Toward Dynamic Epistemic Verification of Zero-Knowledge Protocols*

Gabriele Costa†, Cosimo Perini Brogi*,†

IMT School for Advanced Studies Lucca, Italy

Abstract

We outline a novel approach for formally verifying zero-knowledge protocols building on Dynamic Epistemic Logic (DEL) as an abstract semantics for a low-level protocol specification language called SPEC. One of the main benefits is that our methodology abstracts the logical structure of the interactions from the mathematical subtleties related to cryptographic primitives. Furthermore, we leverage the DEL action structures to verify the knowledge dynamics generated by the protocol runs. We illustrate our methodology by applying it to a new protocol called BKP, and we prove that it meets the participants' goals.

Keywords

Zero-Knowledge Proofs, Security Verification, Dynamic Epistemic Logic, Action models, Protocol Specification

1. Introduction

Zero Knowledge Proofs (ZKP) play a key role in security-critical applications, such as blockchain technology [1] and e-voting platforms. Their goal is to protect privacy and improve security in digital interactions, by allowing one party to prove the authenticity or possession of information without revealing it. They are often crucial for preserving data confidentiality and building trust between interacting parties.

The formal verification of the epistemic properties of ZKP is thus essential *and* challenging at the same time: the definition of a general framework for their design and verification is still an open problem. Very often, only a combination of methods provides robust mathematical proofs that security desiderata are met by specific ZKP.[1]

In this paper, we suggest, by a working example, how epistemic logics can help in formally verifying ZKP of a security protocol, challenging some known limitations (according to [6]) of those systems for characterising knowledge in a cryptographic setting.

ITASEC 2024: The Italian Conference on CyberSecurity, April 8–12, 2024, Salerno (Italy)

*Work partially supported by the project SERICS – SWOPS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union NextGenerationEU.

*Corresponding author.

†These authors contributed equally.

gabriele.costa@imtlucca.it (G. Costa); cosimo.perinibrogi@imtlucca.it (C. Perini Brogi)

https://www.imtlucca.it/it/gabriele.costa (G. Costa); https://www.imtlucca.it/it/cosimo.perinibrogi (C. Perini Brogi)

0000-0002-9385-3998 (G. Costa); 0000-0001-7883-5727 (C. Perini Brogi)

[1] See, e.g., [2, 3] and [4, 5], as well as works mentioned in Section 6 below.

We present an original protocol specification language (SPEC) designed to offer a lucid and exact framework for delineating protocol actions. We then use the semantics of dynamic epistemic logic (DEL), based on action models [7], to interpret statements in SPEC. Next, we show how this DEL-based abstract semantics can correctly model the information flow and knowledge evolution of honest interactive agents in a single run of an original protocol (BKP) we implemented via SPEC statements.

More precisely, we prove that by applying to the epistemic model for the initial configuration of BKP the action model interpretations of (respectively) the honest prover and the honest verifier SPEC-formalisations, we obtain new epistemic models that validate, for each participant, the protocol security desiderata (namely, zero-knowledge, proof-of-knowledge, and no-repudiation, for the honest prover; proof-of-knowledge for the honest verifier) rendered as formulas in the language of epistemic logic.

The results on BKP show that our approach of modelling a low-level specification language, as our SPEC, via a semantics based on mathematical structures for dynamic epistemic reasoning allows us to faithfully represent the evolution of the protocol from the perspectives of both the participants (prover and verifier), and asses the main security features expected from the protocol by each of the interacting agents.

The paper is structured as follows: in Section 2 we present our broken key protocol (BKP) as a working example of zero-knowledge protocol; in Section 3 we introduce the syntax and operational semantics for the Simple Protocol Epistemic Calculus (SPEC), our protocol specification language; Section 4 recalls the basics of dynamic epistemic logic (DEL) which we apply to define an abstract interpretation for statements in SPEC (Definition 6); in Section 5 we show how to model the BKP evolution using that DEL-based abstract semantics and prove that, after a single run of the protocol, the prover's and verifier's goals are satisfied; in Section 6 we discuss related work.

2. Broken Key Protocol

In this section, we introduce the *broken key protocol* (BKP) that will also serve as a working example along the rest of the paper. The execution of a single session of BKP is depicted in Figure 1.

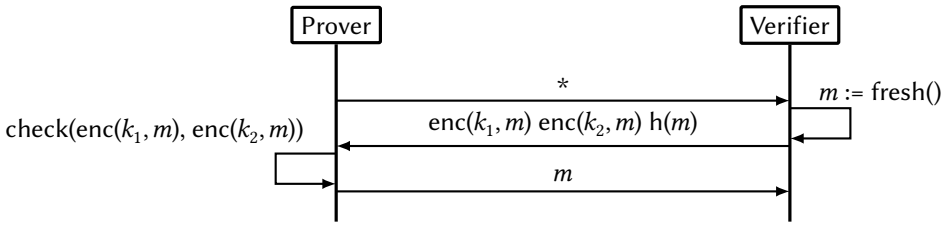


Figure 1: Message sequence chart for BKP.

The protocol involves a verifier V, who owns two secret encryption keys k_1 and k_2, a prover P, who wants to inform V that one between her keys is compromised without revealing which one. To achieve their goals, V and P run the following protocol steps:

1. P initiates the session by pinging V (with a constant message $*$).
2. V generates a fresh, random message m, encrypts it with both k_1 and k_2 ($\mathrm{enc}(k_1, m)$ and $\mathrm{enc}(k_2, m)$, respectively), and computes the hash of m ($h(m)$). Then, she sends the three values $\mathrm{enc}(k_1, m)$, $\mathrm{enc}(k_2, m)$ and $h(m)$ to P.
3. P controls the received values by checking that:
   a. $\mathrm{enc}(k_1, m)$ and $\mathrm{enc}(k_2, m)$ are ciphertexts for the same message (e.g., through a *secure multiparty computation* equality test [8]), and;
   b. $\mathrm{enc}(k_1, m) \neq \mathrm{enc}(k_2, m)$, i.e., two different keys have been used for encrypting m.

   If this is the case, P continues by decrypting one of the two ciphertexts (depending on which between k_1 and k_2 is compromised) and sends m back to V.

3. Protocol language

We present the syntax and operational semantics of our protocol specification language, named *Simple Protocol Epistemic Calculus* (SPEC).

We start by providing the syntax of protocol statements inspired by the Simple Protocol Specification language (SPS) of [9].

