Reconciling operational and epistemic approaches to the formal analysis of crypto-based security protocols

Citation for published version (APA):

Document status and date:
Published: 01/01/2011

Document Version:
Publisher’s PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:
• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the “Taverne” license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.
Reconciling Operational and Epistemic Approaches to the Formal Analysis of Crypto-Based Security Protocols

HamidReza Mahrooghi and MohammadReza Mousavi

1 Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, Email: H.Mahrooghi@tue.nl
2 Department of Computer Science, Eindhoven University of Technology Eindhoven, The Netherlands, Email: M.R.Mousavi@tue.nl.

Abstract. We propose a unifying framework for formal specification and verification of both epistemic and behavioral aspects of security protocols. The main novelty of the proposed framework is the explicit support for cryptographic constructs, which is among the most essential ingredients of security protocols. Due to this feature, the indistinguishability relation for the epistemic constructs gets a dynamic semantics by taking the communicated keys and cryptographic terms in the operational specification into account.

1 Introduction

On one hand, operational specification, e.g., in the Alice-Bob notation [25, 5], provides an intuitive representation of security protocols. Specifying security properties in operational terms, e.g., by using notions of behavioral equivalence or by temporal logic formulae is, however, usually tedious and has led to various definitions for basic properties such as authentication [24] and anonymity [32, 7].

On the other hand, epistemic specification [14, 21, 11] provides inherent support for of basic security properties: most security properties (e.g., secrecy and anonymity) are about impossibility of gaining knowledge about certain facts, or correspondence between a notion of belief and what is actually the case (e.g., authentication). However, specifying the behavior of protocols in the (dynamic) epistemic style is usually cumbersome and there is still ongoing research in finding a generic and compositional approach to protocol specification in the epistemic style (see, e.g., [19, 3, 22, 31, 34]).

Hence, an approach combining the best of the two worlds is highly desirable, i.e., an approach allowing for protocol specification in the operational style and property specification and verification in the epistemic style. This has already been noted by several authors and there already exists a rich literature on such combinational frameworks [9, 6, 23, 33] (cf. related work section below). However, one major shortcoming of most of the existing frameworks is lack of support for cryptographic constructs which are among the essential building blocks for security protocols.

In this paper, we propose a combinational framework with a genuine support for cryptographic constructs. In our framework, we introduce a process-algebraic syntax (with parameterized actions) for expressing the operational specification of the security protocols and an epistemic extension of the modal $\mu$-calculus with past [17] for property specification. Our approach exploits the notion of indistinguishability in [2] and extend it to epistemic indistinguishability to extract the knowledge of the involved (or observing) principals. Moreover, we show that our approach is modular in that any of the components (cryptographic, process algebraic or epistemic constructs) can be extended or replaced as long as their semantics is expressed in the same semantic domain. We illustrate this modularity by starting with the simplest of cryptographic constructs, namely nonces and symmetric encryption (decryption) and after constructing the framework, show how the cryptographic constructs can be extended to asymmetric keys without requiring any change to the rest of the framework.
Related Work Using (dynamic) epistemic logic for the specification and verification of security protocols has been the subject of numerous papers and theses and a comprehensive survey of all these pieces of work goes beyond the scope of this paper. We refer to two recent doctoral theses [20, 34], which provide an extensive survey of the state of the art. Operational techniques and behavioral verification have been the primary focus of protocols designers and hence providing a survey of operational and behavioral approaches is virtually impossible. Hence, in the remainder of this section, we only focus on the most recent approaches that combine the epistemic and behavioral approaches to protocol specification and verification.

In [33] an extension of a dynamic spatial logic with knowledge-related constructs is presented. There are a number of essential differences between the approach of [33] and that of the present paper. Firstly, the notion of knowledge captured in [33], despite being called epistemic, captures an explicit notion of knowledge obtained through the content of the received messages; our notion of knowledge not only addresses the explicit information obtained through receiving the content of messages but also by observing the information exchange among other parties and deriving more facts by comparing the observed traces with possible executions of the protocol (using the prior knowledge of the protocol). Secondly, the notion of principal in [33] refers to separate processes composed using the particular notion of parallel composition. However, in our approach, principals need not be represented by separate processes and even may never have any behavioral specification (i.e., be passive observers). Finally, the notion of symbolic crypto-term used in [33] is coarser than the notion used in [2] and adopted in this paper. To be more specific, in [33] two encrypted terms for which the corresponding keys are distinguishable while in our framework, they are deemed indistinguishable. This is essential for modeling an epistemic notion of implicit knowledge.

In [6] an epistemic logic for the applied π-calculus [1] is presented. We improve upon the approach of [6] in three essential ways. The notion of knowledge captured in [6] (which is indeed epistemic unlike [33]) is geared towards the knowledge of a single principal, i.e., intruder, while we allow for different views of the events by different principals, including the honest principals involved in the protocol. Secondly, we introduce a more expressive logic with fixed points which allows for specifying concepts such as common knowledge in our framework. Thirdly, the approach of [6] uses an explicit construct in the syntax of the operational (process calculus) domain to record events, while we use the semantics of the operational domain to record events. Hence, the operational specifications can be readily used in our approach to reason about their epistemic properties.

In [9] an extension of modal μ-calculus with past is presented with epistemic constructs. Our work is a direct extension of the framework of [9] with cryptographic constructs. In [23], the framework of [9] is extended with probabilities. We believe this is an orthogonal extension which can be unified with the framework of this paper, forming a very rich framework with support for epistemic, probabilistic and cryptographic constructs.

We use the basic notions pattern definition and indistinguishability of expressions from [2] in our framework to define the notion of knowledge for principals. Therefore, we extend the original definitions of [2] to the pattern and indistinguishability of term histories. Our semantic model for the operational domain is based on the notion of computation introduced in [28], which provides a non-deterministic view of the future behavior as well as a deterministic view of the past history.

Structure of Paper In Section 2, we introduce basic cryptographic terms using symmetric encryption. In Section 3, the syntax of our process algebraic language for operational specification. Subsequently the semantics of operational specifications is defined in Section 4. In Section 5, we define the syntax and semantics of our property specification language, which is an epistemic extension of the modal μ-calculus with past. Our basic setting for cryptographic terms is extended in Section 7 with various constructs such as asymmetric keys, blinding and signing show the modularity of our approach by plugging the new term structure into the earlier setting and specifying and reasoning about a case study. We have implemented a prototype to mechanize reasoning about our specifications, about which we report in Section 8. Section 9 concludes the paper and presents some directions for further research.
2 Symbolic Terms and Indistinguishability

In this section, we lay the foundations for introducing cryptographic terms into our framework, following the approach of [2]. We start with defining a simple signature for basic constructs such as plain text, nonces, symmetric keys and encryption. Subsequently, we define the notions of deducibility and indistinguishability, which define an algebraic structure of crypto-terms, equating those terms that are not distinguishable with a certain (local) knowledge of keys. These notions will serve as the basis for defining the syntax and semantics of our process language (for operational specification) and for building the epistemic notion of indistinguishability in the remainder of this paper.

Definition 1 (Symbolic Terms). We assume a syntactic class Key of symmetric keys, typically denoted by \( k, k_i, \ldots \), a syntactic class Id of principal identities, typically represented by \( A, B, 1, 2, \ldots \), a class Var of variables, typically denoted by \( x, x_A, x_k, \ldots \), a classNonce of nonces, denoted by \( n, n_1, \ldots \), and a classMsg of plain-text messages, denoted by \( m, m_1, \ldots \).

The set of crypto-terms, denoted by \( \text{Term} \), is defined by the following grammar:

\[
M, N ::= k \mid id \mid m \mid x \mid n \mid \{M\}_k \mid (M,N)
\]

where \( k \in \text{Key} \), \( id \in \text{Id} \), \( m \in \text{Msg} \), \( x \in \text{Var} \), and \( n \in \text{Nonce} \). The term \( \{M\}_k \) is the result of encrypting \( M \) with the symmetric key \( k \). The term \( (M,N) \) represents pairing of \( M \) and \( N \).

To avoid cluttering the notation, we assume that pairing associates to right and thus, we denote nested pairs of the form \( (M_0, (M_1, \ldots, (M_{n-1}, M_n))) \) by \( (M_0, M_1, \ldots, M_{n-1}, M_n) \).

Example 1. The Needham-Schroeder symmetric-key protocol [27] is used as the running example to illustrate the concepts and definitions. The protocol includes the following steps where \( A \) and \( B \) are identities of Alice and Bob respectively, \( k_{AS} \) (resp. \( K_{BS} \)) is a symmetric key between \( A \) (resp. \( B \)) and \( S, n_A \) and \( n_B \) are nonces generated, respectively, by \( A \) and \( B \), and finally, \( K_{AB} \) is a symmetric key generated to be the session key between \( A \) and \( B \).

1. \( S \rightarrow A : k_{AS} \)
2. \( S \rightarrow B : k_{BS} \)
3. \( A \rightarrow S : (A, B, n_A) \)
4. \( S \rightarrow A : \{(n_A, k_{AB}, B, \{(k_{AB}, A)\}_k_{BS}\})_k_{AS} \)
5. \( A \rightarrow B : \{n_B\}_{k_{AB}} \)
6. \( B \rightarrow A : \{n_B - 1\}_{k_{AB}} \)

It is assumed that Alice and Bob have their separate secure channels to the server in order to receive their keys, i.e., respectively, \( k_{AS} \) and \( k_{BS} \). This is reflected by the notation \( \rightarrow \) in the first two initial steps \( 0_1 \) and \( 0_2 \) of the protocol.

Below, for instance, is a set of terms that is encountered in a run of this protocol. In the remainder of this section, we use this set of terms to illustrate our definitions on crypto-terms.

\[
T_1 = \{(A, B, n_A), \{(n_A, k_{AB}, B, \{(k_{AB}, A)\}_k_{BS})\}_k_{AS}, \{(k_{AB}, A)\}_k_{BS}, \{n_B\}_{k_{AB}}, \{n_B - 1\}_{k_{AB}}\}\.
\]

Term Deduction. Using cryptographic operations, terms can be derived (i.e., (de)constructed) from others. We capture this concept by the deduction rules in Figure 1. A judgement of the form \( T \vdash M \) reads \textit{term } \( M \text{ is derivable from the set } T \text{ of terms}. \) The deduction rules are self-explanatory; we only note that in these rules \( M \) is an arbitrary crypto-term, while \( k \) is a key.

For instance, recall the set of terms \( T_1 \) given in Example 1; take \( T = T_1 \cup \{k_{AS}\} \), then the term \( n_{BS} \) is derivable from \( T \) by the deductions depicted in Figure 2, using the deduction rules given in Figure 1.
The following *Pattern* function (due to [2]) is defined to reduce a given term $M$ to a pattern using a set of keys deduced from the term itself. This function provides an abstract view of encryption, capturing the fact encrypted terms are indistinguishable if their respective keys are not available (denoted by $\square$). Due to their random nature, nonces are also deemed indistinguishable from each other (and from encrypted messages with unavailable keys).

