

Calculus of Cryptographic Communication

Johannes Borgström¹, Simon Kramer², and Uwe Nestmann¹

¹ EECS, Technical University of Berlin, Germany

² Ecole Polytechnique Fédérale de Lausanne (EPFL)

simon.kramer@a3.epfl.ch

Abstract. We define C^3 , a model-based formalism that is one half of a whole framework for the modelling, specification, and verification of cryptographic protocols. C^3 consists of a language of distributed processes and an associated (SOS) notion of concurrent execution. The other, co-designed, half of our framework is a property-based formalism, i.e., a logic, for the specification and verification of such protocols, called CPL. The three primary features of the co-design of C^3 and CPL are that (1) C^3 's notion of execution is a temporal accessibility relation for CPL's temporal modalities, (2) C^3 's notion of observational equivalence and CPL's notion of propositional knowledge have a common definitional basis, namely an epistemic accessibility relation defined in terms of structurally indistinguishable protocol histories, and (3) execution constraints of C^3 -processes are checkable via CPL-satisfaction. Moreover, this co-design permits separation of the concerns of protocol and property description, within the same framework. Other important features of C^3 are explicit out-of-band communication and history-based key lookup, which together give a concrete foundation for authentication and key establishment protocols.

Keywords domain-specific process calculi, cryptographic protocols, formal modelling, model-based specification and verification

1 Introduction

The design of cryptographic protocols is a non-trivial problem. It has even been called “programming Satan’s computer” [1]. These protocols are commonly described by means of an informal notation, and resulting protocol descriptions are commonly called *protocol narrations*, which essentially are global-view transcripts of successful protocol runs [2, 3].

1.1 Motivation

Protocol narrations are not precise enough [4, Page 2–3]:

There is no specification of internal actions of principals. In many protocols the internal actions are fairly obvious; [...] But in others

there are numerous alternatives and the choice can have security-critical relevance.

The implicitness of message reception checks and the involved “magical” knowledge about intended protocol roles and decryption keys leave much room for misunderstanding, incorrect interpretation, and protocol bugs. To tackle this problem, a number of formalisms have been proposed over the last decade. However, we believe that these formalisms have certain important deficiencies.

First, we seek a formalism that not only supports model-based protocol specification and verification, but that is also complementary to a corresponding property-based formalism, i.e., a *logic*, such as BAN [5] or, in our case, the more expressive CPL [6, 7]. An advantage of property-based over model-based protocol specifications is that the former are model-independent. As such, they can be understood, analysed, and compared without explicit reference to a particular model. Technically speaking, the complementary relation between C^3 and CPL is that (1) C^3 's notion of execution is a temporal accessibility relation for CPL's temporal modalities, (2) C^3 's notion of observational equivalence and CPL's notion of propositional knowledge have a common definitional basis, namely an epistemic accessibility relation defined in terms of structurally indistinguishable protocol histories, and (3) execution constraints of C^3 -processes are checkable via CPL-satisfaction.

Second, the formalism must model cryptographic communication and computation at a relevant *level of abstraction*. We aim at a compromise between the amount of detail required to fully understand the logical structure of a cryptographic protocol, and the amount of detail required to address merely technological variations. Cryptographically relevant actions must be expressible succinctly and without reference to implicit magic, but they should also employ the lightest possible technical means and abstract away from non-cryptographic matters. For example, it is crucial that secure communication is well-founded: “As a matter of general principle it is not possible to establish an authenticated session key without existing secure channels already being available.” [4]. Thus, the formalism must have a mechanism for modelling *out-of-band communication* such as trusted couriers, personal contact between communicating parties, and dedicated communication links.

Third, the formalism should be easily *extensible*. For example, sub-protocols (“vertical” composition), and multiple-session scenarios (“horizontal” composition) should all be expressible via “syntactic sugar”. More substantial extensions to the formalism involving modification of the se-

mantics should be backwards compatible (e.g., [8], which is a backwards compatible extension with real time of our present, core calculus).

1.2 Related work

At the current stage, we emphasise ease and clarity of modelling rather than (automated) verification. And our target audience is academia rather than industry. Industry demands push-button tools, i.e., software that pragmatically integrates *programming* and/or *logical formalisms* with *automation methods* (equivalence checking, resp. model checking and theorem proving) and *automation techniques* (rewriting, unification) into practical *engineering frameworks*. A number of such engineering frameworks have been proposed so far, e.g., Athena [9] based on strand spaces and integrating model checking with theorem proving, the AVISPA tool [10] based on a fragment of Lamport’s Temporal Logic of Actions and integrating model checking with rewriting, the ProVerif tool [11] based on Prolog (unification), and the various tools of the EVA project [12].

The principal formalisms related to C^3 are: CPPL [13] based on strand spaces [14]; process calculi such as CSP [15], the Spi Calculus [16], and SPL [17]; and a rich, but not explicitly named model [18] (which we will refer to as CMOS). However, with the important exception of the process model of PCL³ [19], none of the design formalisms for cryptographic protocols we are aware of has been exhibited to have a strong (i.e., model-theoretic) connection to a cryptographic *logic*. Further, none of them explicitly addresses key lookup (magic access to externally defined global functions), nor well-foundedness of secure communication.

On the other hand, CPPL [13] meets several of our requirements. It emphasises programmer usability with sub-protocols and pattern matching. And it provides a formal model of protocol behaviour via strand spaces. However, it is targeted at a particular style of verification, different from ours (satisfaction of CPL-formulae by C^3 -processes), namely the rely/guarantee style: the overall correctness of a protocol results from the truth of individual rely formulae and the successful run-time checking of individual guarantee formulae. Finally, CPPL seems to shift the concerns of well-foundedness of communication and key lookup to a cryptographic library outside the formalism.

³ For comparison, PCL is in the tradition of dynamic — more precisely mono-dimensional (i.e., time) poly-modal (action parameters) — logic. In contrast, CPL is in the tradition of temporal — more precisely, poly-dimensional (i.e., norms, knowledge, space, time) mono-modal — logic. We believe that the two perspectives are complementary.

CMOS [18] embodies a rich collection of explicit concepts, including protocol roles (with local variables), a parametric intruder model, initial knowledge for participants, run identifiers (similar to session identifiers), and pattern matching. In CMOS, security goals are expressed as properties over protocol traces that are generated as sequences of protocol events by the operational semantics. In turn, those trace properties are based on a fine-grained instrumentation of the protocol specification with explicit security claims. CMOS assumes global functions for key lookup, keeping that particular issue at the meta-level. In comparison, CMOS seems quite more heavy-weight, especially w.r.t. the (subjectively) rather complicated handling of instantiations of roles to runs.

The Spi Calculus [16] and SPL [17] are domain-specific variants of the Pi Calculus by the addition of idealised cryptographic primitives. SPL is simpler in that it replaces channel-based directed communication by a public medium. The general-purpose value-passing calculus CSP [15] was used as a variant that includes abstract types in order to deal with encryption and decryption within the underlying message language. When using a process calculus more or less off-the-shelf, the idea is to reuse as much as possible of its available theory. Less advantageous is the fact that many modelling concepts then need to be encoded. This goes clearly against our requirement of explicitness at a suitable level of abstraction. For example, participant names (like `Alice`) and role names (like `Initiator`) are usually confused and jointly represented as process constants. This is tractable as long as one is interested mostly in single-session scenarios. However, the distinction between multiple sessions of multiple protocol roles by multiple participants in changing roles must then usually be coded up using indices of parallel composition operators. However carefully this coding is done, such an indirect way can make it hard to understand the overall structure of non-trivial multiple-session scenarios, and it can make it very hard to track the acquired knowledge of participants that take part in several protocols concurrently. Moreover, it is hardly feasible to compose security arguments for multiple-session scenarios from security properties of their parts.

