A framework for automatically checking anonymity with μ CRL

A Framework for Automatically Checking Anonymity with μCRL

Tom Chothia[1], Simona Orzan[2,1], Jun Pang[3], and Mohammad Torabi Dashti[1]

[1] Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands
[2] Technische Universiteit Eindhoven, Eindhoven, The Netherlands
[3] Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany

Abstract. We present a powerful and flexible method for automatically checking anonymity in a possibilistic general-purpose process algebraic verification toolset. We propose new definitions of a *choice anonymity degree* and a *player anonymity degree*, to quantify the precision with which an intruder is able to single out the true originator of a given event or to associate the right event to a given protocol participant. We show how these measures of anonymity can be automatically calculated from a protocol specification in μCRL, by using a combination of dedicated tools and existing state-of-the-art μCRL tools. To illustrate the flexibility of our method we test the Dining Cryptographers problem and the FOO 92 voting protocol. Our definitions of anonymity provide an accurate picture of the different ways that anonymity can break down, due for instance to coallitions of inside intruders. Our calculations can be performed on a cluster of machines, allowing us to check protocols for large numbers of participants.

1 Introduction

Anonymity, as a security property, refers to the ability of a user to own some data or take some actions without being tracked down. This property is essential in protocols that might involve sensitive personal data, like electronic auctions, voting, anonymous broadcasts, file-sharing etc. Due to its relevance and subtle nature, anonymity has been given many definitions [3,16,17,26] and has been the subject of many theoretical studies and formal analysis [19,21]. However, automatic approaches to the formal verification of anonymity have only treated small examples of individual protocols [10,20,28,30]. We address this situation by investigating the possibility of using a powerful general-purpose explicit-state verification toolset, μCRL [4], to automatically verify anonymity properties. We define two measures of anonymity and set up a framework to calculate them from process specifications.

Our definitions of anonymity are based on a notion of secret choices for participants. These choices may signify actions (e.g., accessing a certain web server) or data (e.g., votes in a voting protocol). We represent a protocol as a composition of a number of players (participants), each given a secret choice. The two types of anonymity that we propose quantify the ability of an intruder to

U. Montanari, D. Sannella, and R. Bruni (Eds.): TGC 2006, LNCS 4661, pp. 301–318, 2007.

deduce the right association of players and choices. Consider a voting protocol with 50 candidates and 1000 voters. There are two types of questions that the intruder may ask: 1) *who* voted for a particular candidate, for instance $candidate_3$ and 2) *what* was the vote of a particular player, for instance $player_1$. In the first case, a choice is fixed ($candidate_3$) and the originator(s) of that choice are sought, in the second case a player is fixed and determining their choice is the object of intruder's attention. The answers obtained are usually not precise, but rather in the form of a set of possibilities — the smaller the set of possibilities, the more exact the intruder's guess and therefore the smaller the degree of anonymity. Namely, there is a quantitative difference between the situation in which the intruder reaches the conclusion that the vote of $player_1$ is in the set $\{candidate_3, candidate_{49}\}$ and the situation in which the intruder considers all 50 candidates as possible choices of $player_1$. Based on these observations, we say that if the intruder considers more than one secret choice possible for a given player then the player has *choice anonymity* and the number of possible choices for the player is the *choice anonymity degree*. In a similar fashion, if the intruder considers more than one player as a possible owner of a given secret choice then that choice has *player anonymity* and the number of possible players for the choice is the *player anonymity degree*.

Our definitions allow for corrupted players that will share their information with the intruder. This means that we can measure the effect of coalitions of corrupted players and the intruder on the anonymity of honest players. We formally define these metrics in terms of bisimulation between processes and provide tool support to compute them automatically, starting from an abstract description of the protocol in the process-algebraic language μCRL. Trace equivalence has also been proposed as an equivalence for checking anonymity [22,28]. Bisimulation is a more discriminating relation than trace equivalence; while it is possible that we will detect false positives, these are better than the possible false negatives. Bisimulation also has the advantage of being more efficient to compute than trace equivalence [18].

We specify the protocols we wish to check in μCRL. This is an expressive language that comes with an extensive toolset and has a long history of successful protocol checking [5,25]. The μCRL toolset includes tools for performing state space reduction modulo bisimulation [6], which we use along with some purpose-built scripts and C programs to generate all possible cases of the model and to calculate our measures of anonymity. The μCRL toolset allows us to distribute the checking over a cluster of machines. Unlike other approaches, we support automatic generation of μCRL models for any given number of participants and any given coalition of corrupt participants.

Our verification approach is possibilistic, rather than probabilistic, i.e., we consider two processes the same if there is the possibility of them performing the same actions. We do not take into account the probability of the actions occurring. While the possibilistic approach may still allow the intruder to make a good guess at the identity of a guilty player, the metrics are much easier

to calculate and it avoids the problem of combining probabilities with non-deterministic choices, such as how often a given player will use the system.

We illustrate our approach by two examples: the Dining Cryptographers problem [7] and the FOO voting protocol [13]. These systems have already been analysed with formal methods, but not within one framework. The Dining Cryptographers problem has been used as a test case for many tools; the largest protocol instance that has been verified, to the best of our knowledge, contains 8 participants, by using symbolic model checking on an epistemic specification [20]. Our approach can check more than 15 cryptographers in a few hours. In contrast, the FOO voting protocol has not previously been checked in a fully automated framework.

The contribution of this paper is threefold: 1) a framework for checking anonymity including the definitions of choice anonymity degree and player anonymity degree, with a treatment of coalitions of corrupt players, 2) demonstrating the flexibility of this framework by testing examples of two well known anonymous systems, 3) demonstrating the power of this framework by showing how we can automatically check the anonymity degrees of our examples on a single machine or on a cluster.

The structure of the paper. We discuss related work in the rest of this section. In Section 2, we define two notions of anonymity degrees, and we present our verification framework. We apply our approach to the analysis of two examples in Sections 3 and 4. The results of the experiments are gathered together in Section 5. Finally, Section 6 concludes the paper and discusses possible future extensions. The code for the examples and the scripts for calculating the anonymity degrees are available online [9].

Related work. Using process equivalences to model anonymity dates back to Meritt [23], whose work was inspired by information flow analysis. Chaum [7] uses the size of an anonymity set to indicate the degree of anonymity provided by a network based on Dining Cryptographers (DC nets). An anonymity set is defined as the set of participants who could have sent a particular message as observed by the intruder. Pfitzmann and Hansen [26] investigate a similar idea. Berthold, Pfiztmann and Standtke [2] define the degree of anonymity as $\ln(N)$, where N is the number of users of the protocols. Our metrics can be thought of as the anonymity set for players and the anonymity set for choices, defined via a behavioural equivalence.

