An Extension of CryptoPAi to the Formal Analysis of E-voting Protocols

Hamid Reza Mahrooghi\textsuperscript{a,}\textsuperscript{*}, Rasool Jalili\textsuperscript{b}

\textsuperscript{a}Department of Computer Engineering, Imamreza International University, Mashhad, Iran.
\textsuperscript{b}Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

**A R T I C L E I N F O.**

Article history:
Received: 16 October 2018
Revised: 4 August 2019
Accepted: 9 October 2019
Published Online: 2 December 2019

**Keywords:**

**A B S T R A C T**

CryptoPAi is a hybrid operational-epistemic framework for specification and analysis of security protocols with genuine support for cryptographic constructs. This framework includes a process algebraic formalism for the operational specification and an epistemic extension of modal $\mu$-calculus with past for the property specification. In this paper, we extend CryptoPAi framework with more cryptographic constructs. The main practical motivation for this work came from the domain of e-voting protocols and then we investigate the applicability of the extended framework in this domain. The framework provides explicit support for cryptographic constructs, which is among the most essential ingredients of security and e-voting protocols. We apply our extended framework to the FOO e-voting protocol. We also promote the prototype model-checker of the framework in the Maude rewriting logic tool and apply it to model-check some specified properties on their corresponding models.

\textcopyright 2019 JComSec. All rights reserved.

1 Introduction

Operational specifications provide an intuitive representation of security protocols (see [1, 2, 3, 4, 5]). Unfortunately, these standard and successful verification schemes use temporal logics that are not well-suited for expressing knowledge-based properties.

Also, epistemic (i.e., knowledge-related) specifications [6, 8] provide inherent support for correctness requirements: most security properties (e.g., secrecy and anonymity) concern impossibility of gaining knowledge about certain facts, or correspondence between a notion of belief and what is actually the case (e.g., authentication). Therefore, many approaches based on epistemic logics have been developed for the analysis of such protocols.

However, modeling protocols using epistemic-logic-based approaches requires a high degree of expertise and verification of functional properties is often very complex. The information updates generating the transitions between epistemic states are especially tedious to specify, because logics are geared to expressing properties rather than operational steps of a protocol. There is still ongoing research in finding a generic and compositional approach to protocol specification in the epistemic style (see,
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 some literature on such combinational frameworks \([14–16]\) (cf. related work section below). The approach proposed in \([17]\) allows one to specify the behavior of a protocol in a process language and verify properties expressed in a logic with both temporal and epistemic operators.

This paper extends and promotes the \textit{CryptoPAi} model checking application based on some essential issues that we have to consider. As defined in \([18]\) and \([19]\), using \(\text{Rand}\) is one of these issues that we need in order to keep track of different invocations of the encryption primitive (representing probabilistic operations). We reason under the assumption of perfect cryptography and assume that the results of two invocations of encryption will be different, even with the same arguments, as opposed to two copies of the same invocation of the encryption operation, which are in fact the same terms. Another issue is related to the definition of pattern that is a parameter to our theory. An alternative definition would be possible, with some complications. Based on this approach, our pattern-definition have to be refined using a type-preserving bijective renaming function of keys, nonces, and random-numbers. This reflects the intuition that two different randomly chosen keys (nonces, or random-numbers) are indistinguishable \([20]\). Although they have different names, they both represent samples from the same distribution and we can replace one of them with the other. These parametericity are further demonstrated by our extension to the term structure and the definition of pattern in Section 2.

We provide some more advanced cryptographic constructs to allow specifying the behavior of more protocols in our process language and then verifying properties expressed in our logic with both temporal and epistemic operators. The domain of e-voting protocols is the main practical motivation for current work. We investigate the applicability of our framework in this domain by applying it to the FOO e-voting protocol. Using our tool, we model-check some specified properties of this protocol on their corresponding models.

Related Work

Using epistemic logic for the specification and verification of security protocols has been the subject of numerous papers and a comprehensive survey of all these pieces of work goes beyond the scope of this paper. We refer to two doctoral theses \([13, 21]\), 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, we only focus on approaches that combine the behavioral and epistemic approaches to protocol specification and verification.

The fact that the two verification approaches, process algebraic and epistemic, are complementary and that they should ideally be combined has already been recognized in \([7]\), where the aim is, just as here, to provide a framework in which both protocol specification and correctness criteria can be specified succinctly and intuitively (and the authors indeed put the two approaches in sharp contrast). However, \([7]\) takes a totally different path towards its goals and develops a domain-theoretic notion of function views to define security protocols and their properties.

Thus, combining the best of the two operational and epistemic styles is highly desirable, i.e., an approach allowing for protocol specification in the operational style and property specification and verification in the epistemic style. \textit{CryptoPAi} is our earlier combinational framework with a genuine support for some cryptographic constructs \([17]\). In the framework, a process-algebraic syntax (with parameterized actions) has been introduced for expressing the operational specification of the security protocols and an epistemic extension of the modal \(\mu\)-calculus with past \([22]\) for property specification.

\textit{Interpreted Systems} \([6, 23, 24]\) are close to the operational semantics of our process language. In fact, it is possible to translate ALTSs defined by our SOS rules to interpreted sys-
In [16] an extension of a dynamic spatial logic with knowledge-related constructs is presented. There are a number of essential differences between the approach of [16] and that of [17] such as: the notion of knowledge captured in [16] captures only an explicit notion of local knowledge through the content of the received messages; the notion of symbolic crypto-term used in [16] is coarser than the notion used in [20] and adopted in [17].

In [14] an epistemic logic for the applied π-calculus [25] is presented. We improved upon the approach of [14] in some essential ways such as: The notion of knowledge captured in [17] allows for different views of the events by different principals and introduces a more expressive logic with fixed points. A prototype model-checker for our framework is provided in the Maude rewriting logic tool (see [17]) while [14] does not introduce a tool.