Definition 1. *A* protocol statement S *is a term generated through the following grammar.*

$$S ::= x := e \mid \twoheadrightarrow_A : e \mid \twoheadleftarrow_B : x \mid [g]S \mid S; S'$$

Briefly, a statement can be an assignment ($x := e$), a sending ($\twoheadrightarrow_A : e$), a reception ($\twoheadleftarrow_B : x$), a conditional ($[g]S$) or a sequence ($S; S'$). We use $x, y, \cdots$ to denote variables. Expressions $e, e', \cdots$ behave as usual and also include uninterpreted function symbols, e.g., $\mathrm{enc}(\ldots)$, and constants, e.g., $*$ and $\blacksquare$. The same holds for boolean guards g, g' in conditional statements. A, B are agent labels used in the communication mechanism detailed in Section 3.1 and Appendix B. Finally, we feel free to use parentheses for grouping terms and we introduce the following abbreviations.

$$\mathtt{skip} \triangleq \_ := *^{2} \qquad \mathtt{fail} \triangleq [x \neq x]\mathtt{skip}^{3} \qquad \twoheadleftarrow_B : e \triangleq \twoheadleftarrow_B : x; [x \neq e]\mathtt{fail}$$

$$\twoheadrightarrow_A : e_1, \ldots, e_n \triangleq \twoheadrightarrow_A : e_1; \ldots; \twoheadrightarrow_A : e_n \qquad \twoheadleftarrow_B : x_1, \ldots, x_n \triangleq \twoheadleftarrow_B : x_1; \ldots; \twoheadleftarrow_B : x_n$$

Example 1. *Consider the BKP protocol described in Section 2. The following statements implement the honest prover S_P and verifier S_V.*

$$S_P \triangleq \quad \twoheadrightarrow_V : *; \twoheadleftarrow_V : x, y, z; [\mathtt{comp}(x, y)][z = \mathrm{h}(\mathtt{trydec}(k, x, y))]\twoheadrightarrow_V : \mathtt{trydec}(k, x, y)$$

$$S_V \triangleq \quad \twoheadleftarrow_P : *; m := \mathtt{fresh}(); \twoheadrightarrow_P : \mathrm{enc}(k_1, m), \mathrm{enc}(k_2, m), \mathrm{h}(m); \twoheadleftarrow_P : x; [x = m]\mathtt{skip}$$

The two statements rely on a few cryptographic functions: h, enc *and* $\mathtt{fresh}$ *are quite traditional, and they represent hashing, encryption and generation of a fresh value, respectively;* $\mathtt{comp}$ *is used by the prover to check whether x and y encrypt the same message with different keys.* $\mathtt{trydec}$ *attempts to decrypt both x and y using key k and returns the cleartext if at least one of the two operations is possible.*

[2] Where _ is a reserved variable name that cannot appear in any other statement.
[3] For some variable x.

In the rest of this paper, with no loss of generality, we also assume protocol statements to be *well typed*. Correct typing is guaranteed statically and dynamically through a utility function $Type(e)$ that assigns the proper type to any expression e. Types can be either *base* or *function*. For instance, we assume basic types to include *Key* and *Msg* (for keys and simple messages, respectively), while function types include *Hash(Msg)* and *Enc(Key,Msg)*. Moreover, we assume that value binding, which occurs in both assignment and receipt statements, always ensures type correctness, e.g., in $x := e$ we have $Type(x) = Type(e)$.

3.1. Operational semantics

In this section, we define the operational semantics of SPEC. We first introduce some preliminary definitions.

Definition 2. *An* agent state σ *is a finite, partial mapping from variable names to values. We use* ε *to denote the empty state and* $\sigma[v/x]$ *for the state* σ *where* x *is (re-)assigned to value* v*. Consequently, the resolution of a variable* x *in a state* σ *is defined as*

$$\sigma(x) = \begin{cases} \emptyset & \text{if } \sigma \text{ is } \varepsilon \\ v & \text{if } \sigma \text{ is } \sigma'[v/x] \\ \sigma'(x) & \text{if } \sigma \text{ is } \sigma'[v/y] \end{cases} \qquad \text{where } \emptyset \text{ stands for the undefined value.}$$

We call the pair $\langle \sigma, S \rangle$ *an* agent configuration.

For explaining the operational semantics rules, we assume that a support function for evaluating expressions and guards (see Definition 1) is defined. We use $[\![e]\!]_\sigma = v$ to denote that, under state σ, expression e evaluates to value v. Similarly, $[\![g]\!]_\sigma = \mathbf{1/0}$ indicates whether, under state σ, the guard g is satisfied (**1**) or not (**0**).

The *structural operational semantics* of an *agent configuration* $\langle \sigma, S \rangle$ is then defined by the rules of Figure 2.

$$\frac{\langle \sigma, S \rangle \longrightarrow \langle \sigma', S'' \rangle}{\langle \sigma, S; S' \rangle \longrightarrow \langle \sigma', S''; S' \rangle}\ (\text{Seq 1}) \qquad \frac{\langle \sigma, S \rangle \longrightarrow \langle \sigma', \cdot \rangle}{\langle \sigma, S; S' \rangle \longrightarrow \langle \sigma', S' \rangle}\ (\text{Seq 2})$$

$$\frac{[\![g]\!]_\sigma = \mathbf{1}}{\langle \sigma, [g]S \rangle \longrightarrow \langle \sigma, S \rangle}\ (\text{Cond 1}) \qquad \frac{[\![g]\!]_\sigma = \mathbf{0}}{\langle \sigma, [g]S \rangle \longrightarrow \text{☠}}\ (\text{Cond 2}) \qquad \frac{[\![e]\!]_\sigma = v}{\langle \sigma, x := e \rangle \longrightarrow \langle \sigma[v/x], \cdot \rangle}\ (\text{Asgn})$$

$$\frac{[\![e]\!]_\sigma = v}{\langle \sigma, \twoheadrightarrow_A : e \rangle \longrightarrow \langle \sigma, \cdot \rangle \uparrow_{A,v}}\ (\text{Send}) \qquad \frac{\langle \sigma, S \rangle \longrightarrow \langle \sigma', S'' \rangle \uparrow_{A,v}}{\langle \sigma, S; S' \rangle \longrightarrow \langle \sigma', S''; S' \rangle \uparrow_{A,v}}\ (\text{Send-P})$$

$$\frac{}{\langle \sigma, \twoheadleftarrow_B : x \rangle \longrightarrow \langle \sigma, \cdot \rangle \downarrow_{B,x}}\ (\text{Recv}) \qquad \frac{\langle \sigma, S \rangle \longrightarrow \langle \sigma', S'' \rangle \downarrow_{B,x}}{\langle \sigma, S; S' \rangle \longrightarrow \langle \sigma', S''; S' \rangle \downarrow_{B,x}}\ (\text{Recv-P})$$

Figure 2: Operational semantics of protocol agents.

Briefly, sequences are evaluated by reducing the first statement (Seq 1) until, eventually, it gets to a terminal configuration (denoted by $\cdot$), and then the second statement is executed (Seq 2). Conditional statements behave as the guarded statement S when the guard is satisfied (Cond 1) or, otherwise, they lead to a faulty configuration ☠ (Cond 2). An assignment updates the current state σ by binding the variable x to the value v obtained from the evaluation of e (Asgn).

Both sending to A (Send) and receiving from B (Recv) lead to special, blocking configurations labelled with $\uparrow_{A,v}$ and $\downarrow_{B,x}$, respectively. Moreover, according to rules (Send-P) and (Recv-P), blocking configurations propagate through the sequences of statements.

Protocol agents compose to form a *choreography*, i.e., the actual execution of a protocol where agents communicate over a network. We provide an SOS for protocol choreographies in Appendix B.

4. Abstract semantics

In this section, we address the task of formalizing through dynamic epistemic logic the epistemic properties of SPEC protocols.

Dynamic Epistemic Logic (DEL) is a branch of modal logic that deals with knowledge change over time. Following [7], here we recall the syntax and semantics of DEL. The following definition describes information flow w.r.t. a dynamic scenario.

