Calculus of Cryptographic Communication

Johannes Borgström[1], Simon Kramer[2], and Uwe Nestmann[1]

[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 framework for the modelling, specification and verification of cryptographic protocols. C^3 consists of a design language of distributed processes and an associated (SOS) notion of concurrent execution. The other half of our framework is a property-based formalism, i.e., a logic for the specification and verification of cryptographic protocols, called CPL. The two primary features of the co-design of C^3 and CPL are that reduction constraints of C^3-processes are checkable via CPL-satisfaction, and that C^3's notion of observational equivalence and CPL's notion of propositional knowledge have a common definitional basis, namely structurally indistinguishable protocol histories. 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 notions of secure (out-of-band) communication and history-based key lookup, which together give a concrete foundation on which to base authentication and key establishment protocols.

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

1 Introduction: on design formalisms for cryptographic protocols

The design of cryptographic protocols is a non-trivial problem. It has even been described as *Programming Satan's Computer* [1]. Unfortunately, these protocols are commonly specified by means of an informal notation that is sometimes called *security protocol engineering notation*; protocols written in this notation are sometimes called *protocol narrations* [2, 3], which essentially are global-view transcripts of successful protocol runs.

1.1 Requirements for a better design formalism

It is commonly agreed that protocol narrations are not precise enough (e.g., [4, pages 2–3]). The implicitness of reception checks and the involved "magical" knowledge about the intended protocol role and the adequate decryption keys leave much room for misunderstandings and false interpretations, not to say protocol bugs. To tackle this problem, a flurry of formalisms have been proposed over the last at least 20 years. Why should we invent another one? Obviously, because we believe that the available ones are not yet good enough and have their deficiencies in one or several ways. However, (probably) all of them were the product of some guiding principles, be it minimality, simplicity, Expressiveness, executability, the support of proof methodologies and tools, etc. In order to structure this discussion, let us explain the guidelines for our own proposal.

First, we seek a formalism that not only supports the task of protocol design, but that is also tightly connected to a *property-based specification formalism*, like BAN [5] or CPL [6]. With such (domain-specific) logical formalisms, one may express security goals directly as succinct logical formulae separately from the protocol description. This is in contrast to common approaches of explicitly incorporating them in the description or using indirect encodings (e.g., equivalences between an actual and a magically correct version of some protocol).

Second, the formalism needs to reflect a *level of abstraction* concerning the aspects of reality that is suitable both for the purpose of design and verification. We aim at a compromise between, on the one hand, the amount of detail required to fully understand the structure of a crypto protocol and, on the other hand, abstracting as much as possible from merely technological variations. For instance, when a protocol participant receives a message then a number of things must happen, e.g., the purpose of the message must be understood, data needs to be extracted, and then it must be decided whether the mesage shall be taken into account. Here, a crucial role is played by the lookup of the cryptographic keys that are required along the way. All of these operations matter—cryptographically—for achieving some overall security goal. Thus, we argue that the formalism shall work at a level of abstraction that expresses the essential operations succinctly and without reference to implicit magic, but that also employs the lightest possible technological means, abstracting from non-cryptographic matters. Here, we go even beyond the *Explicitness Principle* of [1], where it is postulated that "any necessary naming, typing, and freshness information should be explicit in a protocol's messages"; we require in addition that protocol principals should be explicit about all of their *protocol-relevant actions*. Furthermore, it has long been observed that secure communication must be 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." [7]. For the same reasons as above, we argue that a secure communication mechanism shall also be part of the core model. Any reference to abstract magic shall be made explicit.

Third, in our view, process calculi and their standard tools for defining and reasoning about semantics offer—due to the linguistic representation of modelling concepts—the best compromise between mathematical rigour suitable for a core model and usability as a design formalism. Ideally, the core formalism shall also be amenable to variations in its so-called message language, i.e., to prepare for varying underlying cryptosystems and data domains.

Fourth, we aim at pragmatics and usability for both "programming" in the design formalism and verifying protocols written in it. For example, on the programming side, it must be easy to provide extensions of the formalism as syntactic sugar, and it needs to allow for a modular design of crypto protocols by the support of code reuse for subprotocols ("vertical" composition) and multiple sessions ("horizontal" composition). On the verification side, it should not only be easy to describe and reason about single runs of interacting protocol roles, but also to account for multiple sessions with overlapping participant sets and to provide dedicated support for this setting in the design formalism itself; this could be realised by management principles on the use of session identifiers.

1.2 Related work

At this stage, we put an emphasis on ease and clarity of modelling rather than automated protocol verification. Our target audience is academia rather than industry. Industry demands push-button tools, i.e., software that pragmatically integrates *design* 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 [8] based on strand spaces and integrating model checking with theorem proving, the AVISPA tool [9] based on a fragment of Lamport's Temporal Logic of Actions and integrating model checking with rewriting, the ProVerif tool [10] based on Prolog (unification) and the various tools of the EVA project [11].

In the field of protocol design formalisms, i.e., design languages with associated semantics, property languages and notions of correctness, we identify the following related work. CPPL [12] by Guttman et.al., based on strand spaces [13]; process calculi like the Spi Calculus [14] by Abadi and Gordon, the χ-spaces framework and its underlying process calculus SPL [15] by Crazzolara, Milicia and Winskel, CSP [16] as used by Ryan and Schneider with the FDR[] model checker; and a very detailed, but not explicitly named model [17] by Cremers and Mauw, which we simply refer to as CMOS in the remainder. The list is by no means complete, but in our understanding it contains a representative collection. Below, we indicate in how far they do not meet our requirements.

However, with the important exception of the design formalism PCL [18], none of the design formalisms for cryptographic protocols we are aware of (in particular those mentioned above) has been exhibited to have a strong (i.e., model-theoretic) connection to a cryptographic *logic*. This work is connected to CPL, Concerning the integration with some logical formalism, none of the above approaches (nor the other ones that we are aware of) satisfies our requirements.