**Definition 2 (Patterns).** A pattern is a crypto-term with (possibly multiple) occurrences a specific constant $\square$ capturing nonces and those encrypted messages that cannot be decrypted. Hence, the set $P\text{Term}$ of patterns, is defined with the following grammar.

$$M_P, N_P ::= k \mid id \mid m \mid \{M_P\}_k \mid (M_P, N_P) \mid \square$$

The function $\text{Pattern} : \text{Term} \to P\text{Term}$ is defined below by using the auxiliary function $\text{pat} : \text{Term} \times \text{Key} \to P\text{Term}$ which defines the pattern of a message given a set of keys.

$$\text{Pattern}(M) = \text{pat}(M, \{k \in \text{Key} \mid M \vdash k\}).$$

Intuitively, pattern of term maps the observed term to what can be learned and understood from the viewpoint of a principal.

$$\text{pat}(id, K) = id \ (id \in I\text{d}),$$

$$\text{pat}(k, K) = k \ (k \in \text{Key}),$$

$$\text{pat}(m, K) = m \ (m \in M\text{sg}),$$

$$\text{pat}(n, K) = \square \ (n \in N\text{once}),$$
\[
\text{pat}((M, N), K) = (\text{pat}(M, K), \text{pat}(N, K)),
\]
\[
\text{pat}({M}_k, K) = \begin{cases} 
\{\text{pat}(M, K)\}_k & \text{if } k \in K, \\
\Box & \text{otherwise.}
\end{cases}
\]

The notion of pattern is generalized to a set of terms by the following definition of the \textit{SPattern} function.

\[
\text{SPattern}(T) = \bigcup_{M \in T} \text{pat}(M, K), \text{ where } K = \{ k \mid T \vdash k \text{ and } k \in \text{Key} \}.
\]

For example, the patterns of some of the terms in Example 1 are given below.

\[
\text{Pattern}((A, B, n_{A})) = (A, B, \Box),
\]
\[
\text{Pattern}((\{n_{A}, k_{AB}\}, B, \{(k_{AB}, A)\}_{k_{BS}})) = \Box,
\]
\[
\text{SPattern}(\{(A, B, n_{A}), (\{n_{A}, k_{AB}, B, \{(k_{AB}, A)\}_{k_{BS}}\})\}) = \{(A, B, \Box), \Box\}.
\]

Using the ingredients gathered above, we can now define the indistinguishability relation between terms.

**Definition 3 (Indistinguishability of Terms).** Two terms \(M\) are indistinguishable if and only if they yield the same pattern, i.e.,

\[
M \equiv N \text{ if and only if } \text{Pattern}(M) = \text{Pattern}(N).
\]

For instance, recall the set \(T_1\) in Example 1 and let \(T_2\) be another set of terms given below, which can also be encountered in a run of the same protocol

\[
T_2 = \{(A, B, n'_{A}), (\{n'_{A}, k_{AB}, B, \{(k_{AB}, A)\}_{k_{BS}}\})_{k_{AS}} , \{(k_{AB}, A)\}_{k_{BS}}, \{n'_{B}\}_{k_{AB}}, \{n'_{B} - 1\}_{k_{AB}}\}.
\]

By applying Definition 3, we have

\[
\text{SPattern}(T_1) = \text{SPattern}(T_2) = \{(a, b, \Box), \Box, \Box, \Box\}.
\]

As another example, consider the term \(\{(n_{A}, k_{AB}, B, \{(k_{AB}, A)\}_{k_{BS}})\}_{k_{AS}}\) \(\in T_1\); it is indistinguishable from \(\{(A, B)\}_{k_{AB}}\) since both have the pattern \(\Box\). However, the patterns of the terms \(\{(A)_{k_{A}}, k_{A}\}\) and \(\{(B)_{k_{A}}, k_{A}\}\) are identical to their respective original terms and hence, are \textit{not} indistinguishable.

The definition of \textit{Pattern}(\(M\)) is a parameter to our theory. For practical purposes, one might want to use a weaker (coarser) indistinguishability relation which abstracts from random components of terms as suggested briefly in [2]. The rest of the theory remains intact when such a coarser relation is adopted. For example, \(\{(A)_{k_{A}, k_{A}}\}\) \(\{(A)_{k'_{A}, k'_{A}}\}\) can be considered equivalent through a coarser equivalence relation (equivalence up to renaming, which allows for unifying applying a substitution to terms on keys). (This parametericity is further demonstrated by our extension to the term structure and the definition of pattern in Section 7.)

### 3 Crypto – PAi: Syntax

In this section, we present the syntax of our operational specification language \textit{Crypto – PAi}, which is a process algebraic language with value-passing of crypto-terms.

Let \(\text{Act}\) be a finite set of \textit{action names} with fixed arity (for term parameters) which is ranged over by \(a, !a, ?a, b, \ldots\), and let \(\text{Id}\) be a finite set of \textit{identities} typically denoted by \(i, j, \ldots\). We
use the exclamation and the question mark to designate send and receive actions, respectively. Actions that result from a synchronization as well as internal actions are denoted without any annotation. Action \( \tau \in \text{Act} \) denotes the silent action which also represents a message that offers no new information to any observer.

\[
P, Q ::= \begin{align*}
0 & \quad \text{inaction} \\
D; P & \quad \text{action prefixing} \\
P + P & \quad \text{nondeterministic choice} \\
P || P & \quad \text{parallel composition}
\end{align*}
\]

\( D ::= (J)\alpha(\overline{M}) \)

The intuition behind these syntactic constructs is given below. Constant 0 denotes inaction (termination). Action prefixing is denoted by \( D; P \); an action may contain an arbitrary number (zero or more) crypto-terms as parameters (i.e., the arity of an action is not fixed). This comes in handy when modeling actions that may appear to have different information content (vectors of terms of different size) to different principals. \( (J)\alpha(\overline{M}) \in D \) denotes a parametric decorated action, which has the following intuition: action \( \alpha \in \text{Act} \) with parameters \( \overline{M} \) is visible to principal \( i \in J \subseteq \text{Id} \). Other principals \( (j \not\in J) \) observe \( \rho(a(\overline{M})) \) being taken, where \( \rho : \text{Act} \times \text{Term} \rightarrow \text{Act} \times \text{Term} \) is a global renaming function, which assigns a “public appearance” to every parameterized action and should be defined by the specifier of a protocol. We assume that \( \rho(\tau) \) is always defined to be \( \tau \), reflecting the fact that \( \tau \) is invisible to all principals, and if \( \rho(a(\overline{M})) = \rho(a(\overline{M}')) \) then \( \overline{M}' \) is a sub-term of \( \overline{M} \). The combination of identity decoration on actions, action renaming (public appearance, see below), and action parameters provides different views on the the behavior of the protocols, according to different principals. We assume that if \( \rho(a(\overline{M})) = b(\overline{M}') \), then \( \overline{M}' \) is a subterm of \( \overline{M} \) and \( \rho(\tau) \) is always defined to be \( \tau \); these reflect the fact that the public appearance of a message cannot reveal more information what is actually contained in the message. Nondeterministic choice among \( P \) and \( Q \) is denoted by \( P + Q \) and means that the first action taken by either of the two processes determines the choice. \( P || Q \) parallel composition; it allows for interleaving of internal actions and the results of earlier communications as well as hand-shaking synchronization between input and output actions. Hand-shaking synchronization results in a value-passing by replacing the variables of the receiving party with closed terms from the sending party.

To avoid cluttering the syntax, we assume that action prefixing binds stronger than nondeterministic choice and non-deterministic choice binds stronger than parallel composition. Also, we omit the trailing 0 and write, for example, \( D \) for \( D; 0 \).

**Example 2.** Consider the Needham-Schroeder protocol as given in Example 1. Its specification in our syntax is depicted in Figure 3. In this specification, we use the following definitions. In particular, we assume that the server has a finite number of keys (more than one) for its communication with \( A \) and \( B \), denoted by \( k_{AS,i} \) and \( k_{BS,j} \), and generates a shared key \( K_{AB,k} \) between \( A \) and \( B \), with \( i \in I \), \( j \in J \), and \( k \in K \) for some finite (non-empty and non-singleton) sets \( I, J, \) and \( K, \)

\[
\text{Key} = \{ k_{AS,i}, k_{BS,j}, k_{AB,k} \mid i \in I, j \in J, k \in K \},
\]

\[
\text{Id} = \{ A, B, S, E \},
\]

\[
\text{Var} = \{ x_k, x_{k'}, x_A, x_B, x_n \},
\]

\[
\text{Once} = \{ n_{A,m}, n_{B,n} \mid m \in M, n \in N \}.
\]

The behavior of the two clients \( A \) and \( B \), respectively, are specified by the process \( \text{Client}_A \) and \( \text{Client}_B \), and the server \( S \) is specified by the process \( \text{Server}_S \). In the specification of the server, the operator (binder) \( \sum \) is an acronym for a finite number of non-deterministic choices. We consider \( E \) to be the identity of the eavesdropper (passive intruder).\(^3\) Each process executes a series of

\(^3\) An active intruder, e.g., in the Dolev-Yao model, can also be specified in our framework by defining its own process that can intract and interfere with other processes.
Fig. 3. Operational Specification of the NS Protocol.

actions directly corresponding to the steps of the protocol, given in Example 1. We assume that there is a secure and private channel between the server and each of the participants (A and B); this is reflected by the definition of the public appearance function, which is the identity function everywhere, but the for following two actions.

\[ \rho(\text{TakeKey}_{AS}(t)) = \tau \quad \text{and} \quad \rho(\text{TakeKey}_{BS}(t)) = \tau \quad \text{for each crypto-term } t. \]

The public appearance of these two actions is \( \tau \), which means that they are invisible to other principals than the two involved in the hand-shaking synchronization. The fact that the public appearance function is identity elsewhere reflects the fact that other messages are announced publicly and can be eavesdropped, among others, by the eavesdropper \( E \).

4 **Crypto – PAi: Operational Semantics**

The configuration of our operational semantics comprises two components: a process, of which the syntax defined in the previous section, and a computation, which is a sequence of pairs of processes and actions. The first component of the operational semantics determines the possible future behavior of the specified system, while the second component records the history of its past behavior. The operational semantics defines two types of relations among states: a transition relation, defining how a state may evolve by performing actions, and an indistinguishability relation (labeled by principal identities), defining all states that are deemed possible given the current computation observed by each principal. This combination, called an Epistemic Labeled Transition System (ELTS), also serves as the basis for defining the semantics of our logical specification framework.

In order to define our operational semantics, we need a number of auxiliary definitions. Next, we first define the notion of trace histories and then using a number of auxiliary definitions, lift the notion of indistinguishability of terms (Definition 3) to trace histories.

**Definition 4 (Trace History).** A trace history \( \Gamma \in \text{History} \) is a sequence of pairs \((P, D)\), where \( P \) is process in the syntax given in Section 3 and \( D \) is a parameterized decorated action with term
parameter (of the form \((3)\alpha(\vec{M})\), see the same section). By \((P,D) \supseteq \Gamma\), we denote concatenating a pair \((P,D)\) with a trace history \(\Gamma\). The empty trace history is denoted by \(\varepsilon\).

In order to lift the notion of indistinguishability to trace histories, we first lift the notion of pattern to trace histories.

**Definition 5 (Appearance of Actions and Visible Terms).** Given a decorated action \(D\) and an identity \(i\), the appearance of \(D\) to \(i\), denoted by \(\text{Appear}^i(D)\), is defined as follows.

\[
\text{Appear}^i((J)a(\vec{M})) = \begin{cases} 
  a(\vec{M}) & \text{if } i \in J, \\
  b(\vec{M}') & \text{if } i \notin J \text{ and } \rho(a(\vec{M})) = b(\vec{M}')
\end{cases}
\]

In the above definition we assume that \(\vec{M}\) and \(\vec{M}'\) can be empty vectors of terms, denoting actions with no parameter (including \(\tau\)).