There is still ongoing research in finding a generic and compositional approach to protocol specification in the operational style and property specification in the epistemic style. We refer to [17], [13], [7], [26] for a more detailed comparison of epistemic-based vs. process-based protocol verification. As in [17], the concept of indistinguishability used here bears resemblances to the data independence technique in [27]. In fact, runs of a protocol are indistinguishable if they appear equal to a principal (as defined by the visibility range of actions and their public appearance).

We further extend the framework of CryptoPAi with more advanced constructs and it can be extended for forming a very rich framework with support for epistemic, probabilistic and cryptographic constructs as a future work. A part of the framework proposed in [28] has been extended with probabilities in [15].

Structure of the Paper
In Section 2.1 we recall our basic cryptographic terms using symmetric encryption and then this basic setting is extended 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. In sections 2.3 and 2.4 the syntax and the semantics of CryptoPAi is recalled. In Section 3 we present the syntax and semantics of $\text{Ep}_\mu$ (as our property specification language), which is an epistemic extension of the modal $\mu$-calculus with past. We have completed the implementation of our earlier prototype (to mechanize reasoning about our specifications), which we report in Section 4 and further show specifying and reasoning about FOO e-voting protocol as a case study. Section 5 concludes the paper and presents some directions for future research. Some semantic results and details of the implementation are left out and we refer to [17] for complete details of the techniques used in this implementation and the results.

2 The CryptoPAi Calculus
CryptoPAi calculus extended the syntax of CCS with two essential ingredients: identities and cryptographic terms. In the remainder of this section, we first define our new notion of cryptographic terms. Subsequently, we recall the syntax of the CryptoPAi calculus and its operational semantics.

2.1 Cryptographic Terms
Our original signature for cryptographic terms features basic constructs such as plain text, nonces, symmetric keys, and encryption, given in the following definition. This signature can be easily extended, as demonstrated in the following definition, with more constructs, such as asymmetric keys, signing, and blinding.

**Definition 1 (Symbolic Terms).** We assume a syntactic class $\text{Key}$ of symmetric and asymmetric keys, typically denoted by $k$, $k_1$, ..., $\text{pk}_{id}$, $\text{sk}_{id}$, ..., ($k$ and $k_1$ represent symmetric keys, the public $\text{pk}_{id}$ and $\text{sk}_{id}$ represent the public and private keys of a principal with identity $id$ respectively); a syntactic class $\text{Id}$ of principal identities, typically represented by $A$, $B$, 1, 2, ....; a class $\text{Var}$ of variables, typically denoted by $x$, $x_A$, $x_k$, ....; a class $\text{Nonce}$ of nonces, denoted by $n$, $n_1$, ....; and a class $\text{Msg}$ of plain-text messages, denoted by $m$, $m_1$, .... Plain text messages, keys and nonces are different syntactic classes and are indeed assumed to be disjoint.
The set of crypto-terms, denoted by \( T \), is defined by the following grammar:

\[
M, N ::= k \mid id \mid m \mid x \mid n \mid r \mid \{M\}_{k,r} \mid (M, N) \mid pk_id \mid sk_id \mid \{M\}^a_{pk_id, r} \mid \{M\}^b_{sk_id, r}
\]

where \( k, pk_id, sk_id, r, id \in Key, \ id \in Id, \ m \in \text{Msg}, \ x \in \text{Var}, \ id \in Id, \ m \in \text{Msg}, \) and \( n \in \text{Nonce} \). The term \( \{M\}_{k,r} \) is the result of encrypting \( M \) with the symmetric key \( k \) and random-number \( r \in \text{Rnd} \). The term \( (M, N) \) represents pairing of \( M \) and \( N \).

The set of crypto-terms, is extended by asymmetrically encrypted term \( \{M\}^a_{pk_id, r} \), signed term \( \{M\}^s_{sk_id, r} \), and blinded term \( \{M\}^b_{sk_id, r} \). \( \{M\}^a_{pk_id, r} \) denotes the term \( M \) blinded using the random number \( r \in \text{Rnd} \) (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.

Since some of the cryptographic constructors (such as encryption) represent probabilistic operations, we need to use \( \text{Rnd} \) in order to keep track of different invocations of the encryption primitive. Under the assumption of perfect cryptography, we assume that the results of two invocations of encryption will be different, even with the same arguments, as opposed to two copies of the same invocation of the encryption operation, which are in fact the same terms. As the label \( r \) is the invocation identifier of the encryption primitive, whenever we consider two expressions \( \{m\}_{k,r} \) and \( \{m'\}_{k',r} \) with the same label \( r \), then it should also hold that \( m = m' \) and \( k = k' \).

To avoid cluttering the notation, we assume that pairing associates to the right and thus, we denote nested pairs of the form \( (M_0, (M_1, \ldots, (M_{n-1}, M_n))) \) by \( (M_0, M_1, \ldots, M_{n-1}, M_n) \). Informally, we may just write \( \{M\}_{k, r} \), \( \{M\}^a_{pk_id, r} \), and \( \{M\}^s_{sk_id} \) instead of \( \{M\}_{k, r} \), \( \{M\}^a_{pk_id, r} \), and \( \{M\}^s_{sk_id, r} \), when \( r \) is irrelevant or clear from the context.

Plain text messages may be of different types (e.g., strings and integers), each equipped with an algebraic structure (e.g., concatenation and arithmetic operators, respectively). We do not elaborate further on this aspect of term specification and allow for its tacit use in protocol specifications.

### 2.2 Term Deduction

Terms can be derived (i.e., (de)constructed) from others due to using cryptographic operations. This concept has been captured by the deduction rules in Figure 1. A judgement of the form \( T \vdash M \) reads term \( M \) is derivable from the set \( T \) 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.

To apply the extension mentioned in Section 1 we need to extend our term deduction rules by the relevant rules, depicted in Figure 2 which are also relevant to the public-key encryption system.