All of the above approaches deal with key lookup in a rather implicit way where magic access to externally defined global functions is available. Similarly, above-cited well-foundedness principle of [7] is not taken into account properly.

CPPL [12] satisfies many of our other requirements, though. It emphasizes programmer usability issues by the support of subprotocols and the use of pattern matching. It provides a formal model of protocol behavior, building upon a formal semantics within the domain of strand spaces. However, it is targeted at a particular style of verification, different from ours, namely a method based on rely/guarantee pairs of formulae: the overall correctness of a protocol results from the truth of the individual rely formulae and the successful run-time checking of individual gurantee formula. The verification method that *we* have in mind (satisfaction of CPL-formulae by protocols in our formalism) is quite different, so it is no surprise that it requires a quite different and incomparable incorporation into the design formalism. Finally, CPPL seems to shift the concerns of well-foundedness of authentication and key lookup to some cryptographic library outside the formalism.

The Spi Calculus [14] and SPL [15] 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 [16] was used as a variant that includes abstract types in order to deal with encrytion 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 model concepts then need to be encoded. This goes clearly against our requirement on *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 literally understand the overall structure of non-trivial multi-session scenarios, and it can make it very hard to track the acquired knowledge of particpants that take part in several protocols concurrently. Moreover, it is hardly feasible to compose security arguments for multi-session scenarios from security properties of their parts.

CMOS [17] is based on careful domain analysis to determine which cryptographic and other concepts need to be represented. Thus, and in our opinion not surprisingly, the model embodies a rich collection of explicit concepts, including protocol roles (with local variables), parametric intruder models, the initial knowledge of participants, run identifiers (similar to session identifiers) to distinguish between multiple sessions, 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 concerning the (subjectively) rather complicated handling of instantiations of roles to runs.

In summary, our proposal, as introduced in the next section, is closest related to CPPL and CMOS, but it is different: it proposes different decisions concerning the explicitness of concepts, and it is much closer to logics, while carrying out the overall formalisation of the model within a process calculus setting.

2 Definition

C^3 consists of a design 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 classical cryptography and interleaving concurrency. Cryptography is abstract in the sense that communicated information atoms (names) are *logical constants* and communicated information compounds (messages) are *syntactic terms*. We also use uses 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 [19] adversary $\mathtt{Eve}$, i.e., the generation of a history of legitimate resp. illegitimate protocol events, and the evolution of their respective knowledge computed from that history.

There are several areas of C^3 where the codesign with the logic CPL is evident. First, names in C^3 are logical constants. In contrast, the concept of names of process calculi such as the π- 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 quantification in the logic. Second, all logic-based operations, including input guards, pattern matching, key lookup and side conditions to the rules of the operational semantics, are expressed in a sub-language of CPL, which is also equipped with a compatible notion of satisfaction. Finally, C^3's observational equivalence and CPL's epistemic modality have a common definitional basis (structurally indistinguishable protocol histories, cf. Appendix A.2). 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

We assume a typed language of names, representing primitive pieces of data.

Definition 1 (Names and Types). *The different kinds of names n are participant names $c, d \in \mathcal{P}$, the adversary's name $\mathtt{Eve}$, symmetric session and longterm keys $k \in \mathcal{K}^1 \cup \mathcal{K}^\infty$, (asymmetric) private keys $p \in \mathcal{K}^-$, and nonces $x \in \mathcal{X}$ (also used as session identifiers). These kinds of names have corresponding types $\mathtt{P}$, $\mathtt{Adv}$, $\mathtt{K}^1$, $\mathtt{K}^\infty$, $\mathtt{K}^-$ and $\mathtt{X}$, respectively. $\mathtt{K}^+$ is the type of terms of the form p^+, which denotes the public key corresponding to p. Collectively, these types are denoted with the metavariable τ. The types ς freshly generatable by participants are $\mathtt{K}^1, \mathtt{K}^\infty, \mathtt{K}^-$ and $\mathtt{X}$.*

We work with a message language $\mathcal{M}$ with typed names, a distinguished abstract message, hashing, public- and shared-key encryption, and private-key signings of messages. The abstract message $\blacksquare$ represents a term without any particular meaning, and is later also used to denote opaque (i.e., not decryptable) messages. Message forms are open messages, used in process terms where they may instantiate to (transmittable) messages.

Definition 2 (Messages, Message forms). *Protocol messages $M \in \mathcal{M}$ are defined in Table 1. Message forms $F \in \mathcal{M}^{\mathrm{f}}$ are messages with arbitrary subterms replaced by variables $v \in \mathcal{V}$. Key tags A are message forms not containing cryptographic operators. We write $F \preccurlyeq F'$ for F being a subterm of F'.*

Our 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. Regarding the action prefixes, $\mathtt{Out}_a\ F$ and $\mathtt{sOut}_a\ F$ express insecure (to the adversary) resp. secure (unobservable) output of F to a; $\mathtt{In}\ \Pi\ \mathtt{when}\ \varphi$ and

M	$::=$	protocol messages
n	$\mid$	names (logical constants)
$\blacksquare$	$\mid$	the abstract message
p^+	$\mid$	public keys
$\lceil M \rceil$	$\mid$	message hashes
$\{\!\vert M \vert\!\}_k$	$\mid$	symmetric message ciphers
$\{\!\vert M \vert\!\}^+_{p^+}$	$\mid$	asymmetric message ciphers
$\{\!\vert M \vert\!\}^-_p$	$\mid$	signed messages
(M, M)		message tuples

Table 1. Message language

$\mathtt{sIn}_a\ \Pi$ $\mathtt{when}$ φ express insecure resp. secure input (from a) of a message matching the pattern Π and having the property φ; and $\mathtt{New}\ (v : \varsigma, A)$ expresses the generation and binding to the variable v of a fresh name of type ς tagged with ("belonging to") A. The lookup prefix $\mathtt{LookUp}_a\ (v : \tau, A)\ \mathtt{in}\ T$, stands for the lookup and binding to v of a name of type τ generated by a with tag A. Our communication model is based on participants rather than channels for the sake of separating specification and implementation concerns.

