CSP and anonymity

Steve Schneider and Abraham Sidiropoulos

Department of Computer Science
Royal Holloway, University of London
Egham, Surrey TW20 0EX

email: {steve,abraham}@dcs.rhbnc.ac.uk

Abstract. Security protocols are designed to meet particular security properties. In order to analyse such protocols formally, it is necessary to provide a formal definition of the property that they are intended to provide. This paper is concerned with the property of anonymity. It proposes a definition of anonymity within the CSP notation, discusses the approach taken by CSP to anonymity with respect to different viewpoints, and illustrates this approach on some toy examples, and then applies it to a machine-assisted analysis of the dining cryptographers example and some variants.

1 Introduction

The notion of anonymity is used in a wide variety of situations, from anonymous donations to anonymous transactions. Computer systems may be used to support anonymity, but the users have to be confident that their anonymity requirements are actually provided by the system.

This paper aims to provide the foundations of a process algebraic approach to analysing systems with regard to anonymity properties. Such an approach focuses on the interactions between system components and is appropriate for the analysis and verification of protocols designed to achieve these properties. This is in contrast to mathematical characterisations such as that of [Wai90], where attention is focused on the information contained in outputs of communication rounds. This paper fits within the general framework described in [PfW94], where the authors identify the need to discuss properties in terms of the sequences of interactions (traces) possible at the interface between the system and the users. It fits in with the aims of [Sch96] to define a number of security properties within CSP, providing a uniform framework for describing and analysing protocols and their properties.

The principal intention of this paper is to describe the use of CSP to define anonymity properties and to analyse anonymity protocols. CSP is an appropriate formal method for describing and analysing communications protocols because it is designed to describe systems in terms of components which interact by means of message passing. To this end it is important firstly to understand the concept of anonymity by examining the way it is generally used. Once a formal definition has been provided, it is explored and refined by applying it to known situations and confirming that the diagnosis provided by the definition corresponds to what is expected. It can then be used in the analysis of situations which are not already well understood, in order to see how it provides feedback and clarifies understanding.

This paper is structured as follows: Section 2 investigates the nature of anonymity; Section 3 introduces the relevant CSP notation and theory; Section 4 formulates a CSP definition of anonymity which aims to capture the concept, and illustrates the definition by applying it to a variety of simple situations. Notions of abstraction, crucial for the consideration of anonymity with respect to different viewpoints, are also discussed; Section 5 explores and illustrates these definitions on the well-understood Dining Cryptographers anonymity protocol, and discusses how the FDR model-checking tool [FSEL94] provides feedback during protocol analysis.

2 Anonymity notions

If we are to analyse anonymity as a security property it is crucial to define it in a precise way. There are many real life activities which may be done anonymously: examples include donating money, publishing poems, sending mail, voting, informing the police, and posting to bulletin boards. A formal definition should be applicable to this wide variety of situations.

A natural question that arises is whether anonymity is a property of events and messages or a property of agents. The scenarios described above suggest that the anonymity involved in a particular message or event is a property of the agents associated with that event or message. For example, in a specific voting situation where members of a party voted to elect a new leader we might have as a requirement that the voter associated with any particular vote should be anonymous. In another example, if someone informed the police, the informer would like to hide his identity. It is also the case that the police would like in some cases to hide the nature of the information itself. It seems that the hidden information itself (in contrast to the identity of the informant) would be better considered as confidential rather than anonymous. In this case, we use *confidentiality* to refer to messages whose content is to be kept secret, and *anonymity* to refer to messages whose originator or recipient is to be kept secret.

We can identify various aspects of anonymity:

1. It can be provided to agents as in the discussion above, where an agent wishes to hide his identity. In an anonymous mail, poem, donation, or informing police one wants to be anonymous.
2. Viewing the world from particular viewpoints, one may have anonymity with respect to some information but not with respect to other (more privileged) information. For example, with regard to an anonymous donation, the organisation receiving the donation may know the identity of the donor even if the general public does not. Hence the anonymity of the donor will be with respect to the absence of particular privileged information. Furthermore, the

anonymity is with respect to the relationship between the donor and the donation. The identity of the donor may be known in other contexts, but it is the fact that the connection between the donor and the donation is hidden which provides anonymity.

An issue that arises from this concerns who has control over withholding the particular privileged information that is required to provide anonymity. Chaum is concerned with this distinction, and has proposed protocols in which the agent himself is in possession of that information. He has written a number of papers [Cha85, Cha88] on the subject in which two main kinds of transaction are identified: payments, and credentials transactions. In Chaum's electronic cash scheme for example, the author promises anonymity of the digital coins user so that the bank cannot associate a payment with the payer without their consent. The argument of Chaum is that his system permits fraud detection and transaction tracing with the consent of the individual. His system also addresses other problems such as double spending of the same coin.

We can use credentials as another example of demonstrating anonymity. Credentials are usually needed to prove one's credibility and identity. With untraceable credentials using pseudonyms one's credibility is proven without divulging the identity. When one can have only one pseudonym per organisation the problem of double identity is also addressed. Additionally, if one uses different pseudonyms for different organisations nobody can trace him and with a suitable implementation the organisations can be convinced they are dealing with the correct individual. This saves the individual the trouble of giving potentially sensitive information in order to prove its identity. Both examples are published in [Cha85].

This paper is concerned with providing a formal definition which may be applied to this wide variety of situations. Although there are a variety of anonymity protocols, often the property which the protocol aims to guarantee is not explicitly defined. Formal definition provides the starting point for formal analysis. Anonymity protocols can then be described in CSP, and the resulting system can be analysed to show that the anonymity property is present.

3 CSP notation

CSP is an abstract language designed specifically for the description of communication patterns of concurrent system components that interact through message passing. It is underpinned by a theory which supports analysis of systems described in CSP. It is therefore well suited to the description and analysis of network protocols: protocols can be described within CSP, as can the relevant aspects of the network. Their interactions can be investigated, and certain aspects of their behaviour can be verified through use of the calculus. This section introduces the notation and ideas used in this paper. In particular, only the trace model for CSP is used here. For a fuller introduction to the language the reader is referred to [Hoa85].