Reiter and Rubin [27] define the degree of anonymity as the probability that an intruder can assign to a player of being the original sender of a message. This metric does not take into account the number of players in a system. Bhargava and Palamidessi [3] propose a similar definition of anonymity that makes a careful distinction between non-deterministic and probabilistic actions. Deng, Palamidessi and Pang [10] define "weak probabilistic anonymity" and use a probabilistic model checker (PRISM) to analyse the Dining Cryptographers problem. Serjantov and Danezis [29] define an information theoretic anonymity metric

based entropy and Díaz et al. [11] provide a similar metric that is normalised by the number of users.

The FOO voting protocol has been analysed by Kremer and Ryan in the applied pi-calculus [19] and Chothia [8] uses bisimulation to test the anonymity of an anonymous file-sharing system. Also on the possibilistic side, Schneider and Sidiropoulos [28] use FDR to check anonymity via trace equivalence in CSP, and Garcia et al. [14] develop a formal framework for proving anonymity based on epistemic logic.

2 Anonymity Formalisation and Verification Methodology

Anonymity as a security property comes in many flavours. We take the rather general view that when participants in a protocol wish to remain anonymous they wish to hide parts of their behaviour and data. That is, an intruder should not be able to find out what choices, regarding control as well as data, that particular participant has made. We consider the environment as an active attacker that observes protocol runs, hence we need not model the intruder explicitly. We also consider the possibility of a number of corrupt insiders that may leak their observations to the attacker.

The protocol model. Group protocols can usually be written as a parallel composition of participants and an environment process:

$$\texttt{Protocol}(\mathbf{x}) = P_1(x_1) \| P_2(x_2) \| \ldots \| P_n(x_n) \| Q(n) \tag{1}$$

Here $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is the vector of secret choices (e.g., votes in a voting protocol). The choice x_i comes from a known domain and the anonymity refers to the link between this value and the identity of the participant using it. Each P_i $(1 \leq i \leq n)$ describes the behaviour of a single player. Process $Q(n)$ represents the environment, made up from entities that 'oversee' the protocol and, by the nature of their role, do not need to be anonymous. Examples of such entities are the auction house in an auction protocol, or the ballot counter in a voting protocol. In this paper, $P_1, \ldots, P_n, Q$ are models written in the process-algebraic specification language μCRL, a short description of which is given later in this section.

The possible behaviours in our model are grouped in three levels:

1. A *choice* defines the behaviour of a single player. In the Dining Cryptographers protocol, for instance, the possible choices are `T`, to indicate that a player is the payer, and `F`, to indicate that the player is not going to pay.
2. A *choice vector* is an ordered list of choices. It defines one behaviour of the entire system. The vector's ith element defines the behaviour of the ith player.
3. A *choice vector set* is a set of choice vectors that represent all possible behaviours of a system.

A short introduction to μCRL. The specification language μCRL [15] is an extension of the process algebra ACP [1] with abstract data types, which are very handy when describing real-life applications. Processes are built from atomic actions by the ACP operators for sequential composition (.), non-deterministic choice (+) and parallel compositions ($\|$). Synchronisation in ACP is governed by a communication function γ. E.g., synchronisation of actions a and b yields the action $\gamma(a, b)$. There is also an *encapsulation* operator ∂_H, that forces processes to communicate, by making the actions in the set H act exclusively in communication. The *hiding* operator τ_I turns all occurrences of actions from the set I into the internal action τ. The *renaming* operator ρ_R renames actions according to the renaming rules in R. In fact, the precise form of Equation 1 in μCRL is

$$\texttt{Protocol}(\mathbf{x}) = \tau_I \rho_R \partial_H (P_1(x_1) \| P_2(x_2) \| \ldots \| P_n(x_n) \| Q(n)) \tag{2}$$

There are two special actions: δ represents deadlock, and τ the internal action. In order to incorporate abstract data types in a specification, a signature of multiple sorts and functions can be declared, and axiomatised by equations. A number of connectives tie processes up with abstract data types. First, atomic actions can be parameterised with data elements, as in $\texttt{send}(x)$. Then, $\sum_{x:D} P(x)$ denotes alternative (possibly infinite) choice over data domain D, i.e. for any value $x_0 \in D$, the process can behave as $P(x_0)$. Finally, if b is a term of data domain *Bool* and p and q are processes, then the conditional construct $p \lhd b \rhd q$ is the process 'p if b, else q'.

Groups of corrupted players. Since it is not uncommon that the intruder manages to persuade or blackmail some of the participants into revealing their secrets, our models take into account the presence of groups of corrupted players. We use Obs to denote the set of observer that join forces with an external intruder. $\texttt{Protocol}(\mathbf{x})$ specifies the behaviour of a protocol under the assumption that all participants are honest, i.e. Obs $= \emptyset$. If Obs $\neq \emptyset$, it means that the intruder has access to some extra inside information, namely all the secret choices and hidden actions of the players in Obs. Therefore, in order to obtain the behaviour corresponding to this situation, we need to cancel the effect of the action hiding and renaming applied to processes in Obs. The resulting process is denoted by $\texttt{Protocol}_{\text{Obs}}(\mathbf{x})$,

$$\texttt{Protocol}_{\text{Obs}}(\mathbf{x}) = \tau_{(I/AO)} \rho_{(R/AO)} \partial_H (P_1(x_1) \| P_2(x_2) \| \ldots \| P_n(x_n) \| Q(n)) \tag{3}$$

where AO are the actions observed by the members of Obs. We note that, $\texttt{Protocol}_{\emptyset}(\mathbf{x}) = \texttt{Protocol}(\mathbf{x})$.

Anonymity formalisation. We distinguish two ways to look at anonymity requirements, in a protocol specified as above, both are in terms of participants with secret choices: (1) for a given participant P, can the intruder find out its

secret choice, as in "for whom did P vote?", and (2) for a given choice c, can the intruder find out which of the participants owns it, namely "who voted for c?". Following the same two-fold view, we define anonymity notions that are sensitive to quantitative nuances: (1) given a participant P, how many possibilities for the participant's secret choice will the intruder consider, and (2) given a secret choice c, how many participants will the intruder consider as possibly having taken that secret choice? These notions can be thought of as defining the anonymity degree for players in terms of choices (`cad`) and the anonymity degree for choices in terms of players (`pad`).