Definition 3 (Processes). *The syntax of processes $P \in \mathcal{P}$ is as defined below.*

$$\begin{aligned}
P &::= c.x[\,T\,] \mid P \mathbin{|||} P \\
T &::= 1 \mid \pi.T \mid \mathtt{LookUp}_a\ (v : \tau, A)\ \mathtt{in}\ T \\
\pi &::= \mathtt{Out}_a\ F \mid \mathtt{sOut}_a\ F \mid \mathtt{In}\ \Pi\ \mathtt{when}\ \varphi \mid \mathtt{sIn}_a\ \Pi\ \mathtt{when}\ \varphi \mid \mathtt{New}\ (v : \varsigma, A) \\
\Pi &::= F \mid (\Pi, \Pi) \mid \{\!|\Pi|\!\}_{\overline{F}} \mid \{\!|\Pi|\!\}^+_{\overline{F}} \mid \{\!|\Pi|\!\}^-_{\overline{F}} \\
\varphi &::= a\ \mathsf{k}\ F \mid F \preccurlyeq F \mid F : \tau \mid \neg\varphi \mid \varphi \wedge \varphi.
\end{aligned}$$

Here, $a, b \in \mathcal{P} \cup \mathcal{V}$. A process P is epistemically local *:iff for all located threads $c.x[\,T\,]$ in P and for all subterms $a\ \mathsf{k}\ F$ of T, $a = c$. We only consider epistemically local processes.*

Action prefixes π generate events when executed, in contrast to key lookup. Among the action prefixes is secure communication, which we use for the modelling of out-of-band information exchange. Moreover, secure communication is convenient for encoding extensions to the calculus such as *vertical composition* (i.e., subprotocol calls) and *conditionals* (i.e., execution guards). Finally, we intend to use secure communication in the uniform definition of *specification processes* relied upon by equivalence-based verification. The purpose of including session ids x in locations $c.x[\,T\,]$ is to enable factoring the history of a protocol execution into individual sessions (cf. *strands* in strand spaces).

2.2 Semantics

Our reduction calculus (or "operational semantics") is defined on *protocol states*, i.e., pairs of a process term $P \in \mathcal{P}$ and a protocol history $\mathfrak{h} \in \mathcal{H}$. Protocol histories are

$$\begin{aligned}
\mathrm{match}(v, M) &:= \{^{M}/_{v}\}\\
\mathrm{match}(M, M) &:= \emptyset\\
\mathrm{match}((\Pi, \Pi'), (M, M')) &:= \mathrm{match}(\Pi, M) \uplus \mathrm{match}(\Pi', M')\\
\mathrm{match}(\{\!|\Pi|\!\}_{\bar{k}}, \{\!|M|\!\}_{k}) &:= \mathrm{match}(\Pi, M)\\
\mathrm{match}(\{\!|\Pi|\!\}^{+}_{\bar{p}}, \{\!|M|\!\}^{+}_{p^{+}}) &:= \mathrm{match}(\Pi, M)\\
\mathrm{match}(\{\!|\Pi|\!\}^{-}_{\overline{p^{+}}}, \{\!|M|\!\}^{-}_{p}) &:= \mathrm{match}(\Pi, M)
\end{aligned}$$

Table 2. Pattern matching

$\mathfrak{h} \models c.x \circlearrowleft n.A$	:iff $\mathtt{N}(c, x, n, A) \in \mathfrak{h}$
$\mathfrak{h} \models c.x \leftarrow M$	:iff $\mathtt{I}(c, x, M) \in \mathfrak{h}$ or $\mathtt{sI}(c, x, M, d) \in \mathfrak{h}$
$\mathfrak{h} \models c\ \mathsf{k}\ M$	:iff $M \in \mathrm{synth}(\mathrm{analz}(\mathrm{data}\,(\mathfrak{h}\|c)))$
$\mathfrak{h} \models M \preccurlyeq M'$	:iff M is a subterm of M'
$\mathfrak{h} \models M : \tau$	:iff M has type τ
$\mathfrak{h} \models \neg\phi$	: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 3. Satisfaction

records of past protocol events ε, comprising: generation of a fresh name n tagged with A in session x by c, written $\mathtt{N}(c, x, n, A)$; insecure input of M in session x by c, written $\mathtt{I}(c, x, M)$; secure input of M from d in session x by c, written $\mathtt{sI}(c, x, M, d)$; insecure output of M to d in session x by c, written $\mathtt{O}(c, x, M, d)$; and secure output of M to d in session x by c, written $\mathtt{sO}(c, x, M, d)$. Protocol histories $\mathfrak{h} \in \mathcal{H}$ are simply finite words of protocol events ε, i.e., event traces: $\mathfrak{h} ::= \epsilon \,\big|\, \mathfrak{h} \cdot \varepsilon$ where ϵ denotes the empty protocol history.

C^3 relies on pattern matching as an abstraction for cryptographic computation, which is straightforward to implement [20]. When a message matches a pattern, the result is a substitution that relates variables to matched subterms. Substitutions σ are partial functions from variables to messages, and are applied to terms in the usual fashion. Note that attempting to substitute, e.g., a hashed name for a variable in key position, does not result in a valid message form and thus the application fails. Matching is computed by the partial function match defined in Table 2. There, $\uplus$ denotes composition, if the domains of the operands are disjoint, and is otherwise undefined.

Input guards φ are expressed in a sublanguage of the co-designed logic CPL. Their satisfaction is history dependent. Specifically, the knowledge of a participant depends only on the preceeding events witnessed by the participant in question. As an example, this lets us easily implement freshness checks of nonces in order to avoid replay attacks.