Given a history \(\Gamma\) and an identity \(i\), the set of visible terms in \(\Gamma\) according to \(i\) is denoted by \(\text{TermSet}^i(\Gamma)\), and is defined below (\(\varepsilon\) and \(\emptyset\) denote an empty trace and empty set, respectively).

\[
\text{TermSet}^i(\Gamma) = \begin{cases} 
  \emptyset & \text{if } \Gamma = \varepsilon, \\
  \{M\} \cup \text{TermSet}^i(\Gamma') & \text{if } \Gamma = (P,D)^\sim \Gamma' \text{ and } \text{Appear}^i(D) = a(M)
\end{cases}
\]

**Definition 6 (Indistinguishability of Histories).** Let \(\Gamma\) be a trace history, then the pattern of history \(\Gamma\) from the viewpoint of the principal \(i\) is denoted by \(\text{HPattern}^i(\Gamma)\), as defined below.

\[
\text{HPattern}^i(\Gamma) = \begin{cases} 
  \varepsilon & \text{if } \Gamma = \varepsilon, \\
  \text{HPattern}^i(\Gamma') & \text{if } \Gamma = (P,D)^\sim \Gamma' \text{ and } \text{Appear}^i(D) = \tau,
  a(pat(M, K^i))^\sim \text{HPattern}^i(\Gamma') & \text{if } \Gamma = (P,D)^\sim \Gamma' \text{ and } \text{Appear}^i(D) = a(M),
  a()^\sim \text{HPattern}^i(\Gamma') & \text{if } \Gamma = (P,D)^\sim \Gamma' \text{ and } \text{Appear}^i(D) = a() \text{ and } a \neq \tau,
\end{cases}
\]

where \(K^i = \{k | k \in \text{Key} \land \text{TermSet}^i(\Gamma) \vdash k\}\).

Next, we define the indistinguishability relation, denoted by \(\overset{i}{=}\), of histories as follows.

\[
\Gamma \overset{i}{=} \Gamma' \text{ if and only if } \text{HPattern}^i(\Gamma) = \text{HPattern}^i(\Gamma')
\]

We have all the ingredients to define the semantics of our process language \(\text{Crypto} – PAi\). Plotkin-style deduction rules for the structural operational semantics [30] of \(\text{Crypto} – PAi\) are given in Figure 4.

The transition relation \(\rightarrow\) has exactly the same role and meaning as in the standard notion of Labeled Transition system (LTS). The formula \(s^\sigma\) shows the possibility of termination in state \(s\). The indistinguishability between \(s_0\) and \(s_1\) according to principal \(i\) is denoted by expression \(s_0 \overset{i}{=} s_1\) observing of actions depends on the visibility range of actions.

First we define auxiliary relation \(\overset{d}{\leftrightarrow} \subseteq \text{St} \times \text{St}\) for each parametric decorated action \(d \in D\). The deduction rules for \(\overset{d}{\leftrightarrow}\) are mostly self-explanatory and standard to most process algebras. One peculiar deduction rule is \((\text{p3})\) which uses the premise \(\sigma = \text{match}(\vec{M}, \vec{M}')\) in order to match the open term at the receiving process with the (closed) term at the sending side. The
matching (unifying) substitution is then applied to the continuation of the receiving process, thereby replacing its variables with the received values. In the deduction rule \((\text{strip})\), the extra information on the labels (concerning the visibility range) are stripped off, i.e., we block individual send and receive actions and thereby obtain the transition relation \(\rightarrow\). Deduction rule \((\text{I}_C)\) move the indistinguishability relation of histories up to operational states. The symmetric rules \((\text{n1})\), \((\text{n3})\), \((\text{p1})\), and \((\text{p4})\) are omitted for brevity. \((= \text{pat})\) indicates the indistinguishability relation of histories regarding their pattern. \((= \text{refl})\) states that our equivalence relation has the reflexivity property. Termination of a process is orthogonal to its past history, so the different meta-variables are used for the histories in the premises and the conclusion of rules \((\text{s2})\), \((\text{n2})\), and \((\text{p2})\). The transition relation \(\Rightarrow\) and indistinguishability relation \(\cdots\) are the sets of all closed statements provable using the deduction rules (plus their symmetric versions) from Figure 4. These define an Epistemic Labeled Transition System (ELTS) for each process \(p\), as defined below.

**Definition 7 (Semantics of Processes).** Given the sets \(\text{Act}\) and \(\text{Term}\), an ELTS is a 5-tuple \((St, \rightarrow, \checkmark, I_C, s_0)\), where \(St\) is the set of operational states, \(\rightarrow \subseteq St \times \text{Act} \times \text{Term} \times St\) is the transition relation, \(\checkmark \subseteq St\) is the termination predicate, \(I_C \subseteq St \times Id \times St\) is the histories-indistinguishability relation, and \(s_0\) is the initial state.

The semantics of process \(p\) is defined by the ELTS with pairs of processes and trace histories as states, \(\rightarrow\) as transition relation, \(\checkmark\) as termination relation, \(\cdots\) as indistinguishability relation, and \((p, \varepsilon)\) as the initial state, where \(\varepsilon\) denotes the empty sequence of term-history.

The following lemma states that \(\overset{\sim}{\sim}\) is an equivalence relation.

**Lemma 1.** Relation \(\overset{\sim}{\sim}\) is an equivalence relation.

**Proof.** Equality on patterns is an equivalence relation and hence, \(\overset{\sim}{\sim}\) is an equivalence. Equivalence of \(\overset{\sim}{\sim}\) follows trivially from equivalence property of \(\overset{\sim}{\sim}\).

## 5 An epistemic mu-calculus

In this section, we extend the modal \(\mu\)-calculus with past with two epistemic constructs: \(\text{Has}\) and \(K\), representing local and epistemic knowledge. Local knowledge refers to the term a principal has
The syntax of our logic, called $E\Pi$, has the following grammar:

$$\phi ::= \top \mid X \mid \phi \land \phi \mid \neg \phi \mid \langle a(\overrightarrow{M})\rangle \phi \mid \overline{\langle a(\overrightarrow{M})\rangle \phi} \mid K_i \phi \mid Has_i(M) \mid \nu X.\phi(X)$$

(if $X$ occurs only positively in $\phi$),

where $i \in I$ d, $M \in Term$, and $a(M)$ is a parameterized action. $\langle a(\overrightarrow{M})\rangle \phi$ means that “after some $a(\overrightarrow{M})$ transition $\phi$ holds”; $\overline{\langle a(\overrightarrow{M})\rangle \phi}$ has the same meaning as $\langle a(\overrightarrow{M})\rangle \phi$, except that it refers to the past; it intuitively means that there is a state in which $\phi$ holds and from which it is possible to take an $a(\overrightarrow{M})$-step to the current state. $K_i \phi$ indicates that “principal $i$ knows that $\phi$ holds”. $Has_i(M)$ means that the term $M$ can be deduced (tough our term deduction rules) in the current state by the principal $i$. Recursive concepts can be defined using the greatest fixed point operator $\nu X.\phi(X)$, i.e., the current state is in the largest set $X$ of states that satisfy $\phi(X)$.

For convenience, the abbreviations, defined in Figure 5, are used for commonly used logical formulae: common knowledge operator, $C_J \phi$, expresses that agents in $J$ not only know that $\phi$ holds, but also that all agents in $J$ know that $\phi$ holds, and that all agents in $J$ know that all agents in $J$ know that $\phi$ holds, and so on. This property is a very powerful construction and also very interesting, particularly for protocols where trust is an issue and it has posed a real challenge for specification and verification with the standard operational techniques so far. $E\Pi$-forms denotes the set of $E\Pi$-formulae. In the following definition, the satisfaction of a formula $\phi \in E\Pi$-forms in the ELTS $E$ is interpreted.
Theorem 1. The so-called S5 axioms hold in $E\Pi$ [14]:

\[
\begin{align*}
K &: K_i\phi \land K_i(\phi \to \psi) \to K_i\psi \\
T &: K_i\phi \to \phi & \text{(reflexivity)} \\
4 &: K_i\phi \to K_iK_i\phi & \text{(positive introspection)} \\
5 &: \neg K_i\phi \to K_i\neg K_i\phi & \text{(negative introspection)}
\end{align*}
\]

Proof. S5 is a sound and complete axiomatization with respect to Kripke models where the indistinguishability relations are equivalence relations (cf. e.g. [14, Theorem 3.1.5(c)])..

Example 3. In this example, we specify some properties of the Needham-Schroeder protocol specified in Example 2. For instance, the following temporal epistemic formulae states that the key between the server and each client remains secret throughout the protocol for the other client and the eavesdropper.

\[
[\ast] \bigwedge_{i \in I} \{\text{TakeKey}_{AS}(k_{AS,i})\} \land id \in \{B,E\} \models [\ast] \neg K_{id}Has_{id}(k_{AS,i}),
\]

\[
[\ast] \bigwedge_{j \in J} \{\text{TakeKey}_{BS}(k_{BS,j})\} \land id \in \{A,E\} \models [\ast] K_{id}Has_{id'}(k_{BS,j}).
\]

The following property specifies that in all runs of the protocol, after the step 5, the common key between $A$ and $B$ becomes common knowledge among all participants, i.e., not only they (i.e., $A$, $B$, and $S$) all possess the key, each participant knows that the other participant has the key, and so forth.

\[
[\ast] \bigwedge_{n \in N} \bigwedge_{k \in K} \{\text{Stp}_5(\{n_B,n - 1\}k_{AB,k})\} \land id \in \{A,B,S\} \models [\ast] K_j\{Has_j(k_{AB,k})\}.
\]

Finally, the following property specifies that the common key between $A$ and $B$ is never revealed to $E$.

\[
[\ast] \bigwedge_{k \in K} \neg Has_{kAB,k}
\]
Note that although the intruder does not know the key, it does know at some stage, i.e., respectively, after steps 2 and 3, that $A$ and $B$ possess the common key $k_{AB}$, which is due to the intruder’s knowledge of the protocol. To prevent this, the public appearance of all actions may be defined to be identical (while still publicly announcing the communicated term).

6 Conservative extension of existing theories

The framework introduced in this paper is a conservative extension of the traditional process theoretic modeling on the one hand, and epistemic modeling on the other hand. It means that the satisfaction relation defined in Section 5 preserves the standard satisfaction relations of $\mu$-calculus with past) formulae on labeled transition systems and of $E$ (epistemic logic) formulae on Kripke structures. It is straightforward to show (by observing that the semantics of the common sub-languages are identical) that $ELTS + E\overline{\mu}$ model-checking framework extends the $LTS + \overline{\mu}$ model-checking framework since whatever was possible in the latter, is still possible and has the same meaning in the former. Stripping the ELTS from the $IC$ relations, provided by the SOS-rule (strip), can present an LTS hand, and $\overline{\mu}$ can be immediately obtain by ignoring the epistemic aspects of our semantic and our logic. This section aims to present the proof of this claim. Here, we want to show the connections between the existing LTS, KS, $E$, $\overline{\mu}$ and the newly introduced ELTS and $E\overline{\mu}$. To indicate these relations, we consider our framework with its two projection: the temporal part on the one hand and the epistemic part on the other hand. As depicted in Figure 7, our semantics ELTS is a safely extension and combination of labeled transition systems (LTS) and Kripke structures, and also our logic is a conservative combination and extension of $E$ and $\overline{\mu}$. This means:

$$ELTS \models E\overline{\mu} \equiv (LTS + KS) \models (E + \overline{\mu}).$$

In Figure 7, the points in $lts(E)$ are states and the arrows are transitions. In $em(E)$, the points are possible worlds and the lines are indistinguishability relations labeled with identities of agents. In $E$, the points are states and possible worlds simultaneously. Both projections of the ELTS, temporal and epistemic parts, are present. The epistemic valuation in a state is given by the actions executed from the initial state to that state. Learning about facts that were never explicitly given to agents is exactly the power of epistemic logic approaches, that we took over in the combined framework.