The equivalence used in the verification of anonymity models the observation power of the intruder. Our definitions of anonymity degree can use any equivalence relation; in our case studies we use bisimulation ($\approx$) to equate process. In particular we use branching bisimulation when we are modelling processes with hidden actions and for efficiency we use strong bisimulation when there are no hidden actions. This is in contrast to some previous work on anonymity that used trace equivalence [22,28]. While it is often possible, in an asynchronous setting, to implement processes in such a way that an intruder cannot tell the difference between two processes that are trace equivalent but not bisimilar, there also exist reasonable implementations in which the intruder can tell the difference. For instance, the two processes $a.(b + c)$ and $a.b + a.c$ are trace equivalent but not bisimilar. A reasonable implementation of these processes might use sockets for communication, in which case the first process would listen on port "a" for a message and then listen on ports "b" and "c" and accept only the first message that arrives. The second process could be implemented by either listening on port "a" and then port "b" or listening on port "a" and then port "c". All an intruder has to do to tell these processes apart is to send on port "a" and then on port "b". If the intruder then gets a message sent to port "b" rejected they may conclude that they are dealing with the second process. In this sense, using bisimulation rather than trace equivalence is a conservative decision; while it is possible for processes that are trace equivalent, but not bisimilar, to be safe, we cannot guarantee that they do not reveal information to the intruder.

A second advantage of using bisimulation is that it can be much more efficient to check. The added restrictions on bisimilar processes mean that we can reject certain paths as not bisimilar long before we could detect that they are not trace equivalent. In the most extreme cases checking a particular pair of processes for trace equivalence can take exponential time while checking the same processes for bi-simulation can take linear time.

Definition 1 (choice indistinguishability). *Let* `Protocol` *be the specification of a protocol,* $\mathbf{v_1}$ *and* $\mathbf{v_2}$ *two choice vectors, and* Obs *an observer set. The set of all possible choice vector is denoted by CVS. Then the relation* $\approx_{\text{Obs}} : CVS \times CVS$ *is defined as:*

$$\mathbf{v_1} \approx_{\text{Obs}} \mathbf{v_2} \text{ iff } \texttt{Protocol}_{\text{Obs}}(\mathbf{v_1}) \approx \texttt{Protocol}_{\text{Obs}}(\mathbf{v_2}).$$

Definition 2 (choice anonymity degree). *The choice anonymity degree (*`cad`*) of participant i w.r.t. an observer set* Obs *under the choice vector* $\mathbf{x}$ *is:*

$$\mathtt{cad}_{\mathbf{x}}(i) = |\{c \in \mathit{Choices}, \exists \mathbf{v} \in \mathit{CVS} \text{ such that } v_i = c \text{ and } \mathbf{v} \approx_{\mathrm{Obs}} \mathbf{x} \text{ and } (\forall j \in \mathrm{Obs}.v_j = x_j)\}|$$

where $|\cdot|$ *denotes the cardinality of a set, Choices is the set of all possible choices, CVS is the choice vector set,* $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$ *and* $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$*. We define the choice anonymity degree of participant* i *w.r.t.* Obs *as*

$$\mathtt{cad}(i) = \min_{\mathbf{x} \in \mathit{CVS}} \mathtt{cad}_{\mathbf{x}}(i)$$

The set in the above formula for $\mathtt{cad}_{\mathbf{x}}(i)$ is the set of all choices c that could be assigned to player i, as part of a choice vector $\mathbf{v}$ that is indistinguishable from a fixed choice vector $\mathbf{x}$. The "$\exists \mathbf{v} \in$ CVS" and "$v_i = c$" conditions ensure that $\mathbf{v}$ is a choice vector which assigns choice c to player i. The choice vector $\mathbf{x}$ can be thought of as defining the observable behaviour of a particular run of the protocol and the choice vector $\mathbf{v}$ defines the observable behaviour of another run, which the observers represented by Obs, cannot distinguish from the $\mathbf{x}$-run. We look for a value of $\mathbf{x}$ that gives the smallest possible number of choices, i.e., we are looking for the worst case anonymity. Our measure of choice anonymity degree for player i is then the size of the set of possible choices that player i may have been assigned. The condition $\forall j \in \mathrm{Obs}.v_j = x_j$ captures the fact that the choice for Obs is fixed. This is because the players in Obs share their choice values with the attacker.

Definition 3 (player anonymity degree). *The player anonymity degree (*pad*) of secret choice* c*, in a protocol with* n *players, w.r.t. an observer set* Obs *and the choice vector* $\mathbf{x}$ *is:*

$$\mathtt{pad}_{\mathbf{x}}(c) = |\{i \in \{1, \ldots, n\} \setminus \mathrm{Obs}, \exists \mathbf{v} \in \mathit{CVS} \text{ such that } v_i = c \text{ and } \mathbf{v} \approx_{\mathrm{Obs}} \mathbf{x} \text{ and } (\forall j \in \mathrm{Obs}.v_j = x_j)\}|.$$

The player anonymity degree of secret choice c *w.r.t. an observer set* Obs *is*

$$\mathtt{pad}(c) = \begin{cases} 0, & \text{if } \max_{\mathbf{x} \in \mathit{CVS}} \mathtt{pad}_{\mathbf{x}}(c) = 0 \\ \min_{\mathbf{x} \in \mathit{CVS}: \mathtt{pad}_{\mathbf{x}}(c) > 0} \mathtt{pad}_{\mathbf{x}}(c), & \text{otherwise} \end{cases}$$

The set in the definition of $\mathtt{pad}_{\mathbf{x}}$ is the set of all honest players that might, from the perspective of the intruder, have been assigned the choice c. We define the $\mathtt{pad}(c)$ to be the lowest non-zero value assuming that such a value exists. This is because, in some systems, it is possible for a given choice vector to rule out certain choices values, so making the minimum pad value zero, while at the same time any choice vector that allows the choice to happen may allow the choice to be assigned to a number of different players. For instance, if the intruder can see the total number of messages sent then a choice c that results in four messages being sent rules out a choice vector that would only send three messages. So $\mathtt{pad}(c)$ defines the anonymity for a choice c that results from only considering choice vectors that are compatible with c.