Definition 4 (Satisfaction). *Satisfaction is defined in Table 3. There,* $\mathfrak{h}|c$ *denotes the projection of* $\mathfrak{h}$ *onto c's view;* data $(\mathfrak{h}|c)$ *denotes the set of all data that c has generated, received, or sent in* $\mathfrak{h}$*; and* analz *and* synth *denote message analysis and synthesis, respectively (cf. Appendix A.1, [21]).*

During a protocol, the key store of each participant based on secure communication preceding protocol execution and key transfers as part of the protocol (possibly in a different session). Keys are looked up referring to their creator and the tag they were given at creation. We assume that the creator and the tag associated with a key is authentic and universally available (given possession of the key). This gives an approximation of certificate-based key lookup, where we explicitely model key distribution but key metadata is magically protected. As an example, a symmetric key k tagged with (c, d) can be interpreted as that the participants c and d are the intended owners of k; and a (asymmetric) private key p tagged with c declares that the participant c is the intended owner of p. 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 $\mathsf{looksUp}(c, x, n, \tau, d, A)$ defined in Table 4. It intuitively means "c (in session x) looks up n of type τ, generated by d with a tag containing A". The conditions enforce that the retrieved key has the desired type ($n : \tau$), is known by c ($c \; \mathsf{k} \; n$), has a compatible tagging ($A \preccurlyeq v$ where v is the tag used at creation), was generated by d (in some session y, $d.y \circlearrowleft n.v$) and, when looking up a session key, it was received in the current session ($\exists m(c.x \leftarrow m \wedge n \preccurlyeq m)$).

$$\mathsf{looksUp}(c, x, n, \tau, d, A) := n : \tau \wedge c \; \mathsf{k} \; n \wedge \begin{cases} \exists v \exists y(A \preccurlyeq v \wedge d.y \circlearrowleft n.v) & \tau \in \{\mathtt{K}^\infty, \mathtt{K}^-\} \\ \exists p \exists v \exists y(A \preccurlyeq v \wedge p^+ = n \wedge d.y \circlearrowleft p.v) & \tau = \mathtt{K}^+ \\ \exists v \exists y(A \preccurlyeq v \wedge d.y \circlearrowleft n.v) \wedge \exists m(c.x \leftarrow m \wedge n \preccurlyeq m) & \tau = \mathtt{K}^1 \\ \bot & \text{otherwise} \end{cases}$$

Table 4. Key lookup predicate

We are now ready to define our process calculus C^3.

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

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

The generation of a new name (rules New and New-Eve) is possible only if that name does not exist 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 will result in the substitution of the matching message parts for the matched variables in the receiving thread. Secure communication (rules sOut, sIn and sCom-L, with sCom-r being

$$\textsc{New}\ \frac{\mathfrak{h} \models n : \varsigma \wedge \neg\exists(v : \mathtt{P}_{\mathtt{Adv}})(v \circlearrowleft n)}{\begin{pmatrix} c.x[\ \mathtt{New}\ (v : \varsigma, A).T\ ] \\ \mathfrak{h} \end{pmatrix} \longrightarrow \begin{pmatrix} c.x[\ \{^{n}/_{v}\}T\ ] \\ \mathfrak{h} \cdot \mathtt{N}(c, x, n, A) \end{pmatrix}}$$

$$\textsc{New-Eve}\ \frac{\mathfrak{h} \models \mathtt{Eve}\ \mathsf{k}\ A \wedge \neg\exists(v : \mathtt{P}_{\mathtt{Adv}})(v \circlearrowleft n)}{\begin{pmatrix} P \\ \mathfrak{h} \end{pmatrix} \longrightarrow \begin{pmatrix} P \\ \mathfrak{h} \cdot \mathtt{N}(\mathtt{Eve}, \_, n, A) \end{pmatrix}}$$

$$\textsc{Out}\ \frac{}{\begin{pmatrix} c.x[\ \mathtt{Out}_{d}\ M.T\ ] \\ \mathfrak{h} \end{pmatrix} \longrightarrow \begin{pmatrix} c.x[\ T\ ] \\ \mathfrak{h} \cdot \mathtt{O}(c, x, M, d) \end{pmatrix}}$$

$$\textsc{In}\ \frac{\mathfrak{h} \models \mathtt{Eve}\ \mathsf{k}\ M \wedge \mathrm{match}(\Pi, M)\varphi}{\begin{pmatrix} c.x[\ \mathtt{In}\ \Pi\ \mathtt{when}\ \varphi.T\ ] \\ \mathfrak{h} \end{pmatrix} \longrightarrow \begin{pmatrix} c.x[\ \mathrm{match}(\Pi, M)T\ ] \\ \mathfrak{h} \cdot \mathtt{I}(c, x, M) \end{pmatrix}}$$

$$\textsc{sOut}\ \frac{}{\begin{pmatrix} c.x[\ \mathtt{sOut}_{d}\ M.T\ ] \\ \mathfrak{h} \end{pmatrix} \xrightarrow{\mathtt{sO}} \begin{pmatrix} c.x[\ T\ ] \\ \mathfrak{h} \cdot \mathtt{sO}(c, x, M, d) \end{pmatrix}}$$

$$\textsc{sIn}\ \frac{\mathfrak{h} \models \mathrm{match}(\Pi, M)\varphi}{\begin{pmatrix} d.x[\ \mathtt{sIn}_{b}\ \Pi\ \mathtt{when}\ \varphi.T\ ] \\ \mathfrak{h} \cdot \mathtt{sO}(c, x', M, d) \end{pmatrix} \xrightarrow{\mathtt{sI}} \begin{pmatrix} d.x[\ \mathrm{match}(\Pi, M)T\ ] \\ \mathfrak{h} \cdot \mathtt{sO}(c, x', M, d) \cdot \mathtt{sI}(d, x, M, c) \end{pmatrix}}$$