The temporal part ($\overline{\mu}$) The temporal part of our framework, which is obtained by leaving out the two knowledge constructs $Has$ and $K$ from the syntax of our logic presented in Section 5.1, is the same as the modal $\mu$-calculus with past ($\overline{\mu}$) [29]. $\overline{\mu}$ is a very general logical language to reason about processes. We have considered the LTS (standard semantic domain for process-theoretic formalisms) and $\overline{\mu}$ model checking framework as our basic model checking framework ($LTS+\overline{\mu}$) which provides purely temporal aspects of our semantics and our logics. Therefore this part of our satisfaction-relations definition preserves the standard satisfaction relations of $\overline{\mu}$ formulae on labeled transition systems. A Labeled Transition System (LTS) is a standard semantic domain for process-theoretic formalisms. Formally, an LTS over a set of labels $L$ is a tuple $\langle St, \rightarrow, \checkmark, s_0 \rangle$, where $St$ is the set of operational states, $\rightarrow \subseteq St \times L \times St$ is the transition relation, $\checkmark \subseteq St$ is the termination predicate and $s_0$ is the initial state. It typically represents the behavior of a reactive system in terms of states and transitions. Then requirements formulated in a temporal logic are matched against this behavior in the process of model checking. Now, let $T = \langle S, \rightarrow, \checkmark, s_0 \rangle$ be a LTS . The satisfaction relation $T, s \models_{\overline{\mu}} \phi$ (a state $s$ in the LTS satisfies a $\overline{\mu}$ formula $\phi$) is defined inductively as follows:

---

4 Most of the logical results obtained in this chapter carry over from the earlier work in combining operational and epistemic frameworks in the setting without involving cryptographic constructs [10, 9]. Hence, most of the results and some of their proofs remain identical to those reported in [10].
Fig. 7. An ELTS $E$ (down) together with its projections: 'the temporal part' $\text{lts}(E)$ (top-left) and 'the epistemic part' $\text{em}(E)$ (top-right).

Our ELTS + E $\mu$ model checking framework, in fact, is an extension of LTS + $\mu$ model checking framework, which is obtained by embedding the $I_C$ relation into the LTS and also extending the $\mu$ calculus with the knowledge operators $K_i$ and $Has_i$. In other words, the LTS + $\mu$ can be immediately obtained by simply stripping the ELTS from the $I_C$ relations (using the SOS rule strip) and the $E\mu$ logic from the knowledge and epistemic operators $Has_i$ and $K_i$. The following theorem formalizes this.

**Theorem 2.** Given an ELTS $E \equiv \langle St, \to, \cdot, I_C, s_0 \rangle$, let its associated LTS be

\[ \text{lts}(E) = (St, \to, \cdot, s_0). \]

Then it holds for each $\mu$ formula $\phi$ that $E \models \phi$ iff $\text{lts}(E) \models_{\mu} \phi$.

**Proof.** Straightforward, since the definitions of $\models$ and $\models_{\mu}$ are same for the $\mu$ (temporal) part of $E\mu$.

This means that for purely temporal aspects of verification, the epistemic aspects of our semantics and our logic can be safely ignored. In other words, the $\mu$ subset of our logic is verified in the stripped down semantics of processes.

**The epistemic part (E)** Epistemic logics are mainly concerned with expressing subtle properties of communication acts, related to the knowledge, beliefs and intentions of communicating parties. In standard epistemic logic (following [18]), epistemic properties are validated in static rich snapshots of communications (epistemic models), that don't express the temporal evolution of the system. The language of epistemic logic with common knowledge is defined by:

\[ \phi ::= p \mid \neg \phi \mid \phi_1 \land \phi_2 \mid K_i \phi \mid C_J \phi \]
Here the $p$ comes from a given set of propositional variables $\mathcal{P}rop$. These propositions represent the atomic facts the agents may know about. The subscript $i$ ranges over a given set of agents $\mathcal{I}$, and $J \subseteq \mathcal{I}$. The standard reading of the epistemic modalities $K_i$ and $C_i$ is the same as ours in the previous section: “$i$ knows that...” and “it is common knowledge among the agents in $J$ that...”, respectively.

An epistemic ($S5$-)model is a Kripke structure $(W, \{R_i | i \in \mathcal{I}\}, V)$, where $W$ is a nonempty set of possible worlds, $R_i$ is an equivalence relation on $W$ for each $i \in \mathcal{I}$, and $V : \mathcal{P}rop \to \mathcal{P}(W)$ is a valuation function assigning to each propositional variable the set of worlds in which it holds. Given an epistemic model $M$ and world $s \in W$, satisfaction ($\models_{\mathcal{E}}$) is defined recursively as follows:

\[
\begin{align*}
M, s \models_{\mathcal{E}} p & \quad \text{iff } s \in V(p) \\
M, s \models_{\mathcal{E}} \neg \phi & \quad \text{iff it is not true that } M, s \models_{\mathcal{E}} \phi \\
M, s \models_{\mathcal{E}} \phi_1 \land \phi_2 & \quad \text{iff } M, s \models_{\mathcal{E}} \phi_1 \text{ and } M, s \models_{\mathcal{E}} \phi_2 \\
M, s \models_{\mathcal{E}} K_i \phi & \quad \text{iff for all } M, s' \in S, \text{ if } sR_is' \text{ then } M, s' \models_{\mathcal{E}} \phi \\
M, s \models_{\mathcal{E}} C_i \phi & \quad \text{iff } \text{for all } M, s' \in S, \text{ if } (\cup_{j \in J} R_j)^* s' \text{ then } M, s' \models_{\mathcal{E}} \phi
\end{align*}
\]

To isolate ‘the epistemic part’ of our framework, we make suitable choices for the set of propositions, and the set of agents. Let $W := S, R_i := \cdot \cdot \cdot \in I_C, \mathcal{I} := \mathcal{I}_d$. Furthermore, in the context of our Crypto $\rightarrow PAi$-processes we associate with every action $a \in Act$ and its parameter $M \in \mathcal{Term}$ a proposition $a(\overline{M})$ (which can be read as “$a(\overline{M})$ has been executed sometime before”) and its transformation into our $E\mathcal{P}$ frameworks is $a(\overline{M})\gamma$. And we let

$$\mathcal{P}rop := \{a(\overline{M}) | a \in Act, M \in \mathcal{Term}\} \cup \{\top\}.$$ 

The resulting logic is $\mathcal{E}$.

We can then say that our modeling and verification framework is also conservative when it comes to purely epistemic aspects. Namely, if we restrict the ELTS associated with a Crypto $\rightarrow PAi$ process to the $I_C$ relations, we obtain an epistemic model where purely epistemic formulas hold exactly when they hold in the original ELTS, according to the $E\mathcal{P}$ satisfaction relation. This means the epistemic part of our logic (defined by $\langle S, \{\cdot \cdot \cdot | i \in \mathcal{I}_d\}, V\rangle$) has the same reading of the standard epistemic logic. The following theorem formally expresses the conservativeness of $E\mathcal{P}$ w.r.t. $\mathcal{E}$.

**Theorem 3.** *Given an ELTS $E \equiv \langle S, \rightarrow, \bigvee, I_C, s_0 \rangle$, let its associated epistemic be $em(E) = \langle S, \{\cdot \cdot \cdot | i \in \mathcal{I}_d\}, V\rangle$, with propositions from $\mathcal{P}rop$, $V(a(\overline{M})) = \{s \in S | E, s \models (a(\overline{M}) \text{ has been occurred before})\}$ and $V(\top) = S$.*

*For any E formula $\phi$, $E \models \phi$ iff $em(E) \models_{\mathcal{E}} \phi$.*

**Proof.** Straightforward, since the definitions of $|$ and $\models_{\mathcal{E}}$ are same for the E (epistemic) part of $E\mathcal{P}$. 

This means that for purely epistemic aspects of verification, the temporal aspects of our semantics and our logic can be safely ignored. In other words, the epistemic subset of our satisfaction relations (defined in Section 5) preserves the E (epistemic logic) formulae on Kripke structure.

Besides the conservation of standard epistemic model checking, our framework also meets another relevant characteristics of epistemic modeling. Namely, we can show that the behavior part of an ELTS preserves the property of perfect recall (‘no forgetting’ in [16]) for the epistemic part, in the sense that knowledge accumulated is not lost. To formalize this, we need to refine the transition relation $\rightarrow$, for a fixed identity $i$, into update transitions and invisible transitions. Therefore, let $i \rightarrow = \bigcup_{a \in Act, M \in \mathcal{T}erm} a(\overline{M}) \rightarrow \cap i \cdot \cdot \cdot$ be the relation that gives the invisible transitions for $i$, and let $i \rightarrow^*$ be the ‘$i$-update’ relation $\{(s, t) | \exists a \in Act, M \in \mathcal{T}erm, s' \in S t.s.t. s \rightarrow a(\overline{M}) s', s' \not\rightarrow s' \text{ and } s' \rightarrow^* t\}$. 

14
Theorem 4 (perfect recall). Let $E$ be a fixed ELTS generated by the SOS rules, as above, and let $i \in \mathbb{I}$ be any identity. For any states $s,t,s',t' \in St$ and any $n \in \mathbb{N}$, if $s \overset{i}{\sim}^n s'$, $t \overset{i}{\sim}^n t'$ and $s \overset{i}{\sim} t$, then $s' \overset{i}{\sim} t'$, (where $\overset{i}{\sim}^n$ denotes the application of $\overset{i}{\sim}$ $n$ times).

We prove the theorem by a sequence of lemmas:

Lemma 2. For any $d \in D$ and $p,q,\Gamma,\Gamma'$, if $(p,\Gamma) \rightarrow_d (q,\Gamma')$ then $\Gamma' = \Gamma \Rightarrow (p,d)$.

Proof. It follows immediately from rule (d), which is the only rule that affects the construction of histories. \hfill \Box

Lemma 3 (one-step). Let $E$ be an ELTS produced by the SOS. If $s,s',t,t'$ are reachable states in $St$ s.t. $s \rightarrow t$, $s' \rightarrow t'$ and $t \cdots t'$, then either $s \cdots s'$ or $s \cdots s'$ or $s \cdots t'$.

Proof. $s \overset{i}{\rightarrow} t$ and $s' \overset{j}{\rightarrow} t'$ have been generated by rule (strip) and $t \cdots t'$ has been generated by rule (IC), so there exist $x_0,\Gamma_0,\Gamma_0',x_1,\Gamma_1,\Gamma_1',J,J'$ s.t. $s = (x_0,\Gamma_0)$, $s' = (y_0,\Gamma_0')$, $t = (x_1,\Gamma_1)$, $t' = (y_1,\Gamma_1')$, $s \overset{(i)}{\Rightarrow} t$, $s' \overset{(j)}{\Rightarrow} t'$, and $\Gamma_1 = \Gamma_1'$. We need to prove that $\Gamma_0 = \Gamma_0'$.