Events Systems are modelled in terms of the events that they can perform. The set of all possible events (fixed at the beginning of the analysis) is denoted Σ. Events may be atomic in structure or may consist of a number of distinct components. For example, an event $put.5$ consists of two parts: a channel name put, and a data value 5. An example of events used in this paper are those of the form $look.i.j.v$ consisting of a channel $look$, the first participant i, the second participant j, and the value v being communicated. This may be thought of either as a channel $look$ which passes messages consisting of three components, or as a collection of channels $look.i.j$ which each pass a single component message. The CSP model treats these identically, though in this paper we will prefer to think in terms of the second possibility. If M and N are sets of messages, then $M.N$ will be the set of messages $\{m.n \mid m \in M \wedge n \in N\}$. If m is a single message then we elide the set brackets and define $m.N$ to be $\{m\}.N$. Thus for example the set of events $i.N.m = \{i.n.m \mid n \in N\}$. A channel c is said to be of type M if for any message $c.m \in \Sigma$ it is the case that $m \in M$; and for any $m \in M$ it is the case that $c.m \in \Sigma$.

Processes Processes are the components of systems. They are the entities that are described using CSP, and they are described in terms of the possible events that they may engage in. The process $STOP$ is the process that can engage in no events at all. If P is a process then the process $a \rightarrow P$ is able initially to perform only a, following which it will behave in the way described by P. The process $P \,\Box\, Q$ (pronounced 'P choice Q') can behave either as P or as Q: its possible communications are those of P and those of Q. An indexed form of choice $\Box_{i \in I} P_i$ is able to behave as any of its arguments P_i.

Processes may also be composed in parallel. If A is a set of events then the process $P \,|[\, A \,]|\, Q$ behaves as P and Q acting concurrently, with the requirement that they have to synchronise on any event in the synchronisation set A: in other words, any event in the set A can be performed only when both P and Q are simultaneously able to perform it, and they both participate in its occurrence. Events not in A may be performed by either process independently of the other. A special form of parallel operator in which the two components do not synchronise on any events is $P \,|||\, Q$ which is equivalent to $P \,|[\, \{\} \,]|\, Q$.

Events occurring in process descriptions may be renamed by use of an event renaming function $f : \Sigma \rightarrow \Sigma$. The process $f(P)$ performs the event $f(a)$ whenever P would perform a. The process $f^{-1}(P)$ can perform any event from the set $f^{-1}(a)$ whenever P can perform a.

Processes may be recursively defined by means of equational definitions. Process names must appear on the left hand side of such definitions, and CSP expressions which may include those names appear on the right hand side. For example, the definition

$$LIGHT = on \rightarrow off \rightarrow LIGHT$$

defines a process $LIGHT$ whose only possible behaviour is to perform on and off alternately.

Traces For the purposes of this paper we restrict attention to the trace semantics for CSP. This semantics associates a process P with the set of (finite) sequences of events ($traces(P)$) that it may possibly perform. Examples of traces include $\langle\rangle$ (the empty trace, which is possible for any process) and $\langle on, off, on\rangle$ which is a possible trace of $LIGHT$.

Analysing processes A process P is refined by a process Q (written $P \sqsubseteq Q$) if $traces(Q) \subseteq traces(P)$. This means that if P meets a specification then Q will also meet it. It also allows CSP processes to act as specifications: Q meets the specification P if it a refinement of it.

Model-checking techniques allow the refinement relation $P \sqsubseteq Q$ to be checked mechanically (for finite-state processes). There are a number of tools that have been designed to support model-checking. We will use the tool FDR which has been designed specifically for analysis of CSP processes. It takes two processes P and Q as input, and either confirms that Q is a refinement of P, or provides a witness trace tr which is a trace of Q but not of P (which is concrete evidence that $traces(Q) \not\subseteq traces(P)$). The trace tr is useful in debugging Q, since it contains information as to a behaviour of Q that is disallowed by the specification P.

Since two processes are equal if each refines the other, equality of processes can be checked by checking $P \sqsubseteq Q$ and $Q \sqsubseteq P$. The definition of anonymity presented below will require that a process P is equal to another process Q dependent on P. The tool FDR will allow automatic checking for this equality.

4 Formalisation

The point of formalisation is to allow a better analysis of the real-world situation. It is therefore necessary to translate the various aspects involved in anonymity into the formal method. In particular, the CSP approach should be able to model identities of agents, the various ideas of viewpoints of agents on the system, and the idea of sensitive information. Furthermore, the results of the analysis should provide feedback at the real-world level, in the sense that it should provide information concerning why anonymity does not hold in particular cases.

Anonymity is concerned with protecting the identity of agents with respect to particular events or messages. The messages themselves need not be protected. Hence it is natural to consider events in the system under analysis as consisting of two components: the identity of the agent performing that event, and the content itself. For anonymity, we consider events of the form $i.x$, where i is the identity of the agent, and x is the content of the event.

The point of anonymity is that a message that could have originated from one agent could equally have originated from any other (perhaps any other from some set of users). Hence we wish our definition to capture the notion that any message of the form $i.x$ could equally well have been of the form $j.x$. If the set $USERS$ consists of the set of all users whose identities should be masked by the system in providing anonymity, then the set of messages we wish to confuse for a given piece of information x is given by the set A:

$A = \{i.x \mid i \in \mathit{USERS}\}$

Rather than talk directly about the identity of users, we can capture anonymity by requiring that whenever any event from the set A occurs, it could equally well have been any other event. In terms of agent identity and content, this means that if an observer has access only to the content of the message then it is not possible to deduce the identity of the agent associated with it.

This may be encapsulated in an equation for the system P

Definition 1 A process P is *strongly anonymous* on an alphabet A if:

$$f_A^{-1}(f_A(P)) = P$$

where equality is with respect to traces, and

$f_A(x) = \alpha$ if $x \in A$
$f_A(x) = x$ if $x \notin A$

where $\alpha \notin \Sigma$

This definition states that if every occurrence of every event from A were renamed to some new dummy event α (thus considering all events from A to be equivalent) which is the situation in the process $f_A(P)$, then whenever an α is possible in this renamed process, any possible event from A should have been possible in the original process. The process $f_A^{-1}(Q)$ makes every event from A available whenever α is available in Q, so $f_A^{-1}(f_A(P))$ makes all events from A available whenever any such event is possible. The equation states that this process is identical to the original process P, which means that the process P makes all events in A available whenever any of them is.

