Formalizing and Analyzing Security Ceremonies with Heterogeneous Devices in ANP and PDL

Antonio González-Burgueño(✉) and Peter Csaba Ölveczky

University of Oslo, Oslo, Norway
antonigo@ifi.uio.no

Abstract. *Security ceremonies* extend cryptographic protocols with models of human users to allow us to take human behaviors into account when reasoning about security. *Actor-network procedures* (ANPs) are a well-known formal model of security ceremonies, and *procedure derivation logic* (PDL) allows us to reason logically about ANPs. In a security ceremony, different nodes may have different *capabilities*: computers can encrypt and decrypt messages, whereas humans cannot; a biometric device can capture biometric information, whereas a random number generator used in e-banking cannot; and so on. Furthermore, even if a node has the *decryption* capability, it must also know the encryption key to decrypt a message. ANPs do not support explicitly specifying node capabilities. In this paper, we extend ANPs to deal with heterogeneous devices by explicitly specifying the nodes' capabilities. We also modify PDL to take into account the knowledge of participants at different points in time. All this allows us to reason about secrecy and authentication in ceremonies with different kinds of devices and human users.

1 Introduction

Most security breaches nowadays occur not by breaking cryptographic protocols or because of buffer overflow, but through various forms of "social engineering attacks," such as phishing emails, malicious apps and web sites, browser status/address bar spoofing attacks [6], and so on. Furthermore, web applications typically interact with human users. To reason about security, we must therefore include humans as key parts of the security process, which requires defining new models of such processes. For example, in standard crypto-protocol formalisms, the behavior of each actor is typically given as a (deterministic) *sequence* of actions, whereas humans may exhibit *nondeterministic* behaviors (does the user click on the link? does she perform an action in the wrong way?).

Security ceremonies [8] extend cryptographic protocols with models of human users. *Actor-network procedures* (ANPs), introduced by Meadows and Pavlovic, are one of the more popular ways of formalizing security ceremonies (see, e.g., [1, 5,11,12,14,15]), and *procedure derivation logic* (PDL) [11,15] allows us to reason

Published by Springer Nature Switzerland AG 2019
H. Hojjat and M. Massink (Eds.): FSEN 2019, LNCS 11761, pp. 129–144, 2019.
https://doi.org/10.1007/978-3-030-31517-7_9

logically about ANPs. ANPs define the possible behaviors as partial orders over events, and PDL formulas allow nodes to assert the order of events in a protocol run. ANP and PDL have been used to formalize and reason about a wide range of systems, including physical access to secure areas of airports and office buildings, multi-factor multi-channel authentication, and key agreement procedures.

A security ceremony typically includes different kinds of nodes, such as computers, different kinds of humans (expert users, novices, intruders, etc.), and authentication devices like smart cards, random number generators, biometric devices, and so on. Different actors may have very different capabilities: a computer can encrypt and decrypt messages whereas humans cannot; a biometric device can capture biometric information, whereas a random number generator used in e-banking cannot; and so on.

The ANP formalism is fairly abstract, and does not support specifying that different nodes have different capabilities. In this paper, we therefore define *ANPs with capabilities* (ANP-Cs), which extend ANPs with an explicit specification of the capabilities of the different nodes. ANP-Cs also add the following events to APNs' *send* and *receive* events: (a) *learning* events for obtaining information (messages, keys, etc.) from previously received messages, and (b) events *creating* new terms from existing knowledge and the node's capabilities. Learning events are needed to express secrecy: did the intruder learn m from overhearing some (encrypted) message m'?

PDL is a logic for reasoning about the temporal order of *events*, and does not allow us to reason about the *knowledge* of the nodes at certain times. However, a node that has the capability to decrypt an encrypted message can only do so if it currently knows the decryption key for the message. To reason more accurately about security ceremonies, we should keep track of the knowledge of each node throughout the run of the ceremony. We therefore modify PDL to allow reasoning about ANP-Cs. Our new logic PDL-CK allows us to reason about the dynamically evolving knowledge of the participants, and can be used, for example, to reason logically about under what circumstances (i.e., what are the necessary capabilities of the different actors and what must they know initially?) a certain action, such a node decrypting a message, can take place.

The rest of this paper is structured as follows: Sect. 2 gives some background on ANP and PDL. Section 3 shows how different capabilities can be axiomatized as operations in an equational algebraic theory, introduces the new events for learning and creating, and defines ANP-Cs as ANPs with an explicit mapping from devices to capabilities. Section 4 introduces PDL-CK. Finally, Sect. 5 discusses related work, and Sect. 6 gives some concluding remarks.

2 Preliminaries

Meadows and Pavlovic have developed *actor-network procedures* (ANPs) [13,15] to formally specify security ceremonies. This is a quite abstract model, where the possible local behaviors of a group ("configuration") of nodes is specified as a partial order of *localized events*. A localized event is either $\mathsf{send}(t)_P$ or $\mathsf{receive}(t)_P$,

where t is a term of a user-defined algebraic theory (Σ, E) of operations (consisting of an algebraic signature Σ declaring sorts and operations/functions, and a set E of equations axiomatizing those operations), and P is a node or group of collaborating nodes.[1] The set of possible runs in an entire system are then given as the partial order of localized events that "combine" the different local partial orders in a send/receive-consistent way.

Procedure derivation logic (PDL) [13,15] is a logic for localized reasoning about the temporal order of events in an ANPs; for example, "node p knows that if it has received the message t, then some node X previously sent t".

Although ANP and PDL have been used on a number of applications [12,13, 15,16], there is currently no tool support for ANP and PDL.