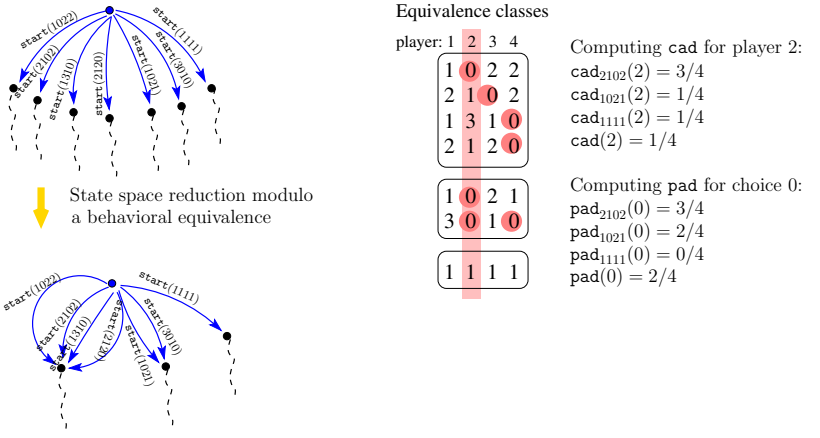

Fig. 1. Left: Computing the equivalence classes of $\approx_{\mathrm{Obs}}$, for a $\mathtt{Protocol}_{\mathrm{Obs}}$. The choices come from the set $\{0, 1, 2, 3\}$, there are 4 players and CVS is the set $\{1022, 2102, 1310, 2120, 1021, 3010, 1111\}$. The end state of every $\mathtt{start}(x)$ transition is the start state of $\mathtt{Protocol}_{\mathrm{Obs}}(x)$. After reduction modulo $\approx_{\mathrm{Obs}}$, the transition $\mathtt{start}(x)$ and $\mathtt{start}(y)$ have the same end state iff $x \approx_{\mathrm{Obs}} y$. The equivalence classes of $\approx_{\mathrm{Obs}}$, listed on the right, are generated in this way. **Right:** Extracting cad and pad information from the equivalence classes, by just applying the definitions.

We believe that these measures give a good impression of the anonymity provided by a protocol. In the rest of the paper, we will write cad and pad together with the actual number of possible choices (i.e., |Choices|) and with the number of players (n), respectively. For instance, we will write $\mathtt{cad}(2) = 3{:}5$ instead of $\mathtt{cad}(2) = 3$, if 5 is the size of the choices' domain. Note that the maximum possible cad or pad are not always of the form $m{:}m$; in fact, this is never the case whenever we consider Obs $\neq \emptyset$, since the intruder knows whether those participants belonging to Obs have performed a certain action and what choices those participants have made.

Verification method. Our goal is to make verification of anonymity properties just as easy as verification of safety or liveness properties. Two difficulties have to be overcome. Firstly, anonymity depends on the point of view of the intruder, therefore new protocol models should be written for every new observer set. Secondly, anonymity is not a properity of a single trace that can be written as a logic formula to be verified via model checking, but requires equivalence checking of several protocol instances. Note that both of these problems are specific to the general verification method of writing process specifications and then model checking temporal properties on the generated behaviour model. For instance, approaches based on epistemic logics [12,24] are able to express anonymity more naturally and do allow its verification by model checking, but they encounter other, mainly modelling, problems and are not supported by such powerful tools as we use here.

We solve these problems by automating the generation of new protocol specifications depending on the observer set and on the different protocol instances. We also support the analysis of anonymity as described above, by automatically generating the equivalence classes of $\approx_{\text{Obs}}$ and computing the anonymity degrees `cad` and `pad`.

All the tool support is available on our website [9]. We start from a base specification `Protocol(x)` describing the behaviour of the protocol for a parameter choice vector `x`. Then a *.rn* file will define, for a generic participant, how its actions are seen from the outside. We choose to implement ρ_R from (2) like this, rather than explicitly using it in the μCRL specification, in order to better control the effects of having a set of corrupted players Obs. The renaming is done by means of rules like `assign(i,x,true)->assign(i)`. This example rule says that the action of i of assigning a true value to its variable x will be observed by the other players and the environment only as an assignment executed by i. Just like the modelling process itself, choosing what the appropriate renamings should be is a subjective task. The actions executed by the observing parties are not renamed, while from the actions executed by the other, honest parties, all private information should be hidden. All information for action renaming are gathered into one *.rn* file, which will be used to automatically generate renamings for particular protocol instances. In order to avoid interferences with this automatically generated renamings, we require the P_i processes to not contain any further renaming operators. Then the tools will generate, for this given μCRL model `Protocol(x)`, the given set of renaming rules, a given set Obs and a given choice vector set CVS, the μCRL specification corresponding to

$$\sum_{v \in \text{CVS}: \forall j \in \text{Obs}. v_j = x_j} \texttt{start}(v).\texttt{Protocol}_{\text{Obs}}(v),$$

namely the sum of all protocol instances corresponding to choice vectors which are in CVS and coincide with `x` on the Obs positions. After that, the μCRL toolset is used to generate the state space of this new process and reduce it modulo a behavioural equivalence. The end states of the `start` actions are the start states of the protocol instances compared, therefore the equivalence classes of these states are exactly the equivalence classes of the relation $\approx_{\text{Obs}}$ on choice vectors (from Definition 1). See also Figure 1 for a scheme of this procedure. Our tools will show these equivalence classes and extract the choice and player anonymity degrees according to Definitions 2 and 3.

In Sections 3 and 4, we will apply our approach to the analysis of the Dining Cryptographers problem and the FOO 92 voting protocol, respectively. Our method can also be used to check the anonymity that protocols provide over a number of rounds, by using choices that represent the behaviour of a participant over a number of rounds. We have tested a simple possibilistic version of Reiter and Rabin's Crowds protocol [27] and shown that, over a number of rounds, an external observer cannot work out which nodes have originated messages, and the observer cannot work out how many of the messages were sent by the same node. Due to page limit, we omit the detailed analysis here. The interested readers can find the μCRL specification of this example online [9].

3 The Dining Cryptographers Problem

The Dining Cryptographers protocol is probably the most well-known example of a protocol where anonymity is the main requirement [7]. The story, which is a metaphor for anonymous broadcast, starts with three cryptographers sitting down at a table to have dinner together. At the end of their meal, they learn that the bill has been paid anonymously by one of them, or perhaps by a shadowy government organisation (the National Security Agency). They respect each other's right to anonymity, but they wish to find out whether the payer was the NSA or not. To achieve this, they come up with the following protocol: each neighbouring pair of cryptographers generates a shared bit, by flipping a coin; then each cryptographer computes the exclusive or (XOR) of the two bits shared with the neighbours, then announces the result - or the opposite result, if that cryptographer was the payer. The XOR of the three publicly announced results indicates whether the payer was an insider or the NSA.

