Attacking Group Protocols by Refuting Incorrect Inductive Conjectures

Graham Steel and Alan Bundy
University of Edinburgh

October 28th, 2004

Abstract.
Automated tools for finding attacks on flawed security protocols often fail to deal adequately with group protocols. This is because the abstractions made to improve performance on fixed 2 or 3 party protocols either preclude the modelling of group protocols all together, or permit modelling only in a fixed scenario, which can prevent attacks from being discovered. This paper describes CORAL, a tool for finding counterexamples to incorrect inductive conjectures, which we have used to model protocols for both group key agreement and group key management, without any restrictions on the scenario. We will show how we used CORAL to discover 6 previously unknown attacks on 3 group protocols.

Keywords: Cryptographic security protocols, counterexamples, superposition

1. Introduction

In terms of analysing the standard corpus of two and three party protocols given in (Clark and Jacob, 1997), the field of cryptographic security protocol analysis can be said to be saturated. Much research attention has now turned to trying to widen the scope of the techniques, e.g. to group protocols, (Meadows, 2000; Taghdiri and Jackson, 2003; Steel et al., 2004). These are protocols designed to prescribe a way in which an unbounded number of principals may establish, and maintain, a secure key for communicating between themselves over an insecure network. The key may have to be updated as the composition of the group changes, and a trusted server may be used to facilitate the process.

This very open scenario causes problems for most automated protocol analysis tools. This is because they rely on concrete abstractions to fixed party protocols, such as restricting the number of agents in the network to 2 or 3, and/or predetermining the exact scenario which is examined in terms of which agent will play the initiator, and which agent the responder. These abstractions are quite suitable for fixed party protocols, but are too restrictive to allow realistic analysis of group protocols. Moreover, in most tools, these restrictions cannot easily be relaxed, as we will explain in §7. In this paper, we describe the CORAL


© 2006 *Kluwer Academic Publishers. Printed in the Netherlands.*

system, an approach to flaw detection based on finding counterexamples to security properties expressed as inductive conjectures. The use of an inductive model, a first-order version of Paulson's, (Paulson, 1998), allows us to naturally model a group protocol without constraining the scenario. Using CORAL we have discovered 6 previously unknown attacks on 3 group protocols: 3 on the Asokan–Ginzboorg protocol, (Asokan and Ginzboorg, 2000), 2 on Taghdiri and Jackson's improved version of the Tanaka-Sato protocol, (Taghdiri and Jackson, 2003), and 1 on the Iolus protocol, (Mittra, 1997). We will describe these case studies in §4, §5 and §6. Our case studies demonstrate that although our system typically takes longer to find attacks on simple protocols than competing systems, the greater expressivity of our first-order inductive model allows us to adapt quickly to different kinds of group protocols with their particular features, such as lists of keys in messages, keys consisting of compound terms, unbounded roles, and messages that change depending on the current composition of the group.

In §2, we will review previous work on group protocol analysis. In §3 we describe the theory and implementation of CORAL. After that we give the 3 case studies: §4 concerns the Asokan–Ginzboorg protocol, §5 the Tanaka–Sato/Taghdiri–Jackson protocol, and §6 the Iolus protocol. We evaluate CORAL in §7, comparing it to related work. In §8, we draw conclusions and discuss further work, such as developing CORAL's ability to verify correct protocols, as well as attack faulty ones.

2. Background

A recent survey of the field of cryptographic protocol analysis can be found in (Meadows, 2003). In this section, we will briefly survey previous work specific to group protocol analysis.

Perhaps the first formal analysis of a group protocol was by Paulson. He investigated a recursive authentication protocol proposed by Bull and Otway, (Bull and Otway, 1997), as part of his development of the inductive method, (Paulson, 1998). Paulson was able to model it in the general case, i.e. for any number of participants, and prove some security properties for arbitrary sized groups. Paulson's model uses a typed higher-order logic formalism, and poses security properties as conjectures about properties of the trace. These properties are then proved by induction on traces in the interactive theorem prover Isabelle/HOL. However, if there is a flaw in the protocol, Paulson's method provides no automated support for finding the attack. It is precisely this task that our tool, CORAL, was designed for.