Actor-Network Procedures. The "static" structure of an ANP is defined as an *actor-network*. A *configuration* is a set of nodes and/or (sub)configurations where all participants need each other to achieve a common goal. A smart card and a card reader may be seen as a configuration: both are needed to validate someone's identity. An actor-network is a network of such (possibly hierarchical) configurations, *principals* that control the configurations, and *channels* between configurations, where each channel has a *type*, and is defined as follows in [15]:

Definition 1 ([15]). *An actor-network consists of: a set $\mathcal{J}$ (of principals); a set $\mathcal{N}$ (of nodes); a set $\mathcal{P}$ of* configurations*, where a configuration can be a finite set of nodes, or a finite set of configurations; a set $\mathcal{C}$ (of channels); a set Θ (of channel types); a partial map $\copyright : \mathcal{P} \to \mathcal{J}$ (denoting the principal controlling a configuration); functions $\delta, \varrho : \mathcal{C} \to \mathcal{P}$ denoting the source and destination of a channel; and a function $\vartheta : \mathcal{C} \to \Theta$ (assigning to each channel its type).*

An algebraic theory (Σ, E) defines the operations, such as encryption, decryption, creating a nonce, etc. An *event* or *action* has the form $a(t)$, where a is an event identifier (such as send or receive) and the term t is its parameter.

An *actor-network procedure* extends an actor-network by adding a *process*, which defines the local behaviors of each configuration as a partially ordered multiset of *localized events*:

Definition 2 ([15]). *A process $\mathcal{F}$ is a partially ordered multiset of localized events, $\mathcal{F} = \langle \mathcal{F}_{\mathbb{E}}, \mathcal{F}_{\mathcal{P}} \rangle : \mathbb{F} \to \mathbb{E} \times \mathcal{P}$, where*

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

such that if $\phi \to \varphi$ in $\mathbb{F}$ then $\mathcal{F}_{\mathcal{P}}\phi \subseteq \mathcal{F}_{\mathcal{P}}\varphi$ or $\mathcal{F}_{\mathcal{P}}\varphi \subseteq \mathcal{F}_{\mathcal{P}}\phi$.

Although a process is defined as a partially ordered *multiset* of localized events, for simplicity, Meadows and Pavlovic assume that each event takes place at most once. We therefore write $e_{1P} \to e_{2Q}$ to denote that there are (time points)

[1] $\mathsf{send}(t)_P$ and $\mathsf{receive}(t)_P$ are written $\langle \cdot t \cdot \rangle_P$ and $(\cdot t \cdot)_P$, resp., in [13,15].

ϕ and φ in $\mathbb{F}$ with $\phi \to \varphi$ such that $\mathcal{F}(\phi) = (e_1, P)$ and $\mathcal{F}(\varphi) = (e_2, Q)$. Informally, this means that e_1 takes place in configuration P before e_2 takes place in configuration Q. The last requirement in Definition 2 implies that a process just orders events *inside* the configuration P (or Q, if $P \subseteq Q$), and hence only define the local behaviors.

A *run* of a process ρ assigns to each receive event $\mathsf{receive}(t)_Q$ a unique *flow* $\mathsf{send}(t)_P \xrightarrow{\tau} \mathsf{receive}(t)_Q$. A run can be seen as a partially ordered (multi)set of localized events that extends the partial order $\to$ in ρ by adding these flows $\mathsf{send}(t)_P \to \mathsf{receive}(t)_Q$. That is, a run extends the internal synchronization in a configuration to the whole network. A network procedure is then defined in [15] as a pair $(\rho, \mathcal{S})$ where ρ is a process and $\mathcal{S}$ is a set of runs of ρ (denoting the "secure" runs).

Procedure Derivation Logic. *Procedure derivation logic* (PDL) [13] is a logic for reasoning about security properties in actor-network procedures. The reasoning of protocol participants is concerned mostly with the order of events in a protocol run. A PDL statement has the form $A : \Phi$, where $A \in \mathcal{J}$ is a participant, and Φ is a predicate asserted by A. The predicate Φ is formed by applying the usual quantifiers and logical connectives (we write $\Longrightarrow$ for implication) to the atomic predicates, which can be: e_P, meaning "the (localized) event e_P happened," or $e_P \to {e'}_Q$, meaning "the event e_P happened before the event ${e'}_Q$". In PDL, the valid statements are derived from the few "generic" PDL axioms, the protocol specification, and protocol-specific assumptions.

One of the generic PDL axioms says that any message that is received must have been sent. That is, if the principal $©P$ controlling P knows $\mathsf{receive}(t)_P$, this principal also knows that there was a corresponding send event $\mathsf{send}(t)$ by some configuration X:

$$©P : \mathsf{receive}(t)_P \Longrightarrow \exists X.\ \mathsf{send}(t)_X \to \mathsf{receive}(t)_P$$

Other PDL axioms axiomatize freshly generated random numbers and continuous flows. In addition, the user can axiomatize her own assumptions about her system. The paper [15] shows many examples of the use of PDL.

3 ANPs with Explicit Device Capabilities

Different devices taking part in a security ceremony can have different capabilities. For example, a security ceremony could include smart cards, biometric devices such as fingerprint readers or iris scanners, a fob device used in online banking to generate one-time passwords, different kids of human users (superuser, standard user, amateur user), computers, and so on.

A security ceremony including many such devices could involve a smart card (or passport) which stores some biometric data of a user. When the user swipes the smart card/passport, the smart card reader sends the biometric data to a central computer, and a biometric device such as a face recognition system takes

a photo of the human user and sends a hash of that information to a central computer. If the biometric data on the smart card and the one taken by the biometric device match, and everything else is OK according to the computer, the user is allowed to enter a certain area/country.

These devices have different capabilities: a fingerprint reader can generate a number by reading your fingerprint, whereas a human or computer cannot; a computer can encrypt and decrypt messages, and a human cannot; only the e-banking one-time password generator can generate one-time passwords; etc.