Definition 3 (Action model). *Let Atm be a set of propositional atoms. Let $\mathscr{L}$ denote the set of formulas built on top of Atm w.r.t. a given grammar. Let Ag be a finite set of agents. An action model $\mathfrak{A}$ is a quadruple $\langle \mathscr{A}, \approx, \mathsf{pre}, \mathsf{post}\rangle$, where*

- *$\mathscr{A}$ is a finite set of action labels $\{\alpha_1, \cdots, \alpha_n\}$;*[4]
- *$\approx$ is a function associating to each agent $i \in Ag$ an accessibility relation $\approx_i \subseteq \mathscr{A} \times \mathscr{A}$ modelling indistinguishability of two actions for the agent i;*
- *the precondition function $\mathsf{pre} : \mathscr{A} \longrightarrow \mathscr{L}$ assigns to each action α a formula $\varphi \in \mathscr{L}$ such that φ is true in a state iff α can be executed on that state;*
- *the postcondition function $\mathsf{post} : \mathscr{A} \longrightarrow Sub_{\mathscr{L}}$ assigns to each action $\alpha \in \mathscr{A}$ a substitution from Atm to formulas in the language $\mathscr{L}$, i.e. a function behaving as the identity map except for a finite number of atoms; we write $a \mapsto \varphi$ for the substitution that maps the atom a to φ, leaving all the other propositional atoms unchanged; the identity substitution that leaves all the atoms unchanged is denoted by idsub.*

Any action model operates on a relational structure that models a static epistemic situation; by performing an action, the structure of the original model changes, and consequently, the truth values of epistemic assertions do. Moreover, the postcondition function implements a notion of factual change, the fact that performing an action changes the truth value of propositional atoms (as well as epistemic statements) by modifying the basic facts of the world.

For *static* epistemic logic (EL), we need to extend a classical propositional language with an indexed modal operator as in the following

$$\varphi \in \mathscr{L}_{\mathrm{EL}} ::= \top \mid a \mid \neg\varphi \mid \varphi \wedge \psi \mid K_i\varphi$$

where a belongs to a given set of propositional atoms and i belongs to a given finite set of agents. The epistemic formula $K_i\varphi$ formalizes the fact that agent i knows that φ holds.[5]

[4] From now on, we use (indexed) initial Greek letters $\alpha, \beta, \gamma, \delta$ to denote action labels, while terminal Greek letters $\varphi, \psi, \theta, \cdots$ denote generic formulas of a formal language for (dynamic) epistemic logic.

[5] Classical connectives are defined as usual in terms of $\neg$ and $\wedge$ (see e.g. [10]).

The relational semantics for $\mathscr{L}_{\mathrm{EL}}$ is standard, and we report its definition below.

Definition 4. *An* epistemic model $\mathscr{M}$ *is a triple* $\langle W, \{R_l\}, ev\rangle$ *made of a non-empty set* W *"of possible worlds", a set of binary "accessibility" relations* $R_l \subseteq W \times W$ *indexed over the given finite set of agents* Ag,[6] *an evaluation function* $ev : W \times Atm \longrightarrow \{0,1\}$ *associating to each* $x \in W$ *and each* $a \in Atm$ *a truth-value* $ev(x,a) \in \{0,1\}$.

The forcing relation $\Vdash$ *holding between a model* $\mathscr{M} \triangleq \langle W, \{R_i\}, v\rangle$, *a world* $x \in W$ *and a formula* φ *of* $\mathscr{L}_{\mathrm{EL}}$ *is inductively defined on the structure of* φ *as follows.*

$x \Vdash_{\mathscr{M}}$ *a iff* $ev(x,a) = 1$;
$x \Vdash_{\mathscr{M}} \top$ *for any* $x, \mathscr{M}$;
$x \Vdash_{\mathscr{M}} \varphi \wedge \psi$ *iff* $x \Vdash_{\mathscr{M}} \varphi$ *and* $x \Vdash_{\mathscr{M}} \psi$;
$x \Vdash_{\mathscr{M}} K_i\varphi$ *iff for all* $y \in W$, *if* xR_iy, *then* $y \Vdash_{\mathscr{M}} \varphi$.

In words, when $x \Vdash_{\mathscr{M}} \varphi$, *we say that* φ is forced by x in $\mathscr{M}$. *When every world in* $\mathscr{M}$ *forces a formula* φ, *we write* $\mathscr{M} \vDash \varphi$.

For *dynamic* epistemic settings, the syntax of $\mathscr{L}_{\mathrm{EL}}$ needs to be extended to $\mathscr{L}_{\mathrm{DEL}}$ by a dynamic modal operator $[\mathfrak{A}, \alpha]\varphi$, where $\mathfrak{A}$ is an action model, and α is an action in $\mathfrak{A}$: the formula formalises the fact that, after the action α in $\mathfrak{A}$ is performed, φ (belonging to the extended language $\mathscr{L}_{\mathrm{DEL}}$) holds.

To give a precise semantics to action performing, we need to formalize the update of a static relational model via an action model. The resulting structure is called *model update*.

Definition 5 (Model update). *Let* $\mathscr{M} \triangleq \langle W, \{R_i\}, ev\rangle$ *be an epistemic model and* $\mathfrak{A} \triangleq \langle \mathscr{A}, \approx, \mathrm{pre}, \mathrm{post}\rangle$ *be an action model. The model for* $\mathscr{M}$ *updated by* $\mathfrak{A}$ *is denoted by* $\mathfrak{A} \circ \mathscr{M}$ *and consists of the triple* $\langle W', R', ev'\rangle$, *where*

- $W' \triangleq \{\langle x, \alpha\rangle : x \Vdash_{\mathscr{M}} \mathrm{pre}(\alpha)\}$;
- $R' \triangleq \lambda l \in Ag.\{\langle\langle x_1, \alpha_1\rangle, \langle x_2, \alpha_2\rangle\rangle : xR_ly \;\&\; \alpha_1 \approx_l \alpha_2\}$
- $ev'(\langle x, \alpha\rangle, a) = 1$ *iff* $x \Vdash_{\mathscr{M}} \mathrm{post}(\alpha)(a)$.

This allows us to define the meaning of all DEL formulas: it suffices to extend Definition 4 by the following forcing clause:

$$x \Vdash_{\mathscr{M}} [\mathfrak{A}, \alpha]\varphi \qquad \text{iff} \qquad x \Vdash_{\mathscr{M}} \mathrm{pre}(\alpha) \text{ implies that } \langle x, \alpha\rangle \Vdash_{\mathfrak{A}\circ\mathscr{M}} \varphi,$$

where $\mathfrak{A} \circ \mathscr{M}$ is the update model of $\mathscr{M}$ with $\mathfrak{A}$ of Definition 5.

Example 2. *Consider again the BKP protocol. Below, we give the security goals of the protocol in* $\mathscr{L}_{\mathrm{EL}}$.

· *Zero knowledge:* $\varphi_{\mathrm{ZK}} \triangleq \neg K_V(\mathsf{has}_P(k_1)) \wedge \neg K_V(\mathsf{has}_P(k_2))$
· *Proof of knowledge:* $\varphi_{\mathrm{PoK}} \triangleq K_V(\mathsf{has}_P(k_1) \vee \mathsf{has}_P(k_2))$
· *No repudiation:* $\varphi_{\mathrm{NR}} \triangleq K_V(K_P(K_V(\mathsf{has}_P(k_1) \vee \mathsf{has}_P(k_2))))$

In words, φ_{ZK} *states that* V *should not know whether* P *owns* k_1 *or* k_2, φ_{PoK} *states that* V *must know that* P *owns one between* k_1 *and* k_2, *and* φ_{NR} *states that* V *and* P *reciprocally acknowledge that* P *owns one of the two keys.*

[6] We write xR_ly to mean that y is accessible to x for l. Further general conventions are collected in Appendix A.