$$\textsc{sCom-l}\ \frac{\begin{pmatrix} P \\ \mathfrak{h} \end{pmatrix} \xrightarrow{\mathtt{sO}} \begin{pmatrix} P' \\ \mathfrak{h}' \end{pmatrix} \qquad \begin{pmatrix} Q \\ \mathfrak{h}' \end{pmatrix} \xrightarrow{\mathtt{sI}} \begin{pmatrix} Q' \\ \mathfrak{h}'' \end{pmatrix}}{\begin{pmatrix} P \mathbin{|||} Q \\ \mathfrak{h} \end{pmatrix} \longrightarrow \begin{pmatrix} P' \mathbin{|||} Q' \\ \mathfrak{h}'' \end{pmatrix}}$$

$$\textsc{LookUp}\ \frac{\begin{pmatrix} c.x[\ \{^{n}/_{v}\}T\ ] \\ \mathfrak{h} \end{pmatrix} \xrightarrow{\alpha} \begin{pmatrix} c.x[\ T'\ ] \\ \mathfrak{h}' \end{pmatrix} \qquad \mathfrak{h} \models \mathsf{looksUp}(c, x, n, \tau, d, A)}{\begin{pmatrix} c.x[\ \mathtt{LookUp}_{d}\ (v : \tau, A)\ \mathtt{in}\ T\ ] \\ \mathfrak{h} \end{pmatrix} \xrightarrow{\alpha} \begin{pmatrix} c.x[\ T'\ ] \\ \mathfrak{h}' \end{pmatrix}}$$

$$\textsc{Par-l}\ \frac{\begin{pmatrix} P \\ \mathfrak{h} \end{pmatrix} \xrightarrow{\alpha} \begin{pmatrix} P' \\ \mathfrak{h}' \end{pmatrix}}{\begin{pmatrix} P \mathbin{|||} Q \\ \mathfrak{h} \end{pmatrix} \xrightarrow{\alpha} \begin{pmatrix} P' \mathbin{|||} Q \\ \mathfrak{h}' \end{pmatrix}}$$

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

Table 5. Process and thread execution

$$\begin{array}{rll} 1. & \mathtt{Alice} \rightarrow \mathtt{Bob} & : \{\!|(x_{\mathtt{Alice}}, \mathtt{Alice})|\!\}^{+}_{p^{+}_{\mathtt{Bob}}} \\ 2. & \mathtt{Bob} \rightarrow \mathtt{Alice} & : \{\!|(x_{\mathtt{Alice}}, x_{\mathtt{Bob}})|\!\}^{+}_{p^{+}_{\mathtt{Alice}}} \\ 3. & \mathtt{Alice} \rightarrow \mathtt{Bob} & : \{\!|x_{\mathtt{Bob}}|\!\}^{+}_{p^{+}_{\mathtt{Bob}}} \end{array}$$

Table 6. Simple NSPK Narration

tacit) is synchronous. To achieve this, we introduce two auxiliary transition relations $\xrightarrow{\mathtt{sI}}$ and $\xrightarrow{\mathtt{sO}}$ not visible on the top level (closed-world assumption). 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 *nondeterminism* 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.

3 Formal modelling with C^3

In this section, we show how a typical protocol would be formulated within C^3. The example we chose does not take full advantage of all the features, but it should nevertheless give an impression of the simplicity and level of explicitness that C^3 provides.

3.1 Core NSPK

In keeping with tradition, we present a version of the well-known Needham Schroeder Public Key (NSPK) authentification protocol from [22], which we give according to the so-called *narration* format in Table 6. In this narration, many tiny operations are implicitly supposed to happen and be understood. More precisely, when Bob receives message 1 that is encrypted with his own public key $p^{+}_{\mathtt{Bob}}$, he is implicitly expected to first decrypt the message, then take the second component denoting the name Alice of the message's sender such that Bob can then look up Alice's public key $p^{+}_{\mathtt{Alice}}$. For the second message, Bob invents a new nonce $x_{\mathtt{Bob}}$, that he then sends back to Alice, together with Alice's nonce $x_{\mathtt{Alice}}$. Finally, Alice confirms the reception of Bob's nonce.

This protocol can be written in C^3 as shown in Table 7. Here, we define named chunks of protocol code like NSPK$_{\mathtt{RESP}}$(slf) := T, which then can be instantiated as NSPK$_{\mathtt{RESP}}$(Bob) $\overset{\text{def}}{=} \{^{\mathtt{Bob}}/_{slf}\}T$. For simplicity, we omitted a CCS-like definitional mechanism for process constants (here: NSPK, NSPK$_{\mathtt{INIT}}$ and NSPK$_{\mathtt{RESP}}$) from the calculus, but there is no problem in adding it. We use New $(n_s : \mathtt{X})$ as a shorthand for New $(n_s : \mathtt{X}, ■)$ and In $\{\!|n_s|\!\}^{+}_{\overline{k_s}}$ for In $\{\!|n_s|\!\}^{+}_{\overline{k_s}}$ when $\top$.