Consequences of the definition A number of aspects of anonymity follow immediately from the definition:

1. If P is anonymous on both A and A', and $A \cap A' \neq \varnothing$ then P is anonymous on $A \cup A'$
2. If P is anonymous on A and $A' \subseteq A$ then P is anonymous on A'
3. Anonymity is not preserved by CSP refinement with respect to nondeterminism.

From these properties it can be seen that if P provides anonymity for A then it need not follow that some event from A *must* have occurred whenever any of them could have occurred. For example, if P provides anonymity for the set $\{0.\mathit{gives}, 1.\mathit{gives}\}$ and in some situation it was possible that $0.\mathit{gives}$ occurred, it need not be the case that either $0.\mathit{gives}$ or $1.\mathit{gives}$ must have occurred; it is also possible that some other event (such as $2.\mathit{gives}$) could have occurred, or even no such event at all. Anonymity on a set simply means that events from that set should be indistinguishable in the sense that if one could have occurred then so could any—it does not mean that this should be a maximal set.

Illustration of the definition As an example, consider a charity which accepts donations. In fact there are only two possible donors, and only one of them will provide a donation. If donor 0 offers to give, then he always gives £ 5; if donor 1 offers to give, then she always gives £ 10[1]. The charity always announces its thanks publicly (in the form 'we have received a donation'). This setup is described by the process $EX0$.

$$\begin{aligned} EX0 = {} & 0.gives \rightarrow £5 \rightarrow thanks \rightarrow STOP \\ & \Box\ 1.gives \rightarrow £10 \rightarrow thanks \rightarrow STOP \end{aligned}$$

The donors require anonymity concerning who decides to *give*. In other words, anonymity is required for the set $A = \{0.gives, 1.gives\}$.

To see whether this setup provides anonymity, we have to consider whether $f_A^{-1}(f^A(EX0)) = EX0$. In fact

$$\begin{aligned} f_A^{-1}(f_A(EX0)) = {} & 0.gives \rightarrow £5 \rightarrow thanks \rightarrow STOP \\ & \Box\ 0.gives \rightarrow £10 \rightarrow thanks \rightarrow STOP \\ & \Box\ 1.gives \rightarrow £5 \rightarrow thanks \rightarrow STOP \\ & \Box\ 1.gives \rightarrow £10 \rightarrow thanks \rightarrow STOP \end{aligned}$$

which has different traces to $EX0$. One of the traces it has is $\langle 0.gives, £10 \rangle$ which is not possible for $EX0$. This indicates that the occurrence of the event £10 allows a distinction to be made between different events in A, and so the system does not provide anonymity. This situation corresponds to the scenario where the donors disguise themselves (so as not to be identified) but all other events are public.

Observation

The definition given above requires that any event from A should be made available whenever any of them is. From the point of view of a possible observer, this is intended to ensure that whenever the observer can *deduce* that one of the events was performed, then no knowledge is obtained about which event it was. The observer is able to make such deductions from the information which is available in the form of seeing events which the system has performed.

Anonymity is often with respect to particular observers or particular viewpoints. In other words, anonymity is provided in cases where an observer has access only to certain kinds of information, and might not be provided in cases where more information is available. For example, a donation to a charity would be anonymous if the only information available is details of the amounts of money passing through particular accounts, but might not be anonymous if all details of particular transactions are available.

[1] Names of donors have been removed to protect their identities

In general, an observer does not have complete access to all of the events occurring in a system, but has only limited or no direct access to some events. The events that an observer has access to could be captured as another set B.

It is an immediate requirement for anonymity that $A \cap B = \varnothing$. If an observer has direct access to the very events that we wish to mask, then it will always be possible to tell some events in A (in particular, those also in B) from some others.

The events that are not in A or B are those events that the observer does not have direct access to. From the point of view of modelling the system in order to analyse for anonymity, the other events should be *abstracted*, since the system to be analysed for anonymity should encapsulate the information available to the observer. There are a number of forms of abstraction, corresponding to various ways in which events can be hidden from the observer. CSP contains a number of abstraction mechanisms.

For example, in the process $EX0$ above an observer might have access only to event *thanks*. In this case $B = \{thanks\}$ and the other events, £5 and £10, should be abstracted away before analysis begins.

If C is the set of events that are to be abstracted from P, then the system to be analysed is $ABS_C(P)$, where ABS_C is one of the abstraction mechanisms to be discussed below. In each case the requirement will be to check

$$f_A(f_A^{-1}(ABS_C(P))) = ABS_C(P)$$

Hiding The most straightforward form of abstraction is *hiding*. In the process $P \setminus C$ (pronounced 'P hide C') all events in the set C that P is able to perform now become internal events, and their performance will no longer be visible to Ps environment.

If a set of events is hidden then they are entirely internal to the process, and observers obtain no direct information about their occurrence. It may of course be possible to deduce such information from the occurrence of visible events.

This form of abstraction would be used in cases where the observer is completely unable to observe the occurrence of events outside B.

In the case of $EX0$ this means that the occurrence of the cash transaction is completely invisible to an observer.

This form of abstraction yields the following analysis

$$\begin{aligned} EX0 \setminus C = {} & 0.gives \rightarrow thanks \rightarrow STOP \\ & 1.gives \rightarrow thanks \rightarrow STOP \end{aligned}$$

It is the case that $EX0 \setminus C$ satisfies the anonymity equation, that

$$f_A^{-1}(f_A(EX0 \setminus C)) = EX0 \setminus C$$

and so the system does provide anonymity on A when £5 and £10 are invisible to the observer. This is exactly what we would expect—all the observer can now see is the occurrence of event *thanks*, and this provides no information about which of the two possible initial events was actually performed.

