A band playing music together is a configuration.

A channel is like a wire, connecting two configurations. The messages and the sources are two types of flows through a channel. If Alice wants to send a message, she needs a channel to send it on. Bob is on the other side of the channel, passively waiting to receive the message. If Bob wants to sample a source, he needs a channel for that. Alice is on the other side of the channel, passively emiting. In both cases, Alice is at the entry of the channel, and Bob is at the exit of the channel.

Besides the channels that connect it to other configurations, a configuration may have internal methods for coordination among its nodes. E.g., a handshake is coordinated by two hands sensing each other. A couple of dancers develop signals that coordinate their dance. In some cases, such signals need to be made explicit as information flows. A configuration may need to send itself a message, or to sample itself as a source: e.g., to assure that genuine randomness is extracted. While the internal signaling and coordination capture controlled processes within a configuration, there are many processes that take place within a single configuration that are not entirely controlled. Dance groups and the bands of musicians have, besides the subtle forms of internal signaling, evolved complex external procedures to synchronize and coordinate. Such procedures can be thought of as primordial *social software*. Analyzing them within a formal model might conceivably open interesting possibilities of electronic support.

3.2. Formalizing data as terms

Each flow carries some data, which contain information. In abstract models, data are represented as terms of an algebra: the content of a message is an element of an algebra. We shall also represent the emission from a source as an element from the same algebra. The algebraic operations correspond to the data processing operations. In the standard symbolic protocol model [20], the messages the terms of a free algebra of encryption and decryption operations. More general algebraic models allow additional operations, and additional equations [12]. 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) [31].

Definition 3.1. 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 $f(x)$. A function of two arguments $e(x, y)$ is thus identified identified with its curried form $e.x.y$, and $e.x$ abbreviates $e(x, -)$. By abuse of notation, the pair (x, y) can thus be written as x, y, and $(x, (y, z)) = ((x, y), z)$ as x, y, z.

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

Tupling. The equation $(x, (y, z)) = ((x, y), z)$ in the above definition implies that there is a unique n-tupling operation for every n. The first two equations imply that the components of any tuple can be recovered.

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 $\mathcal{T}$ [31, §8]. The elements $x, y, z \ldots$ of $\mathcal{X}$ are used to represent nonces and other randomly generated values. This is justified by the fact that indeterminates can be consistently renamed: nothing changes if we permute them. That is just the property required from the random values generated in a run of a protocol. Of course, this is not the only requirement imposed on nonces and random values: the other requirement is that they are known only locally, i.e. only by those principals who generate them, or who receive them in a readable message. This requirement is not formalized within the algebra of messages, but by the binding rules of process calculus [17, 56]. Here we capture it by the freshness axioms in Sec. 4.3.2.

Stores are nodes. While the random values are thus algebraically presented as the indeterminates (i.e. as the variables in the polynomial extension), the stores (i.e. the variables used in computation) can be modeled as network nodes, each with a devoted read channel and a write channel. The property of such a node which is that it can store a value. The term stored in such a node determines its state. In this way, the usual notion of state, as a partial assignment of values to variables, is included within the network model. The state of a configuration is thus the product of the states of its actors, where some of the actors only task is to store some 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 [56]. 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 $E.k.m$. In the Dolev-Yao algebra, it is straightforward to define the easy subterm relation inductively. For general algebraic theories, the task of discerning the subterms gets complicated. A general treatment was attempted in [56].

3.3. Formalizing events and processes

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

3.3.1. Events

An event or action is generally written in the form $a[t]$ where

- a is the event identifier,
- 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 $(\cdot\, t \,\cdot)$,
- 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

- generate a random value $\nu[x]$,

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. Most are able to execute the standard pseudo-code commands, like comparisons $(t = s)$ or assignments $(t := s)$. But the differences in their computational resources will be essential in some of the security analyses of the procedures below. Further examples of application specific events and actions will be introduced in the below.

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 $\copyright P$.

Representing events as terms. How do we represent principals' observations of events and of other principals' actions? The location of an event may be viewed as a source for sampling; the location of an action must be controlled by a principal, who may be viewed as the sender of the message about the action that took place. But since only data can be sent as messages, or sampled from sources, each observable action $a[t]_P$ must be represented as a term $\Big[\!\Big[a[t]_P \Big]\!\Big]$. In general, this is done by adding to the presentation of the algebra $\mathbb{T}$ a mapping

$$[\![-]\!] : \mathbb{E} \to \mathbb{T}$$

which generates a representation of each event from $\mathbb{E}$. A similar map assigning to each action a logical formula leads to dynamic logic [32]. We'll see a typical example of a message about an action in Sec. 3.5.3, where we return to the device handshake procedure. Bob can only conclude that the secret shared by his and Alice's device is authentic if he sees Alice shaking the devices.

Self-sampling. A less typical, but not less interesting example arises if we assume that a configuration has a channel to sample its own events. Such a configuration can then sample its own sampling, i.e. execute the action $(: [\![(::)]\!] :)$. The principal who controls such a configuration can then observe some of her own observations. It is interesting to explore the authenticity of such observations. We shall touch this again in Sec. 4.4.1.

3.3.2. Processes

Definition 3.2. 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 ©$(\mathcal{F}_{\mathcal{P}}\phi)$ 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 $\mathcal{F}_{\mathcal{P}} = P$. Of course, if the there are several points in time $\phi_1, \phi_2, \ldots \in \mathcal{F}$ where the same P executes the same $a[t]$, then this notation is ambiguous, since it is not clear to which ϕ_i does $a[t]_P$ refer. But such situations are rare. On the other hand, with this notation the above conditions become:

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

Remarks. The subset ordering of $(\mathcal{P}, \subseteq)$ arises from Def. 2.1, which says that configurations are finite sets. Partially ordered multisets, or *pomsets* were introduced and extensively studied by Vaughan Pratt and his students [62]. Condition (a) specifies what we already said informally: that the configuration where an action takes place must be controlled. Condition (b), on the other hand, means that *there is no subliminal synchronization*: the ordering of events can only be imposed within a configuration that enables all of them. If Alice performs one action controlling a configuration P_A, and then another action controlling a configuration Q_A, then she must control a configuration $R_A \supseteq P_A \cup Q_A$, that will allow her to control the order of the actions with P_A and with Q_A. The intended use of configurations, including any constraints that would require that some parts of P_A and Q_A should not be used together, must be imposed through axioms, in the PDL language introduced in Sec. 4.