From Lemma 2, $\Gamma_1 = \Gamma_0 \Rightarrow (x_0,(j)a(\overrightarrow{M}))$ and $\Gamma_1' = \Gamma_0' \Rightarrow (y_0,(j')b(\overrightarrow{M}'))$. Further, $\Gamma_1 = \Gamma_1'$ has been generated by the definition of histories indistinguishability relation (Definition 6). So,

$$(\Gamma_1 = \Gamma_0 \Rightarrow (x_0,(j)a(\overrightarrow{M}))) = (\Gamma_1 = \Gamma_0' \Rightarrow (j')b(\overrightarrow{M}'))$$

and

$$H\text{Pattern}^i(\Gamma_1) = H\text{Pattern}^i(\Gamma_1').$$

Now, according to $H\text{Pattern}^i$ definition, let we define $hp$ and $hp'$ as follows:

$$hp = H\text{Pattern}^i(\Gamma_1) = H\text{Pattern}^i(\Gamma_0 \Rightarrow (x_0,(j)a(\overrightarrow{M})))$$

$$= H\text{Pattern}^i(x_0,(j)a(\overrightarrow{M})),$$

$$hp' = H\text{Pattern}^i(\Gamma_1') = H\text{Pattern}^i(\Gamma_0' \Rightarrow (y_0,(j')b(\overrightarrow{M}')))$$

$$= H\text{Pattern}^i(y_0,(j')b(\overrightarrow{M}')).$$

$\sim$ is a string concatenation, so $hp = hp'$ must hold that:

- (i): $H\text{Pattern}^i(\Gamma_0) = H\text{Pattern}^i(\Gamma_0')$ and so, we get $\Gamma_0 = \Gamma_0'$,
- (ii): if $i \notin J$, and $i \notin J'$ then $\Gamma_1 = \Gamma_0$ and $\Gamma_1' = \Gamma_0'$ so, we get $\Gamma_0 = \Gamma_0'$ by applying the transitive property of the equivalency relation $\equiv$ and the result of the item i,
- (iii): if $i \notin J$, and $i \notin J'$ then $\Gamma_1 = \Gamma_0$ and then we get $\Gamma_1 = \Gamma_0'$ similar to the one for ii,
- (iv): if $i \notin J$, and $i \notin J'$ then $\Gamma_1 = \Gamma_0$ and then we get $\Gamma_1' = \Gamma_0'$ similar to the one for ii.

Then we can applying the SOS rule (IC) and obtain either $(x_0,\Gamma_0) \cdots \cdot (y_0,\Gamma_0')$ or $(x_1,\Gamma_1) \cdots \cdot (y_0,\Gamma_0')$ or $(x_0,\Gamma_0) \cdots \cdot (y_1,\Gamma_1')$. In other words, we obtain either $s \cdots s'$ or $s \cdots t'$ or $s \cdots t$.

\textbf{Lemma 4 (update).} Let $E$ be an ELTS produced by the SOS and let $i \in \mathbb{I}$ be an identity. If $s,s',t,t'$ are reachable states in $St$ s.t. $s \overset{i}{\rightarrow} t$, $s' \overset{i}{\rightarrow} t'$ and $t \cdots t'$, then $s \cdots s'$.

Proof. From the definition of $\overset{i}{\rightarrow}$, there exist action labels $a,b \in \mathcal{A}$t and states $u$ and $u'$ s.t. $s \overset{a}{\rightarrow} u$ and $u' \overset{b}{\rightarrow} t$, respectively $s' \overset{b}{\rightarrow} u'$ and $u' \overset{i}{\rightarrow} t'$. With the extra condition that $s' \overset{i}{\rightarrow} u$ and $s' \overset{i}{\rightarrow} u'$.

From the definition of $\overset{i}{\rightarrow}$ and the fact that $\cdots$ is an equivalence relation (Lemma 1), it follows that $u \overset{i}{\rightarrow} t$ and $u' \overset{i}{\rightarrow} t'$. Therefore also $u \overset{i}{\rightarrow} u'$. By applying Lemma 3 to the states $s,s',u,u'$, we obtain that one of the following holds: $s \overset{i}{\rightarrow} u,s' \overset{i}{\rightarrow} u$ or $s \overset{i}{\rightarrow} s'$. The first two options are excluded, because they imply, by $\overset{i}{\rightarrow}$ being an equivalence relation, that $s \overset{i}{\rightarrow} u$ or $s' \overset{i}{\rightarrow} u'$, which are in contradiction with the definition of $\overset{i}{\rightarrow}$. Therefore, $s \overset{i}{\rightarrow} s'$. \hfill \Box
Proof (Theorem 4). By induction on \( n \). If \( n = 0 \), then \( s' \) is identified with \( s \) and \( t' \) with \( t \). So, trivially, \( s \vdash t \) implies \( s' \vdash t' \). In the general case, suppose \( s' \vdash t' \) and let \( s'' \to s' \) be the last step of the update trace \( s \vdash s' \). In the general case, suppose \( s'' \to s' \) be the last step of the update trace \( t \vdash t' \). By Lemma 4, it follows that \( s'' \vdash t'' \) and we can rely on the induction hypothesis to obtain \( s \vdash t' \), which contradicts theorem’s assumption \( s \vdash t \).

\[
\square
\]

7 Asymmetric Cryptography Extension

In this section, we show that our language of cryptographic terms can be extended, e.g., by including concepts such as asymmetric keys, signing and blinding, without requiring any change to the rest of our framework. Let \( \text{Key} \) also includes asymmetric keys of the form \( \text{pk}_{id} \) and \( \text{sk}_{id} \) (to be used for messages asymmetric encryption/decryption). These two terms represent the public and private keys of a principal with identity \( id \) respectively.

The set of crypto-terms, denoted by \( \mathcal{T} \), is defined by the following grammar:

\[
M, N ::= \text{id} \mid m \mid x \mid n \mid \{M\}_k \mid (M, N) \mid \text{pk}_{id} \mid \text{sk}_{id} \mid \{M\}_{\text{pk}_{id}} \mid \{M\}_{\text{sk}_{id}} \mid \{M\}_{r}
\]

The set of crypto-terms, denoted by \( \mathcal{T} \) in Section 2, is extended by asymmetrically encrypted term \( \{M\}_{\text{pk}_{id}}^a \) and signed term \( \{M\}_{\text{sk}_{id}}^s \); \( \{M\}_{r}^b \) denotes the term \( M \) blinded using the random number \( r \) (as blinding factor). Using the blind signature scheme enables a principal to strip the signed term off from the signed blinded term. Another important and relevant detail is that the decryption, unsigning, and unblinding of terms can be operated using the relevant inverted key, respectively. The definition of the function \( \text{pat} \) is then extended with the following.

\[
\begin{align*}
\text{pat}(\text{pk}_{id}, K) &= \text{pk}_{id} \quad (\text{pk}_{id} \in \text{Key}), \\
\text{pat}(\text{sk}_{id}, K) &= \text{sk}_{id} \quad (\text{sk}_{id} \in \text{Key}), \\
\text{pat}(r, K) &= r \quad (r \in \text{Key}), \\
\text{pat}([M]_{\text{pk}_{id}}^a, K) &= \begin{cases} 
\{\text{pat}(M, K)\}_{\text{pk}_{id}}^a & \text{if } \text{sk}_{id} \in K, \\
\square & \text{otherwise}.
\end{cases} \\
\text{pat}([M]_{\text{sk}_{id}}^s, K) &= \begin{cases} 
\{\text{pat}(M, K)\}_{\text{sk}_{id}}^s & \text{if } \text{pk}_{id} \in K, \\
\square & \text{otherwise}.
\end{cases} \\
\text{pat}([M]_{r}^b, K) &= \begin{cases} 
\{\text{pat}(M, K)\}_{r}^b & \text{if } r \in K, \\
\square & \text{otherwise}.
\end{cases}
\end{align*}
\]

To apply this extension, we should just extend our term deduction rules by the following deduction rules which are relevant to the public/private-key encryption system.

The rules (blind), (unblind), and (blindsign), show that a principal who has the key \( k \), can blind or unblind a term, and also obtain \( \{M\}_{sk_i}^s \) from the term \( \{M\}_{pk_i}^b \) blindly signed by the principal \( i \) (i.e., \( \{M\}_{sk_i}^s \) = \( \{M\}_{pk_i}^b \)). The term \( \{M\}_{pk_i}^a \) means that the term \( M \) has been encrypted using the public-key \( \text{pk}_i \) (\text{enc} rule). \( \text{sk}_i \) denotes the relevant private-key of \( \text{pk}_i \) and use to extract the term \( M \) from \( \{M\}_{pk_i}^a \) (\text{dec} rule). The term \( \{M\}_{sk_i}^s \) represents the message \( M \) signed by the private-key \( \text{sk}_i \) (\text{sign} rule). The signed term \( \{M\}_{sk_i}^s \) cannot be unsigned (and indeed checked the sign) using the corresponding public-key \( \text{pk}_i \) (\text{unsig} rule). However, depending on the signature scheme, the \text{unsig} rule may be used to retrieve the message \( M \) from the signature. i.e.,

\[
\frac{(\text{unsig}) \quad T \vdash \{M\}_{sk_i}^s}{T \vdash M}
\]
In the following example, we specify the FOO’92 e-voting protocol [15], which uses the cryptographic constructs introduced earlier in this section. FOO’92 is a well-known e-voting protocol, satisfying some security properties such as eligibility and privacy.

**Example 4.** It is due to compare the new combined framework with using exclusively the operational approach or the epistemic one. FOO’92 protocol has already been analyzed using both operational [12, 26] and epistemic approaches [13]. Anonymous channels are assumed in the protocol for communication between voters and the collector authority. The protocol comprises voter principals who cast their vote, administrator authority, who identifies eligible voters, and the counter authority, who collects and counts votes and finally Publishes the result. For describing the protocol, the following notations are used.

\[V_i: \text{identity of the voter } i,\]
\[A: \text{identity of the administrator authority,}\]
\[C: \text{identity of the counter authority,}\]
\[\xi(v, k): \text{bit-commitment scheme for message } v \text{ using key } k,\]
\[\sigma_i(m): V_i’s \text{ signature scheme,}\]
\[\sigma_A(m): A’s \text{ signature scheme,}\]
\[e = \chi(m, r): \text{blinding message } m \text{ using random number } r, \text{ and}\]
\[v_i: \text{vote of voter } V_i.\]

Public and private keys are distributed during a pre-stage, called registration. Subsequently, the protocol consists these main stages executed by the voter(s), administrator, and counter [15]:

- **Preparation and Administration:** at the first stage, the voter gets the administrator’s signature on her ballot. The voter sends her blinded committed ballot, signed by herself, to the administrator for signing. The administrator signs the authorized ballot and returns to the voter. The blind-signature is used to guarantee the voter privacy; the administrator signs the message in which the vote is hidden.

- **Voting and Collecting:** at this stage, the voter anonymously sends her ballot signed by administrator to the counter. At the end of this stage, the counter publishes the list of the received ballots.

- **Opening and Counting:** finally, the voter has to reveal her random key \(r\) in order for to \(C\) be able to open the votes and publish the result.