Another example is provided by $EX1$, where donor 1 gives two donations.

$$\begin{aligned} EX1 = {} & 0.gives \rightarrow £5 \rightarrow thanks \rightarrow STOP \\ & \quad \Box\ 1.gives \rightarrow £10 \rightarrow £10 \rightarrow thanks \rightarrow STOP \end{aligned}$$

This also provides anonymity when the set C is hidden. Although the second choice will lead to more activity, this will be completely hidden from the observer. In fact, $EX1 \setminus C = EX0 \setminus C$, from which it follows immediately that $EX1 \setminus C$ satisfies the anonymity definition.

Renaming It is also possible that an observer might know that some event is occurring, but unable to detect which. For example, a donation might be posted in an envelope. It may be clear to an observer that a donation is occurring, but its nature remains unknown. This situation can be modelled using the same renaming operation that was used in the definition of anonymity. The process describing the information available to the observer is given as $f_C(P)$, where C is the set of events to be abstracted.

In the case of $EX0$ this form of abstraction yields the following

$$\begin{aligned} f_C(EX0) = {} & 0.gives \rightarrow envelope \rightarrow thanks \rightarrow STOP \\ & 1.gives \rightarrow envelope \rightarrow thanks \rightarrow STOP \end{aligned}$$

It is the case that $f_A^{-1}(f_A(f_C(EX0))) = f_C(EX0)$, and so the system does provide anonymity on A when all that is known about £5 and £10 is when one of them occurs (but not which one). Again, this is exactly what we would expect—all the observer can now see is the occurrence of an *envelope* followed by event *thanks*, and this provides no information about which of the two possible initial events was actually performed.

In the case of $EX1$ an observer does have some indication of the abstracted activity, but no direct information concerning its precise nature. We obtain

$$\begin{aligned} f_C(EX1) = {} & 0.gives \rightarrow envelope \rightarrow thanks \rightarrow STOP \\ & 1.gives \rightarrow envelope \rightarrow envelope \rightarrow thanks \rightarrow STOP \end{aligned}$$

This process does not meet the anonymity definition, since the sequence of events $\langle 0.gives, envelope, envelope \rangle$ is not a trace of this process, but it is a trace of the renamed process $f_A^{-1}(f_A(f_C(EX1)))$. This illustrates the fact that it is possible for anonymity to fail even without any direct information to the observer, if the abstraction mechanism is not strong enough. If the event *thanks* were also abstracted in $EX1$ so the observer does not even have access to any information directly, the process would still fail to provide anonymity, since the observer can detect the level of abstracted activity.

Masking Another third of abstraction, first presented in [Ros94], corresponds to events being *masked* by the occurrence of other events which act as noise. In the charity example we consider the possibility that there are other donations flowing in, so in the case of any particular donation it is unclear whether it is from one of the donors we are interested in or whether it is from somewhere else. Events are observed but it is unclear whether they are events from the process of interest, or whether they are other events.

This form of abstraction is described by running the process P in parallel with $RUN(C)$, the process which can always perform any events from the set C. The resulting description $P \mid\mid\mid RUN(C)$ is always able to perform an event from C, so an observer cannot know whether any particular occurrence of such an event was due to P or due to $RUN(C)$. $RUN(\{£5, £10\})$ models the situation of donations occurring from sources other than those we are considering. The fact that *thanks* is not also in the set indicates that *thanks* can occur only in response to the donors of interest. If *thanks* could also occur elsewhere, then that would also be contained in the set.

In this situation, the observer is in a position to know that P has *not* performed any events from C in the case where none have been performed. In any case, an observer will have an upper bound on the number of events from C that P has performed at any point—the total number of events from C that have been observed.

In the case of $EX0$, the situation is that donations come in from a variety of sources, and their values are known to an observer. The *thanks* event is not masked.

Analysis of $EX0$ in this situation reveals that anonymity is not provided in this case. The process $f_A^{-1}(f_A(EX0 \mid\mid\mid RUN(C)))$ contains the seqnece of events $\langle 0.gives, £10, thanks \rangle$ as a possible trace, though it is not possible for the process $EX0 \mid\mid\mid RUN(C)$. The reason anonymity breaks down is that although £5 and £10 events are masked, it is still possible for an observer to deduce that no such events have occurred, if none are performed by the masking process or by the process under consideration. Hence once a *thanks* event has occurred, if a £10 has occurred but no £5 events then it will be clear to the observer that $0.gives$ cannot have occurred. In such a situation, $0.gives$ is distinguishable from $1.gives$.

The same phenomenon also appears in $EX1$.

Combining masking and renaming These forms of abstraction combine naturally, and it is often appropriate to use a combination in a system where observers have different kinds of access to events. The system $P \setminus C \mid\mid\mid RUN(D)$ for example describes the situation where no events from C can be observed, and events from D are masked.

A particularly natural combination is that of masking with renaming: the identity of particular events is not known, and their occurrence may be observed but may also be masked. In the charity case, this corresponds to donations being provided in envelopes, from the donors of interest and also from other donors.

The situation is described by $f_C(P) \mid\mid\mid RUN(\{envelope\})$ (where f_C maps events to *envelope*). The only information an observer has concerning the abstracted events is an upper bound on the number of such events—a total number of envelopes will have been seen, and so the process in question can have performed no more than this.

This form of abstraction yields anonymity for $EX0$:

$$f_C(EX0) \mid\mid\mid RUN(\{envelope\}) = f_A^{-1}(f_A(f_C(EX0) \mid\mid\mid RUN(\{envelope\})))$$

However, it does not yield anonymity for $EX1$. The process after the renaming, which is

$$f_A^{-1}(f_A(f_C(EX1) \mid\mid\mid RUN(\{envelope\})))$$

has a trace $\langle 1.gives, envelope, thanks \rangle$ which is not possible for the left-hand process $f_C(EX0) \mid\mid\mid RUN(\{envelope\})$. This indicates that if a single envelope is observed, and then *thanks* is observed, then the only possible donor is 0. An observer has an upper bound on the number of donations provided by the process under consideration (i.e. $EX1$) which is the number of *envelope* events that appear in the trace: one in this case. Since *thanks* will not occur for donor 1 after only one *envelope* we have a situation where an observer can distinguish one donor from the other.

