IOS Press

Comparing BDD and SAT based techniques for model checking Chaum's Dining Cryptographers Protocol

Magdalena Kacprzak

Białystok University of Technology, email: mdkacprzak@wp.pl

Alessio Lomuscio [*]

University College London, UK, email: a.lomuscio@cs.ucl.ac.uk

Artur Niewiadomski

University of Podlasie, email: artur@iis.ap.siedlce.pl

Wojciech Penczek [†]

Institute of Computer Science, PAS and University of Podlasie, email: penczek@ipipan.waw.pl

Franco Raimondi[*]

University College London, UK, email: f.raimondi@cs.ucl.ac.uk

Maciej Szreter

Institute of Computer Science, PAS, email: mszreter@ipipan.waw.pl

Abstract. We analyse different versions of the dining cryptographers protocol by means of automatic verification via model checking. Specifically we model the protocol in terms of a network of communicating automata and verify that the protocol meets the anonymity requirements specified. Two different model checking techniques (ordered binary decision diagrams and SAT-based bounded model checking) are evaluated and compared to verify the protocols.

[*] The authors acknowledge support from EPSRC (grants CN04/04 and GR/S49353/01) and the Royal Society.

[†] The authors acknowledge support from the Ministry of Science and Information Society Technologies under grant number 3T11C01128 and the Royal Society.

1. Introduction

A key interest in the analysis of security protocols concerns being able to verify formally and automatically that a protocol meets its intended specifications. This approach differs from the other main stream of research in computer security, namely cryptanalysis, in that it assumes perfect (i.e., unbreakable) cryptographic algorithms and focuses on the properties that a protocol achieves. Formal analysis of security protocols has permitted to find bugs in a variety of security protocols, including the Wide Mouthed Frog [14].

The technique of model checking [5] has recently been used with considerable success [3, 1, 21] to verify properties such as authentication, integrity, secrecy, anonymity, etc., of particular security protocols. Typically, security protocols are analysed in terms of reachability and, occasionally, in terms of full temporal logic. While this is adequate in many instances, the validation of particular properties, such as anonymity, benefit from richer approaches. In particular, the protocol of the dining cryptographers [4] has been successfully analysed [15, 20] by considering a temporal and epistemic language. In this paper we intend to make two contributions in this line. First, we suggest an alternative, often more efficient, formalisation of the dining cryptographer in terms of a network of automata. Second, we compare experimental results for variants of this protocol when analysed by two different model checking techniques: ordered binary decision diagrams and SAT-based bounded model checking. This comparison offers results giving guidance in terms of model checking technology for the security protocol under consideration but also more in general for any verification problem via model checking.

From a technical point of view, we will be working on networks of communicating automata to model the protocol (Section 2). These will generate a branching time semantics on which temporal, epistemic, and correctness modal operators will be interpreted [12], as in Section 3. This syntax will be used as specification language for the properties to be checked in Section 5, by means of VerICS [6] and MCMAS [18], two model checkers for deontic interpreted systems [12], based respectively on bounded model checking (BMC) and ordered-binary decision diagrams (OBDD), techniques briefly summarised in Section 4.

2. Modelling protocols

This paper is concerned with the verification of the protocol of the Dining Cryptographers. This protocol has been modelled in the past by R. van der Meyden and K. Su in [15], using the class of synchronous interpreted systems [9] with perfect recall. In [20], the encoding of the protocol has been extended to deontic interpreted systems [12] to reason about correct behaviour.

In the present work we employ an automata-based approach. Specifically, we interpret formulae of a logic to reason about time, knowledge, and correct behaviour on traces generated by networks of automata. Our choice is motivated by two reasons:

1. We show that an automata-based approach provides a more efficient (i.e., faster) framework for the verification of protocols, as it can be seen by comparing the results in Section 5.3 with the results presented in [20].

2. It provides a common ground for the comparison of experimental results obtained using different model checkers.

Notice that there are formulae that are satisfied in this alternative encoding but not in the former one, especially when reasoning about the temporal evolution of automata. Nevertheless, the *key epistemic properties of the protocol* are satisfied in both the encodings. A formal proof of the above statement is not presented here as it is not essential for our task.

Formally, we proceed as follows. Since our aim is to analyse variants of the dining cryptographers protocols where some participants may cheat, in line with [12, 20], we define a notion of *deontic* automata by colouring the states as either *red* or *green*; we refer to [12] and related papers for an exploration of these concepts. We assume that each agent in the system is formalised by considering a number of automata.

Definition 2.1. (Deontic automaton)
A deontic automaton is a four-tuple $\mathcal{A} = (Act, L, s^0, T)$, where

- Act is a finite set of actions,
- $L = L^{\mathcal{G}} \cup L^{\mathcal{R}}$ is a finite set of states, which is divided into two disjoint sets of green $L^{\mathcal{G}}$ and red $L^{\mathcal{R}}$ states,
- $s^0 \in L$ is the initial state,
- $T \subseteq L \times Act \times L$ is a transition relation.

A set of automata, called a *network of automata*, can be composed into the *product automaton* by a standard multi-synchronisation approach: the transitions that do not correspond to a shared action are interleaved, whereas the transitions labelled with a shared action are synchronised. A synchronised transition is enabled if it is enabled in all the synchronising automata.

Definition 2.2. (Product automaton)
Given a network $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ of deontic automata, where $\mathcal{A}_i = (Act_i, L_i, s_i^0, T_i)$ for $1 \leq i \leq n$, the *product automaton* of the network is defined as a four-tuple $\mathcal{A} = (Act, G, s^0, T)$, where

- $Act = \bigcup_{i=1}^{n} Act_i$ is a finite set of actions,
- $G = G^{\mathcal{G}} \cup G^{\mathcal{R}}$ is a finite set of global states composed by two disjoint sets of green $G^{\mathcal{G}} = L_1^{\mathcal{G}} \times \cdots \times L_n^{\mathcal{G}}$ and red $G^{\mathcal{R}} = (L_1 \times \cdots \times L_n) \backslash G^{\mathcal{G}}$ states,
- $s^0 = (s_1^0, \ldots, s_n^0) \in G$ is the initial state,
- $T \subseteq G \times Act \times G$ is a transition relation such that $((l_1, \ldots, l_n), a, (l'_1, \ldots, l'_n)) \in T$ iff $(\forall i \in \mathcal{A}(a))$ $(l_i, a, l'_i) \in T_i$ and $(\forall i \in \{1, \ldots, n\} \backslash \mathcal{A}(a))$ $l_i = l'_i$, where $\mathcal{A}(a) = \{1 \leq i \leq n \mid a \in Act_i\}$.

Observe that every automaton can perform a local action as soon as it is allowed. However, as we have mentioned before, some of local actions are synchronised and must be executed together. The synchronisation of automata is done via shared labels of transitions. Moreover, two or more unsynchronised actions cannot be realized at the same time. So, we explore the interleaving model of computation.

A global state is coloured green if it consists of green local states only. All other global states are coloured red. Moreover, a global state $(l'_1, \ldots, l'_n)$ is the result of executing the action a at a global state

$(l_1, \ldots, l_n)$ iff for every automaton $\mathcal{A}_i$ whose set of actions contains a, we have $(l_i, a, l'_i) \in T_i$, and for all the remaining automata we have $l'_j = l_j$.

In what follows we denote with $Ix(\mathcal{A})$ a set of the indices of the automata of $\mathcal{A}$, i.e., the set $\{1, \ldots, n\}$.

The product automaton extended with a labelling function is used as a model for interpreting our specification language, but it is not built explicitly for verification purposes as we use symbolic approaches.

In order to reason about multi-agent systems we assume that automata of the network represent agents. However, we do not require a one to one correspondence between automata and agents. Instead, we assume that the behaviour of an agent can be modelled by several automata of the network or, equivalently, by the product automaton of these automata.

Now, let $\mathcal{A}gt = \{1, \ldots, k\}$ be a set of indices of agents. We define a function $Obs : \mathcal{A}gt \to 2^{Ix(\mathcal{A})}$, which assigns to each agent the indices of the automata of $\mathcal{A}$ that are assumed to represent its behaviour.