The CLIQUES protocol suite contains a number of protocols for establishment of groups, and groups within groups, each with their own secure key, (Ateniese et al., 2000). Extensive use is made of Diffie-Hellman style exponentiation, (Diffie and Helman, 1976). Several significant attacks on the CLIQUES protocol suite were found by Pereira and Quisquater as a result of a programme of manual analysis, (Pereira and Quisquater, 2003). Pereira and Quisquater proposed a method for converting the problem of the intruder obtaining a particular term to the solution of a system of linear equations. Using this method, they were able to find weaknesses in every protocol they examined. Often, the attacks required quite imaginative behaviour by the intruder, e.g. being accepted as a member of a group of size 4, and then using the values learnt in that key establishment session to force a group of size 3, intended to exclude him, to accept a key he knew. One clear lesson from Pereira and Quisquater's work is that the design of these kinds of protocols is extremely tricky. It is easy to understand how the designers of the CLIQUES protocols failed to see such attacks. This strengthens the case for the use of formal methods in their design.

Meadows used the NRL protocol analyser (NPA) to tackle the Group Domain of Interpretation (GDOI) protocol suite, as part of an effort to introduce formal methods to the design stage of protocol development, (Meadows and Syverson, 2001). GDOI is a method for group key management involving a trusted key server. NPA could not handle the infinite data structures required for a general model of a group protocol, so a concrete abstraction was made, by restricting phase two to the distribution of one 'security association key'. NPA was able to find a type flaw in the protocol, which was fixed in later versions of GDOI. However, Meadows attempt to use NPA to rediscover the Pereira-Quisquater attacks on the CLIQUES suite was less successful, (Meadows, 2000). NPA was extended to handle the Diffie-Hellman exponentiation operation, but could not find the attacks described in (Pereira and Quisquater, 2003).

The designers of the protocol specification language CAPSL, (Denker and Millen, 2000), have recently turned their attention to group protocols in the development of an extended language, MuCAPSL, (Millen and Denker, 2003). After translating the GDH.2 protocol, (Steiner et al., 1996), to their intermediate language MuCIL they were able to discover a type flaw. However, their analysis requires the number of group members to be set in advance, which can prejudice the chances of discovering an attack.

Taghdiri and Jackson, (Taghdiri and Jackson, 2003), reported results from the modelling of a multicast key management scheme proposed by Tanaka and Sato, (Tanaka and Sato, 2001). This protocol was designed

for a scenario where the group is highly dynamic, i.e. agents join and leave the group frequently. The idea was to minimise the number of key updates required by supplying keys to agents on demand. Taghdiri and Jackson formalised a model for the protocol in the Alloy specification language, (Jackson, 2002), and used Alloy's SAT checker to search for counterexamples to desirable properties of the protocol. Several counterexamples were found, the most serious one indicating that current members of the group will accept as valid messages broadcast by ex-members of the group. Taghdiri and Jackson proposed an improved protocol. However, their formal protocol model differed from the norm established over the last 25 years in that no active attacker was included. In sections 5 and 6 of this paper, we explain how we used Coral to analyse both Taghdiri and Jackson's improved protocol, and the Iolus key management protocol (treated in Taghdiri's MSc thesis, (Taghdiri, 2002), and considered to be secure). We discovered that neither of these protocols are secure in the presence of an active attacker, even if he is weaker than the Dolev-Yao intruder generally used in automated protocol analysis, (Dolev and Yao, 1983).

3. Coral

Coral is the first full implementation of the Comon-Nieuwenhuis method for proof by consistency, (Comon and Nieuwenhuis, 2000). It is based on an adapted version of the theorem prover Spass, (Weidenbach, 2001). We give a summary of the technique and how it is implemented here. More details are available in (Steel, 2004).