Definition 3.3. 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.

3.4. 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 3.4. For configurations $P, Q \in \mathcal{P}$, a *flow channel* $P \xrightarrow{\tau} Q$ can be either

- a channel $P \xrightarrow{\tau} Q$, or
- a flow channel $P \xrightarrow{\tau} Q'$, where $Q' \in Q$, or
- a flow channel $P' \xrightarrow{\tau} Q$, where $P' \in P$, or
- 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 3.5. 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 $b[s]_Q \not\to a[t]_P$ 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. 3.2 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.

Overloading arrows. The view of the runs as order extensions of the processes justifies the overloading of the arrow notation, which is used both

- as $a[t]_P \to b[s]_Q$, saying that $a[t]_P$ precedes $b[s]_Q$ in the partial ordering $(\mathcal{F}, \to)$, and
- $a[t]_P \xrightarrow{\tau} b[s]_Q$, denoting a flow of type τ from $a[t]_P$ to $b[s]_Q$ in a run $\mathcal{E}^{\mathcal{F}}$.

This overloading is consistent, because a flow from $a[t]_P$ to $b[s]_Q$ implies that $a[t]_P$ precedes $b[s]_Q$; and it will be useful when we pass from the runs, e.g. in Figures 6–8, to formal reasoning about them in Figures 9–12. The arrows in the latter family of diagrams arise from the arrows in the former family. But the former represent reality, whereas the latter represent assertions about it.

Definition 3.6. 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.

Procedures generalize protocols. A *protocol* is a special case of a network procedure, where the underlying network is a cyber network. Since cyber channels offer no security guarantees, the security goals of protocols are generally realized by cryptographic functions computed at the nodes. That is why cyber security is largely concerned with *cryptographic* protocols. It is thus based on the *end-to-end* paradigm, where the security tasks are pushed to the local computations at the smart "ends", i.e. nodes, leaving the network simple and efficient. The above definition of a secure procedure generalizes the definition of a secure protocol used in Protocol Derivation Logic [48, 56, 57], as well as in Protocol Composition Logic [24, 17, 15]. Network procedures and their security proofs thus extend cryptographic protocols, and their security proofs. What is the difference? First of all, any node in a cyber network is as good as any other node, so it does not matter which ones you control, or how many. Without loss of generality, configurations can thus be reduced to single nodes, and the channel flows to the messages on cyber channels. A protocol run thus boils down to a sequence of messages among the principals, each usually controlling a single node, with some local computations in-between the messages.

Nontrivial configurations arise in pervasive networks. E.g., a smart card can only compute when inserted into a correct configuration with a card reader. Moreover, the information can flow through a pervasive network in many different ways: by messages sent along a variety of different channels, short range, cellular, social, etc.; or by observations along visual channels, etc. The distinction between computation and communication in pervasive networks becomes blurred, as the two become intertwined in subtle and complicated ways.

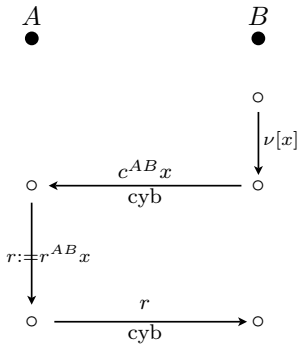

Fig. 6. Challenge-Response (CR) protocol template

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.3.2, 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. 7. 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. 7. The dashed lines represent the data sharing within a configuration.

This discipline of drawing

- the internal ordering of events along the verticals and
- the external ordering, imposed by the flows, along the horizontal lines is

of course, familiar from strand spaces, where the verticals are the strands, and the horizontals the bundles [29]. Our diagrams indeed boil down to strand diagrams whenever the network configurations are single nodes connected by cyber channels, and when the only flows are messages. This graphic convention for depicting the internal and external ordering of events goes back to the early days of distributed computing research, see e.g. [38, 44].

3.5. Examples of procedures

3.5.1. 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. 6. Bob wants to make sure that Alice is online. It is assumed that Alice and Bob share some sort of 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 [48, 11, 56, 57]. 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.

Recall from in Sec. 2.4.1 that cyber networks are degenerate, in the sense that the actors boil down to the principals. Alice's and Bob's unique actors are thus simply denoted by A and B.

3.5.2. Two-factor authentication procedure

Next we describe the first nontrivial procedure, over the actor-network described in Sec. 2.4.2. 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 [23]. The desired run of the challenge-response option of this procedure is depicted on Fig. 7.

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.

This two-factor procedure is thus more secure than the simple password authentication of I_A to D_B because

- there is a fresh challenge, and the attacker cannot impersonate Alice just by recording one session (like phishermen do);
- even in the option without the challenge, the secret s^{AB}, shared between two computing devices, is generally stronger than a human memorable secret p^A, and finally
- the PIN authentication to the smart card is not cryptographically strong, but it is done on a physical channel, which is harder to attack.

If a thief comes in the possession of the smart card S_A, he cannot use it without p^A, stored in I_A. This leads to the muggings, i.e. attacks where S_A is stolen, and then I_A is coerced to disclose the PIN p^A. The authors of [23] point out that introducing the portable, generic readers R simplifies for the attacker the verification that the number given to him by I_A is indeed the correct p^A.