The μCRL model. We specify the behaviour of a cryptographer as a μCRL process *Crypt* and the behaviour of the whole Dining Cryptographers system as a parallel composition of *Crypt*s. With three participants, the global process looks like this:

$$DC(\mathbf{x}{:}ChoiceVector) = \partial_{\{\mathtt{tell},\mathtt{recv}\}}(Crypt_0(x_0)||Crypt_1(x_1)||Crypt_2(x_2))$$

A choice is in this case the decision to pay or not (we will call it the paying bit), represented by the Boolean values $x_i \in \{\mathtt{T},\mathtt{F}\}$. A cryptographer process executes a series of actions corresponding to the three main steps of the protocol: decide whether to pay or not, flip the coins together with the neighbours, and announce the result of XOR-ing the two coins and the own paying bit. The first step is modelled as a statement $\mathtt{pay}(n,i,x_i)$. In other models of this protocol [28,3], the shared coins are represented by separate processes, but we merge the behaviour of ith coin with the behaviour of the ith cryptographer, in order to keep the number of processes small. That is, process $Crypt_i$ will execute a `flip` action and then share the result with the right hand neighbour, by executing an action `tell` while its right hand cryptographer in the ring can get to know the result of this coin flipping by executing the action `recv`. The synchronisation of these two actions results into the communication action `com`.

$$\begin{aligned} Crypt_i(x_i : Bool) = {} & \mathtt{pay}(n,i,x_i).\textstyle\sum_{coin\_left:Bool}(\mathtt{flip}(i,coin\_left).\\ & (\mathtt{tell}(next(i),coin\_left) \parallel \textstyle\sum_{coin\_right:Bool}\mathtt{recv}(i,coin\_right)).\\ & CryptAnnounce(n,0,id,ch \oplus coin\_right \oplus coin\_left)) \end{aligned}$$

CryptAnnounce models broadcasting the result of one's computation and receiving the results from all the others. Since the broadcast implementation is not an actual part of the protocol, we do not discuss *CryptAnnounce* here. The renaming rules specifying how much of a cryptographer's actions is visible for another cryptographer or the intruder are $\{\mathtt{com(i,X)} -> \mathtt{com(i)}, \mathtt{pay(n,i,X)} -> \mathtt{pay(i)}\}$. For

any given observers set Obs, DC_{Obs} will be obtained from the model of DC above, by applying the renaming rules to all $Crypt_i$ processes which are not in Obs, as explained in Section 2 (verification method).

Anonymity verification. Consider an external intruder observing a run of the Dining Cryptographers protocol with 3 participants and trying to conclude who the payer was (in case one of the cryptographers paid). Let us suppose that cryptographer 0 is the payer and let us check whether their anonymity will not be broken. For this, as described in Section 2, we automatically generate the state space corresponding to $\sum_{v \in \text{CVS}} \texttt{start}(v).DC_{\emptyset}(v)$, where CVS is in this case $\Pi(\texttt{TFF})$, the set of permutations of the sequence TFF, since it is a publicly known fact that there is exactly one payer among the cryptographers. Therefore, the above expression becomes $\texttt{start}(\texttt{TFF}).DC_{\emptyset}(\texttt{TFF}) + \texttt{start}(\texttt{FTF}).DC_{\emptyset}(\texttt{FTF}) + \texttt{start}(\texttt{FFT}).DC_{\emptyset}(\texttt{FFT})$. Note that we exclude FFF from CVS, because there is no anonymity claim for this case. The obtained state space will then be reduced modulo strong bisimulation equivalence and one equivalence class will result: $\{\texttt{FFT}, \texttt{FTF}, \texttt{TFF}\}$, meaning that the intruder cannot distinguish between the three possible choice vectors and thus considers that any of the three cryptographers might have been the payer. This situation of maximum anonymity is reflected both by the pad measure $\texttt{pad}(\texttt{T}) = 3{:}3$, and the cad measure $\texttt{cad}(0) = 2{:}2$; the first one says that any of the three players might be the owner of the T paying bit, and the second says that any of the two values T, F might have been assigned to cryptographer 0.

For 5 participants, two of which are corrupted (1 and 3), the state space for $\sum_{v \in \Pi(\texttt{TFFFF}): v_1 = v_3 = \texttt{F}} \texttt{start}(v).DC_{\{1,3\}}(\texttt{TFFFF})$ will automatically be generated and reduced, resulting into the equivalence classes $\{\texttt{FFFFT}, \texttt{TFFFF}\}$ $\{\texttt{FFTFF}\}$. Note that, consistent with the verification method explained in the end of Section 2, the vectors FTFFF and FFFTF are automatically excluded from this check, since it is already known to the intruder that these cannot be the case (because 1 and 3 show their secret paying bit F to the intruder). The computed anonymity degree $\texttt{pad}(\texttt{T}) = 1{:}5$, indicating that in at least one of the scenarios, the payer becomes known to the intruder (namely, when the choice vector is FFTFF). The anonymity degree restricted to the case when 0 pays $\texttt{pad}_{\texttt{TFFFF}}(\texttt{T}) = 2{:}5$, which is much lower than in the case of no corrupted players, indicating that, even if anonymity of 0 is not broken, the set of suspects is reduced to 2 players. The cad and pad degrees give the complete picture of the anonymity of the 3 honest cryptographers $\{0, 2, 4\}$ with respect to the coalition $\{1, 3\}$ of dishonest cryptographers, when cryptographer 0 pays. The conclusion is that 0 remains *partially* anonymous, that is the intruder doesn't know that 0 paid, but does know that one of $\{0, 4\}$ paid and 2 didn't.

4 The FOO 92 Electronic Voting Protocol

In this section, we analyse a more complex protocol, that involves choices (votes) from a larger than binary domain and elaborated cryptographic mechanisms like

anonymous channels, encryption and blind signatures. These mechanisms are very naturally expressible with the abstract datatypes of μCRL.