3.1. Theory

Proof by consistency, (Musser, 1980), was originally designed to prove inductive theorems correct, but later versions of the technique have the property of being refutation complete, i.e. able to refute any incorrect conjecture in finite time. The Comon-Nieuwenhuis method works like this: given a set of Horn clauses E[1] and a conjecture C, the Comon–Nieuwenhuis approach is to produce a first-order axiomatisation, $\mathcal{A}$, of the minimal Herbrand model such that $C \cup \mathcal{A} \cup E$ is consistent if and only if C is an inductive consequence of E. We call $\mathcal{A}$ the *I-Axiomatisation*. This is typically a set of clauses sufficient for deciding inequality of ground terms. To determine the consistency, or

[1] The Comon-Nieuwenhuis method can also be applied to non-Horn specifications, using 'perfect model' semantics to distinguish the intended model, (Bachmair and Ganzinger, 1991). However, the formalisation of the security protocol problem we use consists only of Horn clauses.

otherwise, of C with $\mathcal{A} \cup E$, we use a two part process. In the first part, we pursue a *fair induction derivation*. This is a restricted kind of saturation, (Bachmair and Ganzinger, 1990), where we need only consider overlaps between axioms and conjectures, and produce inferences from an adapted superposition rule called *conjecture superposition*. In the second part, each clause in the induction derivation is checked separately for consistency against the I-Axiomatisation. If any consistency check fails, then the conjecture is incorrect. If all the consistency checks succeed, and the induction derivation procedure terminates, the theorem is proved. The separation of the main inductive derivation from the multiple independent consistency checks gives rise to natural opportunities for parallelising the process.

In the case where a refutation is found, we are able to extract a counterexample by means of a well-known method first proposed by Green, (Green, 1969). When CORAL refutes a security conjecture of the form $\forall \text{trace}.P(\text{trace})$, it has proved in its superposition calculus that $\exists \text{trace}.\neg P(\text{trace})$. We track the instantiations made to the trace variable using an *answer literal*, following Green's method. Green has shown that this will always yield a correct constructive answer for these types of proofs.

Previous versions of the proof by consistency technique were restricted to saturated specifications. An important result of Comon and Nieuwenhuis' work was to relax this restriction. Rather than a saturated specification, they require a *reductive definition*, and a constructor based specification. The definition is reductive if all ground terms headed by defined symbols can be reduced, with respect to our term ordering, by conjecture superposition. CORAL's model for the security protocol problem is a reductive definition, and details of the proof of its refutation completeness can be found in (Steel, 2004, §6.8).

3.2. First-Order Inductive Model

The aim of CORAL's model was a first-order version of Paulson's inductive model for protocol analysis. Though Paulson's formalism is encoded in higher-order logic, no fundamentally higher-order concepts (except induction) are used – in particular, there is no unification of functional objects. Objects have types, and sets and lists are used. All this we model in first-order logic, using a sorted first-order signature instead of higher-order types. Like Paulson's, our model allows an indeterminate and unbounded number of agents to participate, playing any role, and using an arbitrary number of fresh nonces and keys.

A protocol is modelled as the set of all possible traces, i.e. all possible sequences of messages sent by any number of honest users under the

specification of the protocol and, additionally, faked messages sent by the intruder. A trace of messages is modelled as a list. For a specific protocol, we generally require one axiom per protocol message, each one having the interpretation, 'if T is a trace containing message n addressed to agent A, then T may be extended by A responding with message $n+1$'. These axioms define a unary function m, where $m(T) = \mathit{true}$ if T is a valid trace of messages for the protocol specified.

The intruder has the usual capabilities specified by the Dolev-Yao model, (Dolev and Yao, 1983). This includes the ability to intercept all messages in the trace, and to break down and reassemble the messages he has seen. He can only decrypt ciphertext packets if he has the correct key, and is assumed to be accepted as an honest player by the other agents. We specify intruder knowledge in terms of sets. Given a trace of messages exchanged, T, we define $\mathit{analz}(T)$ to be the least set including T closed under projection and decryption by known keys. This is accomplished by an inductive definition using exactly the same rules as the Paulson model, (Paulson, 1998, p. 12). We define the messages the intruder may send, given a trace T, as being members of the set $\mathit{synth}(\mathit{analz}(T))$, where $\mathit{synth}(X)$ is the least set containing X, including agent names and closed under pairing and encryption by known keys. Again, the concept of a 'least set' is defined not by some higher-order quantification over sets, but by an inductive definition, using the same rules as Paulson. We also follow Paulson's method for the modelling of the freshness of nonces and session keys. If T is a list of messages, then $\mathit{parts}(T)$ is the least set containing the messages in T closed under projection and decryption by all keys. A nonce N is considered fresh with respect to T if $N \notin \mathit{parts}(T)$.