Let $loc_i : G \to \prod_{j \in Obs(i)} L_j$, for $i = 1, \ldots, k$, be a function which for each global state s of G returns the local state of s for the agent i, i.e., projects s on the components of $Obs(i)$. Notice that a single automaton of $\mathcal{A}$ may be used for representing a part of the behaviour of several agents. Intuitively, this means that agents can observe and change the same fragments of a world. In such a case, to avoid the agents losing their autonomy, we could require that they are represented by duplicated automata whose appropriate actions are synchronised; we do not insist on this for efficiency reasons.

Interpreted systems [9] are commonly used to interpret temporal and epistemic modalities. Their deontic extensions incorporate the idea of a correct functioning behaviour of some or all the components of systems examined. Following these ideas we define the notion of a model.

Definition 2.3. (Model)
Let $\mathcal{A}gt = \{1, \ldots, k\}$ be a set of indices of agents, $\mathcal{A} = (Act, G, s^0, T)$ be a product automaton. A *(deontic) model* is a tuple $\mathrm{M} = (G, W, s^0, TR, \sim, \sim^O, Obs, \mathcal{V})$, where:

- W is a set of *reachable global states* from s^0, i.e., $W = \{s \in G \mid (s^0, s) \in TR^*\}$[1],
- $TR \subseteq G \times G$ is a binary relation on G such that $(s, s') \in TR$ iff there exists $a \in Act$ such that $(s, a, s') \in T$,
- $Obs : \mathcal{A}gt \to 2^{Ix(\mathcal{A})}$ is a function that assigns a nonempty set of indices of automata to every agent,
- $\sim = \{\sim_i\}_{1 \le i \le k}$, where $\sim_i \subseteq W \times W$ is an *epistemic accessibility relation* for each agent i $(1 \le i \le k)$ defined by: $s \sim_i s'$ iff $loc_i(s') = loc_i(s)$,
- $\sim^O = \{\sim^O_i\}_{1 \le i \le k}$, where $\sim^O_i \subseteq W \times W$ is a *deontic accessibility relation* for each agent i $(1 \le i \le k)$ defined by: $s \sim^O_i s'$ iff $l_{i_j} \in L^{\mathcal{G}}_{i_j}$ for every l_{i_j} of $loc_i(s') = (l_{i_1}, \ldots, l_{i_t})$,
- $\mathcal{V} : G \longrightarrow 2^{\mathcal{PV}}$ is a *valuation function* for a set of propositional variables $\mathcal{PV}$ such that $true \in \mathcal{V}(s)$ for all $s \in G$. $\mathcal{V}$ assigns to each state a set of propositional variables that are assumed to be true at that state.

[1] TR^* denotes the reflexive and transitive closure of TR.

Epistemic relations. Let $\Gamma \subseteq \mathcal{A}gt$. The union of Γ's accessibility relations is defined as $\sim^E_\Gamma = \bigcup_{i \in \Gamma} \sim_i$. By $\sim^C_\Gamma$ we denote the transitive closure of $\sim^E_\Gamma$, whereas $\sim^D_\Gamma = \bigcap_{i \in \Gamma} \sim_i$. The above relations are used to give semantics to the "everyone knows", "common knowledge", and "distributed knowledge" modalities of the logic presented in [9].

Computations paths. A *computation* in M is a maximal sequence $\pi = (s_0, s_1, \ldots)$ of states such that $(s_i, s_{i+1}) \in TR$ for each $i < |\pi|$, where $|\pi|$ denotes the length of π defined as $|\pi| = \infty$ if π is infinite and $|\pi| = k + 1$ if s_k is the last state of π.

A *k-computation* is a prefix of length k of a computation. For a computation $\pi = (s_0, s_1, \ldots)$, let $\pi(k) = s_k$, and $\pi_k = (s_0, \ldots, s_k)$, for each $k \in \mathbb{N}$. By $\Pi(s)$ we denote a set of all computations starting at s in M, whereas by $\Pi_k(s)$ a set of all the k-computations starting at s. Moreover, let $\mathbb{N}_+ = \mathbb{N} \backslash \{0\}$.

3. The Logic CTLKD

We use the logic CTLKD to express properties of protocols. This logic is an extension of *Computational Tree Logic* (CTL) [8], introduced by Emerson and Clarke, enriched with standard epistemic operators [9] and correctness operators [12]. This language enables us to represent temporal flows of time, knowledge of the agents, and what properties hold following the correct execution of prescribed behaviour. We refer to [13] for a detailed example on the use of this formalism.

Definition 3.1. (Syntax of CTLKD)
The set of CTLKD formulae $\mathcal{FORM}$ is defined as follows:

$$\alpha ::= p \mid \neg\alpha \mid \alpha \vee \alpha \mid \mathrm{EX}\alpha \mid \mathrm{EG}\alpha \mid \mathrm{E}(\alpha \mathrm{U} \alpha) \mid \mathcal{P}_i\alpha \mid \overline{\mathrm{K}}^j_i\alpha \mid \overline{\mathrm{K}}_i\alpha \mid \overline{\mathrm{D}}_\Gamma\alpha \mid \overline{\mathcal{C}}_\Gamma\alpha \mid \overline{\mathrm{E}}_\Gamma\alpha$$

where $p \in \mathcal{PV}$, $i, j \in \{1, \ldots, k\}$, and $\Gamma \subseteq \{1, \ldots, k\}$.

Other modalities are derived as follows:

- $\mathrm{EF}\alpha \overset{def}{=} \mathrm{E}(true\mathrm{U}\alpha)$, $\mathrm{AF}\alpha \overset{def}{=} \neg\mathrm{EG}\neg\alpha$, $\mathrm{A}(\alpha\mathrm{R}\beta) \overset{def}{=} \neg\mathrm{E}(\neg\alpha\mathrm{U}\neg\beta)$,
- $\mathrm{AX}\alpha \overset{def}{=} \neg\mathrm{EX}\neg\alpha$, $\mathcal{O}_i\alpha \overset{def}{=} \neg\mathcal{P}_i\neg\alpha$, $\hat{\mathrm{K}}^j_i\alpha \overset{def}{=} \neg\overline{\mathrm{K}}^j_i\neg\alpha$, $\mathrm{K}_i\alpha \overset{def}{=} \neg\overline{\mathrm{K}}_i\neg\alpha$,
- $\mathrm{D}_\Gamma\alpha \overset{def}{=} \neg\overline{\mathrm{D}}_\Gamma\neg\alpha$, $\mathcal{C}_\Gamma\alpha \overset{def}{=} \neg\overline{\mathcal{C}}_\Gamma\neg\alpha$, $\mathrm{E}_\Gamma\alpha \overset{def}{=} \neg\overline{\mathrm{E}}_\Gamma\neg\alpha$.

The remaining boolean connectives are defined in the standard way. Moreover, $false \overset{def}{=} \neg true$. The formula $\mathcal{P}_i\alpha$ stands for "there exists a state where agent i is functioning correctly and α holds". As customary the operators X, G stand for "at the next step", and "forever in the future" respectively. The *Until* operator U, precisely $\alpha\mathrm{U}\beta$, expresses that β occurs eventually and α holds continuously until then. The operators K_i, D_Γ, and $\mathcal{C}_\Gamma$ denote knowledge of the agent i, distributed knowledge, common knowledge, and "everyone knows" knowledge of the group Γ resp. A formula $\mathcal{O}_i\alpha$ represents the fact that following all correct executions of agent i α holds. Moreover, the operator$\hat{\mathrm{K}}^j_i$ expresses knowledge the agent i has on the assumption that the agent j is functioning correctly. A formal interpretation of these formulae is given below.