FOO involves voters, an administrator and a collector and has four stages: registration, voting, opening and counting. During the *registration stage*, a voter id_i prepares his ballot as follows: (1) he chooses a vote v_i and creates the ballot $x_i = \xi(v_i, k_i)$ using the secure bit-commitment ξ and the randomly chosen key k_i; (2) he computes the message $e_i = \chi(x_i, r_i)$ using the blinding technique χ and a random blinding factor r_i; (3) he signs $s_i = \sigma_i(e_i)$ and sends (id_i, e_i, s_i) to the administrator , who signs it $d_i = \sigma_A(e_i)$ and sends it back to the voter as his certification, if the voter is authorised to vote, otherwise the administrator rejects the signature. In the end of the registration stage, the administrator gets to know the number of eligible voters, and publishes the list of (id_i, e_i, s_i). During the *voting stage*, a voter id_i will perform the following steps: (1) he receives d_i and obtains the desired signature y_i of the ballot x_i using the unblinding technique $y_i = \delta(d_i, r_i)$; (2) if y_i is not the administrator's signature of x_i, he claims that (x_i, y_i) is not valid; otherwise (3) he sends (x_i, y_i) to the collector using an anonymous communication channel. The collector receives (x_i, y_i) and verifies the signature y_i of x_i using the administrator's verification key. If this succeeds, the collector enters (ℓ, x_i, y_i) onto a list as the ℓ-th item. After all voters have voted, the collector publishes the list. During the *opening stage*, each voter will send the key k_i and the number ℓ to the collector using an anonymous communication channel, if his vote is on the list, Otherwise, he claims this by revealing the valid ballot x_i and its signature y_i. Finally, during the *counting stage*, the collector opens the ballot x_i, obtains the vote v_i using k_i, counts the votes, and publishes the voting result.

The μCRL model. As in the case of the Dining Cryptographers problem, we will present the μCRL model of FOO at a rather abstract level and only give details on the interesting modelling points. The complete specification is available online [9].

We chose to model the clear and encrypted votes, as well as clear or blinded or signed ballots by one data type: *Data*, with the extension to lists *DataList*. The votes come from a set $V \subseteq Data$ of size N. We model bit commitment using classical symmetric-key encryption and we use the voter's index i to model both key k_i and the blinding factor r_i. This does not introduce problems, since we encode the various laws and restrictions corresponding to k_i, r_i as equations that the functions using them: *commit*, *open*, *blind*, *unblind* have to satisfy. For instance, the cancelling property of the signing procedure is captured by the equation $unblind(i, sign(blind(j, x))) =$ if $eq(i, j)$ then $sign(x)$ else err.

Each voter, the administrator and the collector are modelled as parallel processes communicating by pairs of synchronising actions like (`VfromA`, `AfromV`). The *administrator* waits for blinded ballots from the voters, signs them using the function *sign* and sends them back. We assume that checks by the administrator in the registration stage are always successful.

$$Admin = \sum_{i:Nat} \sum_{m:Data} \texttt{AfromV}(i, m).\texttt{AtoV}(i, sign(m)).Admin$$

A *voter* builds the ballot $x_i = commit(v_i, i)$, blinds it as $e_i = blind(x_i, i)$, and sends (i, e_i) to the administrator. Then he receives a signed ballot m from the administrator, retrieves the desired signature of his ballot using $unblind(id_i, m)$ and sends x_i with the signature to the collector. Then the voter waits for the collector to publish the final list and finally sends k_i (i.e. i) to the collector.

$$
\begin{aligned}
&Voter_i(v_i : Data) = \\
&\texttt{Vdecided}(v_i).\texttt{VtoA}(i, blind(i, commit(i, v_i))). \\
&\quad \textstyle\sum_{m:Data} \texttt{VfromA}(i, m).(\texttt{VtoC}(commit(i, v_i), unblind(i, m)). \\
&\qquad \textstyle\sum_{\ell:DataList} \texttt{VfromClist}(\ell).\texttt{VtoC}(find(\ell, unblind(i, m)), i).\delta \\
&\qquad \lhd\, issigned(unblind(i, m)) \rhd \delta)
\end{aligned}
$$

Note that the construct $\sum_{d:D} P(d) \lhd f(d) \rhd \delta$, with f being a Boolean function of d, forces choosing only the values of d that satisfy f. We model the *collector* with the knowledge of the number n of voters. The action $NVoters(n)$, below, indicates the end of the registration stage. He receives the committed votes as (x_i, y_i), adds them into a list ℓ and stops receiving when the counter ncv reaches n. Then he publishes ℓ by sending it ($CtoVlist$) to all voters. After receiving k_i for each item in the list ℓ, he opens the vote v_i and publishes the voting result by the actions $\texttt{numberof}(v)$ $(v \in V)$.

$$
\begin{aligned}
&Collector = \textstyle\sum_{n:Nat} \texttt{NVoters}(n).Collecting([], n, 0) \\
&Collecting(\ell : DataList, n : Nat, ncv : Nat) = \\
&\quad \textstyle\sum_{x_i:Data} \sum_{y_i:Data} \texttt{CfromV}(x_i, y_i).Collecting(add(x_i, y_i, \ell), n, ncv + 1) \\
&\quad \lhd\, ncv < n \wedge issigned(y_i) \rhd \delta + Opening(\ell, n, 0) \lhd ncv > n \rhd \delta \\
&Opening(\ell : DataList, n : Nat, nv : Nat) = \\
&\quad \texttt{CtoVlist}.Opening(\ell, n, nv + 1) \lhd nv < n \rhd Opening2(\ell, n, 0) \\
&Opening2(\ell : DataList, n : Nat, nv : Nat) = \\
&\quad \textstyle\sum_{label:Nat} \sum_{k_i:Nat} \texttt{CfromV}(label, k_i).Opening2(openvote(\ell, label, k_i), n, nv + 1) \\
&\quad \lhd\, nv < n \rhd Counting(\ell, 0) \\
&Counting(\ell : DataList, v : Nat) = \\
&\quad \texttt{numberof}(v, count(\ell, v)).Counting(\ell, next(v)) \lhd v \in V \rhd \delta.
\end{aligned}
$$

The collector waiting for all n ballots before publishing ℓ is essential for the correctness of the protocol. As noted in [19], the anonymity is broken if the collector were allowed to publish ℓ and continue interacting with the voters before actually having completed collecting all ballots. (In [19], the problem is addressed by introducing synchronisation points between voters after the registration stage. This is equivalent to forcing the collector to wait, because the voters cannot continue without the cooperation of the collector.) We reproduced this problem by modelling a bad collector, which does not wait (see the file `voting.mcrl` at [9]), but we do not insist on this modelling detail here, since our goal is rather to illustrate the anonymity measures on a correct model of the FOO protocol.

Anonymity verification. Let V be the set of possible votes. Then CVS, the allowed set of choice vectors, which is in fact the allowed set of vote outcomes, will

be V^n. Note that since the outcomes become public in the end of the protocol, and thus known to the intruder, the equivalence classes of $\approx_{\text{Obs}}$ will be subclasses of the permutation equivalence relation. Note also that for an external observer, $\texttt{cad}(i) = 1{:}n$, because a choice vector for a unanimous election leaves no doubt as to how anyone voted. So, for this protocol, it is more informative to look at particular instances of choice vectors ($\mathbf{x}$) and evaluate $\texttt{cad}_{\mathbf{x}}(v)$ and $\texttt{pad}_{\mathbf{x}}(i)$. In the bottom table of Figure 2, a number of experiments are shown, involving various numbers of voters and various vote vectors. The votes are taken from the set $\{0, 1, 2\}$.