3.5.3. Device handshake procedure

Going back to Fig. 4, we describe the desired run on the Device Handshake. The run begins by Alice bringing together the nodes for the configuration Q_A. This means that she takes her device D_A and Bob's device D_B into her hand I_A, to shake them together. We also assume that the configuration contains the node S, which is the source of randomness. This node does not correspond to a physical object, but embodies the source that is being sampled by shaking. In reality, the two devices (i.e. their accelerometers) record the joint action of shaking together, and sample the shared secret out of it. The method to achieve this is described in [43], and we take it as given. The desired run on the network from Fig. 4 is depicted on Fig. 8. It consists of five flows, triggered by Q_A's shaking of the devices:

- $(: x :)_{Q_A} \xleftarrow{\text{shk}} \langle : x : \rangle_S$, caused by Q_A's sampling of x from S;
- $\big(: f^A :\big)_{D_A} \xleftarrow{\text{bio}} \big\langle : f^A : \big\rangle_{I_A}$, simmultaneously caused by I_A's sampling of I_A's fingerprint f^A;
- $\Big\langle \cdot \Big[(::)_{Q_A} \Big] \cdot \Big\rangle_{Q_A} \xrightarrow{\text{vis}} \Big(\cdot \Big[(::)_{Q_A} \Big] \cdot \Big)_{I_B}$, where Q_A shows I_B how she shakes the devices to sample x;

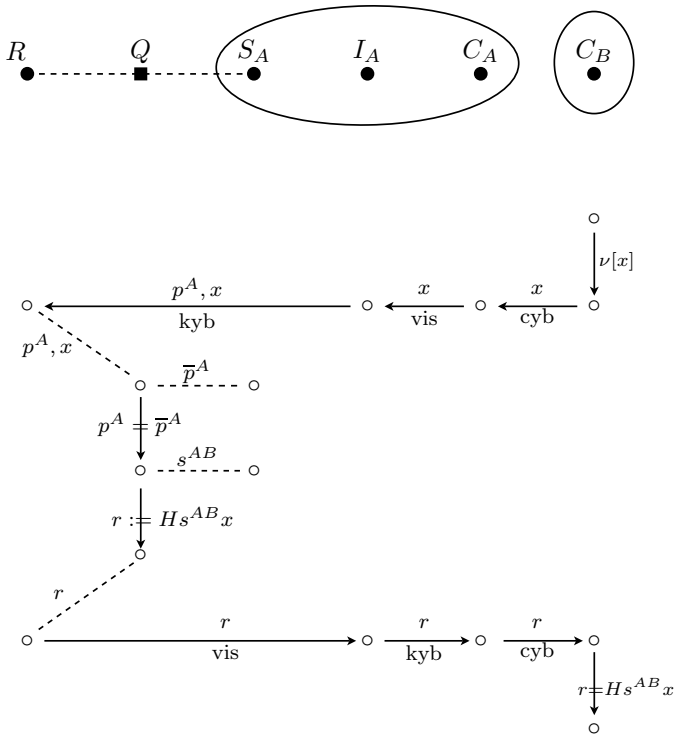

Fig. 7. Chip Authentication Program (CAP) procedure

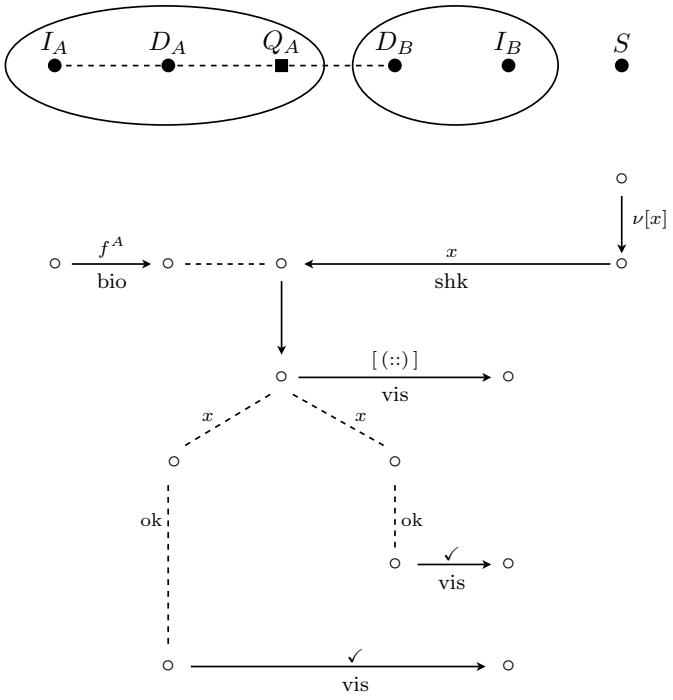

Fig. 8. Device handshake procedure

- $\langle \cdot \checkmark \cdot \rangle_{D_A} \xrightarrow{\text{vis}} (\cdot \checkmark \cdot)_{I_B}$, confirming to I_B that D_A received x and correct f^A; and
- $\langle \cdot \checkmark \cdot \rangle_{D_B} \xrightarrow{\text{vis}} (\cdot \checkmark \cdot)_{I_B}$, confirming to I_B that D_B received x.

Remark. Note that the third flow contains an example of a message about an action, along the lines explained in Sec. 3.3, in the paragraph about representing events as terms. Here Bob's human I_A receives from Alice's configuration Q_A the visual message $[\,(::)\,]$ that she has sampled the randomness.

Two-round device handshake. If both Alice and Bob need to be assured that a secret is shared, then the last two messages, where D_A and D_B confirm that they succeeded to sample x, and one of them confirms that the fingreprint is correct, should be sent to both I_A and I_B. Running two rounds of the device handshake, with both Alice and Bob shaking their devices in the actor-network from Fig. 5, would require such procedure in each of the two rounds.