Definition 3.2. (Interpretation of CTLKD**)**
Let $\mathrm{M} = (G, W, s^0, TR, \sim, \sim^O, Obs, \mathcal{V})$ be a model, $s \in W$ a state, π a computation, and α, β formulae of CTLKD. $\mathrm{M}, s \models \alpha$ denotes that α is true at the state s in the model M. M is omitted, if it is implicitly understood. The relation $\models$ is defined inductively as follows:

$$\begin{array}{lll}
s \models \mathrm{EX}\alpha & \text{iff} & \exists \pi \in \Pi(s)\ \pi(1) \models \alpha, \\
s \models \mathrm{EG}\alpha & \text{iff} & \exists \pi \in \Pi(s)\ \forall_{0\, \leq\, m\, <\, |\pi|}\ \pi(m) \models \alpha, \\
s \models \mathrm{E}(\alpha \mathrm{U} \beta) & \text{iff} & \exists \pi \in \Pi(s)\ (\exists_{0\, \leq\, m\, < |\pi|}\ [\pi(m) \models \beta \ and\ \forall_{j<m}\ \pi(j) \models \alpha]), \\
s \models \mathcal{P}_i \alpha & \text{iff} & \exists s' \in W\ (s \sim_i^O\ s' \ and\ s' \models \alpha), \\
s \models \overline{\mathrm{K}}_i^j \alpha & \text{iff} & \exists s' \in W\ (s \sim_i\ s' \ and\ s \sim_j^O\ s' \ and\ s' \models \alpha), \\
s \models \overline{\mathrm{K}}_i \alpha & \text{iff} & \exists s' \in W\ (s \sim_i\ s' \ and\ s' \models \alpha), \\
s \models \overline{\mathrm{D}}_\Gamma \alpha & \text{iff} & \exists s' \in W\ (s \sim_\Gamma^D\ s' \ and\ s' \models \alpha), \\
s \models \overline{\mathrm{E}}_\Gamma \alpha & \text{iff} & \exists s' \in W\ (s \sim_\Gamma^E\ s' \ and\ s' \models \alpha), \\
s \models \overline{\mathcal{C}}_\Gamma \alpha & \text{iff} & \exists s' \in W\ (s \sim_\Gamma^C\ s' \ and\ s' \models \alpha).
\end{array}$$

For propositions and the boolean connectives the relation $\models$ is defined in the standard manner.

Definition 3.3. (Validity) A CTLKD formula φ is valid in M (denoted $\mathrm{M} \models\ \varphi$) iff $\mathrm{M}, s^0 \models \varphi$, i.e., φ is true at the initial state of the model M.

The logic ECTLKD is the existential restriction of CTLKD such that the negation can be applied only to elements of $\mathcal{PV}$, i.e., $\neg\alpha$ is replaced by $\neg p$ in the Definition 3.1. The logic ACTLKD is the universal restriction of CTLKD such that its language is defined as $\{\neg\varphi \mid \varphi \in \mathrm{ECTLKD}\}$.

4. Methods of verification of CTLKD

In this section we present two symbolic methods of verification of properties of systems and protocols. The first one uses SAT techniques while the second is based on ordered binary decision diagrams (OBDDs).

4.1. Bounded Model Checking

Bounded Model Checking (BMC) was originally introduced for verification of the existential fragment of the logic CTL [17], and then extended to ECTLK [16] and further to ECTLKD [22]. BMC is based on the observation that some properties of a system can be checked over a part of its model only. In the simplest case of reachability analysis, the approach consists in an iterative encoding of a finite symbolic computation as a propositional formula. The satisfiability of the resulting propositional formula is then checked using an external SAT-solver. We present here the main definitions of BMC for ECTLKD, but refer the reader to the literature cited above for more details. In order to restrict the semantics to a part of the model we define k-models.

Definition 4.1. ($k-$model)
Let $M = (G, W, s^0, TR, \sim, \sim^O, Obs, \mathcal{V})$ be a model and $k \in \mathbb{N}_+$. The $k-$model for M is defined as a structure $M_k = (W, s^0, P_k, \sim, \sim^O, Obs, \mathcal{V})$, where P_k is the set of all the k-computations of M over W, i.e., $P_k = \bigcup_{s \in W} \Pi_k(s)$.

We define the function $loop : P_k \rightarrow 2^{\mathbb{N}}$ as: $loop(\pi) \;=\; \{l \;\mid\; 0 \leq l \leq k \; and \; (\pi(k),\; \pi(l)) \in TR\}$, which returns the set of indices l of π for which there is a transition from $\pi(k)$ to $\pi(l)$.

Definition 4.2. (Bounded semantics)
Let M_k be a $k-$model and α, β be ECTLKD formulae. $M_k, s \models \alpha$ denotes that α is true at the state s of M_k. M_k is omitted if it is clear from the context. The relation $\models$ for modal operators is defined inductively as follows:

$s \models \mathrm{EX}\alpha$ iff $(\exists \pi \in P_k(s))\; \pi(1) \models \alpha$,
$s \models \mathrm{EG}\alpha$ iff $(\exists \pi \in P_k(s))(\forall 0 \leq j \leq k)(\pi(j) \models \alpha$ and $loop(\pi) \neq \emptyset)$,
$s \models \mathrm{E}(\alpha \mathrm{U} \beta)$ iff $(\exists \pi \in P_k(s))(\exists 0 \leq j \leq k)\big(\pi(j) \models \beta$ and $(\forall 0 \leq i < j)\pi(i) \models \alpha\big)$,
$s \models \overline{\mathrm{K}}_i^l \alpha$ iff $(\exists \pi \in P_k(s^0))(\exists 0 \leq j \leq k)\big(\pi(j) \models \alpha$ and $s \sim_i \pi(j)$ and $s \sim_l^O \pi(j)\big)$,
$s \models Y\alpha$ iff $(\exists \pi \in P_k(s^0))(\exists 0 \leq j \leq k)\big(\pi(j) \models \alpha$ and $s \sim \pi(j)\big)$,
where $Y \in \{\mathcal{P}_i, \overline{\mathrm{K}}_i, \overline{\mathrm{D}}_\Gamma, \overline{\mathrm{E}}_\Gamma, \overline{\mathcal{C}}_\Gamma\}$ and $\sim \in \{\sim_i^O, \sim_i, \sim_\Gamma^D, \sim_\Gamma^E, \sim_\Gamma^C\}$ resp.

Model checking over models can be reduced to model checking over k-models. The main idea of BMC for ECTLKD is that we can check φ over M_k by checking the satisfiability of a propositional formula $[M, \varphi]_k = [M^{\varphi, s^0}]_k \wedge [\varphi]_{M_k}$, where the first conjunct represents (a part of) the model under consideration and the second a number of constraints that must be satisfied on M_k for φ to be satisfied. Once this translation is defined, checking satisfiability of an ECTLKD formula can be done by means of a SAT-checker. Typically, we start with $k := 1$, test satisfiability for the translation, and increase k by one until either $[M^{\varphi, s^0}]_k \wedge [\varphi]_{M_k}$ becomes satisfiable, or k reaches the maximal depth of M[2].

We provide here some details of the translation. We begin with the encoding of the transitions in the system under consideration. We assume $L_i = L_i^{\mathcal{G}} \cup L_i^{\mathcal{R}} \subseteq \{0,1\}^{k_i}$, where $k_i = \lceil \log_2(|L_i|) \rceil$ and we take $k_1 + \ldots + k_n = m$. Moreover, let Ix_i be an $<$-ordered set of the indices of the bits of the local states of each agent i of the global states, i.e., $Ix_1 = \{1, \ldots, k_1\}, \ldots, Ix_n = \{m - k_n + 1, \ldots, m\}$. Then, each global state $s = (s_1, \ldots, s_m)$ can be represented by $w = (w[1], \ldots, w[m])$ (which we shall call a *global state variable*), where each $w[i]$ for $i = 1, \ldots, m$ is a propositional variable. A sequence $w_{0,j}, \ldots, w_{k,j}$ of global state variables is called a symbolic k-computation j.

The propositional formula $[M^{\varphi, s^0}]_k$, representing the k-computations in the k-model is defined as follows:

$$[M^{\varphi, s^0}]_k := I_{s^0}(w_{0,0}) \wedge \bigwedge_{j=1}^{f_k(\varphi)} \bigwedge_{i=0}^{k-1} TR(w_{i,j}, w_{i+1,j}),$$

where $w_{0,0}$, and $w_{i,j}$ for $0 \leq i \leq k$ and $1 \leq j \leq f_k(\varphi)$ are global state variables. $[M^{\varphi, s^0}]_k$ encodes the initial state s^0 by $w_{0,0}$ and constrains the $f_k(\varphi)$[3] symbolic k-computions to be valid k-computations in M_k.

The next step of the algorithm consists in translating an ECTLKD formula φ into a propositional formula. Let w, v be global state variables. We need the following propositional formulae for the encoding:

[2] The upper approximation is $|W|$.

[3] The function f_k determines the number of k-computations sufficient for checking an ECTLKD formula, see [22] for more details.