The rules \( \text{blind} \), \( \text{unblind} \), and \( \text{blindsign} \), show that a principal who has the key \( k \), can blind or unblind a term, and also obtain \( \{M\}^s_{sk_i} \) from the term \( \{M\}^s_{sk_i} \) blinded by the principal \( i \) (i.e., \( \{M\}^s_{sk_i} \) \( \equiv \{M\}^s_{sk_{id}} \)). The term \( \{M\}^s_{sk_i} \) means that the term \( M \) has been encrypted using the public-key \( pk_i \) \( \text{(enc}_a \text{ rule)} \). \( sk_i \) denotes the relevant private-key of \( pk_i \) and is used to extract the term \( M \) from \( \{M\}^a_{pk_i} \) \( \text{(dec}_a \text{ rule)} \). The term \( \{M\}^s_{sk_i} \) represents the message \( M \) signed by the private-key \( sk_i \) \( \text{(sign) rule)} \). The signed term \( \{M\}^s_{sk_i} \) can be unsigned (and indeed check the sign) using the corresponding public-key \( pk_i \) \( \text{(unsig) rule)} \). However, depending on the signature scheme, the unsign rule may be used to retrieve the message \( M \) from the signature, i.e.,

\[
\text{unsing} \frac{T \vdash \{M\}^s_{sk_i}}{T \vdash M}
\]

### 2.3 CryptoPAi Syntax

In this section we recall the syntax of our operational specification language CryptoPAi, which is a process algebraic language with value-passing of crypto-terms. Let \( \text{Act} \) be a finite set of action names ranged over by \( a, la, ?a, b, \ldots \), and let \( Id \) be a finite set of identities typically denoted by \( i, j, \ldots \). We use the exclamation and the question mark to designate send and receive
\[ (\in_t) \quad T \vdash M \quad M \in T \]

\[ (\text{pair}) \quad T \vdash M \quad T \vdash N \quad (T \vdash (M, N)) \quad (\text{fst}) \quad T \vdash (M, N) \quad T \vdash M \]

\[ (\text{snd}) \quad T \vdash (M, N) \quad T \vdash N \]

\[ (\text{dec}) \quad T \vdash \{ M \}_k \quad T \vdash k \quad T \vdash \{ M \}_k \]

\[ (\text{enc}) \quad T \vdash k \quad T \vdash M \quad T \vdash \{ M \}_k \]

\begin{align*}
\text{Figure 1.} & \quad \text{Term Deduction Rules.} \\
\text{Figure 2.} & \quad \text{Extended Term Deduction Rules.}
\end{align*}

actions, respectively. Actions that result from a synchronization as well as internal actions are denoted without any annotation. (Note that this is a slight deviation from Milner’s CCS, where the result of a synchronization is necessarily denoted by an internal action. In our calculus, agents may be able to observe and derive useful information from synchronizations.) Action \( \tau \in \text{Act} \) denotes the silent action which also represents a message that offers no new information to any observer.

\[
P, Q ::= \begin{cases} 
\text{Processes} \\
0 & \text{inaction} \\
D; P & \text{action prefixing} \\
P + P & \text{nondeterministic choice} \\
P || P & \text{parallel composition} \\
\partial(P) & \text{encapsulation} \\
\end{cases}
\]

\[
D ::= (J)\alpha(M) \quad d = (J)\alpha(M) \in D 
\]

Constant 0 denotes inaction (termination). Action prefixing is denoted by \( d; P \), for each \( d \in D \); an action may contain zero or more crypto-terms as parameters. 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. \( P|Q \) denotes 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. Our language also includes an encapsulation operator, \( \partial \), which prevents unsuccessful communication attempts and turns them into deadlock.

\[
d = (J)\alpha(M) \in D \quad \text{denotes a parametric decorated action, which has the following intuition: action } \alpha \in \text{Act} \text{ with parameters } M \text{ is visible to principal } i \in J \subseteq I_d. \text{ Other principals } (j \notin J) \text{ observe } \rho(\alpha(M)) \text{ being taken, where } \rho : \text{Act} \times \text{Term} \rightarrow \text{Act} \times \text{Term} \text{ is a global renaming function, which assigns a “public appearance” to every parameterized action and should be defined by the specifier of a protocol. We assume that } \rho(\tau) \text{ is always defined to be } \tau, \text{ reflecting the fact that } \tau \text{ is invisible to all princi-}
\]
The combination of identity decoration on actions, action renaming (public appearance), and action parameters provides different views on the behavior of protocols, according to different (participating or observing) principals.

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

### 2.4 **CryptoPAi Semantics: Indistinguishability**

The operational state of CryptoPAi is typically denoted by $(P, \Gamma)$, which comprises two components: a process $P$ and a computation $\Gamma$. 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, i.e., which processes executed which decorated actions leading to the current state. The process $P$ has the syntax described in the previous section. The computation $\Gamma$, formally defined below, is a sequence of pairs of processes and decorated actions.

**Definition 2 (Computation).** A computation $\Gamma \in \text{History}$ is a sequence of pairs $(p, d)$, where $p$ is a process in the syntax given in the previous section of CryptoPAi and $d$ is a parameterized decorated action with term parameter (of the form $(J)\alpha(M)$, see the same section). By $(p, d) \sim \Gamma$, we denote concatenating a pair $(p, d)$ with a computation $\Gamma$. The empty computation is denoted by $\varepsilon$.

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). In order to define the indistinguishability, a number of auxiliary definitions are necessary, presented in the remainder of this section.

**Patterns.** The following Pattern function (as in [20]) 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 $\square$).