In this section we extend ANPs to explicitly define and include the capabilities of the different actors in a security ceremony. The two main reasons for making the capabilities explicit are:

- Specification: making explicit the capabilities of nodes in a ceremony.
- Most importantly, knowing the capabilities of the nodes is necessary to reason about the (dynamically evolving) knowledge of the participants (e.g., a node that cannot decrypt messages cannot know/obtain the plaintext from an encrypted message), as well as reasoning under what circumstances certain runs are possible.

In this section we first show how the different capabilities of different devices can be given as functions in an algebraic theory (Σ, E). We then define an *actor-network procedure with explicit capabilities* (ANP-C) as an actor-network procedure with an associated map from nodes to sets of operations/capabilities. Finally, to make the reasoning about obtained knowledge in Sect. 4 simpler, and in general to make the knowledge obtained or created explicit, ANP-Cs add two new kinds of events to ANPs: *create* event use a node's capability and current knowledge to create new informations, and *learning* events models explicitly obtaining knowledge from other pieces of knowledge. The learning event makes it possible to reason about secrecy and authentication; for example, secrecy means that an intruder cannot obtain certain information m *from an overheard encrypted message*—it does not mean that the intruder does not know m. Therefore, just relying on knowledge is not enough to reason about secrecy; we need to make the learning *from* something explicit.

3.1 Specifying Device Capabilities

We show in this section how different capabilities that devices may have can be given as functions in the algebraic theory (Σ, E) of ANP operations.

Smart Cards. A smart card is a small device that typically can:

- Send and receive information to/from a smart card reader.
- Store (and possibly update) data, such as, e.g., the identity and credentials of a user, a PIN code, the remaining amount of money on the card, and the smart card's public key and private key.
- Encrypt and decrypt data using its private and public keys.

We can specify public-key encryption/decryption as the following algebraic theory, written in the style of the Maude language [7], where the keyword `op` introduces an operator/function, and `eq` introduces an equation:

```
sorts Node Msg EncMsg PbKey PvKey Key .
op pv : Node -> Key .
op enc : Msg PvKey -> EncMsg .
vars X : Msg .
eq dec(enc(X,pb(Y)),pv(Y)) = X .
```

```
subsort  PbKey PvKey < Key .
op pb : Node -> Key .
op dec : EncMsg PbKey  -> Msg .
var Y : Node .
eq dec(enc(X,pv(Y)),pb(Y)) = X .
```

where the sorts `Msg`, `EncMsg`, `PbKey`, `PvKey`, `Key`, and `Node` denote, respectively, messages, encrypted messages, private keys, public keys, keys in general (including both public and private keys), and node identities. $pv(n)$ and $pb(n)$ denote, respectively, the private key of n and the public key of n. Finally, *enc* and *dec* denote public-key encryption and decryption, respectively.

Biometric Devices. A biometric device is an authentication device that verifies the identity of a person based on physiological or behavioral characteristics, such as fingerprints, facial or iris images, and/or voice recognition. A biometric device compares the pre-stored biometric information about the user[2] with the biometric information captured by the sensor of the device during the authentication process. In addition to authenticating a person, biometric keys are also used to encrypt/decrypt sensitive information, for example in smart phones.

The following operations define the capability of turning an "image" (of a person's iris or fingerprint) into biometric data, as well as an operation for checking whether the biometric data of two "images" refer to the same person:

```
sorts Image BioData .
ops fingerPrint irisScan ... : Image -> BioData .
op compare : BioData BioData -> Bool .
```

If biometric data are also used for, say, shared-key encryption, there is an operation *bioKey* that generates a shared key from biometric data; we also axiomatize shared-key cryptography with shared-key encryption and decryption operations *skEnc* and *skDec*:

```
sorts  SharedKey  Key .
op bioKey : BioData -> SharedKey .
op skEnc : Msg SharedKey -> EncMsg .
var SK : SharedKey .
```

```
subsort SharedKey < Key .

op skDec : EncMsg SharedKey -> Msg .
eq skDec(skEnc(X,SK),SK) = X .
```

One-Time PIN Generators. A one-time PIN generator is a device that generates a sequence of "random" numbers used for example in online banking as well as in online services like Google, Facebook, or Dropbox. These devices use a formula that generates pseudo-random numbers based on a seed, e.g., a shared key (such as the device serial number) and the moment in time in which the

[2] The biometric information of the user can be pre-stored at the device itself, e.g., a phone with a biometric sensor, or in an external support such as a passport.

transaction/operation is performed. Since the device (whose owner must push a button or perform an action to activate the device and generate the random value) and the entity at which the user wants to be authenticated both know the seed and the time, both can obtain the same number and hence (partially) authenticate the user. Alternatively, the generated random is instead a function of the seed and the previous random number (or a counter). Since we use an *untimed* framework, we can only define the second option:

```
sort Seed .    subsort Seed < Nat .    op pin : Seed Nat -> Nat .
```

Our longer report [10] defines many more operations used in security ceremonies.

3.2 Actor-Network Procedures with Capabilities

We define an *ANP with capabilities* (ANP-C) to be a pair $(\mathcal{A}, \mathbb{C})$ where $\mathcal{A}$ is an ANP and $\mathbb{C}$ assigns to each node n in $\mathcal{A}$ its capabilities:

Definition 3. *An* ANP with capabilities *(ANP-C) is a pair* $(\mathcal{A}, \mathbb{C})$ *where:*

- $\mathcal{A} = (\mathcal{J}, \mathcal{N}, \mathcal{P}, \mathcal{C}, \Theta, \delta, \varrho, \vartheta, \mathcal{F})$ *is an ANP such that the different capabilities of the devices and their algebraic properties are included in its underlying algebraic theory* (Σ, E)*, and*
- $\mathbb{C}$ *is a* capability distribution $\mathbb{C} : \mathcal{N} \to \wp(\Sigma)$ *assigning to each node* n *in* $\mathcal{A}$ *its capabilities.*