- $p(w)$ encodes a proposition p of ECTLKD.
- $H(w,v)$ represents logical equivalence between global state encodings
(i.e., representing the same global state).
- $HP_i(w,v)$ encodes the set all global states in which agent i is running correctly.
- $HK_i(w,v)$ represents logical equivalence between i-local state encodings,
(i.e., representing the same i-local state).
- $TR(w,v)$ is a formula encoding the transition relation TR.
- $L_{k,j}(l)$ encodes a backward loop connecting the k-th state to the l-th state in the symbolic $k-$computation j, for $0 \leq l \leq k$.

The translation of φ at state $w_{m,n}$ into the propositional formula $[\varphi]_k^{[m,n]}$ is as follows (we give the translation of selected formulas only):

$$\begin{array}{lll}
[\mathrm{EX}\alpha]_k^{[m,n]} & := & \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge \ [\alpha]_k^{[1,i]} \Big), \\
[\mathrm{EG}\alpha]_k^{[m,n]} & := & \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge \ (\bigvee_{l=0}^{k} L_{k,i}(l)) \ \wedge \ \bigwedge_{j=0}^{k} [\alpha]_k^{[j,i]} \Big), \\
[\mathrm{E}(\alpha \mathrm{U} \beta)]_k^{[m,n]} & := & \bigvee_{i=1}^{f_k(\varphi)} \Big(H(w_{m,n}, w_{0,i}) \ \wedge \ \bigvee_{j=0}^{k} \big([\beta]_k^{[j,i]} \ \wedge \ \bigwedge_{t=0}^{j-1} [\alpha]_k^{[t,i]} \big) \Big), \\
[\mathcal{P}_l \alpha]_k^{[m,n]} & := & \bigvee_{i=1}^{f_k(\varphi)} \Big(I_{s^0}(w_{0,i}) \ \wedge \ \bigvee_{j=0}^{k} \big([\alpha]_k^{[j,i]} \ \wedge \ HP_l(w_{m,n}, w_{j,i}) \big) \Big), \\
[\overline{\mathrm{K}}_l^t \alpha]_k^{[m,n]} & := & \bigvee_{i=1}^{f_k(\varphi)} \Big(I_{s^0}(w_{0,i}) \ \wedge \ \bigvee_{j=0}^{k} \big([\alpha]_k^{[j,i]} \ \wedge \ HK_l(w_{m,n}, w_{j,i}) \ \wedge \\
 & & HP_t(w_{m,n}, w_{j,i}) \big) \Big), \\
[\overline{\mathrm{K}}_l \alpha]_k^{[m,n]} & := & \bigvee_{i=1}^{f_k(\varphi)} \Big(I_{s^0}(w_{0,i}) \ \wedge \ \bigvee_{j=0}^{k} \big([\alpha]_k^{[j,i]} \ \wedge \ HK_l(w_{m,n}, w_{j,i}) \big) \Big).
\end{array}$$

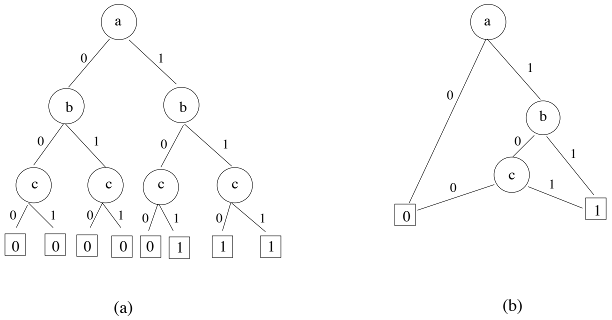
Given the translations above, we can now check φ over M_k by checking the satisfiability of the propositional formula $[M^{\varphi,s^0}]_k \wedge [\varphi]_{M_k}$, where $[\varphi]_{M_k} = [\varphi]_k^{[0,0]}$. The translation above is shown in [22] to be correct and complete.

4.2. Verification via OBDDs

OBDDs are an efficient representation for the manipulation of boolean functions. As an example, consider the boolean function $a \wedge (b \vee c)$. The truth table of this function would be 8 lines long. Equivalently, one can evaluate the truth value of this function by representing the function as a directed graph, as exemplified on Figure 1 (a). As it is clear from the picture, under certain assumptions, this graph can be simplified into the graph pictured on Figure 1 (b). This "reduced" representation is called the OBDD of the boolean function. Besides offering a compact representation of boolean functions, OBDDs of different functions can be composed efficiently. In [2] algorithms are provided for the manipulation and composition of OBDDs.

OBDDs are used in the verification of the model checking of systems specified by means of formulae of CTL, a logic used to reason about branching time [11]. Here states of the model and relations are represented by means of propositional formulae. A CTL formula is identified with a set of states: the states of the model satisfying the formula. As a set of states can be represented as a propositional formula, each CTL formula can be characterised by a propositional formula. Thus, the problem of model checking for CTL is reduced to the construction of propositional formulae. This is achieved by composing OBDDs, or by computing fix-points of operators on OBDDs; we refer to [11] for the details.

We review below the algorithm for the verification of temporal, epistemic, and correctness modalities for MAS presented in [20]. This approach is similar, in spirit, to the traditional model checking tech-



(a) (b)

Figure 1. OBDD representation for $a \wedge (b \vee c)$.

niques for the logic CTL. Indeed, it starts by representing the various parameters of a system by means of propositional formulae. Then, this representation is used for the verification of CTLKD formulae.

The number $nv(i)$ of propositional variables required to encode the local states of an agent i is $nv(i) = \lceil log_2|L_i|\rceil$. Similarly, to encode an agent's action, the number $na(i)$ of propositional variables w_i required is $na(i) = \lceil log_2|Act_i|\rceil$. Thus, a global state s can be encoded as a propositional vector $(v_1, \ldots, v_N)$, where $N = \sum_i nv(i)$. An action $a \in Act$ can be encoded as a propositional vector $(w_1, \ldots, w_M)$, where $M = \sum_i na(i)$. In turn, a propositional vector can be identified with a propositional formula, represented by a conjunction of literals, i.e. a conjunction of propositional variables or their negation. In this way, a set of global states (or actions) can be expressed as the disjunction of the propositional formulae encoding each global state in the set. Having encoded local states, global states, and actions by means of propositional formulae, all the remaining parameters can be expressed as boolean functions, too. Indeed, the transition relation can be translated into propositional formulae. The set of initial states is easily translated, too. In addition to the parameters presented above, the algorithm for model checking presented below requires the definition of n boolean functions $R_i^K(s, s')$ (one for each agent) representing the epistemic accessibility relation, the definition of n boolean functions $R_i^O(s, s')$ representing the accessibility relations for the correctness operator, and the definition of a boolean function $R_t(s, s')$ representing the temporal transitions. The boolean function $R_t(s, s')$ can be obtained from the transition relation TR by quantifying over actions. This quantification can be translated into a propositional formula using a disjunction (see [5] for a similar approach to boolean quantification). The set of *reachable* states is also needed by the algorithm: the set W of reachable global states can be expressed symbolically by a propositional formula, and it can be computed as the fix-point of the operator $\tau(Q) = (I(s) \vee \exists s'(R_t(s', s) \wedge Q(s')))$. The fix-point of τ can be computed by iterating from $\tau(\emptyset)$ as standard (see [5]).

We now have all the ingredients in place to present the algorithm $SAT(\varphi)$ to compute the set of global states (expressed as a propositional formula) in which a formula φ holds, denoted by $[\![\varphi]\!]$. The following are input parameters for the algorithm:

- the propositional variables $(v_1, \ldots, v_N)$ and $(w_1, \ldots, w_M)$ for states and actions;
- the function $\mathcal{V}(p)$ returning the set of global states in which p holds.
- the set of initial states I, encoded as a propositional formula;
- the set of reachable states W, encoded as a propositional formula;

- the boolean function R_t encoding the temporal transition;
- n boolean functions encoding the accessibility relations $\sim_i$;
- n boolean functions encoding the accessibility relations $\sim_i^{\mathcal{O}}$.