**Definition 3 (Patterns).** A pattern is a crypto-term with (possibly multiple) occurrences of a specific constant $\sqcap^r$ (indexed by random-numbers), capturing those encrypted messages that cannot be decrypted. Hence, the set $\mathcal{PTerm}$ of patterns is defined with the following grammar.

$$
M_P, N_P ::= k \mid pk \mid sk \mid id \mid m \mid \{M_P\}_{k,r}^a \mid \{M_P\}_{p,k,r}^a \mid \{M_P\}_{sk,r}^a \mid \{M_P\}_{r}^{b} \mid (M_P, N_P) \mid \square^r
$$

The function $\text{Pattern} : \text{Term} \rightarrow \mathcal{PTerm}$ is redefined below using the auxiliary function $\text{pat} : \text{Term} \times \text{Key} \rightarrow \mathcal{PTerm}$ 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.

$$
\begin{align*}
\text{pat}(id, K) &= id \ (id \in Id), \\
\text{pat}(k, K) &= k \ (k \in \text{Key}), \\
\text{pat}(m, K) &= m \ (m \in \text{Msg}), \\
\text{pat}(n, K) &= n \ (n \in \text{Nonce}), \\
\text{pat}((M, N), K) &= (\text{pat}(M, K), \text{pat}(N, K)), \\
\text{pat}(\{M\}_k, K) &= \begin{cases} 
\{\text{pat}(M, K)\}_k & \text{if } k \in K, \\
\square^r & \text{otherwise}.
\end{cases}
\end{align*}
$$

To apply the extension targeted in this paper (Section 1), our pattern function should be extended by the following statements relevant to the public-key encryption system.
there exist a type preserving bijection \( \sigma \)
\( K \);
\( \text{alent, denoted by} \)
\( m \)
\( \text{terms} \)
\( m \)
\( \text{rating the randomness of keys and nonces, we} \)
\( \text{all keys} \)
\( \text{patey} \)
\( \text{∪} \)
\( \text{N} \)
\( \text{′} \)
\( \text{′} \)
\( \text{we re-use the same notation and write} \)
\( \text{Using this notion of patterns and incorpo-} \)
\( \text{r} \)
\( \text{otherwise.} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{otherwise.} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{otherwise.} \)
\( \text{We re-use the same notation and write} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{otherwise.} \)
\( \text{Example 1. The following examples illustrate the notion of pattern:} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{pat} \)
\( \text{m} \)
\( \text{a} \)
\( \text{r} \)
\( \text{otherwise.} \)
\( \text{Example 2. The following examples illustrate the notion of pattern equivalence:} \)
\( \text{We consider two terms} \)
\( \text{and} \)
\( \text{They hold that for each term} \)
\( \text{and} \)
\( \text{then} \)
\( \text{and} \)
\( \text{Proof.} \)
\( \text{In order to lift the notion of pattern equiva-} \)
\( \text{Definition 5 (Appearance of Actions).} \)
\( \text{Given a decorated action} \)
\( \text{and an identity} \)
\( \text{the appearance of} \)
\( \text{actions with no parameter (including} \)
\( \text{Definition 6 (Visible Terms).} \)
\( \text{Given a computation} \)
\( \text{and an identity} \)
\( \text{the set of visible terms in} \)
\( \text{according to} \)
\( \text{and} \)
\( \text{denote an empty computation and empty set, re-} \)
An Extension of CryptoPAi to the Formal Analysis of E-voting Protocols — H. R. Mahrooghi and R. Jalili

\[ \text{TermSet}^i(\Gamma) = \begin{cases} \emptyset & \text{if } \Gamma = \varepsilon, \\ \{M\} \cup \text{TermSet}^i(\Gamma') & \text{if } \Gamma = (p, d) \not\rightarrow \Gamma' \text{ and } \text{Appear}^i(d) = a(M), \\ \text{TermSet}^i(\Gamma') & \text{if } \Gamma = (p, d) \not\rightarrow \Gamma' \text{ and } \text{Appear}^i(d) = a(). \end{cases} \]

Definition 7 (Indistinguishability of Computations). Let \( \Gamma \) be a computation, then the pattern of computation \( \Gamma \) from the viewpoint of principal \( i \) is denoted by \( \text{HPattern}^i(\Gamma) \), as defined below.

\[ \text{HPattern}^i(\Gamma) = \begin{cases} \varepsilon & \text{if } \Gamma = \varepsilon, \\ \text{HPattern}^i(\Gamma') & \text{if } \Gamma = (P, D) \not\rightarrow \Gamma' \text{ and } \text{Appear}^i(D) = \tau, \\ a(\text{pat}(M, T^i)) \not\rightarrow \text{HPattern}^i(\Gamma') & \text{if } \Gamma = (p, d) \not\rightarrow \Gamma' \text{ and } \text{Appear}^i(d) = a(M), \\ a() \not\rightarrow \text{HPattern}^i(\Gamma') & \text{if } \Gamma = (p, d) \not\rightarrow \Gamma' \text{ and } \text{Appear}^i(d) = a() \text{ and } a \neq \tau, \end{cases} \]

where \( T^i = \{t|t \in \text{Term} \land \text{TermSet}^i(\Gamma) \vdash t\} \).

Next, we define the indistinguishability relation, denoted by \( \not\approx \), of histories as follows.

\[ \Gamma \not\approx \Gamma' \text{ if and only if } \text{HPattern}^i(\Gamma) = \text{HPattern}^i(\Gamma') \sigma' \sigma'' \text{, for some bijections } \sigma : \text{Key} \rightarrow \text{Key}, \sigma' : \text{Nonce} \rightarrow \text{Nonce}, \text{and } \sigma'' : \text{Rnd} \rightarrow \text{Rnd} \]

Some extensions and issues such as composed/related keys, randomness, and cycles (encrypting a key under itself) are useful in modeling realistic protocols, but would complicate our definitions and theorems. So, we leave them for further work.

2.5 CryptoPAi Semantics: Deduction Rules

Plotkin-style deduction rules for the structural operational semantics \[29\] of CryptoPAi are given in Figure 3.

The transition relation \( \rightarrow \) has exactly the same role and meaning as in the standard notion of Labeled Transition System (LTS). The formula \( s \mathcal{A} \) shows the possibility of termination in state \( s \). The indistinguishability between \( s_0 \) and \( s_1 \) according to principal \( i \) is denoted by expression \( s_0 \not\approx s_1 \). Observing of actions depends on the visibility range of actions. The relation \( d \Rightarrow \subseteq \text{St} \times \text{St} \) has been defined for each parametric decorated action \( d \in D \) where \( \text{St} \) is the set of operational states. The deduction rules for \( d \Rightarrow \) are mostly self-explanatory and standard to most process algebras. One peculiar deduction rule is \((p3)\) which uses the premise \( \sigma = \text{match}(\rightarrow M, M') \) 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 \((I_C)\) move the indistinguishability relation of histories up to operational states. The symmetric rules \((n1)\), \((n3)\), \((p1)\), and \((p4)\) are omitted for brevity. Termination of a process is orthogonal to its past history, so the different meta-variables are used for the histories in
Figure 3. SOS Rules of CryptoPAi.

the premises and the conclusion of rules (n2) and (p2). The transition relation \( \Rightarrow \) and indistinguishability equivalence relation \( \equiv \) are the sets of all closed statements provable using the deduction rules (plus their symmetric versions) from Figure 3. These define an Epistemic Labeled Transition System (ELTS) for each process \( p \), as defined below.