In the case described by $EX2$, where donor 1 can receive thanks after a single donation, then anonymity is indeed provided under this form of abstraction, where masking and renaming are combined. The situation where donor 1 gives two donations is masked by donations coming in from other sources.

$$\begin{aligned} EX2 = {} & 0.gives \to £5 \to thanks \to STOP \\ & \Box\ 1.gives \to £10 \to £10 \to thanks \to STOP \ \Box\ £20 \to thanks \to STOP \end{aligned}$$

Note that anonymity is not provided by $EX2$ if the form of abstraction is either simply masking or simply renaming.

5 The Dining Cryptographers

This protocol is taken from [Cha88]. The introductory example of the paper describes a situation in which three cryptographers share a meal. At the end of the meal, each of them is secretly informed by the NSA whether or not she is paying. Either at most one is paying, or else the NSA is itself picking up the bill.

The cryptographers would like to know whether it is one of them who is paying, or whether it is the NSA that is paying; but they also wish to retain anonymity concerning the identity of the payer if it one of them.

They each toss a coin, which is made visible to themself and their right-hand neighbour. Each cryptographer then examines the two coins that they can see. There are two possible announcements that can be made by each cryptographer: that the coins agree, or that they disagree. If a cryptographer is not paying then she will say that they agree if the results on the coins are the same, and that they disagree if the results differ; a paying cryptographer will say the opposite.

If the number of 'disagree' announcements is even, then the NSA is paying. If the number is odd, then one of the cryptographers is paying. The two cryptographers not paying will not be able to identify the payer from the information they have available.

Analysis

This protocol (and variants) will be coded up in CSP and analysed as follows:

1. The protocol will be analysed for functional correctness: that it can be decided from the cryptographers' announcements whether or not the NSA is paying.
2. It will be analysed from the point of view of anonymity of the payer with respect to an eavesdropper at another table
3. Anonymity of the payer with respect to the other cryptographers will be tested
4. The situation of one of the coins being double-headed will be analysed
5. The situation of four dining cryptographers will be analysed for anonymity of the payer where two of the other cryptographers pool their information

Modelling the protocol

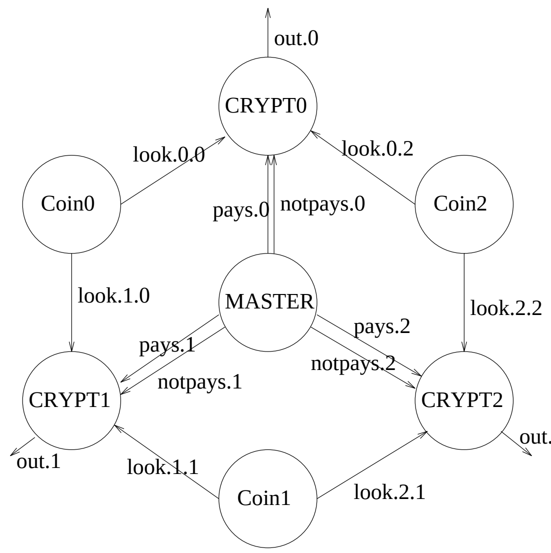

Figure 1: Components of the protocol

The protocol is modelled in CSP as the parallel combination of cryptographers and coins, and a master process dictating who pays, as illustrated in Figure 1. The events of the form $pays.i$ and $notpays.i$ are the instructions from the NSA concerning payment. Events of the form $look.i.j.x$ model cryptographer 1 reading value x from coin j. The channels $out.i$ are used for the cryptographers to make their declaration.

The $MASTER$ process nondeterministically chooses either to pay, or one of the cryptographers to pay.

$$\begin{aligned} MASTER = (\sqcap_{i:CRYPTNAMES}\ & pays.i \rightarrow notpays.((i+1)\ \mathbf{mod}\ 3) \\ & \qquad \rightarrow notpays.((i+2)\ \mathbf{mod}\ 3) \rightarrow STOP) \\ \sqcap\ & notpays.0 \rightarrow notpays.1 \rightarrow notpays.2 \rightarrow STOP \end{aligned}$$

Each cryptographer process follows the protocol. This is described in CSP as follows:

$$\begin{aligned} CRYPT(i) = \ & notpays.i \rightarrow look.i.i?x \rightarrow look.i.((i+1)\ \mathbf{mod}\ 3)?y \rightarrow \\ & \quad (\mathbf{if}\ (x=y)\ \mathbf{then}\ (out.i.agree \rightarrow STOP) \\ & \quad \qquad \mathbf{else}\ (out.i.disagree \rightarrow STOP)) \\ & \square\ (pays.i \rightarrow look.i.i?x \rightarrow look.i.((i+1)\ \mathbf{mod}\ 3)?y \rightarrow \\ & \quad (\mathbf{if}\ (x=y)\ \mathbf{then}\ out.i.disagree \rightarrow STOP \\ & \quad \qquad \mathbf{else}\ out.i.agree \rightarrow STOP)) \end{aligned}$$

Each coin is modelled as a choice between reading heads and reading tails:

$$\begin{aligned} COIN(i) &= HEADS(i) \sqcap TAILS(i) \\ HEADS(i) &= look.i.i!heads \rightarrow HEADS(i) \\ &\quad \square\ look.((i-1)\ \mathbf{mod}\ 3).i!heads \rightarrow HEADS(i) \\ TAILS(i) &= look.i.i!tails \rightarrow TAILS(i) \\ &\quad \square\ look.((i-1)\ \mathbf{mod}\ 3).i!tails \rightarrow TAILS(i) \end{aligned}$$

The master either sends a pay message to one of the cryptographers or a don't pay message to all of them.

The system is constructed from the cryptographers and coins, which are two collections of independent processes.

$$\begin{aligned} CRYPTS &= CRYPT(0)\ |||\ CRYPT(1)\ |||\ CRYPT(2) \\ COINS &= COIN(0)\ |||\ COIN(1)\ |||\ COIN(2) \end{aligned}$$

They must synchronise on the events representing the examination of coins and the MASTER decides who is paying.

$$MEAL = ((CRYPTS\ |[\, look \,]|\ COINS)\ |[\, pays, notpays \,]|\ MASTER) \setminus notpays$$

It is also possible to provide a parametric description of the system for an arbitrary number n of cryptographers; but automatic verification will be possible only once a particular n is chosen.