In summary, our formalism, as introduced in the next section, is related to CPPL and CMOS but differs in important respects. It makes different decisions concerning the explicitness of concepts, and is closely associated to a purely property-based formalism, while carrying out the overall formalisation of the model within a process calculus setting. In our view, process calculi are ideal *design formalisms*. That is, they offer — due to their minimalist, linguistic representation of modelling concepts

and their mathematical, operational semantics — a win-win situation between the pedantic rigour of machine models and the practical usability of programming formalisms.

2 Definition

C^3 consists of a language of distributed processes and an associated notion of concurrent execution in the style of structural operational semantics (SOS). Its *modelling hypotheses* are those of abstract ideal cryptography and interleaving concurrency. Cryptography is abstract in the sense that communicated information atoms (names) are logical constants and communicated information compounds are syntactic terms. We use pattern matching as a linguistic abstraction for cryptographic computation. Cryptography is ideal in the sense that cryptographic primitives are assumed to be perfect, i.e., unbreakable. Process execution engenders the activity of protocol participants and the standard Dolev-Yao [20] adversary Eve, i.e., the generation of a history of legitimate resp. illegitimate protocol events, and the evolution of their respective knowledge computed from that history.

Co-design of C^3 and CPL is manifest in the following ways. First, names in C^3 are logical constants. In contrast, the concept of names of process calculi such as the Pi and the Spi Calculus is hybrid in the sense that their names have both bound-variable character via α -convertibility and logical-constant character via substitutability in free (input) variables. We retain the classical concept of names and work with explicit name generation, in order not to need to introduce an additional, Pitts-style existential quantifier in the logic. Second, execution constraints (e.g., freshness of new names, input guards, key lookup constraints) of C^3 -processes are checkable via CPL-satisfaction. Third, C^3 's notion of execution is a temporal accessibility relation for CPL's temporal modalities. Fourth, C^3 's notion of observational equivalence and CPL's notion of propositional knowledge have a common definitional basis, namely an epistemic accessibility relation defined in terms of structurally indistinguishable protocol histories (cf. Section 2.3). We believe that this co-design will yield logical characterisation results both for C^3 -processes and C^3 's observational equivalence.

2.1 Syntax

Our (sorted) alphabet for information atoms in C^3 is the following.

Definition 1 (Sorted names). Names n are participant names $c, d \in \mathcal{P}$, the adversary's name Eve , symmetric session resp. long-term keys $k \in \mathcal{K}^1 \cup \mathcal{K}^\infty$, private keys $p \in \mathcal{K}^-$, and nonces $x \in \mathcal{X}$ (also used as session identifiers). We assume that given a private key, one can compute the corresponding public key, as in DSA and Elgamal.

The sorts σ corresponding to these kinds of names are P , Adv , K^1 , K^∞ , K^- , and X , respectively. K^1 , K^∞ , K^- , and X are the sorts ς of freshly generable names.

These sorted names (logical constants) are part of the language \mathcal{M} of communicated messages in C^3 . \mathcal{M} contains, besides the usual cryptographic constructions, a distinguished abstract message \blacksquare . This message is a computational artifice to represent the absence of *intelligibility*, just as the number zero is a computational artifice to represent the absence of *quantity*. The abstract message is very useful for doing knowledge-based calculations (e.g., as effectuated by C^3 's observational equivalence, cf. Section 2.3) just as the number zero is very useful (to say the least) for doing number-based calculations. The abstract message is also very useful for adding *backwards-compatible extensions* to the core calculus (e.g., [8]). Symmetric keys may be *compound* for key *agreement* (in addition to mere key *transport*).

Definition 2 (Messages and Message forms). Our protocol messages $M \in \mathcal{M}$ are defined in Table 1. Message forms F are messages with variables $v \in \mathcal{V}$, and are used in process terms where they may instantiate to (transmittable) messages. Sorts of compound messages are macro-definable (cf. Appendix A.2).

$M ::= n$	(names)
\blacksquare	(the abstract message)
p^+	(public keys)
$[M]$	(message hashes)
$\{M\}_M$	(symmetric message ciphers)
$\{M\}_{p^+}^+$	(asymmetric message ciphers)
$\{M\}_p^-$	(signed messages)
(M, M)	(message tuples)

Table 1. Protocol messages

C^3 -processes are parallel compositions of *located threads*. A non-idle thread T located at the participant c and session x , written $c.x[T]$, has either an action prefix or a lookup prefix. The action prefixes $\mathbf{Out}_a F$ and $\mathbf{sOut}_a F$ express insecure (intercepted by the adversary) resp. secure (unobservable by the adversary) output of F to a ; $\mathbf{In} \Pi \mathbf{when} \varphi$ and $\mathbf{sIn}_a \Pi \mathbf{when} \varphi$ express insecure resp. secure input (from a) of a message matching the pattern Π and having the property φ ; and $\mathbf{New}(v : \varsigma, O)$ expresses the generation and binding to the variable v of a fresh name of type ς *tagged* with O , where O (a tuple of participant names) stipulates intended ownership. The lookup prefix $\mathbf{Get}_a(v : \varsigma, O) \mathbf{in}$ expresses the lookup and binding to v of a name of type ς generated by a with ownership tag O .

Our communication model is based on participants rather than channels for separating specification from implementation concerns.

Definition 3 (Processes). C^3 -processes $P \in \mathcal{P}$ are defined in Table 2. There, $a, b \in \mathcal{P} \cup \mathcal{V}$; \circlearrowleft means “freshly generated”, $:$ “has sort”, \mathbf{k} “knows”, and \preceq “is a subterm of”. A process P is *epistemically local* :iff for all located threads $c.x[T]$ in P and for all $a.x \circlearrowleft n.O$ and $a \mathbf{k} F$ in T , $a = c$. Implementation processes must be epistemically local, whereas specification processes need not be. In addition, in implementation processes $\forall v$ may not bind v in any F , whereas in specification processes this need not be.

$$\begin{aligned}
P & ::= c.x[T] \mid P \parallel P \\
T & ::= 1 \mid \pi.T \mid \mathbf{Get}_a(v : \varsigma, O) \mathbf{in} T \\
\pi & ::= \mathbf{Out}_a F \mid \mathbf{sOut}_a F \mid \mathbf{In} \Pi \mathbf{when} \varphi \mid \mathbf{sIn}_a \Pi \mathbf{when} \varphi \mid \mathbf{New}(v : \varsigma, O) \\
\Pi & ::= F \mid (\Pi, \Pi) \mid \{\!\{ \Pi \}\!\}_{\overline{F}} \mid \{\!\{ \Pi \}\!\}_{\overline{F}}^{\pm} \mid \{\!\{ \Pi \}\!\}_{\overline{F}}^{-} \\
\varphi & ::= a.x \circlearrowleft n.O \mid n : \sigma \mid a \mathbf{k} F \mid F \preceq F \mid \neg\varphi \mid \varphi \wedge \varphi \mid \forall v(\varphi)
\end{aligned}$$

Table 2. Protocol processes

Action prefixes π , as opposed to the lookup prefix, generate events when executed. Action prefixes for secure I/O generate unobservable events; they model out-of-band communication such as trusted couriers, personal contact between communicating parties, and dedicated communication links. The same prefixes can also be used for (1) encoding extensions to the core calculus, such as *vertical composition*, i.e., sub-protocol

calls (cf. Section 3.2), *conditionals* (i.e., execution guards); (2) defining *specification processes* relied upon by equivalence-based specification of secrecy; and (3) defining *initialisation traces* (cf. Section 3). The purpose of including session ids x in locations $c.x[T]$ is to enable the factorisation of the history of a protocol execution into individual sessions (cf. *strands* in strand spaces).