The code for a single run of the NSPK protocol with Alice and Bob playing the roles of initiator and responder, respectively, results from the instantiation of the "role names" (the formal parameters *init* and *resp*) by actual participants names (Alice and

```
$\mathrm{NSPK}_{\mathtt{INIT}}(slf, oth) :=$
    $\mathtt{New}\ (n_s : \mathtt{X}).$
    $\mathtt{LookUp}_{oth}\ (k_o : \mathtt{K}^+, oth)\ \mathtt{in}$
    $\mathtt{Out}_{oth}\ \{\!|(n_s, slf)|\!\}^{+}_{k_o}\ .$

    $\mathtt{LookUp}_{slf}\ (k_s : \mathtt{K}^-, slf)\ \mathtt{in}$
    $\mathtt{In}\ \{\!|(n_s, v)|\!\}^{+}_{\overline{k_s}}\ \mathtt{when}\ v : \mathtt{X}\ .$
    $\mathtt{Out}_{oth}\ \{\!|v|\!\}^{+}_{k_o}\ .\ 1$

$\mathrm{NSPK}_{\mathtt{RESP}}(slf) :=$

    $\mathtt{LookUp}_{slf}\ (k_s : \mathtt{K}^-, slf)\ \mathtt{in}$
    $\mathtt{In}\ \{\!|(v, oth)|\!\}^{+}_{\overline{k_s}}\ \mathtt{when}\ v : \mathtt{X} \wedge oth : \mathtt{P}\ .$
    $\mathtt{New}\ (n_s : \mathtt{X})\ .$
    $\mathtt{LookUp}_{oth}\ (k_o : \mathtt{K}^+, oth)\ \mathtt{in}$
    $\mathtt{Out}_{oth}\ \{\!|(v, n_s)|\!\}^{+}_{k_o}\ .$
    $\mathtt{In}\ \{\!|n_s|\!\}^{+}_{\overline{k_s}}\ .\ 1$

$\mathrm{NSPK}(init, resp, x_i, x_r) :=$
    $init.x_i[\ \mathrm{NSPK}_{\mathtt{INIT}}(init, resp)\ ]\ \ |||\ \ resp.x_r[\ \mathrm{NSPK}_{\mathtt{RESP}}(resp)\ ]$
```

Table 7. Protocol code for core NSPK

Bob) as in NSPK(Alice, Bob, x_a, x_b). 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 within the process term using corresponding prefixes for name generation and secure communication.) In our scenario, we start from an initial history $\mathfrak{h}$, defined below, where Alice and Bob generate their own private keys. By redundancy we omit the secure outputs.

$$\mathfrak{h} := \epsilon \cdot \mathtt{N}(\mathtt{Alice}, x_a, k_{\mathtt{Alice}}, a) \cdot \mathtt{N}(\mathtt{Bob}, x_b, k_{\mathtt{Bob}}, b) \cdot \\ \mathtt{sI}(\mathtt{Alice}, x_a, k^{+}_{\mathtt{Bob}}, \mathtt{Bob}) \cdot \mathtt{sI}(\mathtt{Bob}, x_b, k^{+}_{\mathtt{Alice}}, \mathtt{Alice})$$

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

$$\mathrm{NSPK}(\mathtt{Alice}, \mathtt{Bob}, x_a, x_b)\ \ |||\ \ \mathrm{NSPK}(\mathtt{Bob}, \mathtt{Caesar}, y_b, y_c)$$

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 NSPK with non-local 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 for this task. In this case, both participants will have to perform a non-local lookup to aquire the public key of the other. The client and server parts of a subprotocol performing this non-local key lookup are given in Table 8.

In the NSPK protocol definition, the local lookups for the public key of the counterpart are then replaced by $\mathtt{sOut}_{slf}\ oth.\mathtt{sIn}_{slf}\ (k_o, oth)\ \mathtt{when}\ k_o : \mathtt{K}^+$. (This is quite similar

$\mathrm{Lkup}_{\mathtt{CLIENT}}(\mathit{slf}, \mathit{ttp}) :=$	$\mathrm{Lkup}_{\mathtt{SERVER}}(\mathit{slf}) :=$
$\quad \mathtt{sIn}_{\mathit{slf}}\ v\ \mathtt{when}\ v : \mathtt{P}.$	
$\quad \mathtt{Out}_{\mathit{ttp}}\ (v, \mathit{slf}).$	$\quad \mathtt{In}\ (v, \mathit{oth})\ \mathtt{when}\ v : \mathtt{P} \wedge \mathit{oth} : \mathtt{P}.$
$\quad \mathtt{LookUp}_{\mathit{ttp}}\ (k_{\mathit{ttp}} : \mathtt{K}^{+}, \mathit{ttp})\ \mathtt{in}$	$\quad \mathtt{LookUp}_{\mathit{slf}}\ (k_s : \mathtt{K}^{-}, \mathit{slf})\ \mathtt{in}$
	$\quad \mathtt{LookUp}_{\mathit{slf}}\ (k_o : \mathtt{K}^{+}, v)\ \mathtt{in}$
$\quad \mathtt{In}\ \{\!\vert(\mathit{key}, v)\vert\!\}^{-}_{\overline{k_{\mathit{ttp}}}}\ \mathtt{when}\ \mathit{key} : \mathtt{K}^{+}.$	$\quad \mathtt{Out}_{\mathit{oth}}\ \{\!\vert(k_o, v)\vert\!\}^{-}_{k_s}.1$
$\quad \mathtt{sOut}_{\mathit{slf}}\ (\mathit{key}, v).1$	

Table 8. Code for NSPK with non-local key lookup

to the way is dealt with in CPPL [12].) The return values from the lookup subprotocol contain the name of the owner of the sought public key, in order to avoid confusion in a multisession scenario. This is an illustration of addressing messages at the thread level using pattern matching on the message content.

4 Conclusion

We believe having instigated with 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 the *co-designed logic* CPL.

In future work, we wish to extend 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; (3) weak real time for time stamps and timed keys [23]; and (4) computational complexity for cryptographic computation [24]. Regarding meta-theory, we seek logical characterisation results for C^3-processes and C^3's observational equivalence (cf. Appendix A.2). Moreover, we are planning to study equivalence-based specification and verification based on C^3's observational equivalence and to bridge the gap between protocol design and implementation by connecting C^3 to the cryptographic library of [25].