Functional correctness The protocol is functionally correct if it is possible to distinguish the possibilities of a cryptographer paying from the NSA paying. The description of the protocol itself indicates how this is to be achieved: by means of counting the 'disagree' declarations and checking whether there were an odd or an even number of them.

Although this is very simple to express, if it is to be checked using FDR then it is necessary to construct an explicit CSP 'test-harness' to count the number of 'disagree' events and then output the result. This will be the process $COUNTS(0,0)$ below.

The specification then captures the idea that either one of the cryptographers pays, in which case an answer 'crypt' should result, or none of them pays and the answer 'nsa' should result:

$$\begin{aligned} SPEC = & (\sqcap_{i:CRYPTNAMES}(pays.i \to crypt \to STOP)) \\ & \sqcap (nsa \to STOP) \end{aligned}$$

The specification is that if one of the cryptographers is asked to pay, then the protocol yields 'crypt'; otherwise it yields 'nsa'.

The process of telling the difference is captured by $COUNTS(0,0)$ where $COUNTS(i,j)$ is defined as follows:

$$\begin{aligned} COUNTS(i,j) = (&\mathbf{if}\ (i = 3) \\ & \mathbf{then}\ (\mathbf{if}\ (j = 0) \\ & \qquad \mathbf{then}\ (crypt \to STOP) \\ & \qquad \mathbf{else}\ (nsa \to STOP)) \\ & \mathbf{else}\ out.i?x \to (\mathbf{if}\ (x = disagree) \\ & \qquad\qquad \mathbf{then}\ COUNTS(i+1,j) \\ & \qquad\qquad \mathbf{else}\ COUNTS(i+1,((j+1)\ \mathbf{mod}\ 2)))) \end{aligned}$$

The meal together with the protocol captured by $COUNTS(0,0)$ is then captured as follows:

$$SYSTEM = (MEAL \,|[\, out \,]|\, COUNTS(0,0)) \setminus \{look, out\}$$

All of these descriptions are easily coded up within the CSP syntax required for FDR, which confirms that

$$SPEC \sqsubseteq SYSTEM$$

or in other words, that the system really does meet the specification: the protocol is correct.

Anonymity with respect to an outsider Anonymity is required for the system described by *MEAL*.

An observer sitting at the next table sees only the calls made by the cryptographers: such an observer has access only to the set *out*, as illustrated in Figure 2

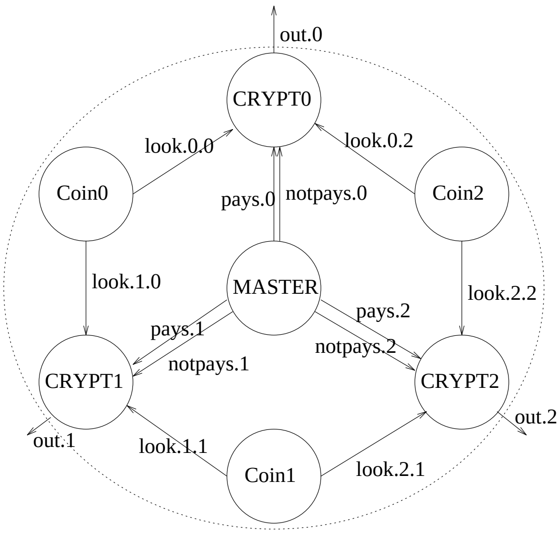

Figure 2 : The abstracted cryptographers

We may test for anonymity between all the cryptographers, by checking for anonymity on the set *pays*. The set of abstracted events in this case is all the internal events except *pays*, i.e. the set *look*.

We will use the abstraction $f_p(pays.i) = \alpha$ to identify all the *pays* events.

If the observer cannot even tell when a cryptographer is looking at a coin, then hiding abstraction is most appropriate. The check required for anonymity is therefore:

$$MEAL \setminus look \sqsubseteq f_p^{-1}(f_p(MEAL \setminus look))$$

On the other hand, if the observer can see when a cryptographer is looking at a coin, then renaming abstraction is appropriate: the observer can tell that some event is occurring, but does not know its precise nature. We may use the renaming function $f_l(look.i.j.x) = look.i.j$. The observer can tell which cryptographer is looking at which coin, but cannot tell what is seen.

The check required in this case is

$$f_l(MEAL) \sqsubseteq f_p^{-1}(f_p(f_l(MEAL)))$$

In both cases, FDR confirms that the relationship holds: that the protocol does provide anonymity.

Anonymity against other cryptographers The protocol is intended not only to provide anonymity against outsiders, but also against the other cryptographers. In this situation we require that any particular cryptographer cannot distinguish between the other two $pay.i$ events on the strength of the available information. Now there is much more information with which to break anonymity. A cryptographer has access to two of the coins, and so has more information.

By symmetry of the system, we have only to consider cryptographer 0. In this case, there is extra information consisting of the $look.0$ events; so the only hidden events are the $look.1$ and the $look.2$ events. We aim to establish anonymity for the set $pays.1, pays.2$.

If the $look.i$ events are completely invisible to $CRYPT0$ then hiding is most suitable, and the appropriate check is

$$MEAL \setminus \{look.1, look.2\} \sqsubseteq f_{p12}^{-1}(f_{p12}(MEAL \setminus \{look.1, look.2\}))$$

where $f_{p12}(pays.1) = f_{p12}(pays.2) = \alpha$.

On the other hand, if the cryptographer can see when the other cryptographers are looking at a coin, then renaming abstraction is more suitable. In this case we use the renaming function

$$\begin{aligned} f_{l12}(look.i.j.x) = \; & look.i.j \quad \text{if } i = 1 \text{ or } i = 2 \\ & look.i.j.x \text{ otherwise} \end{aligned}$$

to rename only $look.1$ and $look.2$.

The check required in this case is

$$f_{l12}(MEAL) \sqsubseteq f_{p12}^{-1}(f_{p12}(f_{l12}(MEAL)))$$

Again, in both cases, FDR confirms that the relationship holds: that the protocol does provide anonymity for each cryptographer against the others.