The algorithm is as follows:

```
$SAT(\varphi)$ {
  $\varphi$ is a proposition: return $\mathcal{V}(\varphi)$;
  $\varphi$ is $\neg\varphi_1$: return $G \setminus SAT(\varphi_1)$;
  $\varphi$ is $\varphi_1 \wedge \varphi_2$: return $SAT(\varphi_1) \cap SAT(\varphi_2)$;
  $\varphi$ is $\mathrm{EX}\varphi_1$: return $SAT_{\mathrm{EX}}(\varphi_1)$;
  $\varphi$ is $\mathrm{E}(\varphi_1 \mathrm{U} \varphi_2)$: return $SAT_{\mathrm{EU}}(\varphi_1, \varphi_2)$;
  $\varphi$ is $\mathrm{EG}\varphi_1$: return $SAT_{\mathrm{EG}}(\varphi_1)$;
  $\varphi$ is $\mathrm{K}_i\varphi_1$: return $SAT_{\mathrm{K}}(\varphi_1, i)$;
  $\varphi$ is $\hat{\mathrm{K}}_i^j\varphi_1$: return $SAT_{\mathrm{KH}}(\varphi_1, i, j)$;
  $\varphi$ is $\mathcal{O}_i\varphi_1$: return $SAT_{\mathcal{O}}(\varphi_1, i)$;
  $\varphi$ is $\mathrm{E}_\Gamma\varphi_1$: return $SAT_{\mathrm{E}}(\varphi_1, \Gamma)$;
  $\varphi$ is $\mathrm{D}_\Gamma\varphi_1$: return $SAT_{\mathrm{D}}(\varphi_1, \Gamma)$;
  $\varphi$ is $\mathcal{C}_\Gamma\varphi_1$: return $SAT_{\mathcal{C}}(\varphi_1, \Gamma)$;
}
```

In the algorithm above, SAT_{EX}, SAT_{EG}, SAT_{EU} are the standard procedures for CTL model checking [11], in which the temporal relation is R_t and, instead of temporal states, global states are considered. We refer to [19] for the definition of the procedures $SAT_{\mathrm{K}}(\varphi, i)$, $SAT_{\mathrm{KH}}(\varphi_1, i, j)$, $SAT_{\mathcal{O}}(\varphi, i)$, $SAT_{\mathrm{E}}(\varphi, \Gamma)$, $SAT_{\mathrm{D}}(\varphi, \Gamma)$, and $SAT_{\mathcal{C}}(\varphi, \Gamma)$. The algorithm SAT can be used to verify whether or not a formula φ holds in a model by comparing two set of states: the set $SAT(\varphi)$ and the set of reachable states W. As sets of states are expressed as OBDDs, verification in a model is reduced to the comparison of the OBDDs for $SAT(\varphi)$ and for W.

5. Dining Cryptographers: modelling, encoding and experimental results

5.1. Protocol description

The anonymous broadcasting of information is one of the main problems discussed in cryptography. The Dining Cryptographers (DC) protocol is a protocol to maintain anonymity in broadcasted information. It was introduced by D. Chaum. The original wording from [4] is included below.

"Three cryptographers are sitting down to dinner at their favorite three-star restaurant. Their waiter informs them that arrangements have been made with the maitre d'hotel for the bill to be paid anonymously. One of the cryptographers might be paying for dinner, or it might have been NSA (U.S. National Security Agency). The three cryptographers respect each other's right to make an anonymous payment, but they wonder if NSA is paying. They resolve their uncertainty fairly by carrying out the following protocol:

Each cryptographer flips an unbiased coin behind his menu, between him and the cryptographer on his right, so that only the two of them can see the outcome. Each cryptographer then states aloud *whether the two coins he can see–the one he flipped and the one his left-hand neighbor flipped–fell on the same side or on different sides. If one of the cryptographers is the payer, he states the opposite of what he sees. An odd number of differences uttered at the table indicates that a cryptographer is paying; an even number indicates that NSA is paying (assuming that dinner was paid for only once). Yet if a cryptographer is paying, neither of the other two learns anything from the utterances about which cryptographer it is."*

The same protocol can be run also for a number of cryptographers greater than three (see [4]). In line with literature in security here we consider a variation of the protocol in which we assume that some cryptographers may be faulty. In particular, we allow them to say the opposite of what they are supposed to, i.e., they can choose to behave correctly or to cheat when announcing the values of the coins they see. The new protocol is called Cheating Dining Cryptographers (CDC) protocol. In the next section we discuss encoding and verification of this protocol.

5.2. Encoding of the CDC protocol

In this section we model the protocol differently from what presented in [15, 10, 20]. While the formalism of interpreted systems is used there, here we decouple the transitions to a finer level, and use asynchronously communicating automata. This description is more convoluted and less intuitive, but as we show below it offers considerable advantages in terms of efficiency. It further allows to present a comparison of different approaches for its verification.

To formalise the protocol we assume that all the events such as coin tosses, determining who is paying and the utterances of the cryptographers can occur in turn, rather then simultaneously. Moreover, instead of enumerating all the possible outcomes of coins tosses, etc., we generate these implicitly using automata that execute independently, and finally synchronizing in order to communicate the result. The aim of the DC protocol is to assure that at the end of the run every cryptographer knows whether it was the NSA or one of cryptographers. Furthermore, if a cryptographer paid, then none of the other cryptographers knows who it was. In the case of CDC protocol we add an assumption that the above properties hold only when all agents behave correctly.

Concretely, we proceed as follows. In the general case, there are n_1 and n_2 automata modelling cheating and honest cryptographers, respectively. An automaton for the honest cryptographer $\mathcal{A}_i$ ($i = n_1 + 1, \ldots, n_1 + n_2$) has got 5 states with the meaning explained by their labels: 0 (the initial state), *seeEqual*$_i$, *seeDifferent*$_i$, *saidEqual*$_i$, and *saidDifferent*$_i$. If the cryptographer can cheat, then the automaton $\mathcal{A}_i$ ($i = 1, \ldots, n_1$) has two additional states: *lieEqual*$_i$ and *lieDifferent*$_i$. The above mentioned automata model what every cryptographer says depending on the coins he sees.

Moreover, there are $n = n_1 + n_2$ automata $\mathcal{A}_i$ ($i = n + 1, \ldots, 2n$) determining who is paying for the dinner. Each of them has three states: 0 (the initial state), *paid*$_{i-n}$, and *notPaid*$_{i-n}$. These automata synchronize in order to determine at most one cryptographer who pays: this automaton moves to the state *paid*, whereas the remaining automata reach the state *notPaid*. In particular, if the NSA is paying, all the automata $\mathcal{A}_i$ ($i = n + 1 \ldots, 2n$) reach the state *notPaid*$_{i-n}$. After determining who pays, every automaton communicates the outcome to the respective cryptographer.

Furthermore, we introduce n automata $\mathcal{A}_i$ ($i = 2n + 1, \ldots, 3n$) that model flipping coins. Each of them has three states: 0 (the initial state), *head*$_{i-2n}$, and *tail*$_{i-2n}$. These automata first determine the result of the flipping, independently of other automata, and next they synchronise with the appropriate automata what corresponds to communicating the outcome to cryptographers – as a result, every cryptographer enters the the state *seeEqual* or *seeDifferent*.

Finally, we have one automaton $\mathcal{A}_{3n+1}$ which models the counter of "different" among the utterances. This automaton also starts from the initial state 0 and then in turn registers what the cryptographers said and how many differences currently there are. In order to do this, it synchronises with the automata modelling cryptographers; it terminates either in the state *even* or *odd*.

The total number of the automata is $3n + 1$. We set all the states of automata to be green with the exception of the states *lieEqual*$_i$ and *lieDifferent*$_i$ ($i = 1, \ldots, n_1$).

An instance of the protocol with two honest and one cheating cryptographer is visualised in Figures 2–6. The network consists of 10 automata: three representing which coins cryptographers can see and what they say (Fig. 4 and 5), three determining who pays (Fig. 2), three modelling the toss-ups (Fig. 3), and one playing the role of the counter of "different" in utterances (Fig. 6). The above automata are composed into the product automaton $\mathcal{A}$ with the initial state $s^0 = \langle 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \rangle$. The global actions, the global states and the transition relation are built according to Definition 2.2. The automaton $\mathcal{A}$ is turned into the model M with a valuation function $\mathcal{V}$ defined over a set of the propositions $\mathcal{PV} = \{paid_1, paid_2, paid_3, even, odd\}$ as follows:

- $paid_i \in \mathcal{V}((l_1, \ldots, l_{10}))$ iff $l_{3+i} = paid_i$, for $i = 1, 2, 3$,
- $even \in \mathcal{V}((l_1, \ldots, l_{10}))$ iff $l_{10} = even$,
- $odd \in \mathcal{V}((l_1, \ldots, l_{10}))$ iff $l_{10} = odd$.