2.2 Semantics

C^3 is a reduction calculus on protocol states (P, \mathfrak{h}) , i.e., pairs of a process term $P \in \mathcal{P}$ and a protocol history $\mathfrak{h} \in \mathcal{H}$. Protocol histories are records of past protocol events ε , comprising: generation of a fresh name n with intended ownership O in session x by c , written $\mathsf{N}(c, x, n, O)$; insecure input of M [..], written $\mathsf{I}(c, x, M)$; secure input of M from d [..], written $\mathsf{sI}(c, x, M, d)$; insecure output of M to d [..], written $\mathsf{O}(c, x, M, d)$; and secure output of M to d [..], written $\mathsf{sO}(c, x, M, d)$. Protocol histories $\mathfrak{h} \in \mathcal{H}$ are simply finite words of protocol events ε , i.e., event traces: $\mathfrak{h} ::= \epsilon \mid \mathfrak{h} \cdot \varepsilon$ where ϵ denotes the empty protocol history.

C^3 employs pattern matching as a linguistic abstraction for cryptographic computation, which is straightforward to implement [21]. When a message matches a pattern, the result is a substitution relating variables to matched subterms. Substitutions are partial functions from variables to messages, and are lifted to terms as usual. Matching is computed by the partial function match defined in Table 3. There, \uplus denotes composition, if the domains of the operands are disjoint, and is otherwise undefined. As an example, the pattern $\{\{II\}_{\overline{v}}\}$ matches any message encrypted with the private key corresponding to v . The line over the letter v is a part of the pattern, intended to recall that the pattern matches a message encrypted with the key dual to v without needing to have this key (e.g., the private key of another participant) appear in the pattern term.

$$\begin{aligned}
\text{match}(v, M) &:= \{^M/v\} \\
\text{match}(M, M) &:= \emptyset \\
\text{match}((II, II'), (M, M')) &:= \text{match}(II, M) \uplus \text{match}(II', M') \\
\text{match}(\{\{II\}_{\overline{M'}}\}, \{\{M\}_{M'}\}) &:= \text{match}(II, M) \\
\text{match}(\{\{II\}_{\overline{p}}\}, \{\{M\}_{p^+}\}) &:= \text{match}(II, M) \\
\text{match}(\{\{II\}_{\overline{p^+}}\}, \{\{M\}_{\overline{p}}\}) &:= \text{match}(II, M)
\end{aligned}$$

Table 3. Pattern matching

Input guards φ are expressed in a decidable sub-language of our co-designed logic CPL. Their satisfaction is history dependent. Specifically, the knowledge of a participant depends only on the preceding events witnessed by the participant in question. As an example, this lets us easily express that keys to be looked up must already be known to the participant looking them up.

Definition 4 (Satisfaction). *Satisfaction is defined in Table 4. There, $\text{data}_c(\mathfrak{h})$ denotes the set of data that c has generated, received, or sent in \mathfrak{h} ; and analz and synth denote message analysis resp. synthesis (cf. Appendix A.1).*

$\mathfrak{h} \models c.x \circlearrowleft n.O$:iff $\mathbf{N}(c, x, n, t, O) \in \mathfrak{h}$
$\mathfrak{h} \models c.x \leftarrow M$:iff $\mathbf{I}(c, x, M) \in \mathfrak{h}$ or $\mathbf{sI}(c, x, M, d) \in \mathfrak{h}$
$\mathfrak{h} \models c \mathbf{k} M$:iff $M \in \text{synth}(\text{analz}(\text{data}_c(\mathfrak{h})))$
$\mathfrak{h} \models M \preceq M'$:iff M is a subterm of M'
$\mathfrak{h} \models n : \sigma$:iff n has sort σ
$\mathfrak{h} \models \neg\phi$:iff not $\mathfrak{h} \models \phi$
$\mathfrak{h} \models \phi \wedge \phi'$:iff $\mathfrak{h} \models \phi$ and $\mathfrak{h} \models \phi'$
$\mathfrak{h} \models \forall v(\phi)$:iff for all $M \in \mathcal{M}$, $\mathfrak{h} \models \{^M/v\}\phi$

Table 4. Satisfaction

The key store of each participant is induced (at the object level) by protocol events, as opposed to having a lookup table on the meta level. This gives us a completely communication-based model of key establishment. Keys are looked up in the protocol history w.r.t. the local view of each participant and by referring to their creator and the tag they were given at creation. We assume that the creator and the tag associated with a key are authentic and universally available (given possession of the key). This gives an approximation of certificate-based key lookup, where we explicitly model key distribution but key meta-data is magically protected. A lookup succeeds when the desired tag is a subterm of the one used at the creation of the key. This lets us model cases where the same shared key is to be used whenever two participants take part in a protocol, independently of their roles, as well as making more complex key-sharing arrangements possible. We model lookup with the predicate $\text{looksUp}(c, x, n, \varsigma, d, O)$ pronounced “ c in session x looks up n of type ς generated by d with a tag containing O ” and defined in Table 5 (cf. Appendix A.2 for the macro-definitions of the employed formulae). The

conditions enforce that the retrieved key (1) has the desired type ($n : \varsigma$); (2) was known by c ($c \text{ k } n$); (3) was generated by d ($d.x' \circ v.o$); (4) has a compatible, intended ownership ($O \preceq o$); and (5) when a session key, was received in the current session ($\exists m(c.x \leftarrow m \wedge n \preceq m)$).

$$\text{looksUp}(c, x, n, \varsigma, d, O) := n : \varsigma \wedge c \text{ k } n \wedge \exists x' \exists v \exists o (d.x' \circ v.o \wedge (n = v \vee n = v^+) \wedge O \preceq o \wedge (n : \mathbb{K}^1 \rightarrow \exists m(c.x \leftarrow m \wedge n \preceq m)))$$

Table 5. Lookup predicate

We are now ready to define our (process) reduction calculus \mathbb{C}^3 .

Definition 5 (Process calculus). Let $\longrightarrow \subseteq (\mathcal{P} \times \mathcal{H}) \times (\mathcal{P} \times \mathcal{H})$, defined in Table 6, denote reduction of protocol states $\mathfrak{s} \in \mathcal{P} \times \mathcal{H}$. Then \mathbb{C}^3 denotes the Calculus of Cryptographic Communication as defined below.

$$\mathbb{C}^3 := \langle \mathcal{P} \times \mathcal{H}, \longrightarrow \rangle$$

The generation of a new name (Rule NEW and NEW-EVE) is possible only if that name has not been generated yet, i.e., names are always *fresh* w.r.t. the current state. The adversary **Eve** may generate a new name at any time. Insecure input (Rule IN) is generated by the adversary and may consist in any message from her knowledge that matches the input pattern Π and that satisfies the input constraint φ . Successful input results in the substitution of the matching message parts for the matched variables in the receiving thread. Secure communication (Rule sOUT, sIN and sCOM-L, with sCOM-R being tacit) is synchronous. To achieve this, we introduce two auxiliary transition relations $\xrightarrow{\text{sI}}$ and $\xrightarrow{\text{sO}}$ not visible on the top level. Insecure communication between two legitimate participants is asynchronous because it goes through the adversary, and secure communication is synchronous because it does not. Execution of parallel processes happens via *interleaving concurrency* (Rule PAR-L, PAR-R being tacit). Finally, observe how *non-determinism* abstracts away three determining choices in the execution of a protocol, i.e., the choice of (1) the message sent by the adversary in an insecure input, (2) the new name selected at name generation time, and (3) the scheduling of located threads.