3.3 Learn and Create Events

As mentioned, to reason about secrecy (what did a bad guy learn by overhearing a message M?), we need some way of saying that someone learnt a particular piece if information from a certain message. Just reasoning about the knowledge of the intruder is not sufficient, since the intruder may know the secret from before, but could not learn it *from the overheard message*. We therefore introduce a new type of event, called a *learning event*, which has the form

$$\mathsf{apply}\ op\ \mathsf{to}\ t\ \mathsf{toLearn}\ t',$$

where op is a function in our signature ($op \in \Sigma$) and $t, t' \in \mathcal{T}_\Sigma$ are two Σ-terms. In this event, an actor which has the capability to perform the operation op applies op to the term t (which could be the overheard message) and learns t'. This event may take additional parameters $u_1, \ldots, u_n$; the actor performing such a learning event should already know t and $u_1, \ldots, u_n$, and $op(t, u_1, \ldots, u_n) =_E t'$; that is $op(t, u_1, \ldots, u_n)$ and t' are equivalent terms in the equational theory (Σ, E). For example, an intruder that has overheard (and hence knows) a message $skEnc(m, sk)$, knows the (shared) encryption key sk, and has the capability to shared-key decrypt messages, can then perform the learning event $\mathsf{apply}\ skDec\ \mathsf{to}\ skEnc(m, sk)\ \mathsf{toLearn}\ m$ to learn m from $skEnc(m, sk)$, since $skDec(skEnc(m, sk), sk) =_E m$.

The creation of a term t in ANP, used e.g., for creating fresh nonces, in [15] does not take into account the capabilities of the node which creates a term. We therefore define a new kind of *create event* which makes explicit also the capability used to create the event. Such a create even has the form

$$\mathsf{apply}\ op\ \mathsf{to}\ t_1, \ldots, t_n,$$

where $op \in \Sigma$ and $t_1, \ldots, t_n \in \mathcal{T}_\Sigma$ are terms of appropriate sorts. Introducing such an event also facilitates the reasoning about how the knowledge of actors evolves during a run. We therefore assume that nodes create terms before using them; e.g., a node knowing both m and sk should perform the event/create the term $\mathsf{apply}\ skEnc\ \mathsf{to}\ m, sk$ before sending this encrypted message.

3.4 Example: Establishing Shared Keys Using SSL/TSL

Figure 1 shows a graphical representation of an ANP-C for the SSL/TSL procedure involving a user, her smartphone with a fingerprint reader, and a computer belonging to, e.g., the bank, for establishing a secret shared key.

The different nodes are represented as filled circles at the far left of the figure. Each time point ϕ_k is written $\textcircled{k}$ and is decorated with the event $\mathcal{F}_{\mathbb{E}}(\phi_k)$ that takes place at the time point. The actor/configuration that performed the event is the actor to the left of the time point.

A run has an internal synchronization inside the same node/configuration; these are written with a standard arrow $\rightarrow$ between two time points. The external synchronization between two different configurations happens when one configuration receives a message sent by another. We write $\textcircled{i} \overset{m}{\Longrightarrow} \textcircled{k}$ for such a communication event, where m is the message transferred. We do *not* write that the events taking place at ϕ_i and ϕ_j are $\mathsf{send}(m)$ and $\mathsf{receive}(m)$, respectively.

According to [13], a node may perform local operations. Specifically, if a node applies a Boolean operation, then it can branch to different time points, depending on whether the result of the previous operation equals *true* or *false*. We use the arrows $\overset{true}{\dashrightarrow}$ and $\overset{false}{\dashrightarrow}$ in this case. Finally, to save space, some expressions are abbreviated, and given as equations $s = t$.

In the example, the user U can check whether a certificate from the bank C looks OK; the smartphone P can read fingerprints and generate biometric data from them, and can generate shared keys and do public-key cryptography; the computer can compare two (biometric data associated to) two fingerprints and decide whether they belong to the same person (finger?), and can generate certificates for the user. The new capabilities added to (Σ, E) are therefore:

```
sort Cert .  subsort Cert < Msg .  op genCert : Bool Node -> Cert .
op visCheck : Cert -> Bool .       op genSk : Bool Node Node -> SharedKey .
```

The ceremony has the following steps: The user U sends her fingerprint *image* to the smartphone P (time points ϕ_0 and ϕ_1); P then uses the operation *fingerprint* to create the biometric data *fingerprint*(*image*), which is abbreviated

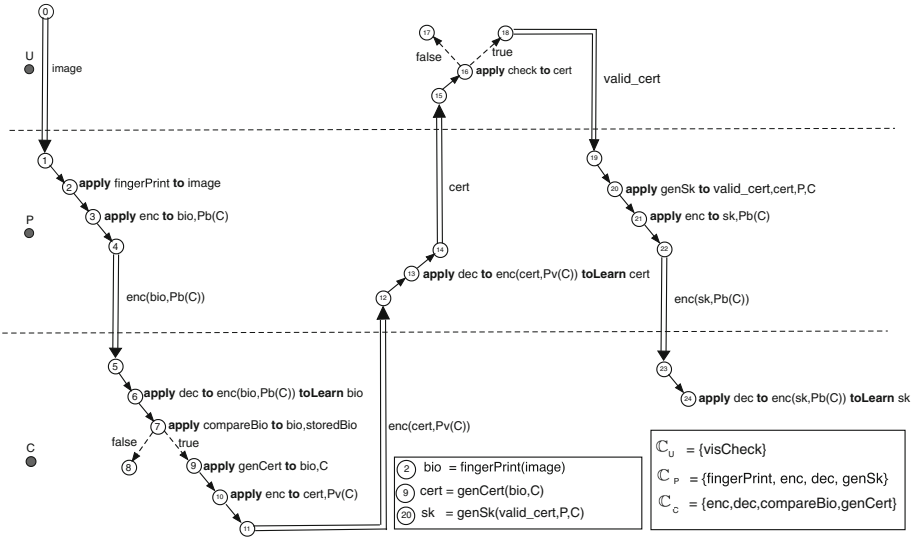

Fig. 1. An ANP-C showing an SSL/TLS procedure involving a user (U), a smartphone with a biometric device (P), and the bank's computer (C).