A double-headed coin If the system contains a double-headed coin, then the situation is subtly changed. Although $CRYPT0$ still has access to the same information, it is within the context of a slightly altered system description, and anonymity may thereby be compromised.

The double headed coin is described by $HEADS(i)$: it can only provide heads. We will consider the situation where it could be any of the three coins. In this case the description of the coins becomes

$$\begin{aligned} COINS' = \; & HEADS(0) \;|||\; COIN(1) \;|||\; COIN(2) \\ & \sqcap\; COIN(0) \;|||\; HEADS(1) \;|||\; COIN(2) \\ & \sqcap\; COIN(0) \;|||\; COIN(1) \;|||\; HEADS(2) \end{aligned}$$

The behaviour of the cryptographers remains unaltered, but the description of the meal must change to reflect the changed description of the coins:

$$MEAL' = ((CRYPTS \,|[\, look \,]|\, COINS') \,|[\, pays, notpays \,]|\, MASTER) \setminus notpays$$

Checking this system for anonymity with respect to $\{pays.1, pays.2\}$ against cryptographer 0 we check

$$MEAL \setminus \{look.1, look.2\} \sqsubseteq f_{p12}^{-1}(f_{p12}(MEAL \setminus \{look.1, look.2\}))$$

Applying FDR to this description reveals that the refinement relation fails to hold. FDR's debugging facilities provide a witness trace: one which is possible for the right hand side but not for the left.

The trace obtained is $tr = \langle pays.1, out.2.agree, look.0.0.tails, look.0.1.tails \rangle$.

This trace is not possible for the system for the following reason: if two coins have both been observed with *tails*, then the knowledge that one of the three coins must read *heads* allows the deduction (provided it is known that the coin is biased) that the third coin $COIN2$ must be the double-headed one. Since $out.2.agree$ has been observed, and it is also known that the two coins seen by $CRYPT2$ read differently, this sequence cannot follow the event $pays.1$.

On the other hand, the events observed by $CRYPT0$ *are* possible if the initial event was $pays.2$, since then the protocol dictates that $CRYPT2$ should announce *agree* if the coins disagree. In other words, the sequence of events $\langle pays.2, out.2.agree, look.0.0.tails, look.0.1.tails \rangle$ is a possible trace of the process $MEAL \setminus \{look.1, look.2\}$. For this reason the trace tr above is possible for the right-hand process.

This illustrates the way in which the feedback from the FDR debugger may be used to gain an understanding of why anonymity does not hold. The messages seen by cryptographer 0 are possible when one of the other cryptographers is paying, but not when the other one is. So there are situations where cryptographer 0 can identify who is paying. In this case it is possible when both visible coins read *tails*. This allows the deduction that the third coin reads *heads*, which means that the coins both sides of each of the other cryptographers are known, enabling cryptographer 0 to tell whether each *agree* or *disagree* declaration is appropriate.

All the FDR checks in this section took about 5 seconds on a sparc 5.

Four cryptographers The situation extends naturally to four cryptographers, and verification that anonymity for any cryptographer against any one of the others is easily established.

A new situation of interest arises when there are four cryptographers: the possibilities of anonymity in the case where two of the participants share their information.

The description alters in the obvious way:

$$
\begin{aligned}
MASTER = (\sqcap_{i:CRYPTNAMES}\, & pays.i \rightarrow notpays.((i+1) \mathbf{\ mod\ } 4) \\
& \rightarrow notpays.((i+2) \mathbf{\ mod\ } 4) \\
& \rightarrow notpays.((i+3) \mathbf{\ mod\ } 4) \rightarrow STOP) \\
\sqcap\, & notpays.0 \rightarrow notpays.1 \rightarrow notpays.2 \rightarrow notpays.3 \rightarrow STOP
\end{aligned}
$$

$$
\begin{aligned}
CRYPT(i) = {} & notpays.i \rightarrow look.i.i?x \rightarrow look.i.((i+1) \mathbf{\ mod\ } 4)?y \rightarrow \\
& (\mathbf{if}\ (x = y)\ \mathbf{then}\ (out.i.agree \rightarrow STOP) \\
& \qquad \mathbf{else}\ (out.i.disagree \rightarrow STOP)) \\
& \Box\ (pays.i \rightarrow look.i.i?x \rightarrow look.i.((i+1) \mathbf{\ mod\ } 4)?y \rightarrow \\
& (\mathbf{if}\ (x = y)\ \mathbf{then}\ out.i.disagree \rightarrow STOP \\
& \qquad \mathbf{else}\ out.i.agree \rightarrow STOP))
\end{aligned}
$$

$$
\begin{aligned}
COIN(i) &= HEADS(i) \sqcap TAILS(i) \\
HEADS(i) &= look.i.i!heads \rightarrow HEADS(i) \\
& \quad \Box\ look.((i-1) \mathbf{\ mod\ } 4).i!heads \rightarrow HEADS(i) \\
TAILS(i) &= look.i.i!tails \rightarrow TAILS(i) \\
& \quad \Box\ look.((i-1) \mathbf{\ mod\ } 4).i!tails \rightarrow TAILS(i)
\end{aligned}
$$

$$
\begin{aligned}
CRYPTS &= CRYPT(0) \ |||\ CRYPT(1) \ |||\ CRYPT(2) \ |||\ CRYPT(3) \\
COINS &= COIN(0) \ |||\ COIN(1) \ |||\ COIN(2) \ |||\ COIN(3)
\end{aligned}
$$

$$
MEAL = ((CRYPTS \ |[\, look \,]|\ COINS) \ |[\, pays, notpays \,]|\ MASTER) \setminus notpays
$$

Consider first the possibility of two adjacent cryptographers pooling their information. Without loss of generality we may consider them to be cryptographers 0 and 1.

In this case, the information available to these cryptographers is given by the set $out \cup look.0 \cup look.1$, and we require anonymity for $\{pays.2, pays.3\}$.

In this case the following check is appropriate:

$$
MEAL \setminus \{look.2, look.3\} \sqsubseteq f_{p23}^{-1}(f_{p23}(MEAL \setminus \{look.2, look.3\}))
$$

where $f_{p23}(pays.2) = f_{p23}(pays.3) = \alpha$.