4.1. Modeling BKP in DEL

This section shows how to interpret SPEC-statements as action models. The actions are applied to epistemic models that formalize the system made of agent states σ *and* the knowledge of each agent, modeled through an accessibility relation.

Intuitively, both an agent state σ_i and an accessibility relation R_i represent information accessible to agent i. Nevertheless, the information encoded by σ_i is *local* to agent i, as it contains i's computational data, and information about any other agent j is not part of σ_i. Instead, the accessibility relation R_i models a more general (though potentially uncertain) information possessed by i, which concerns the status of the whole system of interactive agents. To describe this proper form of knowledge (and uncertainty), agent states do not suffice since, by definition, the information encoded in each state is private. Thus, we start by fixing the set Atm of propositional atoms for BKP, defined as

$$a \in Atm ::= e_1 = e_2 \mid \mathsf{has}_i(e) \mid \mathsf{const}_i(e)$$

where $i \in Ag = \{P, V\}$, e_1, e_2 are expressions, and $=$ denotes identity of values. Formulas for BKP are built according to the grammar for $\mathscr{L}_{DEL}$ on top of this set of atoms.

The atom $\mathsf{has}_i(e)$ expresses the fact that e is stored in the state of agent i. Instead, $\mathsf{const}_i(e)$ denotes that i can build expression e. Trivially, $\mathsf{has}_i(e)$ subsumes $\mathsf{const}_i(e)$; that is encoded by the rule $\dfrac{\mathsf{has}_i(e)}{\mathsf{const}_i(e)}\,(has)$ stating that any local information of i is *per se* constructible by i.

Then, for every uninterpreted function $\mathtt{f}(\ldots)$ appearing in SPEC expressions, we require proper inference rules to define their constructibility. For instance, the rules for the functions used in our working example are the following.

$$\frac{}{\mathsf{const}_i(*)}\,(*) \qquad \frac{}{\mathsf{const}_i(\mathtt{fresh}())}\,(fresh) \qquad \frac{\mathsf{const}_i(m)}{\mathsf{const}_i(\mathsf{h}(m))}\,(h) \qquad \frac{\mathsf{has}_i(m) \quad \mathsf{has}_i(k)}{\mathsf{const}_i(\mathsf{enc}(k,m))}\,(enc)$$

$$\frac{\mathsf{has}_i(\mathsf{enc}(k,m)) \quad \mathsf{has}_i(k)}{\mathsf{const}_i(\mathtt{trydec}(k,\mathsf{enc}(k,m),y))}\,(trydec)_1 \qquad \frac{\mathsf{has}_i(\mathsf{enc}(k,m)) \quad \mathsf{has}_i(k)}{\mathsf{const}_i(\mathtt{trydec}(k,x,\mathsf{enc}(k,m)))}\,(trydec)_2$$

They state that agent i can construct the constant $*$ and $\mathtt{fresh}()$ values. Moreover, if i has a message m, it can compute its hash $\mathsf{h}(m)$ and encrypt it with a key k it also has. Finally, i can decrypt a ciphertext $\mathsf{enc}(k,m)$ when she has the proper key.[7]

For what concerns uninterpreted functions appearing in guards, we assume that an interpretation $(\!|\mathtt{f}(\ldots)|\!)_i$ of $\mathtt{f}(\ldots)$ is defined in terms of a finite set of *literals*, i.e., positive or negative atoms, of our language. Intuitively, $(\!|\mathtt{f}(\ldots)|\!)_i = \{\ell_1, \ldots, \ell_n\}$ represents the fact that the litearals $\ell_1, \ldots, \ell_n$ must be satisfied for $\mathtt{f}(\ldots)$ to be true. For instance, we have that

$$(\!|\mathtt{comp}(\mathsf{enc}(k_1,m_1),\mathsf{enc}(k_2,m_2))|\!)_i \triangleq \{\mathsf{const}_i(\mathsf{enc}(k_1,m_1)), \mathsf{const}_i(\mathsf{enc}(k_2,m_2)), k_1 \neq k_2, m_1 = m_2\}.$$

Next, we need to interpret our protocol statements in terms of action models.

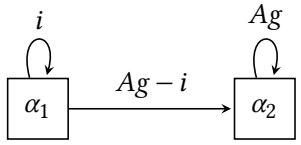
Definition 6. *We define an interpreting function $\langle\!\langle\cdot\rangle\!\rangle$ from protocol statements S (and agent i) to action models $\langle\!\langle S\rangle\!\rangle_i$ by induction on the structure of S as follows.*[8]

[7] It is worth noticing that these rules are not the only candidates. For instance, one may want to model encryption recursively, e.g., to deal with terms such as $\mathsf{enc}(k,\mathsf{enc}(k',m))$. However, nested encryption has no role in the BKP protocol, and thus we omit it.

[8] In these and the following graphics, we adopt some standard visual conventions, detailed in Appendix A.

($x := e$). *The action model $\langle\!\langle x := e \rangle\!\rangle_i$ for the assignment $x := e$ by agent i is given by $\langle \mathscr{A}, \approx, \mathsf{pre}, \mathsf{post} \rangle$ where*

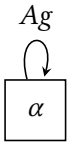
- $\mathscr{A} \triangleq \{\alpha_1, \alpha_2\}$
- $\approx \triangleq \lambda l \in Ag. \begin{cases} \{\langle \alpha_1, \alpha_1 \rangle, \langle \alpha_2, \alpha_2 \rangle\} & \text{when } l = i \\ \{\langle \alpha_1, \alpha_2 \rangle, \langle \alpha_2, \alpha_2 \rangle\} & \text{otherwise} \end{cases}$
- $\mathsf{pre}(\alpha_1) \triangleq \mathsf{const}_i(e)$ *and* $\mathsf{pre}(\alpha_2) \triangleq \top$
- $\mathsf{post}(\alpha_1) \triangleq \mathsf{has}_i(e) \mapsto \top$ *and* $\mathsf{post}(\alpha_2) \triangleq$ *idsub.*



In words, assigning a value to a variable is a private action for the agent performing it: all the other agents involved in the protocol are unaware of the event.

($\rightarrowtail_i : e$). *The action model $\langle\!\langle \rightarrowtail_i : e \rangle\!\rangle_j$ for agent j sending e to agent i is given by $\langle \mathscr{A}, \approx, \mathsf{pre}, \mathsf{post} \rangle$ where*

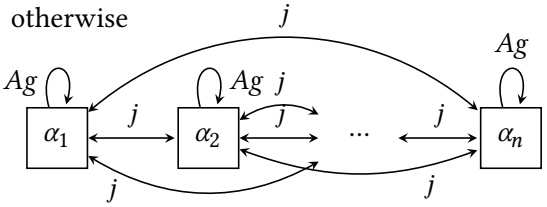
- $\mathscr{A} \triangleq \{\alpha\}$
- $\approx \triangleq \lambda l \in Ag. \{\langle \alpha, \alpha \rangle\}$
- $\mathsf{pre}(\alpha) \triangleq \mathsf{const}_j(e)$
- $\mathsf{post}(\alpha) \triangleq \mathsf{has}_i(e) \mapsto \top$



In words, sending an expression is a public action that can be performed whenever the sender is able to construct the value of that expression; after the event, that value is stored in the local information of the receiver.