2.3 Process equivalence

We define a notion of observational equivalence for \mathbb{C}^3 -processes based on the concepts of *cryptographic parsing* (inspired by [22]) and *structurally*

$$\begin{array}{c}
\text{NEW} \frac{\mathfrak{h} \models n : \varsigma \wedge \neg \exists a \exists x \exists o (a.x \circ n.o)}{\left(\frac{c.x[\text{New}(v : \varsigma, O).T]}{\mathfrak{h}} \right) \longrightarrow \left(\frac{c.x[\{^n/v\}T]}{\mathfrak{h} \cdot \mathbf{N}(c, x, n, O)} \right)} \\
\text{NEW-EVE} \frac{\mathfrak{h} \models \text{Eve k } O \wedge \neg \exists a \exists x \exists o (a.x \circ n.o)}{\left(\frac{P}{\mathfrak{h}} \right) \longrightarrow \left(\frac{P}{\mathfrak{h} \cdot \mathbf{N}(\text{Eve}, \blacksquare, O)} \right)} \\
\text{OUT} \frac{}{\left(\frac{c.x[\text{Out}_d M.T]}{\mathfrak{h}} \right) \longrightarrow \left(\frac{c.x[T]}{\mathfrak{h} \cdot \mathbf{O}(c, x, M, d) \cdot \mathbf{I}(\text{Eve}, \blacksquare, M)} \right)} \\
\text{IN} \frac{\mathfrak{h} \models \text{Eve k } M \wedge \text{match}(II, M)\varphi}{\left(\frac{c.x[\text{In } II \text{ when } \varphi.T]}{\mathfrak{h}} \right) \longrightarrow \left(\frac{c.x[\text{match}(II, M)T]}{\mathfrak{h} \cdot \mathbf{O}(\text{Eve}, \blacksquare, M, c) \cdot \mathbf{I}(c, x, M)} \right)} \\
\text{sOUT} \frac{}{\left(\frac{c.x[\text{sOut}_d M.T]}{\mathfrak{h}} \right) \xrightarrow{\text{sO}} \left(\frac{c.x[T]}{\mathfrak{h} \cdot \text{sO}(c, x, M, d)} \right)} \\
\text{sIN} \frac{\mathfrak{h} \models \text{match}(II, M)\varphi}{\left(\frac{d.x[\text{sIn}_b II \text{ when } \varphi.T]}{\mathfrak{h} \cdot \text{sO}(c, x', M, d)} \right) \xrightarrow{\text{sI}} \left(\frac{d.x[\text{match}(II, M)T]}{\mathfrak{h} \cdot \text{sO}(c, x', M, d) \cdot \text{sI}(d, x, M, c)} \right)} \\
\text{sCOM-L} \frac{\left(\frac{P}{\mathfrak{h}} \right) \xrightarrow{\text{sO}} \left(\frac{P'}{\mathfrak{h}'} \right) \quad \left(\frac{Q}{\mathfrak{h}'} \right) \xrightarrow{\text{sI}} \left(\frac{Q'}{\mathfrak{h}''} \right)}{\left(\frac{P \parallel Q}{\mathfrak{h}} \right) \longrightarrow \left(\frac{P' \parallel Q'}{\mathfrak{h}''} \right)} \\
\text{LOOKUP} \frac{\left(\frac{c.x[\{^n/v\}T]}{\mathfrak{h}} \right) \xrightarrow{\alpha} \left(\frac{c.x[T']}{\mathfrak{h}'} \right) \quad \mathfrak{h} \models \text{lookUp}(c, x, n, \varsigma, d, O)}{\left(\frac{c.x[\text{Get}_d(v : \varsigma, O) \text{ in } T]}{\mathfrak{h}} \right) \xrightarrow{\alpha} \left(\frac{c.x[T']}{\mathfrak{h}'} \right)} \\
\text{PAR-L} \frac{\left(\frac{P}{\mathfrak{h}} \right) \xrightarrow{\alpha} \left(\frac{P'}{\mathfrak{h}'} \right)}{\left(\frac{P \parallel Q}{\mathfrak{h}} \right) \xrightarrow{\alpha} \left(\frac{P' \parallel Q}{\mathfrak{h}'} \right)}
\end{array}$$

Above, $\xrightarrow{\alpha} \in \{\longrightarrow, \xrightarrow{\text{sO}}, \xrightarrow{\text{sI}}\}$.

Table 6. Process and thread execution

indistinguishable protocol histories. Cryptographic parsing captures an agent’s capability to understand the structure of a cryptographically obfuscated message. The idea is to parse unintelligible messages to the abstract message ■.

Definition 6 (Cryptographic parsing). *The cryptographic parsing function $\llbracket \cdot \rrbracket_a^h$ associated with an agent $a \in \mathcal{P}$ and a protocol history $h \in \mathcal{H}$ (and complying with the assumptions of perfect cryptography) is an identity on names, the abstract message, and public keys; and otherwise acts as defined in Table 7.*

$$\begin{aligned}
\llbracket [M] \rrbracket_a^h &:= \begin{cases} \llbracket [M] \rrbracket_a^h & \text{if } h \models a \text{ k } M, \text{ and} \\ \blacksquare & \text{otherwise.} \end{cases} \\
\llbracket \{M\}_{M'} \rrbracket_a^h &:= \begin{cases} \llbracket \{M\}_{M'} \rrbracket_a^h & \text{if } h \models a \text{ k } M', \text{ and} \\ \blacksquare & \text{otherwise.} \end{cases} \\
\llbracket \{M\}_{p^+}^+ \rrbracket_a^h &:= \begin{cases} \llbracket \{M\}_{p^+}^+ \rrbracket_a^h & \text{if } h \models a \text{ k } p \vee (a \text{ k } M \wedge a \text{ k } p^+), \text{ and} \\ \blacksquare & \text{otherwise.} \end{cases} \\
\llbracket \{M\}_{p^-}^- \rrbracket_a^h &:= \begin{cases} \llbracket \{M\}_{p^-}^- \rrbracket_a^h & \text{if } h \models a \text{ k } p^+, \text{ and} \\ \blacksquare & \text{otherwise.} \end{cases} \\
\llbracket (M, M') \rrbracket_a^h &:= (\llbracket M \rrbracket_a^h, \llbracket M' \rrbracket_a^h)
\end{aligned}$$

Table 7. Parsing on cryptographic messages

For notational convenience, we subsequently write $\varepsilon(a)$ for any protocol event as defined in Section 2.2, $\varepsilon(a, n)$ for any of these name-generation events, $\varepsilon(a, M)$ for any of these communication events, and $\hat{\varepsilon}(a)$ for any of these secure events.

Definition 7 (Structurally indistinguishable protocol histories). *Two protocol histories h and h' are structurally indistinguishable from the viewpoint of an agent a , written $h \approx_a h'$, :iff a observes the same event pattern and the same data patterns in h and h' . Formally, for all $h, h' \in \mathcal{H}$, $h \approx_a h'$:iff $h \approx_a^{(h, h')} h'$ where,*

- given that a is a legitimate participant or the adversary Eve,

1. $\overline{\varepsilon \approx_a^{(h, h')} \varepsilon}$

$$2. \frac{\mathfrak{h}_l \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \varepsilon(a, n) \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r \cdot \varepsilon(a, n)}$$

$$3. \frac{\mathfrak{h}_l \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \varepsilon(a, M) \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r \cdot \varepsilon(a, M')} \Downarrow M \Downarrow_a^{\mathfrak{h}} = \Downarrow M' \Downarrow_a^{\mathfrak{h}'}$$

– given that a is a legitimate participant,

$$4. \frac{\mathfrak{h}_l \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \varepsilon(b) \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r} a \neq b \quad \frac{\mathfrak{h}_l \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r \cdot \varepsilon(b)} a \neq b$$

– given that a is the adversary Eve,

$$4. \frac{\mathfrak{h}_l \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \hat{\varepsilon}(b) \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r} \text{Eve} \neq b \quad \frac{\mathfrak{h}_l \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r \cdot \hat{\varepsilon}(b)} \text{Eve} \neq b$$

$$5. \frac{\mathfrak{h}_l \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \text{I}(b, x, M) \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r \cdot \text{I}(b, x, M')} \Downarrow M \Downarrow_{\text{Eve}}^{\mathfrak{h}} = \Downarrow M' \Downarrow_{\text{Eve}}^{\mathfrak{h}'}$$

$$6. \frac{\mathfrak{h}_l \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \text{O}(b, x, M, c) \approx_{\text{Eve}}^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r \cdot \text{O}(b, x, M', c)} \Downarrow M \Downarrow_{\text{Eve}}^{\mathfrak{h}} = \Downarrow M' \Downarrow_{\text{Eve}}^{\mathfrak{h}'}$$

Note that the observations at the different (past) stages \mathfrak{h}_l and \mathfrak{h}_r in \mathfrak{h} and \mathfrak{h}' respectively must be made with the whole (present) knowledge of \mathfrak{h} and \mathfrak{h}' (cf. $\mathfrak{h}_l \approx_a^{(\mathfrak{h}, \mathfrak{h}')} \mathfrak{h}_r$). Learning new keys may render intelligible past messages to an agent a in the present that were not intelligible to her before.