**Definition 8 (Semantics of Processes).** Given the sets \( \text{Act} \) and \( \text{Term} \), an ELTS is a 5-tuple \( \langle St, \rightarrow, \check{}, IC, s_0 \rangle \), where \( \rightarrow \subseteq St \times \text{Act} \times \text{Term}^* \times St \) is the transition relation, \( \check{} \subseteq St \) is the termination predicate, \( IC \subseteq St \times Id \times St \) is the histories-indistinguishability relation, and \( s_0 \) is the initial state.

The semantics of process \( p \) has been defined by the ELTS with pairs of processes and trace histories as states, \( \rightarrow \) as transition relation, \( \check{} \) as termination relation, \( \equiv \) as indistinguishability equivalence relation, and \( (p, \varepsilon) \) as the initial state, where \( \varepsilon \) denotes the empty computation.

An operational state is a pair \( (p, \Gamma) \), where \( p \in \text{Proc} \) is a CryptoPAi process and \( \Gamma \) is a finite computation recording the history of the process executed so far.

## 3 \( E\overline{\mu} \): An Epistemic Mu-Calculus

\( E\overline{\mu} \) is the modal \( \mu \)-calculus extended with past and two epistemic constructs: \( Has \) and \( K \), representing local and epistemic knowledge [17].

The semantics of \( E\overline{\mu} \) has been given on the ELTS [17], thereby connecting the protocol specification and the logical reasoning domain. Epistemic knowledge refers to all facts that a principal can deduce from the particular run of the protocol, set aside all possible runs (also including facts of the form “principal \( i \) knows that principal \( j \) knows the key \( k^j \)”). Now we recall our logic in the following section.

### 3.1 Syntax

The syntax of our logic, called \( E\overline{\mu} \) has the following grammar:
\( \phi ::= \top \mid X \mid \bigwedge_{j \in J} \phi_j \mid \neg \phi \)
\( \mid \langle a(M) \rangle \)
\( \phi \mid \langle a(M) \rangle \)
\( \phi \mid K_i \phi \mid \text{Has}_i(M) \mid \nu X. \phi(X) \)

where \( i \in Td \), \( M \in Term \), and \( a(M) \) is a parameterized action. \( \langle a(M) \rangle \phi \) means that “after some \( a(M) \) transition \( \phi \) holds”; \( \langle a(M) \rangle \phi \) has the same meaning as \( \langle a(M) \rangle \phi \), 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 \( a(M) \)-step to the current state. \( K_i \phi \) indicates that “principal \( i \) knows that \( \phi \) holds”. \( \text{Has}_i(M) \) 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 4, are used for commonly used logical formula: The common knowledge property is a very powerful construction, particularly for protocols where trust is an issue. It has posed a real challenge for specification and verification with the standard operational techniques so far. We have modeled this property by the \( C_J \phi \) operator, expresses that agents in \( J \) not only know that \( \phi \) holds, but also all agents in \( J \) know that \( \phi \) holds, and all agents in \( J \) know that all agents in \( J \) know that \( \phi \) holds, and so on. \( E\overline{P}\)-forms denotes the set of \( E\overline{P} \) formulas. In the following definition, the satisfaction of a formula \( \phi \in E\overline{P}\)-forms in the ELTS \( E \) is interpreted.

**Definition 9 (Satisfaction).** Let \( E \) be an ELTS as \( E = \langle S, \rightarrow, \checkmark, I_C, s_0 \rangle \) and \( s \in S \) be a state of \( E \). The satisfaction relation \( \models \) for formulas \( \phi \in E\overline{P}\)-forms is defined inductively in Figure 5. \( E \models \phi \) if \( s_0 \models \phi \).

For example, the knowledge operator \( K_i \phi \) indicates that \( i \) knows that \( \phi \) if \( \phi \) holds in all states reachable from \( s \) through the \( \checkmark \) relation. The definition of this relation is based on what \( i \) is allowed to observe and relating it to all computations (trace-histories) that are indistinguishable to \( i \).

### 4 Case Study

In the following example, we specify the FOO‘92 e-voting protocol [30], 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 3.** It is due to compare the new combined framework with using exclusively the operational approach or the epistemic one. The FOO‘92 protocol has already been analyzed using both operational [31],[32] and epistemic approaches [33]. 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. To describe the protocol, the following notations are used.

- \( V_i \): identity of the voter \( i \),
- \( A \): identity of the administrator authority,
- \( C \): identity of the counter authority,
- \( \xi(v, k) \): bit-commitment scheme for message \( v \) using key \( k \),
- \( \sigma_i(m) \): \( V_i \)'s signature scheme,
- \( \sigma_A(m) \): \( A \)'s signature scheme,
- \( e = \chi(m, r) \): blinding message \( m \) using random number \( r \), and
- \( v_i \): 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 [30]:

- **Preparation** and **Administration**: at the first stage, the voter aims to get the administrator’s sign on her ballot. The voter sends her blinded committed ballot, which signed by herself, to the administrator for signing. The administrator signs the authorized ballot and returns to
\[ \bigvee_{j \in J} \phi_j \] stands for disjunction defined by \( \neg \bigwedge_{j \in J} \neg \phi_j \).

\[ [a(M)]\phi \] intuitively means that after all \( a(M) \)-transitions, \( \phi \) holds, i.e., \( \neg\langle a(M)\rangle \neg \phi \).