Briefly, after the initial key distribution, the protocol follows these steps, where a broadcast is denoted by the target \(*\):

1. \(V_i \rightarrow A: (V_i, s_i, e_i)\)
2. \(A \rightarrow V_i: V_i, d_i\)
3. \(V_i \rightarrow C: (x_i, y_i)\)
4. \(C \rightarrow *: (l, x_i, y_i)\)
5. \(V_i \rightarrow C: (l, k_i)\)
6. \(C \rightarrow *: (l, v_i)\)

**Fig. 8.** Extended Term Deduction Rules.
RegAuthority = \sum_{m \in M} \sum_{n \in N} (Id)!\text{TakeViPubK}((pk_{V_n}, pk_{A_m}));
(V_i)!\text{TakeViPrivK}(sk_{V_i});
(Id)!\text{TakeAdPubK}((pk_{A_m}, pk_{V_i}));
\{A\}!\text{TakeAdPrivK}(sk_{A_m});
(Id)!\text{TakeCoPubK}(pk_{A_m})

Voter = (Id)!\text{TakeViPubK}((x_{pk_V}, x_{pk_A});
(V_i)!\text{TakeViPrivK}(sk_{V_i});
\sum_{j \in \text{Cand}}(Id)!\text{ViReqAdSign}((V_i, \{s_{i_{kv}}\} \times_{sk_{V_i}}, \{s_{i_{kv}}\} \times_{sk_{V_i}}));
(Id)!\text{AdAckSign}((V_i, \{s_{x_{sk_k}}\} \times_{sk_{V_i}}));
(\{\}\)!\text{ViVoting}((x_{sk_{V_i}}, \{s_{i_{kv}}\} \times_{sk_{V_i}}));
(Id)!\text{CoVoteList}((x_{i_{x_{sk_k}}}, \{s_{i_{kv}}\} \times_{sk_{V_i}}));
(\{\}\)!\text{ViRevealKey}((x_{i_{x_{sk_k}}}, \{s_{i_{kv}}\} \times_{sk_{V_i}}));
(Id)!\text{PublishResult}

Administrator = (Id)!\text{TakeAdPubK}((x_{pk_A}, x_{pk_V});
\{A\}!\text{TakeAdPrivK}(sk_{sk_{A}});
(Id)!\text{ViReqAdSign}((x_{V}, x_{sk_{V}}));
(Id)!\text{AdAckSign}((x_{V}, x_{sk_{V}}));
(\{\}\)!\text{ViVoting}((x_{sk_{V}}, \{s_{i_{kv}}\} \times_{sk_{V_i}}));
(Id)!\text{CoVoteList}((l_r, x_{sk_{V}}, x_{c}));
(\{\}\)!\text{ViRevealKey}((l_r, x_{sk_{V}}));
(Id)!\text{PublishResult}

Counter = (Id)!\text{TakeCoPubK}(x_{pk_A});
(\{\}\)!\text{ViVoting}((x_{sk_{V}}, x_{c}));
\sum_{l_r \in L}(Id)!\text{CoVoteList}((l_r, x_{sk_{V}}, x_{c}));
(\{\}\)!\text{ViRevealKey}((l_r, x_{sk_{V}}));
(Id)!\text{PublishResult}

Fig. 9. Our Model of the FOO’92 E-Voting Protocol.

Model In the specification of the protocol given in Figure 9, we assume that \(m \in M, n \in N, i \in I, j \in J, r \in R\) for some finite (non-empty and non-singleton) sets \(M, N, I, J, R, pk_{A_m}\) and \(sk_{A_m}\) represent, respectively, A’s public and private keys, and similarly, \(pk_{V_n}\) and \(sk_{V_n}\) represent \(V_i\)’s public and private keys; \(c_j \in \text{Cand}\) indicates the \(j\)th candidate of candidates list; \(l_r \in L\) indicates the index of result-list made by the authority Counter.

\[
\begin{align*}
\text{Key} &= \{pk_{V_n}, sk_{V_n}, pk_{A_m}, sk_{A_m}, k_v, r_v \mid m \in M, n \in N\}, \\
\text{Id} &= \{A, C, E, V_i\}, \\
\text{Var} &= \{x_{pk_V}, x_{sk_V}, x_{pk_A}, x_{sk_A}, x_g, x_t, x_{sk_{V_v}}, x_{sk_{V_i}}, x_c, x_{sk_{V_i}}\}, \\
\text{Cand} &= \{c_j \mid j \in J\}, \\
L &= \{l_r \mid i \in R\}.
\end{align*}
\]

We first define the process RegAuthority, which initializes the protocol with keys distribution. Then we specify the behavior of the Administrator and Counter, as well as that of Voter. The principal with identity \(E\) plays the role of the eavesdropper (passive intruder) in our model. The set \(Id\) used in the specification denotes the set of all identities, i.e., \(\{A, C, E, V_i\}\).

Regarding the public appearance of actions, we define for all of the identities and \(m \in M, n \in N, v_i \in \text{Cand}, l_r \in L, \)

\[
\begin{align*}
\rho(\text{TakeViPrivK}(sk_{V_i})) = \tau & \quad \rho(\text{TakeAdPrivK}(sk_{A_m})) = \tau \\
\rho(\text{ViVoting}((\{s_{i_{kv}}\} \times_{sk_{V_i}}, \{s_{i_{kv}}\} \times_{sk_{V_i}}))) = \tau & \quad \rho(\text{ViRevealKey}(l_r, k_v)) = \tau
\end{align*}
\]

The specification of the protocol is the parallel composition of the participating processes:

\[
\text{FOO} = \text{RegAuthority} \parallel \text{Administrator} \parallel \text{Counter} \parallel \text{Voter}
\]
Analysis  
Next, we specify some security properties of the FOO protocol; for sake of brevity, we leave out key secrecy properties that have the same structure as in the Needham-Schroeder protocol.

The following formula is relevant to the eligibility of a voter. It states when one try to cast her vote, she must have already been authenticated by the authority $A$.

$$\forall v_i \in C \land \forall m \in M [\text{ViVoting}((\{v_i\}_k V)_{sk_{A,m}} \cdot \{v_i\}_k V)] \land (\text{AdAckSign}(V_i, \{\{v_i\}_k V\}_{sk_{A,m}})) \land (\text{AdAckSign}(V_i, \{\{v_i\}_k V\}_{sk_{A,m}})).$$

The fairness property is another security requirement for the e-voting protocols and refers to impossibility of affecting the remaining voters by revealing the intermediate results. In other words, it means that the votes cannot be opened cannot be revealed before and while they are being collected. Fairness can be expressed by the following formula.

$$\forall l_r \in L \land \forall v_i \in C \land \forall m \in M [\text{CoVoteList}((l_r, \{\{v_i\}_k V\}_{sk_{A,m}} \cdot \{v_i\}_k V))] \land \forall id \in \{A, C, E\} \neg (\langle \ast \rangle \text{Has}_{id}(v_i)).$$

8 Implementation

We have formalized the semantics of $\text{Crypto-PAi}$ and a subset of our $E\Pi$ logic (including the basic temporal and epistemic operators) in the rewriting logic of Maude and mechanized the model-checking process in its toolset [8]. Instead of defining the general fixed-point construct in our logic, we only formalized the acronyms involving fixed-points defined in Figure 5. This implementation is only meant to be a working prototype for our theory and small examples therein (such as the two case studies presented in this paper); efficiency of our model-checking techniques remains as a topic for future research. Next, we present a brief overview of the implementation of the process language and the logic.

8.1 $\text{Crypto-PAi}$ in Maude

The syntax of $\text{Crypto-PAi}$ is modeled in Maude as the following specification where $\text{Act}$, $\text{TAct}$, $\text{TDAct}$ and $\text{Proc}$ are sorts for basic actions, parametric actions, decorated parametric-actions and processes, respectively. The attribute $\text{ctor}$ designates constructors of each sort. $\text{rcv}$, $\text{snd}$ and $\text{sync}$ define decorated parametric-actions for send, receive and synchronization (or just non-communicating actions), denoted in the original syntax by “?” “!” and without any preceding mark, respectively). $\tau$ is the invisible internal action.

\[
\begin{align*}
\text{op } \tau ( \_ ) & : \rightarrow \text{TDAct} \ [\text{ctor}] . \\
\text{op } _ ( \_ ) & : \text{Act PAiTerm} \rightarrow \text{TAct} \ [\text{ctor}] . \\
\text{op } \text{rcv} ( \_ ) ( \_ ) ( \_ ) & : \text{IdSet Act PAiTerm} \rightarrow \text{TDAct} \ [\text{ctor}] . \\
\text{op } \text{snd} ( \_ ) ( \_ ) ( \_ ) & : \text{IdSet Act PAiTerm} \rightarrow \text{TDAct} \ [\text{ctor}] . \\
\text{op } \text{sync} ( \_ ) ( \_ ) ( \_ ) & : \text{IdSet Act PAiTerm} \rightarrow \text{TDAct} \ [\text{ctor}] . \\
\text{op } \text{rho} ( \_ ) & : \text{TAct} \rightarrow \text{TAct} .
\end{align*}
\]

Subsequently the syntax of our crypto-terms is formalized in Maude as follows.

\[
\begin{align*}
\text{op } ( \_ , \_ ) & : \text{PAiTerm PAiTerm} \rightarrow \text{PAiTerm} \ [\text{ctor}] . \\
\text{op } \text{enc} \{ \_ \} & : \text{PAiTerm PAiTerm} \rightarrow \text{PAiTerm} \ [\text{ctor}] . \\
\text{op } \text{enca} \{ \_ \} & : \text{PAiTerm PAiTerm} \rightarrow \text{PAiTerm} \ [\text{ctor}] . \\
\text{op } \text{sign} \{ \_ \} & : \text{PAiTerm PAiTerm} \rightarrow \text{PAiTerm} \ [\text{ctor}] . \\
\text{op } \text{blind} \{ \_ \} & : \text{PAiTerm PAiTerm} \rightarrow \text{PAiTerm} \ [\text{ctor}] .
\end{align*}
\]

Then, the syntax of our process language is defined using the notion of decorated action. The precedence of operators are defined by the expression $\text{prec } \_ i$, where a lower $i$ indicates stronger binding.
subsort TDAct TBDAct < Proc.

op NIL : -> Proc [ctor].
op _ ; _ : TDAct Proc -> Proc [ctor prec 21].
op _ ; _ : Proc Proc -> Proc [ctor prec 22].