Furthermore, we introduce a set of three agents $\mathcal{A}gt = \{1, 2, 3\}$ representing the three cryptographers: one cheating and two honest. The behaviour of every agent (cryptographer) is modelled by the following automata: one determining what he can see and say, one determining whether he pays, two modelling toss-ups, and one modelling the counter. So, the function *Obs* is defined as follows:

- $Obs(1) = \{1, 4, 7, 8, 10\}$,
- $Obs(2) = \{2, 5, 8, 9, 10\}$,
- $Obs(3) = \{3, 6, 7, 9, 10\}$.

Now we present an example computation. At the beginning, the automata modelling toss-ups execute the actions $h1$, $h2$, $h3$ in turn. These actions set the random results of coin tosses. Therefore, after three steps the global state $s^3 = \langle 0, 0, 0, 0, 0, 0, head_1, head_2, head_3, 0 \rangle$ is reached. Next, the automata determining who pays for dinner execute the synchronised action $s0$ which indicates that the agency pays, so the global state of the model is $s^4 = \langle 0, 0, 0, notPaid_1, notPaid_2, notPaid_3, head_1, head_2, head_3, 0 \rangle$. Next, the cryptographers see the results of coin tosses and every of them says whether he sees equal or different sides of coins. Finally, the counter counts the number of differences. Assuming that the cheating cryptographer does not cheat in this run, the final state of this scenario, after executing 13 transitions, is $\langle saidEqual_1, saidEqual_2, saidEqual_3, notPaid_1, notPaid_2, notPaid_3, head_1, head_2, head_3, even \rangle$. In the general case of n cryptographers, the maximal number of fired transitions is equal to $4n + 1$. This number is called *the maximal depth of the model*.

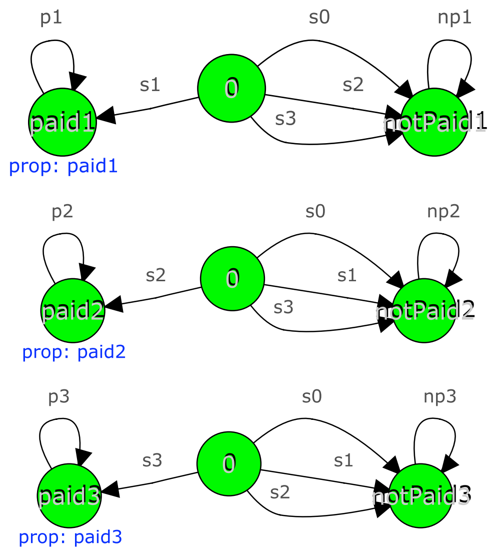


Figure 2. The automata $\mathcal{A}_4$, $\mathcal{A}_5$, $\mathcal{A}_6$ determining who pays for dinner.

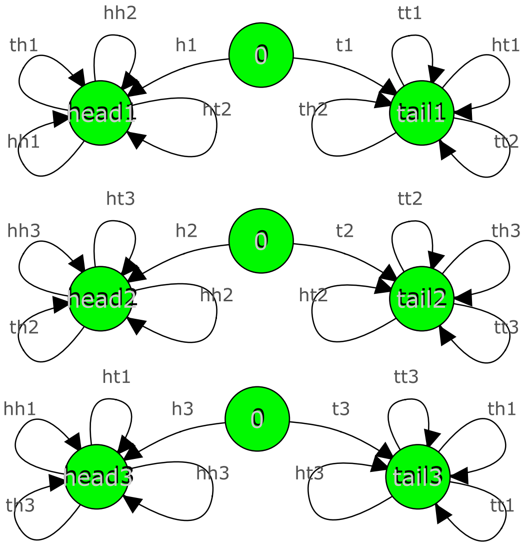


Figure 3. The automata $\mathcal{A}_7$,$\mathcal{A}_8$,$\mathcal{A}_9$ modelling toss-ups.

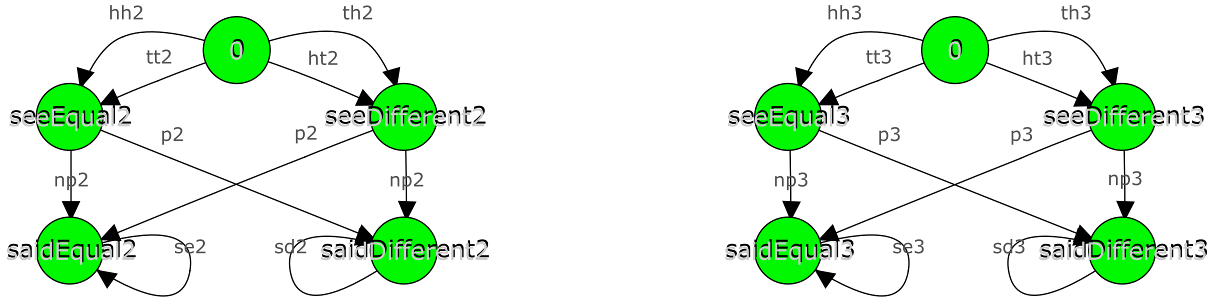


Figure 4. The automata $\mathcal{A}_2$, $\mathcal{A}_3$ modelling what honest cryptographers can see and say.

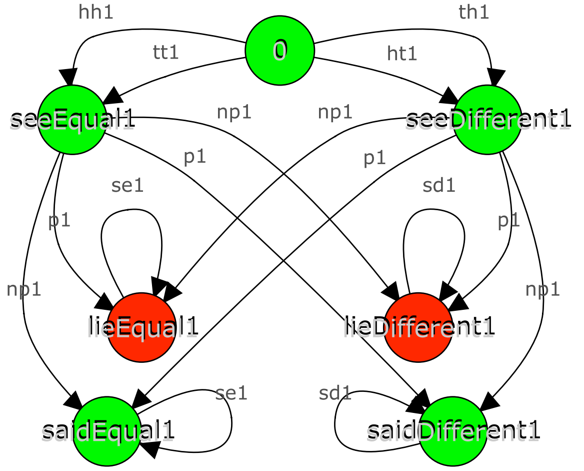


Figure 5. The automaton $\mathcal{A}_1$ modelling what cheating cryptographer can see and say.

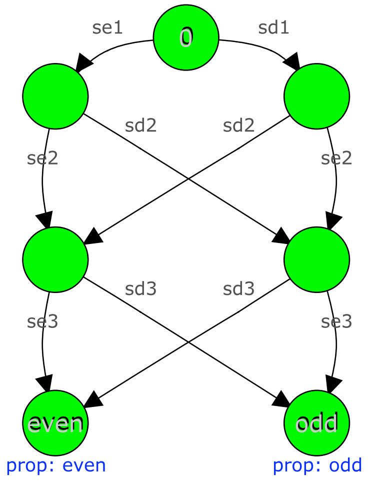


Figure 6. The automaton $\mathcal{A}_{10}$ modelling the counter of differences in the utterances.

Symbol	Formula	Model	Satisfiable
$Form_1$	$\mathrm{AG}(odd \wedge \neg paid_1 \Rightarrow \mathrm{K}_1(\bigvee_{i=2,\ldots,n} paid_i))$	M_{DC_n}	Yes
$Form_2$	$\mathrm{AG}(odd \wedge \neg paid_1 \Rightarrow \bigvee_{i=2,\ldots,n} \mathrm{K}_1(paid_i))$	M_{DC_n}	No
$Form_3$	$\mathrm{AG}(\neg paid_1 \Rightarrow \mathrm{K}_1(\bigvee_{i=2,\ldots,n} paid_i))$	M_{DC_n}	No
$Form_4$	$\mathrm{AG}(even \Rightarrow \hat{\mathrm{K}}^1_n(\bigwedge_{i=1,\ldots,n} \neg paid_i))$	$\mathrm{M}^1_{CDC_n}$	Yes

Table 1. The tested formulae.

5.3. Experimental results

In this section we present the verification results for several properties of the protocols DC and CDC – for n agents. The models are denoted M_{DC_n} and $\mathrm{M}^1_{CDC_n}$ respectively, the latter including only one cheating cryptographer. The presented tests were performed on a workstation equipped with the AMD Athlon XP+ 2400 MHz processor and 2 GB RAM running under Fedora Linux.