4. Procedure Derivation Logic

4.1. Procedures are distributed predicates

Formal methods for software engineering have been built upon Hoare's slogan that *"Programs are predicates"* [33]. The view that computations can be adequately approximated by their logical descriptions, and proven correct, was the guiding principle of theory and practice of software specifications from the outset in the 1960s [30]. The underlying assumption is that software is run within a single computer, with a global observer asserting the predicates. In contrast, the essence of a network is that there is no global observer. The events at each node may be directly observed only by the principal who controls that node. In addition, they may be indirectly observed along an authentic channel that ends at that node.

The upshot is that each participant of network computation may observe different events, and assert different predicates. Each of them sees just bits and pieces of the network process. In most cases of interest, their common goal is to coordinate their observations, and arrive at a coherent joint view of the events in which they jointly participate, so that they can all assert the same predicate, which is the required security property. Towards this goal, they use the channels between them to exchange messages, and to observe each other.

In summary, our goal is thus to extend formal methods from programs as predicates to network procedures as families of predicates, asserted at different network nodes. The task of reconciling these local assertions is usually formalized as *authentication*. In this section, we gradually introduce the language and the axioms that allow participants of network computations to annotate their local observations by global predicates, building up from the basic communication axioms, towards authentication, and beyond.

4.2. 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

- $a[t]_P$ — meaning "the event $a[t]_P$ happened"; or
- $a[t]_P \to b[s]_Q$ — meaning "the event $a[t]_P$ happened before $b[s]_Q$".

Notation: Statements assert events, and events describe points of time. Recall that the expressions like $a[t]_P$ refer to points in time descriptively, i.e. by specifying what happened and where. As explained in Sec. 3.3, this is a notational abuse, but an important one. The descriptions $a[t]_P$ are sometimes refined to $\sqrt{a[[t]]_P}$, which say that the event $a[s]$ took place at the configuration P, for some $s \sqsupseteq t$, *and* that the term t originates there.

The essence of PDL. Here, the event descriptions like $a[t]_P$ and $\sqrt{a[[t]]_P}$ moreover denote the assertions that the described event took place. The descriptions of processes and of their runs are used as the predicates to annotate them, and to reason about them. This notational abuse is justified by the *isomorphism of*

process executions and their logical annotations. This is the basic design decision on which PDL is based: *The descriptions of processes and of their runs as partial orders are used as the logical assertions to annotate these processes and their runs.* This is the formal implementation of the guiding principle of PDL, explained in Sec. 1.4.2.

This isomorphism does not only simplify notation, but substantially simplifies the logic that we work with. A statement in any protocol or procedure logic is an assertion of a participant — *made at a certain point of a run.* In PCL, this dependency on the run was expressed using dynamic modalities. In PDL, though, the process expression within a modality isomorphic to a logical formula. So instead of

- $A : [\psi]\Phi$, saying that A knows that Φ is valid after the execution point ψ is reached, we can write
- $A : \Psi \Longrightarrow \Phi$, saying that A knows that Φ is valid whenever the description Ψ of ψ is valid.

The two formulas are semantically equivalent because the formula Ψ is a complete description of the process ψ. The PDL assertions thus usually appear in the form $A : \Psi \Longrightarrow \Phi$, where Ψ describes A's view of the run, and Φ her conclusion about it.

The examples follow.

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

4.3.1. 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.3. 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

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

$$\copyright P \ : \ (:\, t\, :)_P \Longrightarrow \ \exists X.\ \langle : t : \rangle_X \rightarrow (: t :)_P \qquad \text{(orig.s)}$$

4.3.2. Freshness

In Sec. 3.2 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

- *freshly generated* by an action $\nu[x]$ before it is used anywhere; and
- that it can only be used elsewhere after it has passed in a message or a source.

which formally becomes

$$\copyright P : \quad a[t.x]_P \ \Longrightarrow \ \exists X.\ \nu[x]_X \rightarrow a[t.x]_P \qquad \text{(fresh.1)}$$

$$\begin{aligned} \copyright P : \neg\nu[x]_P \ \wedge \ a[t.x]_P \ \Longrightarrow \ & \exists X.\ \big(\nu[x]_X \rightarrow \sqrt{\langle\langle \cdot\ x\ \cdot \rangle\rangle_X} \rightarrow ((\cdot\ x\ \cdot))_P \rightarrow a[t.x]_P\big) \\ & \vee \ \big(\nu[x]_X \rightarrow \sqrt{\langle\langle : x : \rangle\rangle_X} \rightarrow ((:\ x\ :))_P \rightarrow a[t.x]_P\big) \end{aligned} \qquad \text{(fresh.2)}$$

where, using the easy subterm order $\sqsubseteq$ from Sec. 3.2,

- $\langle\langle \cdot\ x\ \cdot \rangle\rangle_X$ 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.

4.4. Authentication axioms

In classical logic, a statement may be true or false. In classical formal methods, an assertion about a computation is also either true or false, and it is assumed that we can observe which one it is. The idea is that we can, e.g. inspect the program variables in the debug mode. In network computation, an assertion is still either true or false — yet none of the participants may be able to observe whether it is true or false. E.g., when Alice receives a message on a cyber channel, she may not be able to verify whether the statement *"This message is from Bob"* is true or false. The process of verifying such non-local statements by local means is *authentication*. In our model, there are two forms of authentication:

- interactions along authentic channels, and
- challenge-response authentication.

4.4.1. 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:

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

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

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:

- A keyboard channel guarantees to the sender that the device at which she is typing is receiving the message, and thus satisfies (auch.m.1).
- A visual channel used for sending a message allows the receiver to see the sender, and satisfies (auch.m.2).
- When my fingerprints are taken, I observe that they are taken, and can see who is taking them, so this biometric channel satisfies (auch.p.1).
- Moreover, the person taking my fingerprints also observes that they are taking my fingerprints, so (auch.p.2) is also satisfied.
- If a visual channel is used for surveying, then the surveyor sees where the display appears, and thus satisfies (auch.p.2) as well; etc.

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.