to *bio* (ϕ_2); P then applies the function *enc* to *bio* and the computer's public key $Pk(C)$ (ϕ_3), and then sends $enc(bio, Pk(C))$ at time point ϕ_4. This message is read by the computer C at time point ϕ_5. C then applies the *dec* function to learn *bio* (ϕ_6). If this received biometric information *bio* equals the bank's stored biometric data *storedBio* (time point ϕ_7), which the bank hopefully knows before (see Sect. 4 for a discussion on defining initial knowledge), we continue to time point ϕ_9, where C applies *genCert* to generate C's certificate, which is encrypted at time point ϕ_{10} and sent to P at time point ϕ_{11}. The smartphone P receives this message (ϕ_{12}), decrypts the message to learn the certificate (ϕ_{13}), and sends/shows the certificate to the user U (ϕ_{14}). The user receives/sees this certificate (ϕ_{15}) and then checks the certificate visually (ϕ_{16}). If the certificate looks good, the user goes to time point ϕ_{18} where she "sends" an OK message to the smartphone. The smartphone P gets this OK "message" (ϕ_{19}), applies *genSk* to generate a shared key between P and C (ϕ_{20}), encrypt this message (ϕ_{21}) and sends this encrypted message to C (ϕ_{22}). Finally, C receives this message at time point ϕ_{23} and decrypts it at ϕ_{24} to learn the shared key.

4 PDL-CK: Reasoning About ANP-Cs and Knowledge

In this section we define a variation of the PDL logic, called PDL-CK, for reasoning about ANP-Cs.

PDL is typically used to reason about *secure runs*, that is, behaviors that we know are possible. However, one of the main goals of making capabilities explicit is exactly to reason about what runs are possible. For example, can

an intruder obtain a secret? More precisely, under what circumstances can the intruder obtain a secret? And under what circumstances is a security ceremony with the desired event actually possible? For example, in the SSL-TSL example above, under what circumstances is the shared key actually established?

Making capabilities explicit goes half ways towards answering these questions: to perform an action (like decrypting a message), a node needs both a certain capability (such as decrypting messages) *and* certain knowledge (such as knowing the decryption key). We therefore propose to reason about ANP-Cs using a logic which takes into account both the capabilities and the (dynamically evolving) knowledge of the participants of the ceremony. We call this logic PDL-CK ("PDL with capabilities and knowledge"). With this logic we can reason about under what circumstances something can happen or a property holds. More precisely, which capabilities and what initial knowledge are needed for a (good or bad) event to take place? That is, in addition to reasoning about events and their temporal relationship as in PDL, PDL-CK allows us to reason also about the knowledge of the actors when the different events take place.

Notation: assuming that each event only takes place once, we denote by $\mathbb{K}_e$ the knowledge of the nodes at the end of the time point at which e takes place.

This section first introduces such global knowledge. Then we introduce the logic PDL-CK and its axioms, before showing examples of reasoning in PDL-CK.

4.1 Knowledge Distributions and Knowledge Histories

Keys, messages, nonces, and so on, are usually modeled as ground terms in the algebra (Σ, E), and are not identified with their actual numerical values. Therefore, we can represent a node's knowledge as a set of Σ-terms. A *knowledge distribution* defines the current knowledge of each node in the network:

Definition 4. *A* knowledge distribution κ *for a set of nodes* $\mathcal{N}$ *is a function* $\kappa : \mathcal{N} \to \wp(\mathcal{T}_\Sigma)$ *assigning to each node* n *the set* $\kappa(n)$ *of ground terms it knows.*

A *knowledge history* assigns such a knowledge distribution to each time point in the procedure:

Definition 5. *Given a process* $\mathcal{F}$ *with an underlying structure time* $(\mathbb{F}, \to)$*, a* knowledge history $\mathbb{K}$ *for* $\mathcal{F}$ *is a function* $\mathbb{K} : \mathbb{F} \to (\mathcal{N} \to \wp(\mathcal{T}_\Sigma))$ *that assigns to each time point* $\varphi \in \mathbb{F}$ *a knowledge distribution. Furthermore, the function* $\mathbb{K}$ *must be monotonic w.r.t.* $\to$*, i.e.,* $\varphi_i \to \varphi_j \Rightarrow \mathbb{K}(\varphi_i)(n) \subseteq \mathbb{K}(\varphi_j)(n)$ *for all* n*.*

Intuitively, $\mathbb{K}(\phi)$ denotes the knowledge of the different actors at the "end" of time point ϕ; that is, it includes knowledge acquired at time point ϕ.

Notation. Under the usual assumption that an event e takes place at most once, at time point ϕ, we write $\mathbb{K}_e$ for the knowledge distribution $\mathbb{K}(\phi)$.

The initial knowledge of the nodes plays a key role. We denote by $\mathbb{K}_{init}$ the *initial knowledge* in a knowledge history $\mathbb{K}$. Mathematically, this can be seen as adding a new event *init* which takes place at a new time point ϕ_{init} so that $init \to \phi$ for any other time point ϕ (in the run).