References

1. Anderson, R., Needham, R.: Programming Satan's computer. Volume 1000 of Lecture Notes in Computer Science., 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. Boyd, C.: Security architectures using formal methods. IEEE Journal on Selected Areas in Communication **11**(5) (1993)
8. Song, D., Berezin, S., Perrig, A.: Athena, a novel approach to efficient automatic security protocol analysis. Journal of Computer Security **9**(1,2) (2001)
9. 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)
10. Blanchet, B.: An efficient cryptographic protocol verifier based on Prolog rules. In: IEEE Computer Security Foundations Workshop. (2001)
11. Trusted Logic, ENS Cachan, Verimag: (Projet EVA) http://www-eva.imag.fr/.
12. Guttman, J.D., Herzog, J.C., Ramsdell, J.D., Sniffen, B.T.: Programming cryptographic protocols. [26] 116–145
13. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: proving security protocols correct. Journal of Computer Security **7**(2-3) (1999)
14. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The Spi-calculus. Information and Computation **148**(1) (1999)
15. Crazzolara, F., Winskel, G.: Events in security protocols. In: Proceedings of the ACM Conference on Computer and Communications Security. (2001)
16. Ryan, P., Schneider, S., Goldsmith, M., Lowe, G., Roscoe, B.: The Modelling and Analysis of Security Protocols: the CSP Approach. Addison-Wesley (2000)
17. 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
18. Durgin, N., Mitchell, J.C., Pavlovic, D.: A compositional logic for proving security properties of protocols. Journal of Computer Security **11**(4) (2003)
19. Dolev, D., Yao, A.C.: On the security of public key protocols. IEEE Transactions on Information Theory **29**(12) (1983)
20. Haack, C., Jeffrey, A.: Pattern-matching Spi-calculus. In: Proceedings of the Workshop on Formal Aspects in Security and Trust. (2004)
21. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6**(1) (1998)
22. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of Applied Cryptography. CRC Press (1996)
23. Bozga, L., Ene, C., Lakhnech, Y.: A symbolic decision procedure for cryptographic protocols with time stamps. The Journal of Logic and Algebraic Programming **65** (2005)
24. Mitchell, J.C., Ramanathan, A., Scedrov, A., Teague, V.: A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. Theoretical Computer Science **353**(1–3) (2006)
25. Backes, M., Pfitzmann, B., Waidner, M.: A universally composable cryptographic library. In: Proceedings of the ACM Conference on Computer and Communication Security. (2003)
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)

27. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning about Knowledge. MIT Press (1995)

A Auxiliary definitions

A.1 Auxiliary functions

The kernel operator analz [21]

$$\frac{M \in \mathcal{K}}{M \in \mathrm{analz}(\mathcal{K})} \qquad \frac{\{\!|M|\!\}_k \in \mathrm{analz}(\mathcal{K}) \quad k \in \mathrm{analz}(\mathcal{K})}{M \in \mathrm{analz}(\mathcal{K})}$$

$$\frac{\{\!|M|\!\}^+_{p^+} \in \mathrm{analz}(\mathcal{K}) \quad p \in \mathrm{analz}(\mathcal{K})}{M \in \mathrm{analz}(\mathcal{K})} \qquad \frac{\{\!|M|\!\}^-_{p} \in \mathrm{analz}(\mathcal{K}) \quad p^+ \in \mathrm{analz}(\mathcal{K})}{M \in \mathrm{analz}(\mathcal{K})}$$

$$\frac{(M, M') \in \mathrm{analz}(\mathcal{K})}{M \in \mathrm{analz}(\mathcal{K})} \qquad \frac{(M', M) \in \mathrm{analz}(\mathcal{K})}{M \in \mathrm{analz}(\mathcal{K})}$$

The closure operator synth [21]

$$\frac{M \in \mathcal{K}}{M \in \mathrm{synth}(\mathcal{K})} \qquad \frac{M \in \mathrm{synth}(\mathcal{K})}{\lceil M \rceil \in \mathrm{synth}(\mathcal{K})} \qquad \frac{M \in \mathrm{synth}(\mathcal{K}) \quad k \in \mathcal{K}}{\{\!|M|\!\}_k \in \mathrm{synth}(\mathcal{K})}$$

$$\frac{M \in \mathrm{synth}(\mathcal{K}) \quad p^+ \in \mathcal{K}}{\{\!|M|\!\}^+_{p^+} \in \mathrm{synth}(\mathcal{K})} \qquad \frac{M \in \mathrm{synth}(\mathcal{K}) \quad p \in \mathcal{K}}{\{\!|M|\!\}^-_{p} \in \mathrm{synth}(\mathcal{K})}$$

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

A.2 Macro-defined formulae

$\bot$	:=	$\mathtt{Eve} : \mathtt{P}$	false
$\top$	:=	$\neg\top$	true
$\phi \vee \phi'$	:=	$\neg(\phi \wedge \phi')$	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
$\forall(v : \theta)(\phi)$	:=	$\forall v(v : \theta \rightarrow \phi)$	
$\exists(v : \theta)(\phi)$	:=	$\exists v(v : \theta \wedge \phi)$	
$F = F'$	:=	$F \preccurlyeq F' \wedge F' \preccurlyeq F$	F is equal to F'
$F \prec F'$	:=	$F \preccurlyeq F' \wedge \neg F = F'$	F is a strict subterm of F'
$F : \emptyset$	:=	$\bot$	
$F : \mathtt{H}[\theta]$	:=	$\exists(v : \theta)(F = \lceil v \rceil)$	

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

B Process equivalence

We define observational equivalence of cryptographic processes based on the concepts of *structurally indistinguishable protocol histories* and *cryptographic parsing*. Cryptographic parsing captures an agent's capability to understand the structure of a cryptographically obfuscated message.

Cryptographic parsing parses unintelligible messages to the abstract message $\blacksquare$. Indeed, the abstract 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*.