Authenticity of self-sampling. One particular authenticity axiom worth mentioning is the statement that the self-observation channel is authentic, at least for sampling the sampling actions, i.e.

$$\copyright P \ : \ \Big(: \ \big[\, (::)_P \, \big] \ : \Big)_P \Longrightarrow (::)_P \qquad \textsf{(cog)}$$

In other words, "If I observe that I have observed something, then I have really observed something". This is the PDL version of Descartes' authentication of the world: *"Cogito, ergo sum"*.

4.4.2. Challenge-response authentication

The challenge-response axiom is in the form

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

where, using the notation from Sec. 4.3.2

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

Translated into words, (cr) says that the owner $\copyright P$ of the configuration P knows that

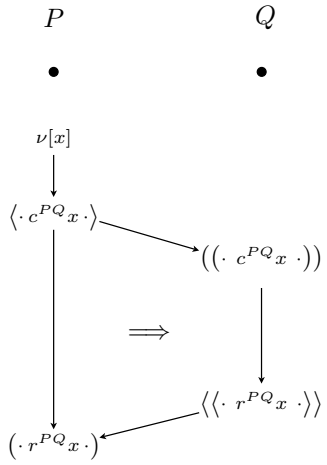

Fig. 9. The graphic view of (cr) axiom

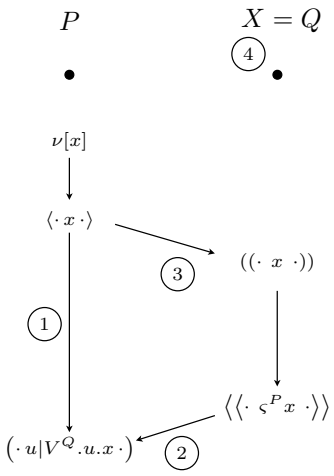

Fig. 10. Challenge-response using signatures

- if he generates a fresh x, sends the challenge $c^{PQ}x$, and receives the response $r^{PQ}x$,
- then 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. 9 shows depicts this reasoning diagrammatically. This axiom should be viewed as an assertion about the functions c^{PQ} and r^{PQ}. They must be such that Q can compute $r^{PQ}x$ from $c^{PQ}x$, but no one else can do it.[5]

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 [48, 11], through to [57].

5. Examples of reasoning in PDL

5.1. 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* [42, 54]. 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 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.

5.2. Cryptographic (single-factor) authentication

We begin with a very simple example of diagrammatic reasoning, present already in [48].

[5] In the cases when c^{PQ} and r^{PQ} are based on a secret shared between P and Q, then P can compute r^{PQ} as well. In such cases, the soundness of (cr) depends on ©P's observation that P has not done that.

Theorem. The functions

$$c^{PQ}x = x \qquad\qquad r^{PQ}x = \varsigma^Q x$$

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 $\nu[x]_P \to \langle\cdot\, x\,\cdot\rangle_P \to \big(\cdot\, r|V^Q rx\,\cdot\big)$, i.e. after sending a fresh value x, he receives a response u which passes the verification $V^Q rx$.
2. Using the axioms (c) and (orig.m), he concludes that there is some X such that $\big\langle\cdot\, V^Q x\,\cdot\big\rangle_X \to \big(\cdot\, r|V^Q rx\,\cdot\big)_P$.
3. Using (fresh.2) he further derives that for the same X holds $\langle\cdot\, x\,\cdot\rangle_X \to ((\cdot\ x\ \cdot))_X \to \big\langle\cdot\, V^Q x\,\cdot\big\rangle_X$.
4. Using (a) and (b), Bob concludes that $V^Q x$ must have originated from Q.

Bob can, of course, only be sure that Q was online between his $\langle\cdot\, x\,\cdot\rangle_P$ and $\big(\cdot\, r|V^Q rx\,\cdot\big)$, and *not* that Alice = ©Q really intended to respond to his challenge. It is well known that this form of authentication is open to impersonation, since $r = V^Q x$ contains no reference to Bob or to P. □

5.3. Pervasive (two-factor) authentication

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

Theorem. The procedure on Fig. 7 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) $\big\langle\cdot\, Hs^{AB}x\,\cdot\big\rangle_Q \implies \Big(\big(\cdot\, p^A, x\,\cdot\big)_Q \to \big\langle\cdot\, Hs^{AB}x\,\cdot\big\rangle_Q\Big) \wedge p^A = \overline{p}^A$, 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. 11. The enumerated steps in the diagram chase correspond to the following steps in Bob's reasoning:

1. Bob observes $\nu[x]_{C_B} \to \langle\cdot\, x\,\cdot\rangle_{C_B} \to \big(\cdot\, Hs^{AB}x\,\cdot\big)_{C_B}$.
2. Using (orig.m) he concludes that there is some X such that $\langle\cdot\, Hs_A x\,\cdot\rangle_X \to \big(\cdot\, Hs^{AB}x\,\cdot\big)_{C_B}$.
3. Using (fresh.2) he further derives that for the same X holds $\langle\cdot\, x\,\cdot\rangle_{C_B} \to ((\cdot\ x\ \cdot))_X \to \big\langle\cdot\, Hs^{AB}x\,\cdot\big\rangle_X$.
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), $\big\langle\langle\cdot\ p^A\ \cdot\big\rangle\big\rangle_{I_A} \to \big((\cdot\ p^A\ \cdot)\big)_Q \to \big(p^A = \overline{p}^A\big) \to \big(Hs^{AB}x\big)$, 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. (See Remark below.)
6. Since Q had to also receive x before computing the response in $\big(\cdot\, p^A, x\,\cdot\big)_R \to \big(Hs^{AB}x\big)$ 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 $\big\langle\cdot\, p^A, x\,\cdot\big\rangle_Y \to \big(\cdot\, p^A, x\,\cdot\big)_R$. By (e), $\big\langle\langle\cdot\ p^A\ \cdot\big\rangle\big\rangle_{I_A}$ from 5 must be $\big\langle\cdot\, p^A, x\,\cdot\big\rangle_{I_A}$.