($\leftarrowtail_i : x$). *The action model $\langle\!\langle \leftarrowtail_i : x \rangle\!\rangle_j$ for agent j receiving values on variable x from agent i is given by $\langle \mathscr{A}, \approx, \mathsf{pre}, \mathsf{post} \rangle$ where*

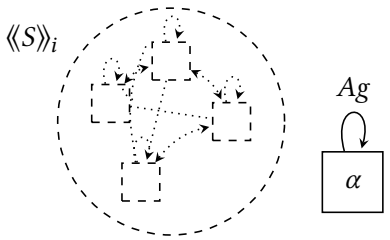
- $\mathscr{A} \triangleq \{\alpha_h : \mathsf{const}_i(e_h) \,\&\, \mathit{Type}(e_h) = \mathit{Type}(x)\}$
- $\approx \triangleq \lambda l \in Ag. \begin{cases} \{\langle \alpha_h, \alpha_m \rangle : \alpha_h \in \mathscr{A} \,\&\, \alpha_m \in \mathscr{A}\} & \text{when } l = j \\ \{\langle \alpha_h, \alpha_h \rangle : \alpha_h \in \mathscr{A}\} & \text{otherwise} \end{cases}$
- $\mathsf{pre} \triangleq \lambda \alpha_h \in \mathscr{A}. \mathsf{const}_i(e_h)$
- $\mathsf{post} \triangleq \lambda \alpha_h \in \mathscr{A}. \mathsf{has}_j(e_h) \mapsto \top$



Thus, we can informally interpret the receiving statement from the agent i as an equivalence class of sending statements from the same agent.

($[g]S$). *The action model $\langle\!\langle [g]S \rangle\!\rangle_i$, for a guarded statement with $\langle\!\langle S \rangle\!\rangle_i = \langle \mathscr{A}^S, \approx^S, \mathsf{pre}^S, \mathsf{post}^S \rangle$, is given by $\langle \mathscr{A}, \approx, \mathsf{pre}, \mathsf{post} \rangle$ where*

- $\mathscr{A} \triangleq \mathscr{A}^S \cup \{\alpha\}$, *where α does not occur in $\mathscr{A}^S$*
- $\approx \triangleq \lambda i \in Ag. \approx_i^S \cup \{\langle \alpha, \alpha \rangle\}$
- $\mathsf{pre} \triangleq \lambda \beta \in \mathscr{A}. \begin{cases} \bigwedge (\!|g|\!)_i \wedge \mathsf{pre}^S(\beta) & \text{when } \beta \neq \alpha \\ \neg \bigwedge (\!|g|\!)_i & \text{otherwise} \end{cases}$
- $\mathsf{post} \triangleq \lambda \beta \in \mathscr{A}. \begin{cases} \mathsf{post}^S(\beta) & \text{when } \beta \neq \alpha \\ \text{☠} \mapsto \top & \text{otherwise} \end{cases}$



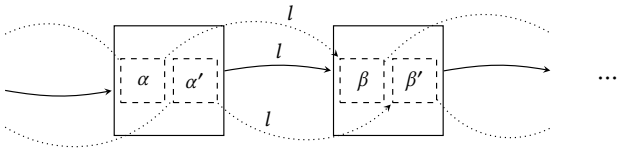
where the expression ☠ $\mapsto \top$ denotes the substitution mapping, for each $l \in Ag$, $\mathsf{has}_l(\blacksquare)$ to $\top$.

In words, we model the fact that in SPEC guards are external*: in* $[g]S$ *whenever the guard* g *evaluates to true by agent* i*, we proceed with the action formalized by* S*; when the same guard* g *evaluates to false, we have a public announcement of protocol failure.*

($S; S'$**).** *The action model* $\langle\!\langle S; S' \rangle\!\rangle_i$ *is given by* $\langle \mathscr{A}, \approx, \mathsf{pre}, \mathsf{post} \rangle$*, obtained by the composition of the action model* $\langle\!\langle S \rangle\!\rangle_i = \langle \mathscr{A}^{\mathcal{S}}, \approx^S, \mathsf{pre}^S, \mathsf{post}^S \rangle$ *with the action model* $\langle\!\langle S' \rangle\!\rangle_i = \langle \mathscr{A}^{\mathcal{S}'}, \approx^{S'}, \mathsf{pre}^{S'}, \mathsf{post}^{S'} \rangle$*, as detailed in e.g. [11, Def. 6.7]:*

- $\mathscr{A} \triangleq \mathscr{A}^S \times \mathscr{A}^{S'}$
- $\approx \triangleq \lambda l \in Ag.\{\langle\langle \alpha, \alpha' \rangle, \langle \beta, \beta' \rangle\rangle \,:\, \alpha \approx_l^S \beta \,\&\, \alpha' \approx_l^{S'} \beta'\}$
- $\mathsf{pre} \triangleq \lambda\langle \alpha, \alpha' \rangle \in \mathscr{A}.\mathsf{pre}^S(\alpha) \wedge [\langle\!\langle S \rangle\!\rangle_i, \alpha]\mathsf{pre}^{S'}(\alpha')$
- $\mathsf{post} \triangleq \lambda\langle \alpha, \alpha' \rangle \in \mathscr{A}.\mathsf{post}^S(\alpha) \cdot \mathsf{post}^{S'}(\alpha')$*, where* $f \cdot g$ *is an abbreviation for* $\lambda x.g(f(x))$*.*

Graphically, we have



Finally, we model the agent states by an epistemic structure. Assuming that the given protocol involves n agents, the whole system state corresponds to an epistemic model with possible worlds as n-tuples of lists assigning values to variables (defined on the basis of the current σ_i for each $i \in Ag$). The epistemic uncertainty of an individual agent i regarding the actual state of the system is represented by an accessibility relation R_i among such worlds. The evaluation function ev for worlds and propositional atoms is given by the information thus encoded in each n-tuple and according to the semantics of $\mathsf{const}_i(e)$ defined in Section 4.1.

Graphically, to distinguish the local information of, say, agent i from agent j, we use colors: when agent i assigns a value to a variable of hers, we write it inside the node's area with a color conventionally assigned to i; similarly for j. Epistemic possibility and uncertainty for agent i are represented by (bi-)directed arrows, labeled by i.[9] The figures in the next section will make these graphical conventions more concrete.

5. Assessing knowledge in BKP

We can finally prove that each agent implementation complies with the respective goal. First, we fix the epistemic scenario as described by the initial assumptions of the protocol: V has two keys (k_1 and k_2), and P has at most one[10] of them. We call this model $\mathscr{I}_{\mathrm{BKP}}$ and we depict it in Figure 3 (we assigned blue to P and red to V).

Performing S_P**.** We compute the update of $\mathscr{I}_{\mathrm{BKP}}$ with $\langle\!\langle S_P \rangle\!\rangle_i$, where S_P is the SPEC-statement of Example 1 for the honest prover in BKP. S_P starts by sending $*$: this does not change the epistemic model, apart from adding $*$ to the local information of V, which we skip for brevity.