\( \mu X.\phi(X) \) (with \( X \) occurring positively in \( \phi \)) is the least fixed point operator, which is defined by \( \neg\nu X.\neg\phi(\neg X) \) (\( X \) also occurs positively in \( \neg \phi \)). The current state is in the smallest set of states satisfying \( \phi(X) \).

\( \langle \phi \rangle \) stands for \( \bigvee_{a \in Act, M \in Term} (a(M))\phi \), which is by itself an abbreviation for a number of disjunctions (similarly, \( \langle \langle \phi \rangle \rangle \) stands for \( \bigvee_{a \in Act, M \in Term} (a(M))\phi \)). Intuitively, it means that after (before) some transition \( \phi \) holds.

\[ [\phi] \] \( \neg \langle \phi \rangle \) after all transitions \( \pi \) holds

\( ^\pi a(M) \) (similarly, \( a(M) \uparrow \)) is an abbreviation for \( \mu X.\langle a(M)\rangle \top \lor (x).X \) (or \( \mu X.\langle a(M)\rangle \top \lor (\pi).X \)). So, it is possible to reach a state in the future where an \( a(M) \)-transition is possible (go back to a state in the past that results from an \( a(M) \)-transition).

\[ [\ast] \phi \] (similarly, \( \lceil \rangle \phi \)) is an abbreviation for \( \mu X.\phi \lor \langle \rangle X \) (or \( \mu X.\phi \lor \lceil \rangle \phi \)). The intuition behind this abbreviation is that all future paths will (paths in the past) lead to a state, in which there is a state satisfying \( \phi \). (\( [\ast] \phi \) and \( \lceil \rangle \phi \) are defined accordingly.)

\( C_J \phi \) stands for \( \forall X.\circlearrowleft_{j \in J} K_i(X \land \phi) \), meaning: “it is common knowledge among the principals in the set \( J \) that \( \phi \) holds” [6].

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 received ballots.

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

Briefly, after the initial key distribution, the protocol follows these steps, where the message broadcasting is denoted by *:
1. \( V_i \rightarrow A : (V_i, s_i, c_i) \)
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. \( C \rightarrow * : (l, v_i) \)

Model

In the specification of the protocol given in Figure 6, we assume that \( m \in M, n \in N \), \( i \in I, j \in J \), and \( r \in R \) for some finite (non-empty and non-singleton) sets \( M, N, I, J \), and \( R \). \( pk_{A,m} \) and \( sk_{A,m} \) represent \( A \)'s public and private keys, and similarly, \( pk_{V_i,n} \) and \( sk_{V_i,n} \) represent \( V_i \)'s public and private keys; \( c_j \in Cand \) indicates the \( j \)th candidate of candidates list; \( l_r \in L \) indicates the index of result-list made by the authority Counter.

\[
Key = \{pk_{V_i,n}, sk_{V_i,n}, pk_{A,m}, sk_{A,m}, k_V, r_V\} \\
\sum_{m \in M} \sum_{n \in N} (Id)！DistViPubK((pk_{V_i,n}, pk_{A,m})); \\
(Id)！DistViPrivK((sk_{V_i,n})); \\
(Id)！DistAdPubK((pk_{A,m}, pk_{V_i,n})); \\
(Id)！DistAdPrivK((sk_{A,m}, pk_{V_i,n})); \\
(Id)！DistCoPubK(pk_{A,m}) \\
Voter = \\
(Id)！DistViPubK((x_{pk_V}, x_{pk_A})); \\
((V_i)！DistViPrivK((x_{sk_V})); \\
\sum_{c_j \in Cand} (Id)！ViReqAdSign((V_i, \{ \{v_i\} \}_{V_i})); \\
(Id)！AdAckSign((V_i, \{ \{v_i\} \}_{V_i})); \\
(Id)！CoVoteList((x_V, \{x_v\})); \\
((Id)！VRevealKey((x_V, \{x_v\})); \\
(Id)！PublishResult \\
Administrator = \\
(Id)！DistAdPubK((x_{pk_A}, x_{pk_V})); \\
((A)！DistAdPrivK((x_{sk_A})); \\
(Id)！ViReqAdSign((x_V, \{x_{sk_V}, x_{vbc}\})); \\
(Id)！AdAckSign((x_V, \{x_{sk_V}, x_{vbc}\}); \\
Counter = (Id)！DistCoPubK(x_{pk_A}); \\
((Id)！VRevealKey((x_V, \{x_vbc\})); \\
\sum_{l_r \in L} (Id)！CoVoteList((l_r, x_{sk_V}, x_v)); \\
((Id)！VRevealKey((l_r, x_{sk_V})); \\
(Id)！PublishResult \\
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, V_i, E\}.

Regarding the public appearance of actions, we define for all of the identities and \( m \in M \),

Figure 6. Our Model of the FOO’92 E-Voting Protocol.

\( n \in N, v_i \in Cand \), and \( l_r \in L \),

\[ \rho(\text{DistViPrivK}(sk_{V_i,n})) = \tau \]
\[ \rho(\text{DistAdPrivK}(sk_{A,m})) = \tau \]
\[ \rho(\text{ViVoting}(\{v_i\}_{sk_{A,m}}, \{v_i\}_{sk_{V_i}})) = \tau \]
\[ \rho(\text{VRevealKey}(l_r, x_{sk_V})) = \tau \]

The specification of the protocol is the parallel composition of the participating processes:
$\text{FOO} = \text{RegAuthority} || \text{Administrator} || \text{Counter}$

### 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 a voter try to cast her vote, she must have already been authenticated by the authority $A$.

$$[\text{\_\_} \forall v_i \in \text{Cand} \forall m \in \text{M}]$$
$$\text{ViVoting}((\{v_i\}_{k_V}, \{v_i\}_{k_V}))$$
$$\text{AdAckSign}(V_i, \{\{v_i\}_{k_V}\}_{s_{sk_A,m}}).$$

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 before and while they are being collected. Fairness can be expressed by the following formula.

$$[\text{\_\_} \forall l_r \in \text{L} \forall v_i \in \text{C} \forall m \in \text{M}]$$
$$\text{CoVoteList}((l_r, \{\{v_i\}_{k_V}\}_{s_{sk_A,m}}), \{\{v_i\}_{k_V}\}_{s_{sk_A,m}}).$$

### 5 Implementation

In [17], we have formalized the semantics of CryptoPAi 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 [34].