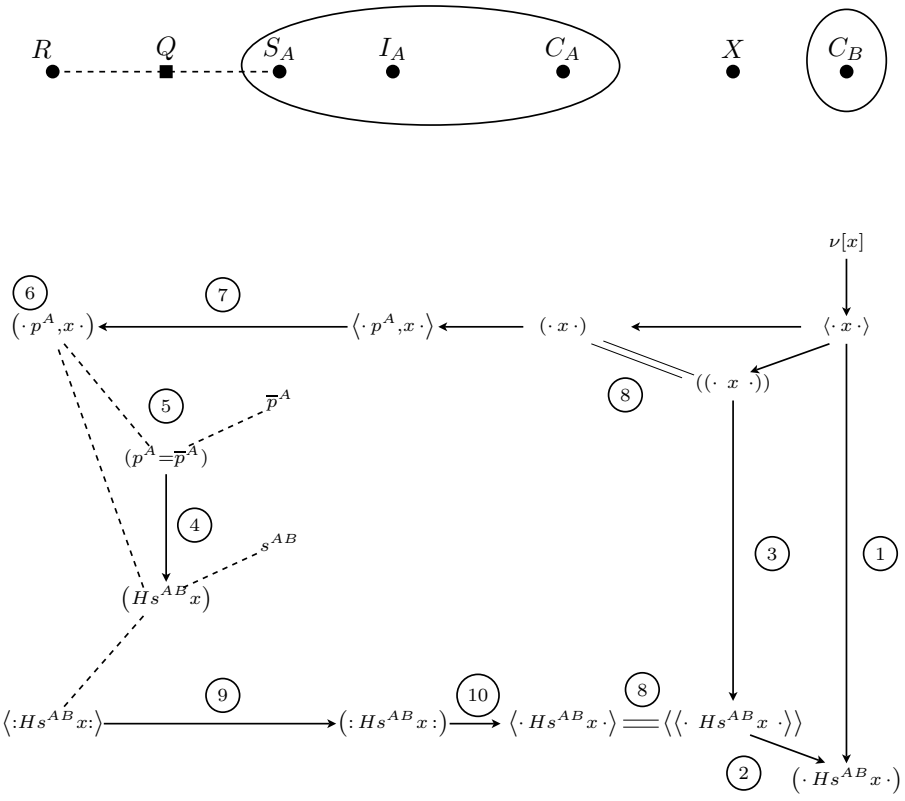

Fig. 11. B's reasoning in CAP

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. 11, 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.

More explicitly, Bob's conclusion in Fig. 11 is

$$\begin{aligned}\langle \cdot\, x\, \cdot \rangle_{C_B} \to (\cdot\, x\, \cdot)_{C_A}\, \langle \cdot\, x\, \cdot \rangle_{C_A} \to (\cdot\, x\, \cdot)_{I_A} \to \langle \cdot\, p^A, x\, \cdot \rangle_{I_A} \to (\cdot\, p^A, x\, \cdot)_R \to \\ (p^A = \overline{p}^A)_Q \to (Hs^{AB}x)_Q \to \langle : Hs^{AB}x : \rangle_R \to (: Hs^{AB}x :)_{I_A} \to \\ \langle \cdot\, Hs^{AB}x\, \cdot \rangle_{I_A} \to (\cdot\, Hs^{AB}x\, \cdot)_{C_A} \to \langle \cdot\, Hs^{AB}x\, \cdot \rangle_{C_A} \to (\cdot\, Hs^{AB}x\, \cdot)_{C_B}\end{aligned}$$

where we also added the trivial relay flows omitted from the figure, namely

- $\langle \cdot\, x\, \cdot \rangle_{C_A} \xrightarrow{\text{cyb}} (\cdot\, x\, \cdot)_{I_A}$ on the way out, and
- $\langle \cdot\, Hs^{AB}x\, \cdot \rangle_{I_A} \xrightarrow{\text{kyb}} (\cdot\, Hs^{AB}x\, \cdot)_{C_A}$ on the way back.

Again, it is clear that this conclusion is clearly an instance of (cr) and thus realizes authentication. □

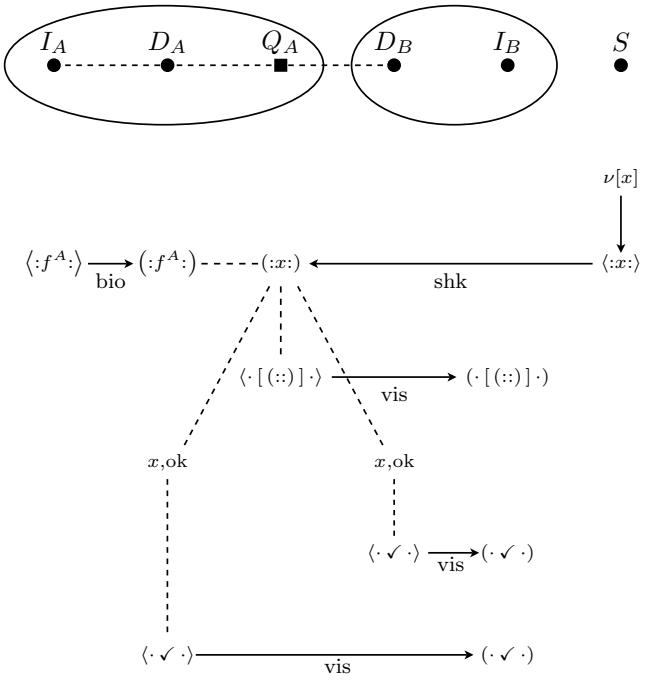

Fig. 12. B's view of A's round of device handshake

Remarks. The reader may note that the store r, to which the response $Hs^{AB}x$ is assigned at run time, was present in Fig. 7, but elided in Fig. 11. This is a minor quirk here, but we wanted to adher to the custom established in the informal protocol analyses, and propagated, e.g. through the strand space notation: stores and program variables are by convention avoided, and denoted by the terms assigned to them at runtime. Although such a term, strictly speaking, does not have a static value, and denoting the stores ready to receive it when it is evaluated by its unevaluated expression is abuse of notation — it seems to be an eminently reasonable abuse, since it displays the path of the term during the run, whereas the names of all the local stores ready to receive it, only conceal its path. So the diagrammatic convention wins the day.