Remark 1. For all agents a including Eve, $\approx_a \subseteq \mathcal{H} \times \mathcal{H}$ is

1. an equivalence with an infinite index due to fresh-name generation
2. not a right-congruence due to the possibility of learning new keys
3. a refinement on the projection $\mathcal{H}|_a$ of \mathcal{H} onto a 's view [23]
4. decidable

We lift structural indistinguishability from protocol histories to protocol states, i.e., tuples of a protocol term and a protocol history.

Definition 8 (Structurally indistinguishable protocol states). *Let P_1 and P_2 denote two C^{β} -processes. Then two protocol states (P_1, \mathfrak{h}_1) and (P_2, \mathfrak{h}_2) are structurally indistinguishable from the viewpoint of an agent a , written $(P_1, \mathfrak{h}_1) \approx_a (P_2, \mathfrak{h}_2)$, :iff $\mathfrak{h}_1 \approx_a \mathfrak{h}_2$.*

This relation coincides with the relation of epistemic accessibility defining the epistemic modality of CPL [7], which reflects the intimate co-design of C^3 and CPL. We are finally ready to define observational equivalence for C^3 -processes.

Definition 9 (Trace-equivalent cryptographic processes). *For all agents a including Eve and for all $\mathfrak{s}_1, \mathfrak{s}_2 \in \mathcal{P} \times \mathcal{H}$,*

- \mathfrak{s}_2 trace-refines \mathfrak{s}_1 from the viewpoint of a , written $\mathfrak{s}_1 \gtrsim_a^* \mathfrak{s}_2$, :iff for all $\mathfrak{s}'_2 \in \mathcal{P} \times \mathcal{H}$, if $\mathfrak{s}_2 \longrightarrow^* \mathfrak{s}'_2$ then there is $\mathfrak{s}'_1 \in \mathcal{P} \times \mathcal{H}$ s.t. $\mathfrak{s}_1 \longrightarrow^* \mathfrak{s}'_1$ and $\mathfrak{s}'_2 \approx_a \mathfrak{s}'_1$; and
- \mathfrak{s}_1 and \mathfrak{s}_2 are trace-equivalent from the viewpoint of a , written $\mathfrak{s}_1 \approx_a^* \mathfrak{s}_2$, :iff $\mathfrak{s}_1 \gtrsim_a^* \mathfrak{s}_2$ and $\mathfrak{s}_2 \gtrsim_a^* \mathfrak{s}_1$.

3 Formal modelling with C^3

We show how a typical protocol can be formally modelled in C^3 . The chosen example does not take full advantage of C^3 's features, but it should nevertheless give an impression of C^3 's simplicity and explicitness.

3.1 Core NSPuK

In keeping with tradition, we present the well-known Needham Schroeder Public Key (NSPuK) authentication protocol, stated as a protocol narration in Table 8. In this narration, many tiny operations are implicitly

1. Alice \rightarrow Bob	:	$\{(x_{\text{Alice}}, \text{Alice})\}_{p_{\text{Bob}}^+}^+$
2. Bob \rightarrow Alice	:	$\{(x_{\text{Alice}}, x_{\text{Bob}})\}_{p_{\text{Alice}}^+}^+$
3. Alice \rightarrow Bob	:	$\{x_{\text{Bob}}\}_{p_{\text{Bob}}^+}^+$

Table 8. Protocol narration for core NSPuK

supposed to happen and be understood. More precisely, when Bob receives Message 1 encrypted with his own public key p_{Bob}^+ , he is implicitly expected to first decrypt the message, then take the second component, the name (**Alice**) of the message sender, and look up **Alice**'s public key p_{Alice}^+ . For the second message, Bob invents a new nonce x_{Bob} , which he then sends back to **Alice**, together with **Alice**'s nonce x_{Alice} . Finally, **Alice** confirms the reception of Bob's nonce.

This protocol can be expressed in C^3 as shown in Table 9. Here, we define named chunks of protocol code like $\text{NSPuK}_{\text{RESP}}(slf) := T$, which then can be instantiated as $\text{NSPuK}_{\text{RESP}}(\text{Bob}) \stackrel{\text{def}}{=} \{\text{Bob}/slf\}T$. For simplicity, we did not include a CCS-like definitional mechanism for process constants (here: NSPuK , $\text{NSPuK}_{\text{INIT}}$ and $\text{NSPuK}_{\text{RESP}}$) in the core calculus; the formal definition is well-known and easy to add. We use $\text{New}(v : \varsigma)$ as a shorthand for $\text{New}(v : \varsigma, \blacksquare)$ and $\text{In } \Pi$ as a shorthand for $\text{In } \Pi \text{ when } \top$. The code $\text{NSPuK}(\text{Alice}, \text{Bob}, x_a, x_b)$ for a single run of the NSPuK proto-

$\text{NSPuK}_{\text{INIT}}(slf, oth) :=$ $\text{New}(n_s : X).$ $\text{Get}_{oth}(k_o : K^+, oth) \text{ in}$ $\text{Out}_{oth} \{(n_s, slf)\}_{k_o}^+ .$ $\text{Get}_{slf}(k_s : K^-, slf) \text{ in}$ $\text{In} \{(n_s, v)\}_{k_s}^+ \text{ when } v : X .$ $\text{Out}_{oth} \{v\}_{k_o}^+ . 1$	$\text{NSPuK}_{\text{RESP}}(slf) :=$ $\text{Get}_{slf}(k_s : K^-, slf) \text{ in}$ $\text{In} \{(v, oth)\}_{k_s}^+ \text{ when } v : X \wedge oth : P .$ $\text{New}(n_s : X) .$ $\text{Get}_{oth}(k_o : K^+, oth) \text{ in}$ $\text{Out}_{oth} \{(v, n_s)\}_{k_o}^+ .$ $\text{In} \{n_s\}_{k_s}^+ . 1$
$\text{NSPuK}(init, resp, x_i, x_r) := \text{init}.x_i[\text{NSPuK}_{\text{INIT}}(init, resp)] \parallel \parallel$ $\text{resp}.x_r[\text{NSPuK}_{\text{RESP}}(resp)]$	

Table 9. Protocol code for core NSPuK

col with **Alice** and **Bob** playing the role of initiator resp. responder is obtained via instantiation, i.e., substitution of the participant names **Alice** and **Bob** for the “role names” *init* resp. *resp* and session identifier values x_a and x_b for session identifier variables x_i resp. x_r in the code chunk $\text{NSPuK}(init, resp, x_i, x_r)$. Such an instance is executed starting from an initial history, where the generation of private keys and the secure distribution of the corresponding public keys is made explicit. (This generation and distribution can equivalently be expressed with process terms using corresponding prefixes for name generation and secure communication.) In our scenario, we start from an initial history \mathfrak{h} , defined in Table 10, where **Alice** and **Bob** generate their own private keys (tagged with their own names) and transmit the corresponding public keys to each other and the adversary.