Then, P receives x. Since the actual message sent by V is unknown, P must assume that any constructible (and type-compatible) message might be received. This results in an unfolding of

[9] As for the $\approx$ symbol of Definition 6, we omit the R symbol to enhance picture readability.
[10] We neglect the case when P knows both the keys since it is irrelevant for our scenario.

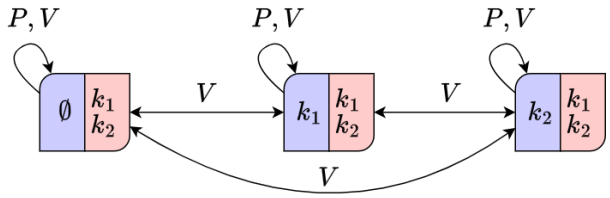


Figure 3: Initial epistemic configuration, modelled via $\mathcal{I}_{\text{BKP}}$. Initially, although V has both k_1 and k_2, three configurations are possible, i.e., P has no key (left node), P has k_1 (middle node) or P has k_2 (right node). In terms of knowledge, P can distinguish between these three states (self-loops), but V cannot.

the epistemic model where, in each world, P's local information is extended with the possible message from V: from P's perspective, all the states obtained by extending the same initial state with V's message are epistemically indistinguishable. Next, P receives another message on variable y. This event is analogous to the previous one and leads to a similar effect: P cannot distinguish what message V will send. The third receiving, stored in z, behaves in the same way. Because of our typing discipline, the resulting model is then made of $3 \times 4 \times 4 \times 2 = 96$ worlds. Each is distinct from all the others only because of the local information of P.[11]

The first guard $[\texttt{comp}(x, y)]$ allows us to select from the 96 nodes of the epistemic model those containing the local information of P triggering the guard. The subsequent guard $[z = \texttt{h}(\texttt{trydec}(k, x, y))]$ further narrows down the possibilities. After calculating (based on Definitions 6 and 5) the appropriate accessibility relations for P and V, we obtain the final model $\mathcal{F}^P_{\text{BKP}}$, as depicted in Figure 4.

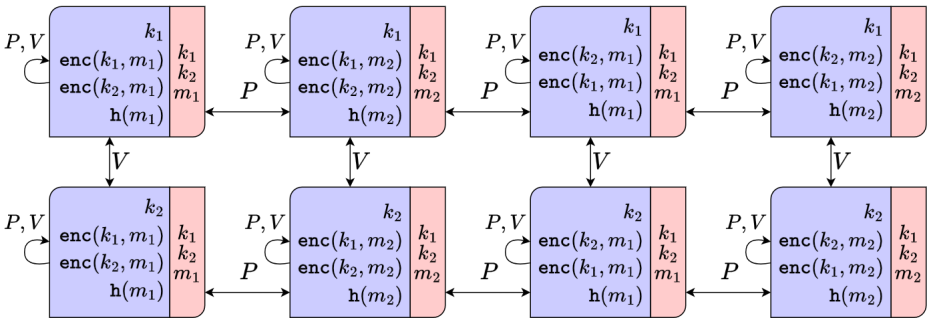


Figure 4: The model $\mathcal{F}^P_{\text{BKP}}$ generated from S_P.

Notice that V's epistemic uncertainty is narrowed by eliminating the worlds where P's key variable is unassigned but unaltered for the remaining worlds: there are bi-directed V-arrows connecting the worlds where k has value k_1 with worlds where k has value k_2. The bi-directed P-arrows connect worlds in which P does not need to differentiate between scenarios where V has swapped the order of sent ciphertexts. Likewise, these arrows eliminate distinctions between worlds where cryptographic functions have been applied to one message (m_1) versus

[11] Those epistemic possibilities are summarized in the three-part table given in Appendix C and omitted here for brevity.

another (m_2). By an easy inspection of $\mathscr{F}^P_{\text{BKP}}$, we see that each component of P's goal (see Example 2) is valid. In symbols, $\mathscr{F}^P_{\text{BKP}} \vDash \varphi_{\text{ZK}}$, $\mathscr{F}^P_{\text{BKP}} \vDash \varphi_{\text{PoK}}$, and $\mathscr{F}^P_{\text{BKP}} \vDash \varphi_{\text{NR}}$.

Performing S_V. We now compute the update of $\mathscr{I}_{\text{BKP}}$ with $\langle\!\langle S_V \rangle\!\rangle_i$, where S_V is the SPEC-statement of Example 1 for the honest verifier in BKP. The first statement applied to $\mathscr{I}_{\text{BKP}}$ consists of receiving $*$ from P; as before, we skip this event, which only adds $*$ to V's local information. After generating a new message, V assigns its value to a variable named m. Given that value assignment is a private action, the resulting model is essentially a replica of $\mathscr{I}_{\text{BKP}}$. The nodes are P-accessible from their counterparts in the model derived from $\mathscr{I}_{\text{BKP}}$ by removing P-loops and incorporating the value of m into V's local information. After that, V sequentially sends three messages to P: $\text{enc}(k_1, m)$, $\text{enc}(k_2, m)$, and $\text{h}(m)$. These actions consist of three public announcements. Since each of those values is constructible in any of its worlds, the structure of the previous model is preserved, though the values $\text{enc}(k_1, m)$, $\text{enc}(k_2, m)$ and $\text{h}(m)$ join P's local information for any world.

At this point, V receives a message from P, whose value is assigned to x. Since V does not know the actual value she will receive, she must consider the epistemic possibility where P sends m and the one where P sends another value m_2. This situation unfolds the previous model, whose worlds differ for the local information available to V; still, P's epistemic possibilities are preserved.

Finally, V checks whether the value stored in x is equal to the value of m: if that is the case, BKP terminates, leading to the final model $\mathscr{F}^V_{\text{BKF}}$ depicted in Figure 5.

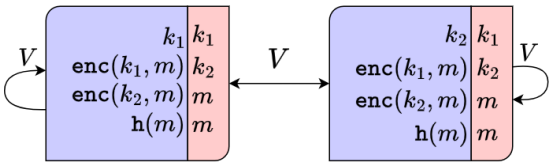


Figure 5: The model $\mathscr{F}^V_{\text{BKP}}$ generated from S_V.

Again, one can check that $\mathscr{F}^V_{\text{BKF}} \vDash \varphi_{\text{PoK}}$, i.e., the honest verifier achieves a PoK.

6. Concluding remarks

We outlined the application of dynamic epistemic logic (DEL) to verify zero-knowledge proofs (ZKP) formally. We consider it a first step towards a comprehensive verification framework of ZKP. The approach's potential is demonstrated through a practical example, illustrating how DEL semantics can capture the perspectives of protocol participants on the protocol's evolution.

Formal verification of security protocols is a vast research field, where several methodologies co-exist. Model checking [12] is often adopted for spotting vulnerabilities (expressed in some temporal logic such as LTL [13]) by visiting a finite state model representing all the possible runs of a protocol [14].[12] In those contexts, adversarial networks are represented via the Dolev-Yao

[12]Among several existing model checkers for protocol verification we mention ProVerif [15] and SATMC [16].

(DY) attacker model [17]

A DY-based implementation of zero-knowledge in the Tamarin prover is introduced in [18], where the author is able to formally prove *some* security properties of the Direct Anonymous Attestation (DAA) protocol. She also discusses some difficulties (and their possible solutions) in automating rigorous formal verification of ZKP in Tamarin [19].