Example 1. A possible initial knowledge of the ANP-C in Fig. 1 could be

$$\mathbb{K}_{init}(U) = \{image, ok\}$$
$$\mathbb{K}_{init}(P) = \{Pb(C), Pv(P), Pb(P)\}$$
$$\mathbb{K}_{init}(C) = \{true, false, Pb(C), Pv(C), Pb(P), storedBio\}.$$

For example, the computer initially knows a (pre-stored) biometric key of the user. This history at time point ϕ_9, after the event apply *genCert* to $true, C$, is:

$$\mathbb{K}_{\mathsf{apply}\ldots}(U) = \mathbb{K}_{init}(U)$$
$$\mathbb{K}_{\mathsf{apply}\ldots}(P) = \mathbb{K}_{init}(P) \cup \{bio(= fingerprint(image)), image, enc(bio, Pb(C))\}$$
$$\mathbb{K}_{\mathsf{apply}\ldots}(C) = \mathbb{K}_{init}(C) \cup \{bio, enc(bio, Pb(C)), cert\}.$$

4.2 PDL-CK

The *procedure derivation logic with capabilities and knowledge* (PDL-CK) modifies and extends PDL to reason not only about the temporal order of events, but also of the participants' knowledge at each point in time. To simplify the exposition, in the rest of this paper we assume that we do not have "composite" configurations. That is, any configuration is a single node.

The difference between PDL and PDL-CK is that PDL reasons about an ANP $\mathcal{A}$, whereas PDL-CK reasons about a pair $((\mathcal{A}, \mathbb{C}), \mathbb{K})$, where $(\mathcal{A}, \mathbb{C})$ is an ANP-C and $\mathbb{K}$ is a knowledge history for $\mathcal{A}$. (In practice, we are interested in whether *there exists* a $\mathbb{K}$ such that $\Phi(\mathbb{K})$ holds for a given $(\mathcal{A}, \mathbb{C})$).

Therefore, e_p (the event e took place at p) and $e_{1p} \rightarrow e_{2q}$ (the event e_1 took place at p before e_2 took place at q) are still atomic propositions in PDL-CK; the difference is that the PDL-CK formulas also may include $\mathbb{C}$ and $\mathbb{K}$. However, the axioms in PDL are replaced with others to take also the capacities and the knowledge into account.

Some generic PDL-CK axioms for global (bird's-eye view) reasoning are given in Table 1 (where we use the symbol $\Longrightarrow$ for logical implication). The axiom Equality says that if p knows t_1, and t_1 and t_2 are E-equivalent, then p must also know t_2. The axiom Send says that if p sends z, then p must have known z before, and that nothing new was learnt anywhere as a result of performing this action. The Receive axiom says that if p receives z, then: p knows z at the end of this time point, that the only thing learnt globally during this time point is that z learnt p, and the receive event must have been preceded by the corresponding send event at some actor q. The axiom Learn says that if p applies O to a term u to learn t, then p knows t at (the end of) this time point, that p has the capability to perform O, that p knows u before, that there are additional parameter values $u_1, \ldots, u_n$ previously known by p so that $O(u, u_1, \ldots, u_n) =_E t$, and that the only thing learnt by performing this event is that p learnt t. Likewise, Creation says that if you "generate" a new term $O(t_1, \ldots, t_n)$, then you have learnt this new term, must have the capability O and must know $t_1, \ldots, t_n$ earlier, and that the only new knowledge added is that p has learnt the generated term. Finally, we add new axioms for test-and-branch.

(Note that the nodes do not learn anything by taking a branch. We can encode such knowledge by adding two new capabilities $valid, inValid : \texttt{Bool} \to Flag$ and transform a branch $boolExp \overset{true}{\dashrightarrow} e'$ to $e \overset{true}{\dashrightarrow} \mathsf{apply}\ valid\ \mathsf{to}\ boolExp \to e'$, and transforming $boolExp \overset{false}{\dashrightarrow} e''$ to $e \overset{false}{\dashrightarrow} \mathsf{apply}\ inValid\ \mathsf{to}\ boolExp \to e''$).

Table 1. PDL-CK Axioms.

Equality	$\forall t_1, t_2, e, p.\ t_1 =_E t_2 \implies t_1 \in \mathbb{K}_e(p) \Leftrightarrow t_2 \in \mathbb{K}_e(p)$
Send	$\mathsf{send}(z)_p \implies p\ \mathbf{knows}\ z\ \mathbf{before}\ \mathsf{send}(z)_p\ \wedge\ nothingLearnt(\mathsf{send}(z)_p)$
Receive	$\mathsf{receive}(z)_p \implies z \in \mathbb{K}_{\mathsf{receive}(z)}(p)\ \wedge\ onlyLearnt(\mathsf{receive}(z)_p, z, p)$ $\wedge\ (\exists q.\ \mathsf{send}(z)_q \to \mathsf{receive}(z)_p)$
Learn	$(\mathsf{apply}\ O\ \mathsf{to}\ u\ \mathsf{toLearn}\ t)_p$ $\implies\ t \in \mathbb{K}_{(\mathsf{apply}\ O\ \mathsf{to}\ u\ \mathsf{toLearn}\ t)}(p)\ \wedge\ O \in \mathbb{C}(p)$ $\wedge\ p\ \mathbf{knows}\ u\ \mathbf{before}\ (\mathsf{apply}\ O\ \mathsf{to}\ u\ \mathsf{toLearn}\ t)_p$ $\wedge\ \exists\ u_1, \ldots, u_n.\ (O(u, u_1, \ldots, u_n) =_E t$ $\wedge\ \forall_{1 \le i \le n}.\ p\ \mathbf{knows}\ u_i\ \mathbf{before}\ (\mathsf{apply}\ O\ \mathsf{to}\ u\ \mathsf{toLearn}\ t)_p)$ $\wedge\ onlyLearnt((\mathsf{apply}\ O\ \mathsf{to}\ u\ \mathsf{toLearn}\ t)_p, t, p)$
Creation	$(\mathsf{apply}\ O\ \mathsf{to}\ t_1, \ldots, t_n)_p$ $\implies\ O(t_1, \ldots, t_n) \in \mathbb{K}_{(\mathsf{apply}\ O\ \mathsf{to}\ t_1, \ldots, t_n)}(p)\ \wedge\ O \in \mathbb{C}(p)$ $\wedge\ \forall_{1 \le i \le n}.\ p\ \mathbf{knows}\ t_i\ \mathbf{before}\ (\mathsf{apply}\ O\ \mathsf{to}\ t_1, \ldots, t_n)_p$ $\wedge\ onlyLearnt((\mathsf{apply}\ O\ \mathsf{to}\ t_1, \ldots, t_n)_p, O(t_1, \ldots, t_n), p)$
Branch.True	$bExpr_p \overset{true}{\dashrightarrow} e_q \implies (e_q \implies bExpr)\ \wedge\ nothingLearnt(bExpr_p)$
Branch.False	$bExpr_p \overset{false}{\dashrightarrow} e_q \implies (e_q \implies \neg bExpr)\ \wedge\ nothingLearnt(bExpr_p)$