5.3.1. BMC: Verifying CDC with Verics

The verification system VerICS has been used to perform the experiments with BMC. VerICS [6] is a verification tool for real-time and multi-agent systems. It offers three complementary methods of model checking: SAT-based Bounded Model Checking (BMC), SAT-based Unbounded Model Checking (UMC), and an on-the-fly verification while constructing abstract models of systems. The theoretical background for its implementation has been presented in several papers [7, 17, 23].

All the tested formulae are listed in Table 1. In order to provide a better intuition behind the properties they express, the formulas are given in the universal form. However, notice that BMC handles universal formulae indirectly by looking for counterexamples to their negations (i.e., the existential formulae). The results of verification are presented in Table 2 where the number of cryptographers (n), the length of the symbolic paths (k), time and memory needed for BMC translations (BMC[s], BMC[MB]) and time of SAT verification (SAT[s]) as well as the number of generated variables (Vars) and clauses (Clauses) are given.

The first three properties are checked for the DC protocol, i.e., without cheating cryptographers. The formula $Form_1$ expresses Chaum's property which states that always when the number of differences is odd and the first cryptographer has not paid for the dinner, then he knows that some other cryptographer paid for dinner. The formula $\neg Form_1$ has been tested over two symbolic paths.

The formula $Form_2$ states that always when the number of differences is odd and the first cryptographer has not paid for dinner, then he knows the cryptographer who paid for dinner. This formula is obviously not true in the model because, following the protocol, none of the cryptographers can possess such an information. If odd holds, then the first cryptographer knows that one of the cryptographers has paid but he does not know which one. In this case the number of symbolic paths is equal to n.

The formula $Form_3$ states that it is always true that if the first cryptographer has not paid for dinner, then he knows that some other cryptographer pays. Since this property is true only if the number of differences is odd, the formula is not true in the model. In this case one symbolic path suffices to check that the formula $\neg Form_3$ is true. Moreover this property can be checked on a very small depth of the symbolic path. This example shows that BMC is very powerful in such cases. We can verify this formula

Formula	n	k	BMC[s]	BMC[MB]	SAT[s]	Vars	Clauses
$Form_1$	3	13	0.69	7.06	267.67	21922	67855
	4	17	1.33	9.80	3010.89	40107	60200
	5	21	2.11	13.45	19729.38	62637	194244
	6	25	3.08	19.25	87224.75	90959	281919
$Form_2$	3	13	1.5	10.53	84.98	44450	140017
	4	17	5.19	26.41	730.72	139015	442119
	5	21	13.59	58.18	4019.71	335877	1070691
	6	25	31.1	123.00	31675.79	706087	2250907
$Form_3$	100	1	3.75	22.17	0.12	96561	284528
	500	1	109.16	436.09	3.98	2082111	6220764
	1000	1	499.20	1889.8	19.34	8164166	24441422
$Form_4$	3	13	0.72	7.19	1602.18	23050	71323
	4	17	1.40	10.56	30979.78	41815	129693
	5	21	2.19	14.14	106624.84	65038	201664

Table 2. The results of verification of DC and CDC using VerICS.

for even 1000 cryptographers (4001 automata)!

The formula $Form_4$ says that the last cryptographer knows that always when the first cryptographer behaves correctly and the number of differences is even, then any of the cryptographers is not a payer. Unlike the other properties it is verified in the model with one cheating cryptographer. Observe that in such a model the formula $\mathrm{AG}(even \Rightarrow \mathrm{K}_n(\bigwedge_{i=1,\ldots,n} \neg paid_i))$ is not valid since the even number of differences does not ensure that NSA paid for dinner. Therefore in $Form_4$ operator $\hat{\mathrm{K}}^1_n$ instead of K_n is used. Now the formula $\hat{\mathrm{K}}^1_n(\bigwedge_{i=1,\ldots,n} \neg paid_i)$ expresses that agent n knows that any of cryptographers did not pay on condition that agent 1 does not cheat. This change makes the whole property true. The formula $\neg Form_4$ has been tested on two symbolic paths of the maximal length.

We should underline that BMC method is usually used for checking satisfiability of existential formulae, i.e., checking that a universal formula does not hold. In this case, since all the computations of CDC model are finite, checking validity of universal formulae is also possible. However such tests must be performed on the whole model (in particular on the maximal length paths), thereby invalidating the main BMC idea of finding counter-examples without exploring the whole model. This is the main cause of the long time of verification for the given properties.

5.3.2. OBDD: Verifying CDC with MCMAS

Now, we present the experimental results obtained with MCMAS – a tool that implements the OBDD-based algorithms presented in Subsection 4.2. MCMAS is released under the terms of the GNU General Public License (GPL) and it is available for download [18].

In MCMAS, multi-agent systems are described using the language ISPL (Interpreted Systems Pro-

```
Agent SampleAgent
 Lstate = {s0,s1,s2,s3};
 Lgreen = {s0,s1,s2};
 Action = {a1,a2,a3};
 Protocol:
   s0: {a1};
   s1: {a2};
   s2: {a1,a3};
   s3: {a2,a3};
 end Protocol
 Ev:
   s2 if ((AnotherAgent.Action=a7);
   s3 if Lstate=s2;
 end Ev
end Agent
```

Figure 7. ISPL example

gramming Language). Figure 7 gives a short example of this language. We refer to the files available online [18] for the full syntax of ISPL. Formulae to be checked are provided at the end of the specification file, using an intuitive syntax.

A given network of communicating automata can be encoded using the language ISPL by associating each automaton to an agent (in the sense of MCMAS); synchronisation is achieved in MCMAS using the appropriate evolution function for the agents. MCMAS can implement the function Obs for a network of automata by taking the distributed knowledge of a set of automata (encoded as agents). The encoding of the protocol of the dining Cryptographers using a network of automata, as presented in the previous Section, is available for download [18].

Following standard conventions, we define the size of a system as $|W| + |R|$, where $|W|$ is the size of the state space and $|R|$ is the size of the relations. In our case, we define $|W|$ as the number all the possible combinations of local states and actions.

Experimental results for the verification of $\mathrm{M}^1_{CDC_n}$ are reported in Table 3. Differently from the SAT-based Bounded Model Checking techniques presented above, time results for model checking using OBDDs are not affected by the structure of the formula being verified. Typically, the time required to execute the algorithm presented in Figure 4.2 is a fraction (in the order of 0.1% - 0.5%) of the time required for the construction of the OBDDs representing the temporal relation, the set of reachable states, etc. The temporal results in Table 3 refer to the verification time for various formulas, also discussed later in of Table 1. Further, we verified the following:

$$\mathrm{AG}((odd \wedge \neg paid_1) \rightarrow \mathrm{AF}(\mathrm{K}_1(paid_2 \vee paid_3) \wedge \neg \mathrm{K}_1(paid_2) \wedge \neg \mathrm{K}_1(paid_3)))$$

This formula expresses the idea that, if the first cryptographer did not pay for dinner and the number of “different” utterances is odd, then eventually the first cryptographer knows that either the second or the third cryptographer paid for dinner; moreover, in this case, the first cryptographer does not know which of these two is the payer (notice that this formula holds when the first cryptographer is behaving correctly). Intuitively this entirely captures the specification of the protocol.

N.	sec	Memory (bytes)	BDD vars	$\|S\|$
3	1	5281140	53	1.91E+07
4	4	6524788	69	1.48E+09
5	6	7229988	85	1.01E+11
6	424	56056516	101	6.48E+12
7	78	22589412	117	3.92E+14
8	8101	134174996	133	2.29E+16
9	508	39823892	149	1.29E+18
10	4841	60021380	165	7.15E+19
11	991	57448372	181	3.88E+21

Table 3. Experimental results with MCMAS

5.4. Discussion and comparison with existing work

As mentioned at the beginning of Section 2, the protocol of the Dining Cryptographers has been modelled in different ways by other authors [15, 20].