The paper [20] extends the DY model to compare information leakage between protocol implementations. Our method differs from that approach since we are focused on reasoning about agents' knowledge and security goals.

In [21], the authors present a symbolic semantics for a language including a ZKP operator aimed at sharing a proof tree without revealing the prover's identity. Their language is well suited for scenarios where anonymous proofs are published among a set of participants but does not include message-sending/receiving primitives. Therefore, they cannot model protocols such as our BKP.

The seminal papers [22, 6] approached first ZKP via epistemic logic. They identify some criticalities that emerge when modeling cryptographic primitives and suggest overcoming them by combining epistemic and temporal operators.

Differently, [23, 24, 25] assessed the security of some cryptographic protocols using dynamic epistemic logic. We borrowed some notations from those papers, but none of the previous proposals consider ZKP, focusing only on security aspects of cryptographic operations.

References

[1] X. Sun, F. R. Yu, P. Zhang, Z. Sun, W. Xie, X. Peng, A survey on zero-knowledge proof in blockchain, IEEE Network 35 (2021) 198–205. doi:10.1109/MNET.011.2000473.

[2] S. Goldwasser, S. Micali, C. Rackoff, The knowledge complexity of interactive proof systems, SIAM J. Comput. 18 (1989) 186–208. URL: https://doi.org/10.1137/0218012. doi:10.1137/0218012.

[3] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity for all languages in NP have zero-knowledge proof systems, J. ACM 38 (1991) 691–729. URL: https://doi.org/10.1145/116825.116852. doi:10.1145/116825.116852.

[4] O. Goldreich, The Foundations of Cryptography - Volume 1: Basic Techniques, Cambridge University Press, 2001. URL: http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol1.html. doi:10.1017/CBO9780511546891.

[5] O. Goldreich, The Foundations of Cryptography - Volume 2: Basic Applications, Cambridge University Press, 2004. URL: http://www.wisdom.weizmann.ac.il/%7Eoded/foc-vol2.html. doi:10.1017/CBO9780511721656.

[6] J. Y. Halpern, R. Pass, V. Raman, An epistemic characterization of zero knowledge, in: A. Heifetz (Ed.), Proceedings of the 12th Conference on Theoretical Aspects of Rationality and Knowledge (TARK-2009), Stanford, CA, USA, July 6-8, 2009, 2009, pp. 156–165. URL: https://doi.org/10.1145/1562814.1562837. doi:10.1145/1562814.1562837.

[7] A. Baltag, B. Renne, Dynamic Epistemic Logic, in: E. N. Zalta (Ed.), The Stanford Encyclopedia of Philosophy, Winter 2016 ed., Metaphysics Research Lab, Stanford University, 2016.

[8] D. Bogdanov, M. Niitsoo, T. Toft, J. Willemson, High-performance secure multi-party computation for data mining applications, International Journal of Information Security 11 (2012) 403–418.
[9] O. Almousa, S. Mödersheim, L. Viganò, Alice and bob: Reconciling formal models and implementation, Programming Languages with Applications to Biology and Security: Essays Dedicated to Pierpaolo Degano on the Occasion of His 65th Birthday (2015) 66–85.
[10] P. Blackburn, M. de Rijke, Y. Venema, Modal Logic, volume 53, Cambridge University Press, 2002.
[11] H. Van Ditmarsch, W. van Der Hoek, B. Kooi, Dynamic epistemic logic, volume 337, Springer Science & Business Media, 2007.
[12] E. M. Clarke, E. A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, in: D. Kozen (Ed.), Logics of Programs, Springer Berlin Heidelberg, Berlin, Heidelberg, 1982, pp. 52–71.
[13] A. Pnueli, The temporal logic of programs, in: 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), 1977, pp. 46–57. doi:10.1109/SFCS.1977.32.
[14] D. Basin, C. Cremers, C. Meadows, Model Checking Security Protocols, Springer International Publishing, Cham, 2018, pp. 727–762. URL: https://doi.org/10.1007/978-3-319-10575-8_22. doi:10.1007/978-3-319-10575-8_22.
[15] M. Abadi, B. Blanchet, Analyzing Security Protocols with Secrecy Types and Logic Programs, Journal of the ACM 52 (2005) 102–146.
[16] A. Armando, L. Compagna, SATMC: A SAT-Based Model Checker for Security Protocols, in: J. J. Alferes, J. Leite (Eds.), Logics in Artificial Intelligence, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 730–733.
[17] D. Dolev, A. C. Yao, On the security of public key protocols, IEEE Trans. Inf. Theory 29 (1983) 198–207. URL: https://doi.org/10.1109/TIT.1983.1056650. doi:10.1109/TIT.1983.1056650.
[18] S. Fischlin, Formalising Zero-Knowledge Proofs in the Symbolic Model, Master's thesis, ETH Zurich, 2021.
[19] S. Meier, B. Schmidt, C. Cremers, D. Basin, The TAMARIN prover for the symbolic analysis of security protocols, in: Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings 25, Springer, 2013, pp. 696–701.
[20] A. W. Baskar, R. Ramanujam, S. P. Suresh, A Dolev-Yao Model for Zero Knowledge, in: Asian Computing Science Conference, 2009. URL: https://www.cmi.ac.in/~spsuresh/pdfs/zero-know-jun09.pdf.
[21] M. Backes, F. Bendun, M. Maffei, E. Mohammadi, K. Pecina, Symbolic malleable zero-knowledge proofs, in: C. Fournet, M. W. Hicks, L. Viganò (Eds.), IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015, IEEE Computer Society, 2015, pp. 412–426. URL: https://doi.org/10.1109/CSF.2015.35. doi:10.1109/CSF.2015.35.
[22] J. Y. Halpern, Y. Moses, M. R. Tuttle, A knowledge-based analysis of zero knowledge (preliminary report), in: J. Simon (Ed.), Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, ACM, 1988, pp. 132–147. URL: https://doi.org/10.1145/62212.62224. doi:10.1145/62212.62224.

[23] X. Chen, H. Deng, Efficient verification of cryptographic protocols with dynamic epistemic logic, Applied Sciences 10 (2020) 6577.
[24] M. Gattinger, J. van Eijck, Towards model checking cryptographic protocols with dynamic epistemic logic, in: Proc. LAMAS, Citeseer, 2015, pp. 1–14.
[25] F. Dechesne, Y. Wang, Dynamic epistemic verification of security protocols: framework and case study, in: A Meeting of the Minds: Proceedings of the Workshop on Logic, Rationality, and Interaction, Beijing, Citeseer, 2007.
[26] D. E. Knuth, Backus normal form vs. backus naur form, Commun. ACM 7 (1964) 735–736. URL: https://doi.org/10.1145/355588.365140. doi:10.1145/355588.365140.

A. Notational and graphical conventions

In this appendix, we recap the principal conventions adopted in the paper to facilitate understanding and maintain consistency in our presentation.[13]

Mathematical notations. From the logical viewpoint, we will use different formal languages that interact at various levels. To distinguish these abstraction layers, we adopt some general conventions: operators of the epistemic language we use in our abstract semantics are denoted by standard symbols in logical practice;[14] operators of the higher-order/set-theoretic language we use to *define* the relational semantics of the epistemic language are denoted by standard set-theoretic symbols extended by logical symbols that are kept distinct from those used in the epistemic language;[15] operators of the meta-language, i.e. used to *reason* about the epistemic system (language or semantics), are preferably written in plain English.[16]

Whenever we formally define the basic syntax of a language, we recur to (an easier-to-read version of) the Backus-Naur form convention [26].