The formulas $p\ \mathbf{knows}\ z\ \mathbf{before}\ e_q$ (p must know z before then localized event e_q takes place), $onlyLearnt(e_q, t, p)$ (the only knowledge added to the system as a result of performing the event e is that p learnt t), and $nothingLearnt(e_q)$ (nothing was learnt by performing the event e) are defined as follows:

$$p\ \mathbf{knows}\ z\ \mathbf{before}\ e_q\ \triangleq\ z \in \mathbb{K}_{init}(p)\ \vee\ \exists e', r.\ (e'_r \to e_q\ \wedge\ z \in \mathbb{K}_{e'}(p))$$

$$onlyLearnt(e_q, z, p)\ \triangleq\ \forall x, r.\ x \in \mathbb{K}_e(r) \implies (x =_E z\ \wedge\ p = r)\ \vee\ r\ \mathbf{knows}\ x\ \mathbf{before}\ e_q$$

$$nothingLearnt(e_p)\ \triangleq\ \forall t, q.\ t \in \mathbb{K}_e(q) \implies q\ \mathbf{knows}\ t\ \mathbf{before}\ e_p.$$

4.3 Examples

This section gives some small examples of reasoning with PDL-CK.

Example 2. Figure 2 shows an ANP-C run where a computer C_A sends a shared-key encrypted message $skEnc(msg_1, sk_1)$ to a smart card reader R which decrypts the message (e.g., to receive an update). The encrypted message is overheard/received by a Trojan virus T. The algebraic theory (Σ, E) of operations is (a subset of) the one in Sect. 3.1. The run of this ANP-C is as follows:

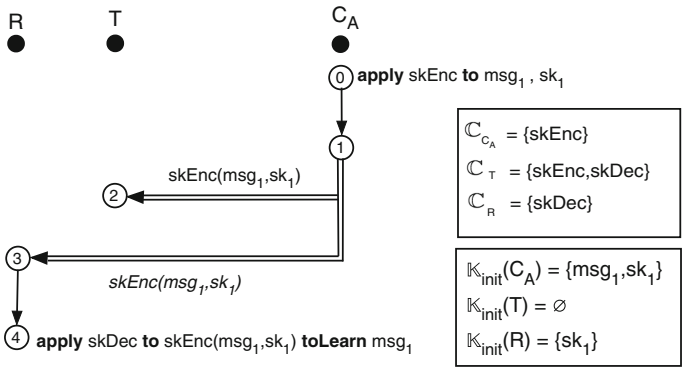

Fig. 2. An actor-network run for updating information.

0. The computer C_A creates the encrypted message $skEnc(msg_1, sk_1)$.
1. C_A sends the shared-key encrypted message $skEnc(msg_1, sk_1)$.
2. The trojan virus T receives/overhears the message $skEnc(msg_1, sk_1)$.
3. The smart card reader R receives the message $skEnc(msg_1, sk_1)$.
4. R learns sk_1 from the message $skEnc(msg_1, sk_1)$ by applying the $skDec$ capability with parameter sk_1.

The desired property is that if R learns msg_1 from the shared-key encrypted message $skEnc(msg_1, sk_1)$, then *initially* R knows the shared key, $sk_1 \in \mathbb{K}_{init}(R)$, and R can perform the shared-key decryption operation, $skDec \in \mathbb{C}(R)$:

$$(\mathsf{apply}\ skDec\ \mathsf{to}\ skEnc(msg_1, sk_1)\ \mathsf{toLearn}\ msg_1)_R \implies skDec \in \mathbb{C}(R)\ \wedge\ sk_1 \in \mathbb{K}_{init}(R).$$

Example 3. An interesting property to prove about the ceremony in Fig. 1 is that if a shared key is established between the phone and the bank, then:

- the bank *initially* knows U's biometric data: $storedBio \in \mathbb{K}_{init}(C)$; and
- the biometric data of the user matches the biometric data stored by the bank: $compare(fingerPrint(image), storedBio)$.

That is, the formula to prove is:

$$(\mathsf{apply}\ dec\ \mathsf{to}\ enc(sk, Pb(C))\ \mathsf{toLearn}\ sk)_C \implies storedBio \in \mathbb{K}_{init}(C) \wedge compare(fingerPrint(image), storedBio).$$

Our longer report [10] contains many more examples, including reasoning about secrecy and authentication.

5 Related Work

Most papers on ANP and PDL [5,11,12,14] show how ANP and PDL can be applied to reason about *protocol runs*, but do not use a dynamic representation of the knowledge of the different actors, and do not differentiate between the capabilities of different devices. Fiadeiro et al. [9] use ANP to describe a logic for reasoning about the different states and state transitions of an actor network. In this formalization, the different actors, interaction channels and knowledge are static, whereas in our work, the knowledge of each actor evolves during the execution of the run.