Definition 6 (Cryptographic parsing). *The cryptographic parsing function* $(\!|\cdot|\!)^{\mathfrak{h}}_a$ *associated with an agent a and a protocol history* $\mathfrak{h}$ *(and complying with the assumptions of perfect cryptography) is defined by Table 5.*

For notational convenience, we are going to write $\varepsilon(a)$ for any protocol event as mentioned in Section **??**, $\varepsilon(a, n)$ 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* $\mathfrak{h}$ *and* $\mathfrak{h}'$ *are* structurally indistinguishable from the viewpoint of *an agent a, written* $\mathfrak{h} \approx_a \mathfrak{h}'$*, :iff a observes the same* event pattern *and the same* data patterns *in* $\mathfrak{h}$ *and* $\mathfrak{h}'$*. Formally, for all* $\mathfrak{h}, \mathfrak{h}' \in \mathcal{H}$*,* $\mathfrak{h} \approx_a \mathfrak{h}'$ *:iff* $\mathfrak{h} \approx^{(\mathfrak{h},\mathfrak{h}')}_a \mathfrak{h}'$ *where,*

– *given that a is a legitimate participant or the adversary* Eve*,*

1. $$\frac{}{\epsilon \approx^{(\mathfrak{h},\mathfrak{h}')}_a \epsilon}$$

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

3. $$\frac{\mathfrak{h}_l \approx^{(\mathfrak{h},\mathfrak{h}')}_a \mathfrak{h}_r}{\mathfrak{h}_l \cdot \varepsilon(a, M) \approx^{(\mathfrak{h},\mathfrak{h}')}_a \mathfrak{h}_r \cdot \varepsilon(a, M')} \quad (\!| M |\!)^{\mathfrak{h}}_a = (\!| M' |\!)^{\mathfrak{h}'}_a$$

$$
\begin{aligned}
\llparenthesis\, n \,\rrparenthesis_a^{\mathfrak{h}} &:= n\\
\llparenthesis\, \blacksquare \,\rrparenthesis_a^{\mathfrak{h}} &:= \blacksquare\\
\llparenthesis\, \ulcorner M \urcorner \,\rrparenthesis_a^{\mathfrak{h}} &:= \begin{cases} \ulcorner \llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}} \urcorner & \text{if } \mathfrak{h} \models a \mathrel{\mathsf{k}} M \text{, and}\\ \blacksquare & \text{otherwise.} \end{cases}\\
\llparenthesis\, \{\!|M|\!\}_k \,\rrparenthesis_a^{\mathfrak{h}} &:= \begin{cases} \{\!|\llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}}|\!\}_k & \text{if } \mathfrak{h} \models a \mathrel{\mathsf{k}} k \text{, and}\\ \blacksquare & \text{otherwise.} \end{cases}\\
\llparenthesis\, \{\!|M|\!\}^+_{p^+} \,\rrparenthesis_a^{\mathfrak{h}} &:= \begin{cases} \{\!|\llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}}|\!\}^+_{p^+} & \text{if } \mathfrak{h} \models a \mathrel{\mathsf{k}} p^+ \wedge (a \mathrel{\mathsf{k}} p \vee a \mathrel{\mathsf{k}} M)\\ \{\!|\llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}}|\!\}^+_{\blacksquare} & \text{else if } \mathfrak{h} \models \neg\, a \mathrel{\mathsf{k}} p^+ \wedge a \mathrel{\mathsf{k}} p \text{, and}\\ \blacksquare & \text{otherwise.} \end{cases}\\
\llparenthesis\, \{\!|M|\!\}^-_{p} \,\rrparenthesis_a^{\mathfrak{h}} &:= \begin{cases} \{\!|\llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}}|\!\}^-_{p} & \text{if } \mathfrak{h} \models a \mathrel{\mathsf{k}} p \wedge (a \mathrel{\mathsf{k}} p^+ \vee a \mathrel{\mathsf{k}} M)\\ \{\!|\llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}}|\!\}^-_{\blacksquare} & \text{else if } \mathfrak{h} \models \neg\, a \mathrel{\mathsf{k}} p \wedge a \mathrel{\mathsf{k}} p^+ \text{, and}\\ \blacksquare & \text{otherwise.} \end{cases}\\
\llparenthesis\, (M, M') \,\rrparenthesis_a^{\mathfrak{h}} &:= (\llparenthesis\, M \,\rrparenthesis_a^{\mathfrak{h}}, \llparenthesis\, M' \,\rrparenthesis_a^{\mathfrak{h}})
\end{aligned}
$$

Table 9. Parsing on cryptographic messages

– *given that a is a legitimate participant,*

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

– *given that a is the adversary* Eve,

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

$$
5.\quad \frac{\mathfrak{h}_l \approx_{\mathtt{Eve}}^{(\mathfrak{h},\mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \mathtt{I}(b, x, M) \approx_{\mathtt{Eve}}^{(\mathfrak{h},\mathfrak{h}')} \mathfrak{h}_r \cdot \mathtt{I}(b, x, M')}\; \llparenthesis\, M \,\rrparenthesis_{\mathtt{Eve}}^{\mathfrak{h}} = \llparenthesis\, M' \,\rrparenthesis_{\mathtt{Eve}}^{\mathfrak{h}'}
$$

$$
6.\quad \frac{\mathfrak{h}_l \approx_{\mathtt{Eve}}^{(\mathfrak{h},\mathfrak{h}')} \mathfrak{h}_r}{\mathfrak{h}_l \cdot \mathtt{O}(b, x, M, c) \approx_{\mathtt{Eve}}^{(\mathfrak{h},\mathfrak{h}')} \mathfrak{h}_r \cdot \mathtt{O}(b, x, M', c)}\; \llparenthesis\, M \,\rrparenthesis_{\mathtt{Eve}}^{\mathfrak{h}} = \llparenthesis\, M' \,\rrparenthesis_{\mathtt{Eve}}^{\mathfrak{h}'}
$$

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. $\mathfrak{h}_l \approx_{\cdot}^{(\mathfrak{h},\mathfrak{h}')} \mathfrak{h}_r$). Learning new keys may render intelligible past messages to an agent a in the present that were not 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 [27]
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 cryptographic processes, i.e., models of cryptographic protocols, of some set $\mathcal{P}$. 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 in CPL, and is another element reflecting 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 $\mathfrak{s}_1 \gtrapprox_a^* \mathfrak{s}_2$, :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 $\mathfrak{s}_1 \gtrapprox_a^* \mathfrak{s}_2$ and $\mathfrak{s}_2 \gtrapprox_a^* \mathfrak{s}_1$.*