In this scenario, the key lookup $\text{Get}_{\text{Bob}}(k_o : K^+, \text{Bob})$ in T performed by **Alice** is effectuated via the $\text{lookup}(\text{Alice}, x_a, n, K^+, \text{Bob}, \text{Bob})$ predicate. As no other asymmetric keys are generated, this predicate is true iff $n = k_{\text{Bob}}^+$, since this key is known to **Alice** and was generated by **Bob**

$$\begin{aligned}
\mathfrak{h} := & \epsilon \cdot \mathbf{N}(\text{Alice}, x_a, k_{\text{Alice}}, \text{Alice}) \cdot \mathbf{N}(\text{Bob}, x_b, k_{\text{Bob}}, \text{Bob}) \cdot \\
& \mathbf{sO}(\text{Alice}, x_a, k_{\text{Alice}}^+, \text{Bob}) \cdot \mathbf{sI}(\text{Bob}, x_b, k_{\text{Alice}}^+, \text{Alice}) \cdot \\
& \mathbf{sO}(\text{Bob}, x_b, k_{\text{Bob}}^+, \text{Alice}) \cdot \mathbf{sI}(\text{Alice}, x_a, k_{\text{Bob}}^+, \text{Bob}) \cdot \\
& \mathbf{O}(\text{Alice}, x_a, k_{\text{Alice}}^+, \text{Eve}) \cdot \mathbf{I}(\text{Eve}, \blacksquare, k_{\text{Alice}}^+) \cdot \\
& \mathbf{O}(\text{Bob}, x_b, k_{\text{Bob}}^+, \text{Eve}) \cdot \mathbf{I}(\text{Eve}, \blacksquare, k_{\text{Bob}}^+)
\end{aligned}$$

Table 10. Initialisation trace for core NSPuK

with an appropriate ownership tag. The choice of session identifiers x_a and x_b is arbitrary.

We may now also consider multiple-session scenarios, such as

$$\text{NSPuK}(\text{Alice}, \text{Bob}, x_a, x_b) \parallel \text{NSPuK}(\text{Bob}, \text{Caesar}, y_b, y_c)$$

where we can easily distinguish events in the two different sessions of Bob by referring to the different session identifiers x_b and y_b .

3.2 NSPuK with remote key lookup

Let us now assume that, instead of generating their own private keys and sharing public keys beforehand, the participants in this protocol relied on a trusted third party **Trent** for this task. In this case, both participants will have to perform a remote lookup to acquire the other's public key. The client and server parts of a sub-protocol performing this remote key lookup are given in Table 11. The corresponding initial history \mathfrak{h}' of this

$ \begin{aligned} & \text{Lkup}_{\text{CLIENT}}(slf, srv) := \\ & \mathbf{sIn}_{slf} v \text{ when } v : \text{P}. \\ & \mathbf{Out}_{srv} (v, slf). \\ & \mathbf{Get}_{srv} (k_{srv} : \text{K}^+, srv) \text{ in} \\ & \mathbf{In} \{(key, v)\}_{k_{srv}}^- \text{ when } key : \text{K}^+. \\ & \mathbf{sOut}_{slf} (key, v).1 \end{aligned} $	$ \begin{aligned} & \text{Lkup}_{\text{SERVER}}(slf) := \\ & \mathbf{In} (v, oth) \text{ when } v : \text{P} \wedge oth : \text{P}. \\ & \mathbf{Get}_{slf} (k_s : \text{K}^-, slf) \text{ in} \\ & \mathbf{Get}_{slf} (k_o : \text{K}^+, v) \text{ in} \\ & \mathbf{Out}_{oth} \{(k_o, v)\}_{k_s}^- .1 \end{aligned} $
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Table 11. Additional (parallel) code for NSPuK with remote key lookup

key generation scenario is defined in Table 12. In the core NSPuK protocol definition, the local lookups for the public key of the counterpart are then replaced by

$$\mathbf{sOut}_{slf} oth. \mathbf{sIn}_{slf} (k_o, oth) \text{ when } k_o : \text{K}^+$$

$$\begin{aligned}
\mathfrak{h}' := & \epsilon \cdot \mathbf{N}(\mathbf{Trent}, x_t, k_{\mathbf{Alice}}, \mathbf{Alice}) \cdot \mathbf{N}(\mathbf{Trent}, x_t, k_{\mathbf{Bob}}, \mathbf{Bob}) \cdot \mathbf{N}(\mathbf{Trent}, x_t, k_{\mathbf{Trent}}, \mathbf{Trent}) \cdot \\
& \mathbf{s0}(\mathbf{Trent}, x_t, k_{\mathbf{Trent}}^+, \mathbf{Alice}) \cdot \mathbf{sI}(\mathbf{Alice}, x_a, k_{\mathbf{Trent}}^+, \mathbf{Trent}) \cdot \\
& \mathbf{s0}(\mathbf{Trent}, x_t, k_{\mathbf{Trent}}^+, \mathbf{Bob}) \cdot \mathbf{sI}(\mathbf{Bob}, x_b, k_{\mathbf{Trent}}^+, \mathbf{Trent}) \cdot \\
& \mathbf{0}(\mathbf{Trent}, x_t, k_{\mathbf{Trent}}^+, \mathbf{Eve}) \cdot \mathbf{I}(\mathbf{Eve}, \blacksquare, k_{\mathbf{Trent}}^+)
\end{aligned}$$

Table 12. Initialisation trace for NSPuK with remote key lookup

The return values from the lookup sub-protocol contain the name of the owner of the sought public key, in order to avoid confusion in a multiple-session scenario. This is an illustration of addressing messages at the thread level using pattern matching on the message content. Observe that in this example of sub-protocol calls, the sub-protocol (callee) inherits the session id from its parent (caller) who, in turn, is suspended until the sub-protocol returns.

4 Towards formal specification and verification with \mathbf{C}^3

We illustrate how to use \mathbf{C}^3 on the well-known failure of NSPuK. The attack involves two different, interleaved sessions. It consists in Eve tricking Alice into a session with Eve and impersonating Alice in the face of Bob, who is lead to believe that he is talking to Alice whereas he is in fact talking to Eve (cf. Table 13). As a result, the protocol fails to achieve its authentication goal.

$$\begin{array}{ll}
1. \quad \mathbf{Alice} \rightarrow \mathbf{Eve} & : \{ \{ (x_{\mathbf{Alice}}, \mathbf{Alice}) \} \}_{P_{\mathbf{Eve}}}^+ \\
1'. \quad \mathbf{Eve}_{\mathbf{Alice}} \rightarrow \mathbf{Bob} & : \{ \{ (x_{\mathbf{Alice}}, \mathbf{Alice}) \} \}_{P_{\mathbf{Bob}}}^+ \\
2'. \quad \mathbf{Bob} \rightarrow \mathbf{Eve}_{\mathbf{Alice}} & : \{ \{ (x_{\mathbf{Alice}}, x_{\mathbf{Bob}}) \} \}_{P_{\mathbf{Alice}}}^+ \\
2. \quad \mathbf{Eve} \rightarrow \mathbf{Alice} & : \{ \{ (x_{\mathbf{Alice}}, x_{\mathbf{Bob}}) \} \}_{P_{\mathbf{Alice}}}^+ \\
3. \quad \mathbf{Alice} \rightarrow \mathbf{Eve} & : \{ \{ x_{\mathbf{Bob}} \} \}_{P_{\mathbf{Eve}}}^+ \\
3'. \quad \mathbf{Eve}_{\mathbf{Alice}} \rightarrow \mathbf{Bob} & : \{ \{ x_{\mathbf{Bob}} \} \}_{P_{\mathbf{Bob}}}^+
\end{array}$$

Table 13. Attack narration for NSPuK

We recall that model-based (e.g., process algebraic) correctness statements of cryptographic protocols enunciate an observational equivalence that is supposed to hold between two process models (terms) of the protocol under scrutiny. The choice of the actual process terms depends on

the cryptographic goal that the correctness statement is intended to encode. For example, authenticity goals for a cryptographic protocol can be encoded as an observational equivalence between, on the one hand, an obviously (via different kinds of “magic”) correct specification process and, on the other hand, an implementation process expressing the protocol as it would be coded in a realistic implementation.