Honesty assumption. It would be easy and natural to eliminate the assumption that S_A is honest by storing p^A also at I_A, and including it into the response. That would reduce the above reasoning to "S_A has been on the path of the message" and added a separate thread "I_A has also been on the path of the message". We chose to present the above version as slightly more informative, albeit slightly weaker.

Does Bob need to be authenticated? In practice, the attackers usually impersonate Bob, to steal Alice's credential and use that to impersonate her to the actual Bob, the bank. The most frequent form of that attack is, of course, phishing. Two factor authentication is devised to avoid that: here Alice does not give her credentials to Bob, but to the smart card reader. Why is that better? How does Alice know that reader R's request for PIN is authentic? She *sees* on the visual channel that the only card in R is S_A, which she had put there herself. This is a simple but typical example of use of an authentic channel.

5.4. Device pairing by handshake

For the final example, we return to the device handshake procedure. For reasons of space, we omit proof of the theorem, but present that diagram that is used to supply the proof.

Theorem. Upon the completion of a run of the procedure described in Sec. 3.5.3, Bob can be sure that his and Alice's device share a key, provided that he knows that the following assumptions are valid:

(a) $©P : \langle\cdot\,[\,a_Q\,]\,\cdot\rangle_Q \xrightarrow{\text{vis}} (\cdot\,[\,a_Q\,]\,\cdot)_P$, i.e. the visual channel satisifies (auch.m.2) at least for events;
(b) $(:x:)_{Q_A} \Longrightarrow x \in \Gamma_{D_A} \cap \Gamma_{D_B}$, i.e., Q_A distributes the same value to D_A and D_B;
(c) $(:t:)_{Q_A} \Longrightarrow \nu[x]_S \to \langle:x:\rangle_S \to (:t:)_{Q_A} \wedge t = x$, i.e. Q_A is honest ;
(d) $\langle\cdot\checkmark\cdot\rangle_{D_A} \Longrightarrow \text{ok}_{D_A} \wedge \langle:f^A:\rangle_{I_A} \to \big(:f^A:\big)_{D_A} \to \langle\cdot\checkmark\cdot\rangle_{D_A}$, i.e. D_A is honest; and
(e) $\langle\cdot\checkmark\cdot\rangle_{D_B} \Longrightarrow \text{ok}_{D_B}$, i.e. D_B is honest.

Bob's formal reasoning is displayed on Fig 12.

6. Conclusion, discussion, and some last minute philosophy

Summary 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. But we also believe that it should be possible to develop tool support as well, in which a proof engine would execute the logic, and the proof itself would be demonstrated on a graphical template. Both avenues are a question for future research.

Post hoc ergo propter hoc? Although we speak of complete runs, as specified in Def. 3.5, and the information flows seem completely displayed in such runs, it is always possible to raise the question whether a demonstrated temporal order of events reflects their causal order. Can it be that Bob has received the same message that Alice had sent, but that by some coincidence Carol had sent an identical message, and that Bob actually received Carol's message? If the two messages are really identical, how can we tell?

Should we try to distinguish the causal connections from temporal coincidences? More specifically, would it be possible to refine PDL by working with statements that would specify not just the order of events, but also which interactions occurred through which channels? The idea would thus be to discern whether Bob has received Alice's or Carol's message by following the messages as they travel from channel to channel, and seeing which one goes where.

We believe that this would just complicate logic, without bringing essentially more information. Even if we could follow two messages, with the same payload say m_1 and m_2 on their respective paths across the network, could we be sure that their paths never cross? Networks are busy places, a hop from node to node may conceal many intermediary steps, and m_1's hop $a_1 \to b_1$ may pass through an invisible node x at the same time as m_2's hop $a_2 \to b_2$ passes through x. If that happens, then m_1 may emerge at b_2 and m_2 at b_1. Or the other way around. We will never know. It will remain uncertain which of the two messages has reached Bob in the end.

Logic has no business chasing the ghost of causality. The best we can do is describe the order of the events. Partially ordered multisets will remain a robust foundation for reasoning about network interactions and procedures for many applications to come.

Can we really model social interactions and computations within the same model? People are not computers and society is not a computer network. How could I ever predict the behavior of a social group, when I am unable to predict the behaviors within my own family? Sometime I don't even understand my own behavior. Humans are hopelessly irrational. End of the story.

But science has strange tricks. It is unable to model the trajectory of a single particle in the air, but it models hurricanes and predicts weather. Similarly, the polling experts and the web advertisers have developed impressive techniques to predict *and influence* the behaviors of large groups of people with a significant precision, although noone seems to be able to predict or influence what any particular individual will do.

Most sciences try to model reality. But people do not obey models, groups of people do not obey models,

so the sciences concerned with people and with social groups largely took exception to the modeling task. It turns out, though, that the task may become easier when social networks are interleaved with networks of computers and devices, and when people are modeled together with other social and computational actors. Can it be that models better fit people because people better fit models?

Acknowledgement. The first author would like to thank Wolter Pieters [61] for introducing him to Bruno Latour's ideas about actor-networks.

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] Martín Abadi, Bruno Blanchet, and Hubert Comon-Lundh. Models and proofs of protocol security: A progress report. In Ahmed Bouajjani and Oded Maler, editors, *CAV*, volume 5643 of *Lecture Notes in Computer Science*, pages 35–49. Springer, 2009.

[3] 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.

[4] 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.

[5] David A. Basin, Manuel Clavel, and Marina Egea. A decade of model-driven security. In Ruth Breu, Jason Crampton, and Jorge Lobo, editors, *SACMAT*, pages 1–10. ACM, 2011.