Next, we extend our earlier implementation based on our extension in current paper.

#### 5.1 CryptoPAi in Maude

Let $\text{Act}$, $\text{TAct}$, $\text{TDAct}$ 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).

$\text{tau}$ is the invisible internal action. The syntax of our actions, crypto-terms, and process language are as follows.

- $\text{op \tau :} \rightarrow \text{TDAct [ctor].}$
- $\text{op \_ (_ ) :} \text{Act PAiTerm} \rightarrow \text{TAct [ctor].}$
- $\text{op \text{rcv ( _ ) ( _ ) :} IdSet Act PAiTerm} \rightarrow \text{TDAct [ctor].}$
- $\text{op \text{snd (_ ) _ :} IdSet Act PAiTerm} \rightarrow \text{TDAct [ctor].}$
- $\text{op \text{sync (_ ) _ :} IdSet Act PAiTerm} \rightarrow \text{TDAct [ctor].}$
- $\text{op \rho (_ ) :} \text{TAct} \rightarrow \text{TAct [ctor].}$
- $\text{op ( _ , _ ) :} \text{PAiTerm PAiTerm} \rightarrow \text{PAiTerm [ctor].}$
- $\text{op \text{enc { _ }_ _ :} PAiTerm PAiTerm PAiTerm} \rightarrow \text{PAiTerm [ctor].}$
- $\text{op \text{enca { _ }_ _ :} PAiTerm PAiTerm PAiTerm} \rightarrow \text{PAiTerm [ctor].}$
- $\text{op \text{sign { _ }_ _ :} PAiTerm PAiTerm PAiTerm} \rightarrow \text{PAiTerm [ctor].}$
- $\text{op \text{blind { _ }_ :} PAiTerm PAiTerm} \rightarrow \text{PAiTerm [ctor].}$

The operational semantics (given in Figure 3) has been implemented through translation of the deduction rules. A set of Maude’s conditional rewrite rules have been used to model transitions in the formalization. For example deduction rules (0), (d) and (n0) have been implemented as follows. The auxiliary symbol $\$$ used at the left-hand side of the rewrite rules designates the transition $\Rightarrow$.

- $\text{op \_ \_ :} \text{TDAct Proc} \rightarrow \text{Proc [ctor prec 21].}$
- $\text{op \_ \_ :} \text{Proc Proc} \rightarrow \text{Proc [ctor prec 22].}$
- $\text{op \_ \_ :} \text{Proc Proc} \rightarrow \text{Proc [ctor prec 23].}$
- $\text{op \_ \_ :} \text{Proc Proc} \rightarrow \text{Proc [ctor prec 24].}$

The operational semantics (given in Figure 3) has been implemented through translation of the deduction rules. A set of Maude’s conditional rewrite rules have been used to model transitions in the formalization. For example deduction rules (0), (d) and (n0) have been implemented as follows. The auxiliary symbol $\$$ used at the left-hand side of the rewrite rules designates the transition $\Rightarrow$.

- $\text{var \_ \_ :} \text{TDAct.}$
- $\text{vars p0 p1 pp1 :} \text{Proc.}$
- $\text{vars ga gap :} \text{Hist.}$

...$\text{***(0) and other terminating rules}$

- $\text{eq (tick NIL) = true.}$
- $\text{eq (tick d) = false.}$
- $\text{eq (tick (p0 + p1)) = ((tick p0) or (tick p1)).}$
finally, the following equalities, represent deduction rules (= ref1) and (= pat) in Figure 3. in fact, they formalize the introduced indistinguishability concept in maude. ded operator has been defined to check the deducibility of a term from a given term-set. pat, iCPattern, and iTerm implement the introduced functions pat (in section 2.1), CPattern and Term (in section 2.4), respectively. Regarding Definition 4, the function Sigma implements type-preserving bijection σ of Key ∪ Nonce ∪ Rand (i.e., σ: Key → Key, Nonce → Nonce, Rand → Rand). ...

Finally, the following equalities, represent deduction rules (= ref1) and (= pat) in Figure 3. In fact, they formalize the introduced indistinguishability concept in Maude. ded operator has been defined to check the deducibility of a term from a given term-set. pat, iCPattern, and iTerm implement the introduced functions pat (in Section 2.1), CPattern and Term (in Section 2.4), respectively. Regarding Definition 4, the function Sigma implements type-preserving bijection σ of Key ∪ Nonce ∪ Rand (i.e., σ: Key → Key, Nonce → Nonce, Rand → Rand).

vars P Pp Ppp : PAiPair.
op Box( , ) : PAiTerm → Pattern [ctor].
op _ ded _ : PAiTerm PAiTerm → Bool .
op pat ( , , ) : PAiTerm PAiTerm → Pattern [ctor].
op iTerm( , , ) : Id TDAct → PAiTerm [ctor].
op iTermSet( , , ) : Id History → PAiPair [ctor].
op iAppear ( , , , ) : Id TDAct → TAct [ctor].
op iCPattern( , , , ) : Id History History → PHistory [ctor].

... *** Term^i and TermSet^i Functions ***
eq iTerm(i, snd(I) ac( α(P)) ) = empty if not (i in I) and rho(ac( P)) = bc(Pp).
ceq iAppear(i, snd(I) ac( P) ) = bc(Pp).

... *** CPattern^i Function ***
ceq iCPattern(i,h,((prc: tda) ^ hp)) = (ac (pat(P,iTermSet(i,h)))

... *** Indistinguishability Rel. ***

5.2 Eμ in Maude

The formalization defined in [17] for the syntax of the logic defined in Section 3 is used without any changes in this paper.