Moreover, variants of the identity symbol = are omnipresent in the following pages: plain = between expressions of our protocol language denotes identity between values within the type of both the expressions; we reserve the symbol $:=$ for value assignment to a variable of the protocol language; $\triangleq$ denotes definitional equality.

Functions are preferably defined by λ-abstraction: e.g., $\lambda x.x+1$ denotes the successor function. Nevertheless, we occasionally recur to the standard notation to enhance readability. Whenever needed, we also recur to standard conventions to denote the domain and target of a given function: e.g., $f : Atm \longrightarrow \mathscr{L}$ expresses the fact that the function f maps elements of Atm into elements of $\mathscr{L}$.[17] Function definition by cases is expressed by the standard notation distinguishing values for each case considered.

[13] Further conventions, abbreviations, and notation overload are promptly and appropriately signaled in the main body of the paper as soon as they are introduced.
[14] E.g., conjunction is denoted by the symbol '$\wedge$'.
[15] E.g., the expression $\{\alpha \in \mathscr{A} \times \mathscr{B} : C_1(\alpha)\ \&\ C_2(\alpha)\}$ denotes the set of pairs α of elements in $\mathscr{A}$ and $\mathscr{B}$ satisfying a given condition C_1 *and* a given condition C_2.
[16] However, we may write, e.g., $l \in Ag$ to express that the item with label l denotes an agent of our protocol.
[17] Notice the difference between the long arrow symbol $\longrightarrow$ used for functions, from the arrow symbol $\rightarrow$ we use for sending statements in our protocol language.

Graphical conventions. We use squares to denote actions in action models, reserving more rounded nodes for the worlds of static epistemic models. Whenever $\alpha \approx_l \beta$ in the action model, we draw an arrow with label l from the square labeled by α to the square labeled by β; whenever $\alpha \approx_l \beta$ and $\beta \approx_l \alpha$ we draw a bi-directed arrow between α and β; an arrow labeled by Ag denotes a collection of labeled arrow, namely one for each agent in our given set; when we remove the i-labelled arrows from that collection, we obtain the collection represented by the $(Ag - i)$-labeled arrow. Similar conventions are applied to arrows between worlds rendered in Section 5.

B. Choreographies

A *protocol choreography* in SPEC is defined as $\Big\|_M \langle \sigma_i, S_i \rangle$ where $i \in \{1, \dots, n\}$. M is a function mapping each agent label appearing in each protocol agent to another agent in the choreography. In symbols, $M(i, A) = j$ denotes that agent i will send to (and receive from) agent j when using label A.

For the sake of presentation, we use $\langle \sigma_1, S_1 \rangle_A \| _B \langle \sigma_2, S_2 \rangle$ for $\langle \sigma_1, S_1 \rangle \|_M \langle \sigma_2, S_2 \rangle$ where (i) A and B are the only labels appearing in S_2 and S_1 (respectively), and (ii) $M(1, B) = 2$ and $M(2, A) = 1$.

We can now introduce the operational semantics of protocol choreographies, given in Figure 6. Briefly, choreographies allow internal reduction of their protocol agents (Step) and synchronous

$$\frac{\langle \sigma_i, S_i \rangle \longrightarrow \langle \sigma_i', S_i' \rangle}{\langle \sigma_1, S_1 \rangle \|_M \dots \|_M \langle \sigma_i, S_i \rangle \|_M \dots \|_M \langle \sigma_n, S_n \rangle \rightsquigarrow \langle \sigma_1, S_1 \rangle \|_M \dots \|_M \langle \sigma_i', S_i' \rangle \|_M \dots \|_M \langle \sigma_n, S_n \rangle} \text{(Step)}$$

$$\frac{M(i, A) = j \qquad M(j, B) = i}{\dots \|_M \langle \sigma_i, S_i \rangle \uparrow_{A,v} \|_M \dots \|_M \langle \sigma_j, S_j \rangle \downarrow_{B,x} \|_M \dots \rightsquigarrow \dots \|_M \langle \sigma_i, S_i \rangle \|_M \dots \|_M \langle \sigma_j[v/x], S_j \rangle \|_M \dots} \text{(Sync)}$$

Figure 6: Operational semantics of choreographies

communications between agents (Sync) when the agent labels mapping permits it.

We say that a choreography $\Big\|_M \langle \sigma_i, S_i \rangle$ is *successful* when $\forall i. S_i = \cdot$ and we call *stuck* a choreography that (i) is not successful and (ii) does not allow further reductions (denoted by $\not\rightsquigarrow$). Furthermore, we use $\rightsquigarrow^\star$ for the transitive closure of $\rightsquigarrow$ and, given two choreographies C and C', we say that $C \rightsquigarrow^\star C'$ is a *run* of C if $C' \not\rightsquigarrow$.

Then, a *successful run* of C is a run $C \rightsquigarrow^\star C'$ such that C' is successful. For instance, by defining the choreography $C \triangleq \langle \varepsilon[\mathbf{k_1}/k], S_P \rangle_P \| _V \langle \varepsilon[\mathbf{k_1}/k_1][\mathbf{k_2}/k_2], S_V \rangle$, where $\mathbf{k_1}$ and $\mathbf{k_2}$ denote actual cryptographic keys, we have

$$C \rightsquigarrow^\star \langle \varepsilon[k_x^{-1}/k][\mathrm{enc}(k_x, m)/x][\mathrm{enc}(k_y, m)/y][\mathrm{h}(m)/z][m/m'], \cdot \rangle_P \| _V \langle \varepsilon[k_x/k_1][k_y/k_2][m/x], \cdot \rangle.$$

C. Detailed example

Here, we provide full details on the intermediate model omitted in Section 5 when modeling S_P. We render the model through the three-part table in Figure 7 (denoting that the variable k is either k_1, k_2, or unassigned $\emptyset$) where:

- rows represent the possible values of x (namely: $\mathrm{enc}(k_1, m_1)$, $\mathrm{enc}(k_1, m_2)$, $\mathrm{enc}(k_2, m_1)$, $\mathrm{enc}(k_2, m_2)$);
- columns represent the possible values of y ((namely: $\mathrm{enc}(k_1, m_1)$, $\mathrm{enc}(k_1, m_2)$, $\mathrm{enc}(k_2, m_1)$, $\mathrm{enc}(k_2, m_2)$));
- each cell is diagonally split to represent the two possible values of z (namely: $\mathrm{h}(m_1)$, $\mathrm{h}(m_2)$).

Figure 7: Modelling the epistemic evolution of $\mathcal{I}_{\mathrm{BKP}}$ when P sequentially receives from V messages in x, y, and z.

Then, the first guard $[\mathtt{comp}(x, y)]$ allows us to select from the 96 nodes of the epistemic model those containing the local information of P triggering the guard, represented by the blue cells in the table.[18] The subsequent guard $[z = \mathrm{h}(\mathtt{trydec}(k, x, y))]$ further narrows down the possibilities, enabling us to identify, among the blue cells, the sub-cells highlighted with red text that satisfy the guard and allow P to send $\mathtt{trydec}(k, x, y)$ to V.

[18]Recall from Section 4.1 that $\mathtt{comp}(\cdot)$ is true when keys are different and the message is the same.