[6] Giampaolo Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography. Springer Verlag, 2007.

[7] Yochai Benkler. *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. Yale University Press, 2006.

[8] Bruno Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Trans. Dependable Sec. Comput.*, 5(4):193–207, 2008.

[9] 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.

[10] Ileana Buhan, Bas Boom, Jeroen Doumen, Pieter H. Hartel, and Raymond N. J. Veldhuis. Secure pairing with biometrics. *IJSN*, 4(1/2):27–42, 2009.

[11] 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.

[12] V. Cortier, S. Delaune, and P. Lafourcade. A survey of algebraic properties used in cryptographic protocols. *J. Comput. Secur.*, 14(1):1–43, 2006.

[13] Véronique Cortier and Steve Kremer, editors. *Formal Models and Techniques for Analyzing Security Protocols*, volume 5 of *Cryptology and Information Security Series*. IOS Press, 2011.

[14] 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.

[15] A. Datta, A. Derek, J. Mitchell, and A. Roy. Protocol Composition Logic (PCL). *Electronic Notes in Theoretical Computer Science*, 172:311–358, April 2007.

[16] Anupam Datta, Ante Derek, John Mitchell, and Dusko Pavlovic. Secure protocol composition. *E. Notes in Theor. Comp. Sci.*, pages 87–114, 2003.

[17] 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.

[18] 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.

[19] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 523–540. Springer, 2004.

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

[21] Marco Dorigo and Thomas Stützle. *Ant colony optimization*. MIT Press, 2004.

[22] S. N. Dorogovtsev and J. F. F. Mendes. *Evolution of Networks: From Biological Nets to the Internet and WWW*. Oxford University Press, 2003.

[23] 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.

[24] 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.

[25] 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.

[26] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning About a Highly Connected World*. Cambridge University Press, 2010.
[27] Carl Ellison. Ceremony design and analysis. Cryptology ePrint Archive, Report 2007/399, October 2007.
[28] Santiago Escobar, Catherine Meadows, and José Meseguer. Maude-NPA: Cryptographic protocol analysis modulo equational properties. In Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri, editors, *FOSAD*, volume 5705 of *Lecture Notes in Computer Science*, pages 1–50. Springer, 2007.
[29] Javier Thayer Fabrega, Jonathan Herzog, and Joshua Guttman. Strand spaces: What makes a security protocol correct? *Journal of Computer Security*, 7:191–230, 1999.
[30] Robert W. Floyd. Assigning meaning to programs. In J.T. Schwartz, editor, *Proceedings of the Symposium on Applied Maths*, volume 19, pages 19–32. AMS, 1967.
[31] George A. Gratzer. *Universal Algebra*. Van Nostrand Princeton, N.J.,, 1968.
[32] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. MIT Press, Cambridge, MA, 2000.
[33] C. A. R. Hoare. Programs are predicates. *Phil. Trans. R. Soc. Lond.*, A 312:475–489, 1984.
[34] 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.
[35] 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.
[36] Richard A. Kemmerer, Catherine Meadows, and Jonathan K. Millen. Three system for cryptographic protocol analysis. *J. Cryptology*, 7(2):79–130, 1994.
[37] 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.
[38] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
[39] Bruno Latour. *Reassembling the Social: An Introduction to Actor-Network Theory*. Oxford University Press, 2005.
[40] 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.
[41] Sven Laur and Sylvain Pasini. User-aided data authentication. *IJSN*, 4(1/2):69–86, 2009.
[42] Saunders Mac Lane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.
[43] Rene Mayrhofer and Hans Gellersen. Shake well before use: Intuitive and secure pairing of mobile devices. *IEEE Trans. Mob. Comput.*, 8(6):792–806, 2009.
[44] D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.
[45] Catherine Meadows. The NRL Protocol Analysis Tool. In *Proceedings of the Computer Security Foundations Workshop*, page 227. IEEE, 1991.
[46] Catherine Meadows. A model of computation for the NRL Protocol Analyzer. In *Proceedings of the Computer Security Foundations Workshop*, pages 84–89. IEEE, 1994.
[47] Catherine Meadows. The NRL Protocol Analyzer: An overview. *J. Log. Program.*, 26(2):113–131, 1996.
[48] 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.
[49] 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.
[50] Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010.
[51] 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.
[52] Bernhard O. Palsson. *Systems Biology: Properties of Reconstructed Networks*. Cambridge University Press, 2006.
[53] 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.
[54] Dusko Pavlovic. Maps II: Chasing diagrams in categorical proof theory. *J. of the IGPL*, 4(2):1–36, 1996.
[55] Dusko Pavlovic. The unreasonable ineffectiveness of security engineering: An overview. In José Luiz Fiadeiro and Stefania Gnesi, editors, *Proceedings of IEEE Conference on Software Engineering and Formal Methods, Pisa, Italy, 2010*, pages 12–18. IEEE, 2010.
[56] 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.
[57] Dusko Pavlovic and Catherine Meadows. Bayesian authentication: Quantifying security of the Hancke-Kuhn protocol. *E. Notes in Theor. Comp. Sci.*, 265:97 – 122, 2010.
[58] 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.
[59] 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.

[60] Chris Peltz. Web services orchestration and choreography. *Computer*, 36:46–52, October 2003.

[61] 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.

[62] Vaughan Pratt. Modelling concurrency with partial orders. *Internat. J. Parallel Programming*, 15:33–71, 1987.

[63] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 2:277–288, November 1984.

[64] 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.

[65] 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.

[66] Duncan J. Watts. *Six Degrees: The Science of a Connected Age*. W. W. Norton, New York, 2003.

[67] Barry Wellman, Janet Salaff, Dimitrina Dimitrova, Laura Garton, Milena Gulia, and Caroline Haythornthwaite. Computer networks as social networks: Collaborative work, telework, and virtual community. *Annual Review of Sociology*, 22(1):213–238, 1996.