In the Dining Cryptographers problem, the question was "who pays?", translated to "which of the players has the choice T?"; $\texttt{pad}_{\texttt{TFF}}(\texttt{T})$ was in that case the appropriate anonymity measure, since it indicates the number of players (worst case: 1) who, according to the intruder, might possibly be associated with the choice T. In voting protocols however it is more relevant how many different votes the intruder might associate to a given voter, so $\texttt{cad}(\textit{voter})$ is a more realistic measure of anonymity. It is possible to get $\texttt{pad}_{\mathbf{x}}(v) > 1$ and $\texttt{cad}_{\mathbf{x}}(i) = 1$, for a voter i who voted v; this indicates that player anonymity is not a sensitive enough measure, while choice anonymity detects that the intruder precisely knows which candidate voter i chose. This situation is illustrated by $\mathbf{x} = 1121, \text{Obs} = \{2\}, i = 0, v = 1$.

5 Experiences with the Distributed Toolset

In order to assess the efficiency of our approach, we ran some experiments with both examples, various number of players and various coalition sizes. We generated and reduced the state spaces both with the sequential and the distributed tools. For the latter, a cluster with 16 machines (32 processors) was used.

Our initial DC specification gives a very realistic model of the protocol, allowing maximal action interleaving and including a handshake implementation of the final broadcast announcements. Unfortunately, this leads to a fast explosion of the state space, limiting the number of cryptographers that can be handled to 10. Note however, that this is already a larger instance than ever been formally analysed before; Schneider and Sidiropoulos [28] analyse four players, and the maximum we could find in the literature was 8, by means of symbolic epistemic model checking [20]. In order to deal with even larger instances in our explicit tool, we also experimented with a simpler DC model, where an order is imposed on the cryptographers' actions, and a synchronisation of all cryptographers takes place between the three protocol phases (flipping the coins, sharing the coins, broadcasting the XOR results). This requires some more computing effort during generation (and thus more time), but the state spaces resulted are much smaller, and therefore verification of instances with 17 players and more can be achieved. This simplified version occurs in Figure 2 as *DCso*.

Note also that using the distributed tool is not always more efficient. For small state spaces like DC11so, communication between the machines consuming much of the overall time and therefore the sequential tool actually performs better.

	Size	Size after red.	Time	cad(0)	pad(T)
DC3, Obs = $\emptyset$	229 469	65 112	1.5s	2:2	3:3
DC3, Obs = $\{1\}$	184 362	71 132	1.3s	2:2	2:3
DC5, Obs = $\{1, 3\}$	5189 14679	1620 4567	4.9s	2:2	2:5
DC7, Obs = $\emptyset$	185 769 695 551	27 180 85 763	8m53s	2:2	7:7
DC9, Obs = $\emptyset$	5 194 659 22 961 789	1 034 142 4 088 977	(s) - (db) 7h5m	2:2	9:9
DC10, Obs = $\emptyset$	27 436 022 130 031 220	5 002 490 21 535 547	(db) 17h20m	2:2	10:10
DC11so, Obs = $\emptyset$	33 876 41 035	6 156 7 188	(s) 6m (db) 11m	2:2	11:11
DC12so, Obs = $\{1, 3, 5, 7, 9, 11\}$	58 749 67 612	17 467 21 219	(s) 7m	1:2	1:12
DC15so, Obs = $\emptyset$	606 388 721 067	98 320 114 716	(s) 7h2m (db) 44m	2:2	15:15
DC17so, Obs = $\emptyset$	2 556 144 3 014 887	393 234 458 784	(s) - (db) 5h40m	2:2	17:17

	Size	Size after red.	Time	cad	pad
FOO4, Obs = $\{2\}$	58 749 67 612	17 467 21 219	17s	$\mathtt{cad}_{1121}(0) = 1{:}3$	$\mathtt{pad}_{1121}(1) = 3{:}4$
FOO6, Obs = $\{2\}$	3 423 841 10 518 810	29 451 92 835	22m36s	$\mathtt{cad}_{010102}(0) = 3{:}3$	$\mathtt{pad}_{010102}(1) = 5{:}6$
FOO7, Obs = $\{2\}$	65 282 690 221 299 564	3 676 249 9 628 686	(db) 4h48m	$\mathtt{cad}_{0101022}(0) = 3{:}3$	$\mathtt{pad}_{0101022}(1) = 6{:}7$

Fig. 2. Experiments with various protocol and coalition instances. The sizes are given as pairs (no. states, no. transitions). The times include both state space generation and reduction. *(db)* marks that the distributed toolset was used. *(s)* - marks that on a single-machine, the state-space generation ran out of memory or didn't stop within 10 hours. For the **FOO** experiments, votes are taken from the set $\{0, 1, 2\}$.

6 Conclusions

Whether a protocol satisfies an anonymity requirement depends not only on the protocol itself, but also on the particular scenarios in which the protocol is used. Namely, there are two influential factors: the strength of the intruder (that is, if/how many participants it has corrupted) and the exact data or actions that need to be protected (in DC, if the NSA pays, the intruder learns that no cryptographer paid; in voting protocols, if the vote is unanimous, there is obviously no choice anonymity). We captured these observations into two definitions of anonymity degrees, parameterised with the aforementioned factors. *Player anonymity* measures the number of participants that the intruder might

consider guilty for a given event, and *choice anonymity* measures the number of different pieces of data or events that the intruder might consider as belonging to or being generated by a given participant. We have built tool support for tailoring generic protocol specifications to particular instantiations and coalitions of corrupted participants.

We demonstrated the use of a modern powerful verification toolset, μCRL, to automatically check the anonymity of the generated models. Due to the distributed state space generation and reduction tools, we were able to analyse large instances of known protocols.

Our definitions of anonymity may be based on any equivalence relation. One interesting future direction may be to attempt to model probabilistic systems using our definitions with a probabilistic bisimulation. Finding the right definition of probabilistic bisimulation would be key to making this work. We know of no tools for checking probabilistic bisimulation, so automatically calculating probabilistic anonymity degrees would not be trivial.