In our case, we use our notion of observational process equivalence w.r.t. Eve’s point of observation \approx_{Eve}^* and create the specification and implementation via a minor adaptation of the responder process. More precisely, we introduce an additional parameter oth' in the template $\text{NSPuK}_{\text{RESP}}(slf)$ and an additional conjunct φ in the guard of its first (insecure) input prefix $\text{In } \{(v, oth)\}_{k_s}^\pm \text{ when } v : X \wedge oth : P$. The modified responder process thus is

$$\text{NSPuK}_{\text{RESP}}(slf, oth') := [\dots] \text{In } \{(v, oth)\}_{k_s}^\pm \text{ when } v : X \wedge oth : P \wedge \varphi. [\dots]$$

where ‘[. . .]’ designates the parts left unchanged by the modification. The specification process $\text{NSPuK}_{\text{spec}}$ is then obtained from $\text{NSPuK}_{\text{RESP}}$ by stipulating that $\varphi := oth' = oth$ (the authentication guarantee), and the implementation process $\text{NSPuK}_{\text{impl}}$ by stipulating that $\varphi := \top$. Observe that the kind of magic in our specification is represented by the passing of the name of the actual correspondent oth' to the responder *at the meta-level*, i.e., as a formal parameter.

An implementation set-up $\mathfrak{s}_{\text{impl}}$ in \mathbb{C}^3 that potentially yields, via process reduction, a trace corresponding to the above attack narration is

$$\mathfrak{s}_{\text{impl}} := (\text{Alice}.x_a[\text{NSPuK}_{\text{INIT}}(\text{Alice}, \text{Eve})] \parallel \parallel \text{Bob}.x_b[\text{NSPuK}_{\text{impl}}(\text{Bob}, \text{Eve})], \mathfrak{h}_{\text{Eve}})$$

where

$$\begin{aligned} \mathfrak{h}_{\text{Eve}} &:= \mathfrak{h} \cdot \text{N}(\text{Eve}, x_e, k_{\text{Eve}}, \text{Eve}) \cdot \\ &\quad \text{sO}(\text{Eve}, x_e, k_{\text{Eve}}^+, \text{Alice}) \cdot \text{sI}(\text{Alice}, x_a, k_{\text{Eve}}^+, \text{Eve}) \cdot \\ &\quad \text{sO}(\text{Eve}, x_e, k_{\text{Eve}}^+, \text{Bob}) \cdot \text{sI}(\text{Bob}, x_b, k_{\text{Eve}}^+, \text{Eve}) \end{aligned}$$

and \mathfrak{h} is the initialisation trace of Section 3.1. The trace being considered is defined in Table 14.

In contrast, our specification set-up

$$\mathfrak{s}_{\text{spec}} := (\text{Alice}.x_a[\text{NSPuK}_{\text{INIT}}(\text{Alice}, \text{Eve})] \parallel \parallel \text{Bob}.x_b[\text{NSPuK}_{\text{spec}}(\text{Bob}, \text{Eve})], \mathfrak{h}_{\text{Eve}})$$

$$\begin{aligned}
\mathfrak{h}_{impl} := & \mathfrak{h}_{Eve} \cdot \mathbf{N}(\mathbf{Alice}, x_a, x_{\mathbf{Alice}}, \blacksquare) \cdot \\
& \mathbf{O}(\mathbf{Alice}, x_a, \{(x_{\mathbf{Alice}}, \mathbf{Alice})\}_{k_{Eve}^+}^+, \mathbf{Eve}) \cdot \mathbf{I}(\mathbf{Eve}, \blacksquare, \{(x_{\mathbf{Alice}}, \mathbf{Alice})\}_{k_{Eve}^+}^+) \cdot \\
& \mathbf{O}(\mathbf{Eve}, \blacksquare, \{(x_{\mathbf{Alice}}, \mathbf{Alice})\}_{k_{\mathbf{Bob}}^+}^+, \mathbf{Bob}) \cdot \mathbf{I}(\mathbf{Bob}, x_b, \{(x_{\mathbf{Alice}}, \mathbf{Alice})\}_{k_{\mathbf{Bob}}^+}^+) \cdot \\
& \mathbf{N}(\mathbf{Bob}, x_b, x_{\mathbf{Bob}}, \blacksquare) \cdot \\
& \mathbf{O}(\mathbf{Bob}, x_b, \{(x_{\mathbf{Alice}}, x_{\mathbf{Bob}})\}_{k_{\mathbf{Alice}}^+}^+, \mathbf{Alice}) \cdot \mathbf{I}(\mathbf{Eve}, \blacksquare, \{(x_{\mathbf{Alice}}, x_{\mathbf{Bob}})\}_{k_{\mathbf{Alice}}^+}^+) \cdot \\
& \mathbf{O}(\mathbf{Eve}, \blacksquare, \{(x_{\mathbf{Alice}}, x_{\mathbf{Bob}})\}_{k_{\mathbf{Alice}}^+}^+, \mathbf{Alice}) \cdot \mathbf{I}(\mathbf{Alice}, x_a, \{(x_{\mathbf{Alice}}, x_{\mathbf{Bob}})\}_{k_{\mathbf{Alice}}^+}^+) \cdot \\
& \mathbf{O}(\mathbf{Alice}, x_a, \{x_{\mathbf{Bob}}\}_{k_{Eve}^+}^+, \mathbf{Eve}) \cdot \mathbf{I}(\mathbf{Eve}, \blacksquare, \{x_{\mathbf{Bob}}\}_{k_{Eve}^+}^+) \cdot \\
& \mathbf{O}(\mathbf{Eve}, \blacksquare, \{x_{\mathbf{Bob}}\}_{k_{\mathbf{Bob}}^+}^+, \mathbf{Bob}) \cdot \mathbf{I}(\mathbf{Bob}, x_b, \{x_{\mathbf{Bob}}\}_{k_{\mathbf{Bob}}^+}^+)
\end{aligned}$$

Table 14. Attack trace for NSPuK

cannot, due to the failure of the authentication guarantee ($\mathbf{Eve} \neq \mathbf{Alice}$), yield a trace \mathfrak{h}_{spec} that matches the implementation trace \mathfrak{h}_{impl} beyond the event $\mathbf{I}(\mathbf{Eve}, \blacksquare, \{(x_{\mathbf{Alice}}, \mathbf{Alice})\}_{k_{Eve}^+}^+)$. Hence, $\mathfrak{s}_{impl} \not\approx_{Eve}^* \mathfrak{s}_{spec}$.

5 Conclusion

We believe having instigated with \mathbf{C}^3 a promising design formalism for the modelling, specification, and verification of cryptographic protocols. In particular, we have (1) invented a dynamic *key lookup mechanism* based on the explicit declaration of key ownership via tagging, (2) defined a comprehensive *message reception mechanism* integrating pattern matching and conditional input as linguistic abstractions for cryptographic computation, and (3) defined an operational semantics with reduction constraints that are checkable via the relation of satisfaction of a *co-designed logic*, namely CPL. Further, in [8] we demonstrate that \mathbf{C}^3 is easily extensible with real-time; only two additional axioms (and no modified axioms/rules !) are needed in its operational semantics.

In future work, we wish to extend \mathbf{C}^3 with (1) object-orientation, e.g., method-based sub-protocol calls and dynamic session creation; (2) further key management capabilities, e.g., public-key certificate issuing and revocation; and (3) computational complexity for cryptographic computation. We also wish to bridge the gap between protocol design and implementation by connecting \mathbf{C}^3 to the cryptographic library of [24]. Finally, we are planning to investigate different degrees of communication: first-order (communication of plain messages), second-order (communication of mes-

sage types), third-order (communication of processes, i.e., protocols), and fourth-order (communication of process types).