5.3 Modeling the FOO Protocol in Maude

In this section we briefly present the modeling of Example 4 in the Maude implementation. 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 computation.
- Rew s0 (p,niks) = phi : is used to verify an Emu formula phi on a process p (with the empty computation), in a state space reachable from the initial state s0.

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))) (v;i;E)
Sum ((pkA,skA) :
((pkA1,skA1),(pkA2,skA2))) (v;i;A;C;E)
TakeViPubK ((pki,pkA));
(and (v;i) TakeViPrivK (ski));
(and (v;i,A;C;E) TakeAdPubK ((pkA,pki)));
(and (A) TakeAdPrivK (skA));
(and (v;i,A;C;E) TakeCoPubK (pkA))).

eq Admin = ((((rcv (v;i,A;C;E) TakeAdPubK ((xpka,xpki)));
(rcv (A) TakeAdPrivK (xskA)));
(rcv (v;i,A;C;E) ViReqAdSign ((xVi,(xsvbc , xsvbc))).manage);
(and (v;i,A;C;E) AdAckSign
((xVi,(xsvbc,xsvbc,skA)))).

eq Voteri = Sum ((vi : (vi1 , vi2)) (v;i;A;C;E)
 (((((rcv (v;i,A;C;E) TakeViPubK ((xpki , xpki)));
(rcv (v;i) TakeViPrivK (xski)));
(and (v;i,A;C;E) ViReqAdSign
((v;i,(sign (blind (enc (v;i) ki )))ri));
(blind (enc (v;i) ki )))ri));
(rcv (v;i,A;C;E) AdAckSign
((v;i,(blind (xsvbc,xsvbc)ki))));
(and (v;i) ViVoting ((xsc , (enc(v;i)ki)));
(rcv (v;i,A;C;E) CoVoteList ((x1, (xsc , (enc(v;i)ki)));
(and (v;i,A;C;E) ViRevealKey ((x1 , ki)));
(rcv (v;i,A;C;E) PublishResult (empty))).

eq Counter = Sum ((l : (l1 , l2)) (v;i;A;C;E)
 (((((rcv (v;i,A;C;E) TakeCoPubK (xpka));
(rcv (v;i) ViVoting ((xsc , xc))));
(and (v;i,A;C;E) CoVoteList ((l , (xsc , xc)));
(rcv (v;i) ViRevealKey ((l , xki)));
(and (v;i,A;C;E) PublishResult (empty)))).

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 (6156315ms real)
(3 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 |=
[[*][ViRevealKey(l1,ki)]].
rewrites: 6929255 in
1628036047000ms cpu (76864ms real)
(0 rewrites/second)
result Satform: * true

- rew FOOState0 (FOOProtocol , niks) |=
[[*][ViRevealKey(l1,ki)]].
rewrite in EmuBasic :
FOOState0 FOOProtocol , niks |=
[[*][ViRevealKey(l1,ki)](Vi Has sk1)
emor (Vi Has sk2)].
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 |=
  [...]([ViRevealKey(l1,ki)]
  (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 |=
  [...]([ViRevealKey(l1,ki)]
  ((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)) <-|).
rewrite in
  EmuBasic: FOOState0 FOOProtocol, niks |=
  [...]([ViVoting(sign{enc{vi1}ki}skA1, enc{vi1}ki})
  (AdAckSign(Vi, sign{blind{enc{vi1}ki}ri}skA1)) <-|).
rewrites: 888839
  in 1628036047000ms cpu (56553ms real)
  (0 rewrites/second)
result Satform: * true
- rew FOOState0 (FOOProtocol, niks) |=
  [...]([ViVoting(sign{enc{vi2}ki}skA2, enc{vi2}ki})
  (AdAckSign(Vi, sign{blind{enc{vi2}ki}ri}skA2)) <-|).
rewrite in
  EmuBasic: FOOState0 FOOProtocol, niks |=
  [...]([ViVoting(sign{enc{vi2}ki}skA2, enc{vi2}ki})
  (AdAckSign(Vi, sign{blind{enc{vi2}ki}ri}skA2)) <-|).
rewrites: 888839
  in 1628036047000ms cpu (55946ms real)
  (0 rewrites/second)
result Satform: * true

6 Conclusions
In this paper, we presented an extension of our hybrid operational-epistemic framework, introduced in [17], for specification and analysis of security protocols with genuine support for cryptographic constructs. The framework includes a process algebraic formalism for the operational specification and an epistemic extension of modal µ-calculus with past for the property specification. The extended framework has been further re-formalized and re-implemented in the rewriting logic of Maude.

The main practical motivation for this paper came from the domain of e-voting protocols. In fact, we extended CryptoPAi framework with more cryptographic constructs and then investigated its applicability in this domain through analysis of the FOO e-voting protocol as a case study. We plan to further investigate the applicability of our framework in this domain by applying it to more case studies (see [35] for some ideas in this direction). Extending our framework with probabilities, along the lines of [15], is another avenue for future research.

References


[27] P.J. Broadfoot. *Data Independence in the
An Extension of CryptoPAi to the Formal Analysis of E-voting Protocols — H. R. Mahrooghi and R. Jalili


Hamid Reza Mahrooghi received his bachelor’s degree in Computer Engineering from Isfahan University of Technology in 1993, and his master’s degree in Computer Engineering from Sharif University of Technology in 1996. He received his Ph.D. in Computer Engineering from Sharif University of Technology in 2013. He joined the Group of Computer Engineering, Imamreza University, Mashhad, Iran, in 2004. His research interests include electronic-voting, mobile & network security, block-chain-which he conducts at his network security laboratory in Imamreza University.

Rasool Jalili received his bachelor’s degree in Computer Science from Ferdowsi University of Mashhad in 1985, and his master’s degree in Computer Engineering from Sharif University of Technology in 1989. He received his Ph.D. in Computer Science from The University of Sydney, Australia, in 1995. He then joined the Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, in 1995. He has published more than 150 papers in Computer Security and Pervasive Computing in international journals and conferences proceedings. He is now an associate professor in Sharif. His research interests include access control, vulnerability analysis, and database security-which he conducts at his network security laboratory (DNSL, dnsl.sharif.ir).