We model intruders by making their private communications visible to the attacker. This is done by removing the actions that the intruders see from the sets of actions that are hidden and renamed. We do not remove any actions from those that are encapsulated, i.e., those that are forced to synchronise. If we did remove names from this set too then the communications that were forced to happen with the intruder could instead happen with the outside environment. This would allow us to model active inside attackers that deviate from the protocol. However as actions can in general synchronise with a range of other actions care would have to be taken to ensure that the communications that the attacker is given access to are exactly those that are used by the intruder.

Another possible extension of our framework would be to automatically search for the worst anonymity for a fixed number of intruders. For instance, in the Dining Cryptographers protocol the intruders can be much more effective when spaced out, rather than when they are direct neighbours. We could generate all possible placements of intruders and then test each system to find the lowest anonymity degrees. Such an analysis might help to identify weak points in anonymity protocols that could be strengthen to make inside attacks harder.

The μCRL toolset includes state space optimisation tools that we haven't yet taken into account. Confluence reduction, for instance, has been successfully employed in keeping state spaces manageable [5] and might be useful in our case as well.

References

1. Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theoretical Computer Science 37(1), 77–121 (1985)
2. Berthold, O., Pfiztmann, A., Standtke, R.: The disavantages of free mix routes and how to overcome them. In: Federrath, H. (ed.) Proc. Workshop on Design Issues in Anonymity and Unobservability. LNCS, vol. 2009, pp. 30–45. Springer, Heidelberg (2001)

3. Bhargava, M., Palamidessi, C.: Probabilistic anonymity. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 171–185. Springer, Heidelberg (2005)
4. Blom, S.C.C., Fokkink, W.J., Groote, J.F., van Langevelde, I., Lisser, B., van de Pol, J.C.: μCRL: A toolset for analysing algebraic specifications. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 250–254. Springer, Heidelberg (2001)
5. Blom, S.C.C., Groote, J.F., Mauw, S., Serebrenik, A.: Analysing the BKE-security protocol with μCRL. In: Proc. 6th AMAST Workshop on Real-Time Systems. ENTCS, vol. 139, pp. 49–90 (2004)
6. Blom, S.C.C., Orzan, S.M.: A distributed algorithm for strong bisimulation reduction of state spaces. Software Tools for Technology Transfer 7(1), 74–86 (2005)
7. Chaum, D.: The dining cryptographers problem: Unconditional sender and recipient untraceability. Journal of Cryptology 1, 65–75 (1988)
8. Chothia, T.: Analysing the mute anonymous file-sharing system using the pi-calculus. In: Najm, E., Pradat-Peyre, J.F., Donzeau-Gouge, V.V. (eds.) FORTE 2006. LNCS, vol. 4229, pp. 115–130. Springer, Heidelberg (2006)
9. Chothia, T., Orzan, S.M., Pang, J.: μCRL specifications. `http://www.win.tue.nl/~sorzan/anonymity`
10. Deng, Y., Palamidessi, C., Pang, J.: Weak probabilistic anonymity. In: Proc. 3rd Workshop on Security Issues in Concurrency (2005)
11. Díaz, C., Seys, S., Claessens, J., Preneel, B.: Towards measuring anonymity. In: PET 2002. LNCS, vol. 2482, pp. 54–68. Springer, Heidelberg (2002)
12. van Eijck, J., Orzan, S.M.: Epistemic verification of anonymity. In: Proc. Views On Designing Complex Architectures (VODCA'06) (2006)
13. Fujioka, A., Okamoto, T., Ohta, K.: A practical secret voting scheme for large scale elections. In: AUSCRYPT 1992. LNCS, vol. 718, pp. 244–251. Springer, Heidelberg (1992)
14. Garcia, F.D., Hasuo, I., Pieters, W., van Rossum, P.: Provable anonymity. In: Proc. 3rd ACM Workshop on Formal Methods in Security Engineering, pp. 63–72. ACM Press, New York (2005)
15. Groote, J.F., Reniers, M.A.: Algebraic process verification. In: Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, North-Holland, pp. 1151–1208 (2001)
16. Halpern, J.Y., O'Neill, K.R.: Anonymity and information hiding in multiagent systems. Journal of Computer Security, 483–514 (2005)
17. Hughes, D., Shmatikov, V.: Information hiding, anonymity and privacy: A modular approach. Journal of Computer Security 12(1), 3–36 (2004)
18. Hüttel, H., Shukla, S.: On the complexity of deciding behavioural equivalences and preorders - a survey. Technical Report RS-96-39, BRICS (1996)
19. Kremer, S., Ryan, M.: Analysis of an electronic voting protocol in the applied pi-calculus. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 186–200. Springer, Heidelberg (2005)
20. Lomuscio, A., Raimondi, F.: MCMAS: A tool for verifying multi-agent systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006 and ETAPS 2006. LNCS, vol. 3920, pp. 450–454. Springer, Heidelberg (2006)
21. Mauw, S., Verschuren, J., de Vink, E.P.: A formalization of anonymity and onion routing. In: Samarati, P., Ryan, P.Y A, Gollmann, D., Molva, R. (eds.) ESORICS 2004. LNCS, vol. 3193, pp. 109–124. Springer, Heidelberg (2004)
22. Mauw, S., Verschuren, J., de Vink, E.P.: Data anonymity in the FOO voting scheme. In: Proc. Views On Designing Complex Architectures (VODCA'06) (2006)

23. Meritt, M.J.: Cryptographic Protocols. PhD thesis, Georgia Institute of Technology (1983)
24. van der Meyden, R., Su, K.: Symbolic model checking the knowledge of the dining cryptographers. In: Proc. 17th IEEE Computer Security Foundations Workshop, pp. 280–291. IEEE Computer Society Press, Los Alamitos (2004)
25. Pang, J.: Analysis of a security protocol in μCRL. In: George, C.W., Miao, H. (eds.) ICFEM 2002. LNCS, vol. 2495, pp. 396–400. Springer, Heidelberg (2002)
26. Pfitzmann, A., Hansen, M.: Anonymity, unobservability, and pseudonymity: A proposal for terminology, draft v0.23 (August 2005)
27. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for Web transactions. ACM Transactions on Information and System Security 1(1), 66–92 (1998)
28. Schneider, S., Sidiropoulos, A.: CSP and anonymity. In: Martella, G., Kurth, H., Montolivo, E., Bertino, E. (eds.) Computer Security - ESORICS 96. LNCS, vol. 1146, pp. 198–218. Springer, Heidelberg (1996)
29. Serjantov, A., Danezis, G.: Towards an information theoretic metric for anonymity. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 41–53. Springer, Heidelberg (2003)
30. Shmatikov, V.: Probabilistic model checking of an anonymity system. Journal of Computer Security 12(3/4), 355–377 (2004)