FDR confirms that the check indeed holds. This check took approximately 35 seconds on a sparc 5.

The other possibility is that opposite cryptographers pool their information. In this case we will assume that they are cryptographers 0 and 2.

In this case, the information available to these cryptographers is $out \cup look.0 \cup look.2$, and we require anonymity for $\{pays.1, pays.3\}$.

The following check is performed:

$$MEAL \setminus \{look.1, look.3\} \sqsubseteq f_{p13}^{-1}(f_{p13}(MEAL \setminus \{look.1, look.3\}))$$

where $f_{p13}(pays.1) = f_{p13}(pays.3) = \alpha$.

In this case FDR establishes that the check fails, (also in approximately 35 seconds) and provides a witness trace, possible for the right hand process but not for the left:

$$\langle pays.1, out.3.disagree, look.2.2.tails, look.2.3.tails, look.0.0.tails \rangle$$

The information obtained by $look.2.3.tails$ and $look.0.0.tails$ is that the two coins adjacent to $CRYPT3$ both read $tails$. Hence it is not possible to have observed $out.3.disagree$ following $pays.1$, since if cryptographer 3 is not paying then the declaration should match the readings on the coins.

However, such observations as have been made by cryptographers 0 and 2 are possible following an occurrence of $pays.3$. Hence they are able to distinguish the events $pays.1$ and $pays.3$ and hence to deprive cryptographer 3 of anonymity.

6 Discussion

This paper has proposed a formal CSP definition of anonymity and has illustrated its use with the example of the dining cryptographers. The availability of the FDR model-checking tool [FSEL94] for CSP means that anonymity properties can be checked for systems in a mechanical fashion. Furthermore, in cases where anonymity fails, the CSP analysis provides useful feedback as to the particular behaviour of the system which violates the anonymity requirement. Use of a small example such as the dining cryptographers acts as a feasibility study for the approach and tests the definition against our general understanding of anonymity. We intend next to apply this approach to larger systems and to real-world anonymity protocols.

This notion of anonymity is appropriate for situations where every anonymous event is entirely independent of every other such event, as might be the case in cash-transactions. But there are some situations where this requirement is too strong, such as anonymous voting protocols. Even if anonymity is required for the identity of the voter, it must still be ensured that different votes such as $a_1.v_1$ and $a_2.v_2$ are generated by different agents. Hence two different votes are not entirely independent, since a_1 and a_2 must be distinct. Such a voting system would fail our definition above, which requires that each of two votes could independently have originated from the same agent: that the trace $\langle a_3.v_1, a_3.v_2 \rangle$ should be possible. It appears likely that subtly different definitions of anonymity will be required for different situations depending on the particular requirements. The formalisation of these definitions will allow the distinctions to be understood explicitly, and will facilitate the correct choice. The authors are currently exploring further notions of anonymity, and their relationship to the definition of this paper.

Treatments of anonymity often include an analysis of the probabilistic behaviour of the system. For example, in [Wai90] the scheme is proven to have the property that the conditional probability for an input vector given the observed output vector is the same as the a priori probability for that input vector: in other words the output vector provides no additional information about the probabilities associated with the inputs. This paper has not considered probability, focusing instead on simple possibilities. This identifies anonymity for an agent with the *possibility*, however unlikely, that other agents could equally have performed the same event. In this analysis, if a set of biased coins was used where each had a 99% chance of landing *heads*, then any cryptographer *may* have be paying given any particular outputs, but it is much more likely in the case of two *agree* calls that the one who called *disagree* is actually the one who is paying: the conditional probability given the outputs is not the same as the a priori probability.

It seems likely that the use of probabilistic CSP [Sei92, Low93] could directly address this issue. This would allow the association of probability values with particular traces. It may be the case that the same definition of anonymity will extend directly to probabilistic models for CSP: that $f_A^{-1}(f_A(P)) = P$. In a probabilistic model this would mean that the identification of different events should not make any difference to the probabilities associated with each of those event. The definition captures the idea that it is not possible to tell the difference between the outcomes of a number for different events. What is meant precisely by 'telling the difference' is defined by the CSP model being used. In the traces model it simply means that all resulting processes should have the same traces; a model with probabilities will also require that resulting probabilistic behaviours should also be the same.

7 Acknowledgements

Thanks are due to Peter Ryan for posing the dining cryptographers as a challenge for anonymity, and for his comments; to Birgit Pfitzmann for a thorough reading and detailed comments; and also to Irfan Zakiuddin, Dieter Gollmann, Bill Roscoe, Gavin Lowe, and Michael Goldsmith for other comments on various stages of this work. We are also grateful to the anonymous referees for their reviews and suggestions.

The authors are grateful to the DRA and to Peter Ryan for funding this research.

References

[Cha85] D. Chaum, *Security without Identification: Card Computers to make Big Brother Obsolete*, CACM 28(10), 1985.

[Cha88] D. Chaum, *The dining cryptographers problem: unconditional sender and recipient untraceability*, J. Cryptology (1), 1988.

[FSEL94] Formal Systems (Europe) Ltd, *Failures Divergences Refinement User Manual and Tutorial*, 1994.

[Hoa85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[Low93] G. Lowe, *Probabilities and priorities in timed CSP*, D.Phil thesis, Oxford, 1993.

[PfW94] B. Pfitzmann and M. Waidner, *A general framework for formal notions of "secure" system*, Hildesheimer Informatik-Berichte 11/94, Institut für Informatik, Universität Hildesheim, 1994.

[Ros94] A.W. Roscoe, *CSP and determinism in security modelling*, Submitted for publication, 1994.

[Sch96] S.A. Schneider, *Security properties and CSP*, IEEE Symposium on Security and Privacy, 1996.

[Sei92] K. Seidel, *Probabilistic Communicating Processes*, D.Phil Thesis, Oxford, 1992.

[Wai90] M. Waidner, *Unconditional sender and recipient untraceability in spite of active attacks*, Eurocrypt'89, LNCS 434, Springer, 1990.

This article was processed using the LaTeX macro package with LLNCS style