A Auxiliary definitions

A.1 Auxiliary functions

The closure operator synth [25]

$$\frac{M \in \mathcal{K}}{M \in \text{synth}(\mathcal{K})} \quad \frac{M \in \text{synth}(\mathcal{K})}{\lceil M \rceil \in \text{synth}(\mathcal{K})} \quad \frac{M \in \text{synth}(\mathcal{K}) \quad M' \in \mathcal{K}}{\{\!\{M\}\!\}_{M'} \in \text{synth}(\mathcal{K})}$$

$$\frac{M \in \text{synth}(\mathcal{K}) \quad p^+ \in \mathcal{K}}{\{\!\{M\}\!\}_{p^+}^+ \in \text{synth}(\mathcal{K})} \quad \frac{M \in \text{synth}(\mathcal{K}) \quad p \in \mathcal{K}}{\{\!\{M\}\!\}_p^- \in \text{synth}(\mathcal{K})} \quad \frac{p \in \text{synth}(\mathcal{K})}{p^+ \in \text{synth}(\mathcal{K})}$$

$$\frac{M \in \text{synth}(\mathcal{K}) \quad M' \in \text{synth}(\mathcal{K})}{(M, M') \in \text{synth}(\mathcal{K})}$$

The kernel operator analz [25]

$$\frac{M \in \mathcal{K}}{M \in \text{analz}(\mathcal{K})} \quad \frac{\{\!\{M\}\!\}_{M'} \in \text{analz}(\mathcal{K}) \quad M' \in \text{synth}(\text{analz}(\mathcal{K}))}{M \in \text{analz}(\mathcal{K})}$$

$$\frac{\{\!\{M\}\!\}_{p^+}^+ \in \text{analz}(\mathcal{K}) \quad p \in \text{analz}(\mathcal{K})}{M \in \text{analz}(\mathcal{K})} \quad \frac{\{\!\{M\}\!\}_p^- \in \text{analz}(\mathcal{K}) \quad p^+ \in \text{analz}(\mathcal{K})}{M \in \text{analz}(\mathcal{K})}$$

$$\frac{(M, M') \in \text{analz}(\mathcal{K})}{M \in \text{analz}(\mathcal{K})} \quad \frac{(M', M) \in \text{analz}(\mathcal{K})}{M \in \text{analz}(\mathcal{K})}$$

A.2 Macro-defined formulae

\top	:=	Eve : Adv	true
\perp	:=	$\neg \top$	false
$\phi \vee \phi'$:=	$\neg(\phi \wedge \phi')$	ϕ or ϕ'
$\phi \rightarrow \phi'$:=	$\neg\phi \vee \phi'$	if ϕ then ϕ'
$\phi \leftrightarrow \phi'$:=	$(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$	ϕ if and only if ϕ'
$\exists v(\phi)$:=	$\neg \forall v(\neg\phi)$	there is a v s.t. ϕ
$\forall(v : \theta)(\phi)$:=	$\forall v(v : \theta \rightarrow \phi)$	
$\exists(v : \theta)(\phi)$:=	$\exists v(v : \theta \wedge \phi)$	
$F = F'$:=	$F \preceq F' \wedge F' \preceq F$	F is equal to F'
$F \prec F'$:=	$F \preceq F' \wedge \neg F = F'$	F is a strict subterm of F'
$F : \emptyset$:=	\perp	
$F : \mathbf{K}^+$:=	$\exists(v : \mathbf{K}^-)(F = v^+)$	
$F : \mathbf{H}[\theta]$:=	$\exists(v : \theta)(F = \lceil v \rceil)$	
$F : \mathbf{SC}_{F'}[\theta]$:=	$\exists(v : \theta)(F = \{\!\{v\}\!\}_{F'})$	

$$\begin{aligned}
F : \mathbf{AC}_{p^+}[\theta] &:= \exists(v : \theta)(F = \{\!|v|\!\}_{p^+}^+) \\
F : \mathbf{S}_p[\theta] &:= \exists(v : \theta)(F = \{\!|v|\!\}_p^-) \\
F : \mathbf{T}[\theta, \theta'] &:= \exists(v : \theta)\exists(v' : \theta')(F = (v, v')) \\
F : \theta \cup \theta' &:= F : \theta \vee F : \theta' \\
F : \theta \cap \theta' &:= F : \theta \wedge F : \theta' \\
F : \theta \setminus \theta' &:= F : \theta \wedge \neg F : \theta' \\
F : \mathbf{M} &:= \top \\
F : \mathbf{SC}[\theta] &:= \exists v(F : \mathbf{SC}_v[\theta]) \\
F : \mathbf{AC}[\theta] &:= \exists(v : \mathbf{K}^+)(F : \mathbf{AC}_v[\theta]) \\
F : \mathbf{C}[\theta] &:= F : \mathbf{SC}[\theta] \cup \mathbf{AC}[\theta] \\
F : \mathbf{S}[\theta] &:= \exists(v : \mathbf{K}^-)(F : \mathbf{S}_v[\theta])
\end{aligned}$$

References

1. Anderson, R., Needham, R.: Programming Satan's computer. In: Computer Science Today: Recent Trends and Developments. Volume 1000 of LNCS., Springer-Verlag (1996)
2. Abadi, M.: Security protocols and their properties. In: Foundations of Secure Computation, IOS Press (2000)
3. Briais, S., Nestmann, U.: A formal semantics for protocol narrations. [26] 163–181
4. Boyd, C., Mathuria, A.: Protocols for Authentication and Key Establishment. Springer (2003)
5. Burrows, M., Abadi, M., Needham, R.: A logic of authentication. ACM Transactions on Computer Systems **8**(1) (1990)
6. Kramer, S.: Cryptographic Protocol Logic. In: Proceedings of the LICS/ICALP-Affiliated Workshop on Foundations of Computer Security. (2004)
7. Kramer, S.: Logical concepts in cryptography. submitted (2006)
8. Borgström, J., Grinchtein, O., Kramer, S.: Timed Calculus of Cryptographic Communication. submitted (2006)
9. Song, D., Berezin, S., Perrig, A.: Athena, a novel approach to efficient automatic security protocol analysis. Journal of Computer Security **9**(1,2) (2001)
10. Armando, A., Basin, D., Boichut, Y., Chevalier, Y., Compagna, L., Cuellar, J., Drielsma, P.H., Heàm, P., Kouchnarenko, O., Mantovani, J., Mödersheim, S., von Oheimb, D., Rusinowitch, M., Santiago, J., Turuani, M., Viganò, L., Vigneron, L.: The AVISPA tool for the automated validation of Internet security protocols and applications. In: International Conference on Computer Aided Verification. (2005)
11. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: IEEE Computer Security Foundations Workshop. (2001)
12. Trusted Logic, ENS Cachan, Verimag: (Projet EVA) <http://www-eva.imag.fr/>.
13. Guttman, J.D., Herzog, J.C., Ramsdell, J.D., Sniffen, B.T.: Programming cryptographic protocols. [26] 116–145
14. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: proving security protocols correct. Journal of Computer Security **7**(2-3) (1999)
15. Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, B.: The Modelling and Analysis of Security Protocols: the CSP Approach. Addison-Wesley (2000)
16. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The Spi-calculus. Information and Computation **148**(1) (1999)

17. Crazzolaro, F., Winskel, G.: Events in security protocols. In: Proceedings of the ACM Conference on Computer and Communications Security. (2001)
18. Cremers, C.J.F., Mauw, S.: Operational semantics of security protocols. In Leue, S., Systä, T., eds.: Scenarios: Models, Transformations and Tools. Volume 3466 of Lecture Notes in Computer Science., Springer (2003) 66–89
19. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for proving security properties of protocols. *Journal of Computer Security* **11**(4) (2003)
20. Dolev, D., Yao, A.C.: On the security of public key protocols. *IEEE Transactions on Information Theory* **29**(12) (1983)
21. Haack, C., Jeffrey, A.: Pattern-matching Spi-calculus. In: Proceedings of the Workshop on Formal Aspects in Security and Trust. (2004)
22. Abadi, M., Rogaway, P.: Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology* **15**(2) (2002)
23. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press (1995)
24. Backes, M., Pfizmann, B., Waidner, M.: A universally composable cryptographic library. In: Proceedings of the ACM Conference on Computer and Communication Security. (2003)
25. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. *Journal of Computer Security* **6**(1) (1998)
26. De Nicola, R., Sangiorgi, D., eds.: Trustworthy Global Computing, International Symposium, TGC 2005, Edinburgh, UK, April 7-9, 2005, Revised Selected Papers. In De Nicola, R., Sangiorgi, D., eds.: TGC. Volume 3705 of Lecture Notes in Computer Science., Springer (2005)