Our operational semantics (given in Figure 4) has been implemented through translation of the deduction rules. A set of Maude’s conditional rewrite rules have been used to model transitions in our formalization. Termination is modeled by a set of conditional equations. For example deduction rules (0), (a) and (s1) are implemented as follows, where \( \text{eq} \) stands for equality and \( \text{rl} \) and \( \text{crl} \) stand for rewrite and conditional rewrite rules, respectively. The auxiliary symbol \( $* \) used at the left-hand side of the rewrite rules designates the transition \( \Rightarrow \).

\[
\begin{align*}
\text{var} \quad \text{td} & : \text{TDAct}. \\
\text{vars} \quad p0, p1, pp1 & : \text{Proc}. \\
\text{vars} \quad ga, gap & : \text{Hist}. \\
\end{align*}
\]

\[\text{*** (0) and other terminating rules}\]

\[
\begin{align*}
\text{eq} \ (\text{tick} \ NIL) & = \text{true}. \\
\text{eq} \ (\text{tick} \ d) & = \text{false}. \\
\text{eq} \ (\text{tick} \ (p0 + p1)) & = ((\text{tick} \ p0) \text{ or} \ (\text{tick} \ p1)). \\
\text{eq} \ (\text{tick} \ (p0 ; p1)) & = ((\text{tick} \ p0) \text{ and} \ (\text{tick} \ p1)). \\
\text{eq} \ (\text{tick} \ (p0 || p1)) & = ((\text{tick} \ p0) \text{ and} \ (\text{tick} \ p1)). \\
\end{align*}
\]

\[\text{*** (a)}\]

\[
\begin{align*}
\text{rl} \ \$* \ (\text{td}, \text{ga}) & \Rightarrow \ (\{\text{td}\} \text{ R} \ (\text{NIL}, \text{ga}^*(\text{td}: \text{td}))). \\
\text{crl} \ \$* \ (p0 \ ; p1, \text{ga}) & \Rightarrow \ (\{\text{td}\} \text{ R} \ (\text{pp1}, \text{gap})) \\
& \quad \text{if} \ ((\text{tick} \ p0) \text{ and} \ \$* \ (p1, \text{ga}) \Rightarrow \ (\{\text{td}\} \text{ R} \ (\text{pp1}, \text{gap}))). \\
\end{align*}
\]

In the semantics, we have another transition relation, denoted by \( \rightarrow \). Instead of defining \( \rightarrow \) in terms of \( \Rightarrow \), as in \( \text{strip} \), for the sake of efficiency, we implemented \( \rightarrow \) directly by stripping down the decorated action in each semantic rule. For example, for the three deduction rules for \( \Rightarrow \) specified above, we get the following corresponding rules for \( \rightarrow \) (in the following rules \$ designates the transition relation \( \rightarrow \)).

\[
\begin{align*}
\text{var} \quad \text{ta} & : \text{TAct}. \\
\text{rl} \ \$ \ (\text{sync} \ (\text{I}) \text{ta}, \text{ga}) & \Rightarrow \ (\{\text{ta}\} \text{ r} \ (\text{NIL}, \text{ga}^*(\text{ta} : \text{sync} \ (\text{I}) \text{ta}))). \\
\end{align*}
\]

\[
\begin{align*}
\text{crl} \ \$ \ (p0 \ ; p1 , \text{ga}) & \Rightarrow \ (\{\text{ta}\} \text{ r} \ (\text{pp1}, \text{gap})). \\
\text{if} \ \$ \ (p1, \text{ga}) \Rightarrow \ (\{\text{ta}\} \text{ r} \ (\text{pp1}, \text{gap})). \\
\end{align*}
\]

Finally, the following equalities, represent deduction rules (= \text{refl}) and (= \text{pat}) in Figure 4. In fact, they formalize our introduced indistinguishability concept in Maude. \text{ded} operator is defined to check the deducibility of a term from a given term-set. \text{pat}, \text{ihpat}, and \text{iterm} implement the introduced functions \text{pat} (in Section 2), \text{HPattern} \text{i} and \text{Term} \text{i} (in Section 4), respectively.

\[
\begin{align*}
\text{vars} \quad P, \ Pp, Ppp & : \text{PAiPair}. \\
\text{op} \ \text{Box} & : \rightarrow \text{Pattern} [\text{ctor}]. \\
\text{op} \ _ \text{ ded} _ & : \text{PAiTerm} \text{ PAiTerm} \rightarrow \text{Bool} \quad , \\
\text{op} \ \text{pat} (\_, \_, \_) & : \text{PAiTerm} \text{ PAiTerm} \rightarrow \text{Pattern} [\text{ctor}]. \\
\text{op} \ \text{ iTermSet}(\_,\_,\_) & : \text{Id} \text{ History} \rightarrow \text{PAiPair} [\text{ctor}]. \\
\text{op} \ \text{ taction}(\_,\_,\_) & : \text{Id} \text{ TDAct} \text{ PAiTerm} \rightarrow \text{TAct [ctor]}. \\
\text{op} \ \text{ ihpattern}(\_,\_,\_) & : \text{Id} \text{ History History} \rightarrow \text{PHistory [ctor]}. \\
\end{align*}
\]
*** Term\^i Function ***

eq iTerm(i, snd (I) ac (Emp) ) = empty .

\text{ceq }iTerm(i, snd (I) ac (P) ) = empty \text{ if not (i in I) .}

\text{ceq }iTerm(i, snd (I) ac (P) ) = P \text{ if (i in I) .}

*** pat Function ***

\text{ceq }pat((Pp,Ppp),P) = ((pat(Pp), P), (pat(Ppp), P))
\text{if (Pp \neq empty) and (Ppp \neq empty) .}

\text{ceq }pat((enc(Pp)k), P) = (enc((pat(Pp)k)k), P) \text{ if (P ded k) .}

\text{eq }pat((enc(Pp)k), P) = Box \text{ [otherwise] .}

*** TAction\^i Function ***

\text{ceq }taction(i, snd (I) ac (P),Pp) = bc (Pp)
\text{if (not (i in I) and rho(ac(P))==bc(Ppp) .}

\text{ceq }taction(i, snd (I) ac (P),Pp) = ac (Pp) \text{ if (i in I) .}

*** HPattern\^i Function ***

\text{eq }ihpattern(i,h,((prc : tda) ^ hp))=
\text{taction(i,tda,pat(iTerm(i,tda),iTermSet(i,h))))} ~ ihpattern(i,h,hp) .

*** Indistinguishability Rel. ***

\text{op }_\text{obseq ( ) }_ : \text{History Id History } \to \text{Bool .}

\text{eq }(_\text{ga obseq (i) ga }) = \text{true .}

\text{eq }(_\text{ga obseq (i) gap }) = \text{true if (ihpattern(i,ga,ga) == ihpattern(i,gap,gap)) .}

The following part of the implementation refers to our substitution definition which is presented by the SOS rules. By \textit{Match}, the possibly substitution rules of two term-sets are generated, the rule-set’s consistency is checked using \textit{CheckMatch} operation, and finally, \textit{Apply} and \textit{Substit} are used for applying the substitution rules (if match).

\textbf{sorts} Sbst ExSbst .
\textbf{subsort} Sbst \textless ExSbst .
\textbf{op} Apply (_ , _) : Sbst Proc -> Proc [ctor].
\textbf{op} Substit (_ , _) : Sbst PAiPair -> Pair [ctor].
\textbf{op} Match (_ ? _) : PAiPair PAiPair -> ExSbst [ctor].
\textbf{op} CheckMatch (_ ) : ExSbst -> Bool.

\textbf{**** (blindsign) ****}

\textbf{eq} Match ((blind(X) xp) ? (blind(P)trp)) = ((Match (X ? P)) (Match (xp ? trp))).
\textbf{eq} Match ((blind( x )k) ? (sign{(blind(P)k)sk})) = (x ! (sign{P}sk)).

\textbf{**** (p3)-SOS ********}

\textbf{crl}$* ((p0 || p1) , ga) =>
\{sync (I ; J) a (P)\} R (((Apply((Match(X ? P)) , pp0)) || pp1) ,
\text{(ga ~ ((p0 || p1) : (sync (I ; J) a (P))))}))
\text{if }$* (p0 , ga) => ( {rcv (I) a (X)} R ( pp0 , gap0 )) /\$
\text{$* (p1 , ga) => ( {snd (J) a (P)} R ( pp1 , gap1 )) /\$
\text{( CheckMatch((Match(X ? P))))}.

\textbf{**** (summation) *****}

\textbf{op} Sum (_ : _) : PAiPair PAiPair Proc -> Proc [ctor].
8.2 $E\mu$ in Maude

The following constructors formalize the syntax of our logic defined in 5. In what follows, `emnot`, `emand`, and `emor` denote the basic logical connectives “not”, “and”, and “or”, respectively. Also, “$<a>$” and “$[a]$” denote the modalities “$(a)\_” and “$[a]_\_\_”’, respectively. “$K_\_”’ and “$C_\_”’ denote the knowledge and common knowledge operators, respectively. “$\_ \leftrightarrow \_” and “$[\_\_]_\_”’ are respectively notations which used in Maude for the two acronyms “$\_\_” and “$[\_\_]_\_”’.

sort Emuf.
...
op top : -> Emuf [ctor].
op bot : -> Emuf [ctor].
op emnot _ : Emuf -> Emuf.
op _emand_ : Emuf Emuf -> Emuf [ctor].
op _emor_ : Emuf Emuf -> Emuf [ctor].
...
op _Has_ : Id Term -> Emuf [ctor].
op <_>_ : TAct Emuf -> Emuf [ctor].
op _[]_ : TAct Emuf -> Emuf [ctor].
op K{...} : Id Emuf -> Emuf [ctor].
op C{...} : IdSet Emuf -> Emuf [ctor].
op _ <-| : TAct -> Emuf [ctor].
op [.*]_ : Emuf -> Emuf [ctor].

The expression “s0 s |= phi” denotes that formula “phi” holds at state “s” in an LTS with the initial state “s0” (with the semantics given before). For example, the semantics of `emor` is specified as the following conditional rewrite rule.

vars s s0 : State.
vars phi psi : Emuf.
vars b0 b1 : Bool.
...
crl (s0 (p,ga) |= (phi emor psi)) => * (b0 or b1)
    if (s0 (p,ga) |= phi) => * b0 /
    (s0 (p,ga) |= psi) => * b1.

The semantics of temporal and epistemic constructs is a bit more involved and makes use of the reflective semantics of Maude. For example, consider the following semantics of the operator “$<a>$”.

var ta : TAct.
crl (s0 (p,ga) |= (i Has P)) => * true
    if ( iTermSet(i,ga) ded P).
crl (s0 (p,ga) |= (i Has P)) => * false
    if not ( iTermSet(i,ga) ded P).
...
crl [atrue]:
(s0 (p,ga) |= (<ta> phi)) => * true
    if trans {ta}(p,ga)=>target(pp,gap) /
    (s0 (pp,gap) |= phi) =>* true.
...
crl (s0 s |= (<ta> phi)) => * false
    if not (isaTrue (upTerm (s0 s |= (<ta> phi)))).

The first conditional rewrite rule is related to Has(i, trm) operator and states that the principal i has the term trm in the state (p, ga) if the term can be deduced from the relevant history. The second conditional rule, labeled atrue, expresses that if at state (p, ga) a transition labeled with a is enabled and a transition labeled with a at the target of which formula phi holds, then “$<a>$
phi" will be true. However, to establish that "<a> phi" is false, we need to check if all transitions labeled with a lead to a state in which phi does not hold, i.e., there is no transition labeled with a enabled at (p, pi). The impossibility of a certain transition can be achieved through the reflective semantics of Maude. In other words, propositions like isaTrue and rMove involve using the meta-level function metaXapply which gives us information about possibility of applying a certain rewrite rule on a term. The following is the implementation of function rMove that returns whether it is possible to apply rewrite rule atrue (given before) on term t [9] and other constructs such as "[a]" or "K{[.]_}" can be implemented similarly.

ceq isaTrue (t)= isfalse? :: Result4Tuple
  if isfalse? := metaXapply(['EmuBasic'],t,'atrue,none,0,unbounded,0).

8.3 Our example protocols in Maude

In this section section we briefly present the implementation our case studies reported in Examples 2 and 4 in our Maude implementation framework. We use the following commands in the Maude command-line in order to simulate the protocols and verify them against the specified formulae:

- rew + (p,niks) : calculates a trace of a CryptoPAi process p, where niks stands for the empty history,
- rew s0 (p,niks) |= phi : is used to verify an Emu formula phi on a process p from an initial state s0.