Basin et al. [3] use a node topology for the analysis of security protocols that specifies the node's capabilities, initial knowledge, honesty, and available communication channels. They group the different agents in three different groups based on their capabilities and knowledge, i.e., honest, dishonest and restricted, but do not distinguish between different types of restricted agents (human participants) and their capabilities and knowledge. Their security ceremony formalization is linked to the Tamarin tool, whereas our work is not yet linked to a tool.

Bella and Coles-Kemp [4] present a security ceremony model focused on the human-computer interoperation, whereas our framework deals with the interactions between different kinds of devices and humans, and we explicitly define the different participants of the security ceremony (human and non-humans) whereas they use a general model to represent the different actors.

Radke et al. [17] define an attacker model for security ceremonies in which they use a recognize function to formalize human capabilities. In contrast to our work, they do not focus on representing knowledge (explicitly).

Finally, Belfanz et al. [2] and Creese et al. [18] define different threat models in different communications channels, but do not define the capabilities nor the knowledge or the participants. We do not take into account channel features, but we explicitly define the different participants and analyze a communication process independently of the kind of channel used.

6 Concluding Remarks

Many different kinds of devices and humans, with different rights and capabilities, participate in today's security processes. We have therefore extended the well-known and general model of security ceremonies by Meadows and Pavlovic by explicitly representing the user-definable capabilities of each actor. We have also defined a new logic, PDL-CK, to reason about our models. This logic allows reasoning about the dynamically changing knowledge of the participants. We believe that this is the first formalism for security ceremonies that makes explicit the different user-definable capabilities of the participants. PDL-CK allows us to reason, for example, under what circumstances (i.e., initial knowledge and capabilities) certain actions, such as decrypting a message, can be performed.

Much work remains. Like the work of Meadows and Pavlovic that we extend, our model does not yet have an executable formal semantics, and hence no

tool support. We should develop verification strategies and should apply our methods on state-of-the-art applications. We should also consider non-monotonic knowledge and dynamic node capabilities.

References

1. Anlauff, M., Pavlovic, D., Waldinger, R., Westfold, S.: Proving authentication properties in the Protocol Derivation Assistant. In: FCS-ARSPA 2006. ACM (2006)
2. Balfanz, D., Smetters, D.K., Stewart, P., Wong, H.C.: Talking to strangers: authentication in ad-hoc wireless networks. In: NDSS 2002. The Internet Society (2002)
3. Basin, D.A., Radomirovic, S., Schläpfer, M.: A complete characterization of secure human-server communication. In: CSF 2015. IEEE Computer Society (2015)
4. Bella, G., Coles-Kemp, L.: Layered analysis of security ceremonies. In: Gritzalis, D., Furnell, S., Theoharidou, M. (eds.) SEC 2012. IAICT, vol. 376, pp. 273–286. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30436-1_23
5. Cervesato, I., Meadows, C.A., Pavlovic, D.: An encapsulated authentication logic for reasoning about key distribution protocols. In: CSFW 2005, vol. 1. IEEE (2005)
6. Chen, S., Sasse, R., Meseguer, J., Wang, H., Wang, Y.M.: A systematic approach to uncover security flaws in GUI logic. In: IEEE SSP 2007. IEEE (2007)
7. Clavel, M., et al.: All About Maude - A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic. LNCS, vol. 4350. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-71999-1
8. Ellison, C.: Ceremony design and analysis. IACR Cryptology ePrint Archive (2007)
9. Fiadeiro, J., Ţuţu, I., Lopes, A., Pavlovic, D.: Logics for actor networks: a case study in constrained hybridization. In: Madeira, A., Benevides, M. (eds.) DALI 2017. LNCS, vol. 10669, pp. 98–114. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-73579-5_7
10. Gonzalez-Burgueño, A., Ölveczky, P.C.: Formalizing and analyzing security ceremonies with heterogeneous devices in ANP and PDL (2018). http://folk.uio.no/antonigo/Security_Ceremonies_Heterogeneous_Devices.pdf
11. Meadows, C., Pavlovic, D.: Deriving, attacking and defending the GDOI protocol. In: Samarati, P., Ryan, P., Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 53–72. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30108-0_4
12. Meadows, C., Pavlovic, D.: Formalizing physical security procedures. In: Jøsang, A., Samarati, P., Petrocchi, M. (eds.) STM 2012. LNCS, vol. 7783, pp. 193–208. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38004-4_13
13. Pavlovic, D., Meadows, C.: Actor-network procedures. In: Ramanujam, R., Ramaswamy, S. (eds.) ICDCIT 2012. LNCS, vol. 7154, pp. 7–26. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28073-3_2
14. Pavlovic, D., Meadows, C.: Deriving secrecy in key establishment protocols. In: Gollmann, D., Meier, J., Sabelfeld, A. (eds.) ESORICS 2006. LNCS, vol. 4189, pp. 384–403. Springer, Heidelberg (2006). https://doi.org/10.1007/11863908_24
15. Pavlovic, D., Meadows, C.: Actor-network procedures: modeling multi-factor authentication, device pairing, social interactions. CoRR abs/1106.0706 (2011)
16. Pavlovic, D., Meadows, C.: Deriving ephemeral authentication using channel axioms. In: Christianson, B., Malcolm, J.A., Matyáš, V., Roe, M. (eds.) Security Protocols 2009. LNCS, vol. 7028, pp. 240–261. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36213-2_27

17. Radke, K., Boyd, C., Nieto, J.G., Manulis, M., Stebila, D.: Formalising human recognition: a fundamental building block for security proofs. In: AISC 2014. CRPIT, vol. 149. Australian Computer Society (2014)
18. Roscoe, A.W., Goldsmith, M., Creese, S.J., Zakiuddin, I.: The attacker in ubiquitous computing environments: formalising the threat model. In: FAST 2003 (2003)