In particular, [15] present and OBDD-based algorithm for the verification of a particular class of interpreted systems (synchronous with perfect recall). Their algorithm accepts the class of formulae whose structure is $X^k(K_i p)$ (where p is an atomic proposition and X^k denotes a concatenation of k temporal operators X. It is shown that the problem of model checking this class of formulae can be reduced to the verification of the equivalence of Boolean formulae, manipulated using OBDDs. This methodology is applied to the verification of the protocol of the dining cryptographers. However, the modelling appearing in [15] differs substantially from the modelling presented in this paper: indeed, in synchronous systems with perfect recall all the information about coin tosses, utterances, etc., is stored in a special agent, the Environment. The remaining agents do not have “local” states, but they are only allowed to *observe* the environment, and to perform actions based on their observations. This restriction, together with the restriction of limiting verification to a particular class of formulae, results in a much smaller encoding. No tool is presented in [15], but partial experimental results are provided for the verification of formulae in examples with up to 20 cryptographers.

The tool MCMAS is used in [20] to verify the protocol of the dining cryptographers, but with a different encoding of the example. That encoding models each cryptographer using a single agent, with an additional agent for the environment. Such an encoding is less efficient than the one we present, in that various parameters are repeated for each agent. For instance, the number of utterances is stored separately in each agent, while our approach encodes this information with a single automaton representing the counter. It is clear that while [20] follows the formalism of interpreted systems to the letter, the efficient decoupling presented here offers speed advantages. This fact is reflected in the experimental results of MCMAS: as shown in Table 3, we verified scenarios with up to 11 cryptographers, while the encoding proposed in [20] allowed for the verification of 8 cryptographers only.

Therefore, we argue that our encoding offers a substantial improvement with respect to previous works in two respects:

- Our encoding permits the verification of a larger class of formulae than [15], and it represents more "autonomous" agents, characterised by private local states.
- Our encoding is more efficient than the one presented in [20], by enabling an improvement of nearly 50% in performance when the same tool is used.

6. Conclusions and future work

In this paper we have presented a scenario of modelling and model checking of the dining cryptographers protocol, in the presence of cheaters. In particular we have compared the performance of MCMAS and VERICS using a common representation based on a network of automata.

Our experimental results, summarised in Tables 3 and 2, paint the following picture: first, both checkers were able to check a variety of complex formulae correctly and efficiently. Specifically, MCMAS calculates the (symbolic representation of the) whole model before actually performing the checks. It proves to be faster for many formulae and enables the verification of the full CTLKD syntax. On the other hand, the experiments with VERICS confirmed that BMC is in general not complete and performs best when finding shallow counterexamples. In this case, it can handle really huge models. The overall conclusion coincides with the common belief that the OBDD and BMC techniques complement each other.

It should be noted that our results are preliminary, as the main effort was focused on modelling issues and not on the performance. Many optimisation techniques are widely used in model checking, usually enabling verification of realistic systems, and both the presented methodologies should be combined with these standard approaches, extended the presented framework.

While our results are limited to these two model checkers and each checker may benefit from additional optimisation techniques, it seems to us that these results may be generalised to the techniques behind the checkers, i.e., BMC for VERICS and OBDD for MCMAS.

In other words, what we found is that depending on the model in hand one technique may be more efficient than another. To check satisfaction on models up to a size of about 10^{20} it seems that MCMAS has an advantage. Checking satisfiability of ECTLKD formulae only on large models is clearly something that is better handled by VERICS.

A further novelty of this paper lies in the analysis of the protocol in terms of deontic, epistemic and temporal properties (as opposed to temporal properties only). This allows to represent violations (i.e., cheating) in the behaviour of the cryptographers in a natural way. When comparing our approaches to other available in the literature, we find that this considerably simplifies the specifications to be checked against, while still maintaining the feasibility of the model checking approach. We plan to continue evaluating this approach by means of other protocols of interest and to pursue ideas resulting from the novel formalisation of the DC protocol presented here when modelling other security protocols so that possible efficiency advantages may be replicated.

References

[1] Biere, A., Clarke, E., Raimi, R., Zhu, Y.: Verifying Safety Properties of a PowerPC Microprocessor Using Symbolic Model Checking without BDDs, *Proc. of the 11th Int. Conf. on Computer Aided Verification (CAV'99)*, 1633, Springer-Verlag, 1999.

[2] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, **35**(8), August 1986, 677–691.

[3] Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L., Hwang, L. J.: Symbolic Model Checking: 10^{20} States and Beyond, *Information and Computation*, **98**(2), June 1992, 142–170.

[4] Chaum, D.: The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability, *Journal of Cryptology*, **1(1)**, 1988, 65–75.

[5] Clarke, E. M., Grumberg, O., Peled, D.: *Model Checking*, MIT Press, 1999.

[6] Dembiński, P., Janowska, A., Janowski, P., Penczek, W., Półrola, A., Szreter, M., Woźna, B., Zbrzezny, A.: VerICS: A Tool for Verifying Timed Automata and Estelle Specifications, *Proc. of the 9th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, 2619, Springer-Verlag, 2003.

[7] Doroś, A., Janowska, A., Janowski, P.: From Specification Languages to Timed Automata, *Proc. of CS&P the Int. Workshop on Concurrency, Specification and Programming (CS&P'02)*, 161(1), Humboldt University, 2002.

[8] Emerson, E. A., Clarke, E. M.: Using Branching-Time Temporal Logic to Synthesize Synchronization Skeletons, *Science of Computer Programming*, **2(3)**, 1982, 241–266.

[9] Fagin, R., Halpern, J. Y., Moses, Y., Vardi, M. Y.: *Reasoning about Knowledge*, MIT Press, Cambridge, 1995, ISBN 0-262-06162-7.

[10] van der Hoek, W., Wooldridge, M., van Otterloo, S.: Model Checking Knowledge and Time via Local Propositions: Cooperative and Adversarial Systems, 2004, Submitted.

[11] Huth, M. R. A., Ryan, M. D.: *Logic in Computer Science: Modelling and Reasoning about Systems*, Cambridge University Press, Cambridge, England, 2000, ISBN Hardback: ISBN 0-521-65200-6, Paperback: ISBN 0-521-65602-8.

[12] Lomuscio, A., Sergot, M.: Deontic Interpreted Systems, *Studia Logica*, **75**(1), 2003, 63–92.

[13] Lomuscio, A., Sergot, M.: A formalisation of violation, error recovery, and enforcement in the bit transmission problem, *Journal of Applied Logic*, **2**(1), March 2004, 93–116.

[14] Lowe, G., Roscoe, A. W.: Using CSP to Detect Errors in the TMN Protocol, *Software Engineering*, **23**(10), 1997, 659–669.

[15] van der Meyden, R., Su, K.: Symbolic Model Checking the Knowledge of the Dining Cryptographers, *17th IEEE Computer Security Foundations Workshop*, 2004.

[16] Penczek, W., Lomuscio, A.: Verifying Epistemic Properties of Multi-Agent Systems via Bounded Model Checking, *Proc. of the 2nd Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS'03)*, ACM, July 2003.

[17] Penczek, W., Woźna, B., Zbrzezny, A.: Bounded Model Checking for the Universal Fragment of CTL, *Fundamenta Informaticae*, **51(1-2)**, 2002, 135–156.

[18] Raimondi, F., Lomuscio, A.: MCMAS - A tool for verification of multi-agent systems, Http://www.cs.ucl.ac.uk/staff/f.raimondi/MCMAS/.

[19] Raimondi, F., Lomuscio, A.: Verification of multiagent systems via ordered binary decision diagrams: an algorithm and its implementation, *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'04)* (N. R. Jennings, C. Sierra, L. Sonenberg, M. Tambe, Eds.), II, ACM, July 2004.

[20] Raimondi, F., Lomuscio, A.: Automatic verification of multi-agent systems by model checking via OBDDs, *Journal of Applied Logic*, 2005, To appear in Special issue on Logic-based agent verification.

[21] Visser, W., Havelund, K., Brat, G., Park, S.: Model Checking Programs, *Proc. of the 15th IEEE Int. Conf. on Automated Software Engineering (ASE'00)*, IEEE Computer Society, 2000.

[22] Woźna, B., Lomuscio, A., Penczek, W.: Bounded Model Checking for Deontic Interpreted Systems, *Proc. of the 2nd Workshop on Logic and Communication in Multi-Agent Systems (LCMAS'04)*, 126, Elsevier, 2004.

[23] Woźna, B., Penczek, W., Zbrzezny, A.: Reachability for Timed Systems Based on SAT-Solvers, *Proc. of the Int. Workshop on Concurrency, Specification and Programming (CS&P'02)*, 161(2), Humboldt University, 2002.