Modeling the NS protocol The following module models the NS protocol in Example 2 (the include expression stands for importing the syntax of Crypto – PAi and Eμ).

mod NS_Protocol is
  include CryptoPAi.
  include EmuBasic .
...

The processes Server, Clienta, and Clientb are specified below, and the protocol NSProtocol is the parallel composition of these participating processes. The initial state of the protocol (with empty history) is denoted by NSPState0.

eq NSProtocol = (( ClientA || ClientB ) || Server ) .
eq NSPState0 = ( NSProtocol , niks ) .
eq Server = Sum ( (kAS : (kAS1 , kAS2)) ( 
  Sum ( (kBS : (kBS1 , kBS2)) ( 
    Sum ( (kAB : (kAB1 , kAB2)) ( 
      (((snd ( A;S ) TakeKeyAS (kAS)) ;
        (snd ( B;S ) TakeKeyBS (kBS))) ;
      (rcv ( A;B;S;E ) Stp1 ((xA , (xB , xnA)))) ;
      (snd ( A;B;S;E ) Stp2 (enc{(xA , (kAB , (xB ,
                        (enc{(kAB , xA)})kBS))))kAS)))))))) )

eq Clienta = Sum ( (nA : (nA1 , nA2)) ( 
  ((((rcv ( A;S ) TakeKeyAS (xk)) ;
    (snd ( A;B;S;E ) Stp1 ((A , (B , nA)))) ;
    (rcv ( A;B;S;E ) Stp2 (enc{(nA , (xkB , (B , xB)))xk})) ;
    (snd ( A;B;S;E ) Stp3 (xB))) ;
    (rcv ( A;B;S;E ) Stp4 (enc{(xB)}xkB)))) ;
    (snd ( A;B;S;E ) Stp5 (enc{nBb}kB)))) )

23
eq Clientb = \text{Sum} ( (nB : (nB1, nB2)) ( (((rcv (B;S) TakeKeyBS (xk)) ;
    (rcv (A;B;S;E) Stp3 (enc{(xkAB, A)}xk)));
    (snd (A;B;S;E) Stp4 (enc{(nB)}xkAB)));
    (rcv (A;B;S;E) Stp5 (enc{(xnBb)xkAB}))))).

Now, we can specify and verify the properties in this framework. For instance, the following property specifies that the shared key between A and B is never revealed to E:

- rew NSPState0 (NSProtocol, niks) |= [.*] (emnot (E Has kAB1)).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*]emnot (E Has kAB1).
  rewrites: 429335 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * true

- rew NSPState0 (NSProtocol, niks) |= [.*] (emnot (E Has kAB2)).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*]emnot (E Has kAB2).
  rewrites: 429335 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * true

- rew NSPState0 (NSProtocol, niks) |= [.*] ((Stp5 (enc{(nBb)xkAB1})(A Has kAB1))).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*][Stp5(enc{(nBb)xkAB1}](A Has kAB1).
  rewrites: 586479 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * true

- rew NSPState0 (NSProtocol, niks) |= [.*] (K{(A; B; S)}(K{A}(A Has kAB1))).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*][Stp5(enc{(nBb)xkAB1}](K{A}A Has kAB1).
  rewrites: 810901 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * true

- rew NSPState0 (NSProtocol, niks) |= [.*] (C{(A; B; S)}(K{A}(A Has kAB1))).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*][Stp5(enc{(nBb)xkAB1}](C{A; B; S}(K{A}A Has kAB1).
  rewrites: 810901 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * true

- rew NSPState0 (NSProtocol, niks) |= [.*] ((K{(B)(A Has kAB1)) emand (K{S}(S Has kAB1))).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*][Stp5(enc{(nBb)xkAB1}](C{A; B; S}(K{A}A Has kAB1) emand (K{S}(S Has kAB1)).
  rewrites: 810901 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * true

- rew NSPState0 (NSProtocol, niks) |= [.*] (K{(B)(B Has kAS1))).
  rewrite in EmuBasic : NSPState0 NSProtocol,niks |= [.*][TakeKeyAS(kAS1)][K{(B)(B Has kAS1))].
  rewrites: 8186 in 1628036047000ms cpu (0 rewrites/second)
  result Satform: * false
rew NSPState0 (NSProtocol, niks) |= [.\*] ([TakeKeyAS(kBS2)] ([.\*] ( (B Has kBS2)))).
rewrite in EmuBasic: NSPState0 NSProtocol, niks |= [.\*] ([TakeKeyAS(kBS2)] [.\*] (B Has kBS2)).
rewrites: 1496309 in 1628036047000ms cpu (51503ms real) (0 rewrites/second)
result Satform: * true

rew NSPState0 (NSProtocol, niks) |= [.\*] ([TakeKeyAS(kAS1)] ([.\*] ((emnot(K{B}(B Has kAS1))) emand (emnot(K{E}(E Has kAS1)))))).
rewrite in EmuBasic: NSPState0 NSProtocol, niks |= [.\*] ([TakeKeyAS(kAS1)] [.\*] (emnot(K{B}(B Has kAS1)) emand emnot(K{E}(E Has kAS1)))).
rewrites: 1045736625 in 1628036047000ms cpu (13023863ms real) (0 rewrites/second)
result Satform: * true

Modeling the FOO protocol

The remainder of this section presents Maude's module specification of the FOO protocol in Example 4). In the following specification, FOOProtocol and FOOState0 stand for the specification of the protocol and its initial state, respectively.

mod FOO_Protocol is
include CryptoPAi.
include EmuBasic.
...
eq FOOProtocol = ( Reg || ( Voteri || ( Admin || Counter ) ) )
eq FOOState0 = ( FOOProtocol , niks )
eq Reg = Sum ( ((pki,ski) : ((pki1,ski1),(pki2,ski2))) (Sum ( ((pkA,skA) : ((pkA1,skA1),(pkA2,skA2))) ((((((snd ( Vi;A;C;E ) TakeViPubK ((pki , pkA))) ; (snd ( Vi ) TakeViPrivK (ski))) ; (snd ( A ) TakeAdPrivK (skA))) ; (snd ( Vi;A;C;E ) TakeCoPubK (pkA))) )).
eq Admin = (((rcv ( Vi;A;C;E ) TakeAdPubK (xpki,xpki))) ; (rcv ( A ) TakeAdPrivK ( xskA ))) ; (rcv ( Vi;A;C;E ) ViReqAdSign ((xVi,(xsvbc , xvbc)))) ; (snd ( Vi;A;C;E ) AdAckSign ((xVi,(sign{xvbc}xskA)))).
eq Voteri = Sum ( (vi : (vi1 , vi2)) ( ((((((rcv ( Vi;A;C;E ) TakeViPubK ((xpki , xpki))) ; (rcv ( Vi ) TakeViPrivK (xski))) ; (rcv ( Vi;A;C;E ) ViReqAdSign ((Vi,(sign{(blind{enc{vi}ki}ri})xski),(blind{(enc{vi}ki}ri)}))) ; (rcv ( Vi;A;C;E ) AdAckSign ((Vi,(blind{xsc}ri)))) ; (rcv ( Vi;A;C;E ) CoVoteList ((xl,(xsc,(enc{vi}ki)))))) ; (snd ( Vi ) ViRevealKey ((xl , ki))))) ; (rcv ( Vi;A;C;E ) PublishResult (empty))).
eq Counter = Sum ( (l : (l1 , l2)) ( (((rcv ( Vi;A;C;E ) TakeCoPubK (xpki)) ; (rcv ( Vi ) ViVoting ((xsc , xc)))) ; (snd ( Vi;A;C;E ) CoVoteList ((l , (xsc , xc)))) ; (rcv ( Vi ) ViRevealKey ((l , xki)))).

25
Some of the properties of the protocol have been verified on the above specification and their result are reported below.

- rew FOOState0 (FOOProtocol, niks) |= [*] ([TakeViPrivK(ski1)](emnot (E Has ski1)))
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*]([TakeViPrivK(ski1)]emnot (E Has ski1))
  rewrites: 915479 in 1628036047000ms cpu (55030ms real) (0 rewrites/second)
  result Satform: * true

- rew FOOState0 (FOOProtocol, niks) |= [*] ([TakeViPrivK(ski1)](emnot (K{E} (TakeViPrivK(ski1) <-|)))).
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*]([TakeViPrivK(ski1)]emnot (K{E}(TakeViPrivK(ski1)) <-|)).
  rewrites: 5276064119 in 1628036047000ms cpu (61563115ms real) (3 rewrites/second)
  result Satform: * true

- rew FOOState0 (FOOProtocol, niks) |= [*] ([TakeViPrivK(ski1)]([K{Vi} (TakeViPrivK(ski1) <-|)))).
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*](K{Vi}(TakeViPrivK(ski1)) <-|).
  rewrites: 13857468327 in 1628036047000ms cpu (56267527ms real) (8 rewrites/second)
  result Satform: * true

- rew FOOState0 (FOOProtocol, niks) |= [*]([ViRevealKey(l1,ki)] ((Vi Has vi1) emor (Vi Has vi2)))
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*](Vi Has vi1 emor Vi Has vi2))
  rewrites: 6929255 in 1628036047000ms cpu (76864ms real) (0 rewrites/second)
  result Satform: * true

- rew FOOState0 (FOOProtocol, niks) |= [*]([ViRevealKey(l1,ki)] (Vi Has ski1 emor (Vi Has ski2)))
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*](Vi Has ski1 emor Vi Has ski2))
  rewrites: 6887015 in 1628036047000ms cpu (78038ms real) (0 rewrites/second)
  result Satform: * true

- rew FOOState0 (FOOProtocol, niks) |= [*]([ViRevealKey(l1,ki)] (E Has vi1 emor (E Has vi2)))
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*](E Has vi1 emor E Has vi2))
  rewrites: 6891623 in 1628036047000ms cpu (78116ms real) (0 rewrites/second)
  result Satform: * true

- rew FOOState0 (FOOProtocol, niks) |= [*]([ViRevealKey(l1,ki)] (E Has ski1 emor (E Has ski2)))
  rewrite in EmuBasic: FOOState0 FOOProtocol,niks |= [*](E Has ski1 emor E Has ski2))
  rewrites: 518997 in 1628036047000ms cpu (22068ms real) (0 rewrites/second)
  result Satform: * false

- rew FOOState0 (FOOProtocol, niks) |= [*] ([ViVoting(sign{enc{vi1}ki}skA1, enc{vi1}ki}) (AdAckSign(Vi,sign{blind{enc{vi1}ki}ri}skA1) <-| )).
9 Conclusions

In this paper, we have presented a framework which supports operational specification and temporal as well as epistemic analysis of security protocols. This enables an intuitive description of both protocols and their security properties. The framework including a process algebraic formalism for the operational specification and an epistemic extension of modal $\mu$-calculus with past for the property specification. The framework has been further formalized and mechanized in the rewriting logic of Maude.

Our logic is very expressive thanks to the expressive power of modal $\mu$-calculus. However, for most practical purposes a subset of modal $\mu$-calculus (e.g., corresponding to LTL or CTL) suffices. The less expressive logics enable both more readable specifications and more efficient verification. We plan to investigate efficient model-checking algorithms after defining a practical subset of our logical specification language. The main practical motivation for this work came from the domain of e-voting protocols. We plan to further investigate the applicability of our framework in this domain by applying it to more case studies (see [4] for some ideas in this direction). Extending our framework with probabilities, along the lines of [23], is another avenue for future research.

References