3.3. Implementation

Coral is implemented on top of the Spass prover, (Weidenbach, 2001). We added the conjecture superposition inference rules and answer literals. Since the fair induction derivation procedure may not terminate, we must carry out the consistency checks in parallel to retain refutation completeness. Since the consistency checks are independent from each other, we can also make them concurrently. Coral uses a parallel architecture to achieve this, using a socket interface to send clauses for I-Axiomatisation checking to separate Spass processes running in parallel. For specific I-Axiomatisations, we can often filter out a lot of these clauses to avoid having to check them. For example, if we are using a free constructor specification, the I-Axiomatisation will just specify the inequality of non-identical constructor terms. If a clause to

be checked contains defined symbols, no axioms in the I-Axiomatisation will apply, so it can be ignored.

CORAL also includes a small number of domain specific redundancy rules to prune the search space. These are:

Spy only sends protocol messages: We restrict the spy to sending only messages which match the pattern of the protocol. It has been formally proved, by Syverson et al., that a spy who only sends messages which an honest player might accept as being part of the protocol is just as powerful as the full 'Dolev-Yao' intruder, (Syverson et al., 2000).

Spy only expects protocol messages: We can prune away any clauses representing a state where the spy is expecting to see something in the trace which cannot possibly occur - for example, an honest agent sending his long-term keys in cleartext.

Eager pruning of unsatisfiable *parts* literals: As described above, we model freshness of nonces using the *parts* operator, following Paulson's model. We can prune out any clauses of the form $in(nonce(N), parts(T)) = false$, and the term $nonce(N)$ occurs in T, using the occurs check already implemented in SPASS.

These pruning rules improve CORAL's performance greatly on all the protocol problems we have tried. Indeed, without them, CORAL would certainly not have found the attacks described in this paper. They also account for only a tiny proportion of time spent on reduction, with the standard subsumption checking and rewriting rules taking up the vast majority of the time.

3.4. RESULTS ON STANDARD PROTOCOLS

After using 4 fixed party protocols for development, CORAL was tested on 10 different standard protocols from the standard corpus, (Clark and Jacob, 1997). The corpus contains examples of 5 different kinds of attack, so 2 of each kind were used. CORAL found all 10 of the attacks we searched for, 5 of them in under 10 seconds[2], 8 of them in under a minute, 9 in under 15 minutes and all ten in under half an hour. The total run time for all ten is a little under 43 minutes and 30 seconds. The longest search time was for the parallel session attack on the Andrew-RPC protocol. This took a long time to find because it is

[2] All timings in this paper are quoted for a 2 GHz Pentium IV PC with 512Mb RAM running Linux 2.4.20.

8 steps long, and requires the symmetry of the key to be exploited 4 times.

CORAL is slower to find these attacks than some competing systems, e.g. (Song et al., 2001; Basin et al., 2003), which take at most a few seconds. However, it is not so slow as to be impractical. Additionally, CORAL is discovering the appropriate scenario for the attack for itself whereas some tools, e.g. (Basin et al., 2003), require the user to decide in advance what role the spy and the honest agents will play. This quality is vital for realistic analysis of group key protocols, as we will demonstrate in the case studies that follow.

4. The Asokan–Ginzboorg Protocol

Asokan and Ginzboorg of the Nokia Research Center have proposed an application level protocol for use with Bluetooth devices, (Asokan and Ginzboorg, 2000). The scenario under consideration is this: a group of people are in a meeting room and want to set up a secure session amongst their Bluetooth-enabled laptops. They know and trust each other, but their computers have no shared prior knowledge and there is no trusted third party or public key infrastructure available. The protocol proceeds by assuming a short group password is chosen and displayed, e.g. on a whiteboard. The password is assumed to be susceptible to a *dictionary attack*, but the participants in the meeting then use the password to establish a secure secret key.

Asokan and Ginzboorg describe two protocols for establishing such a key in their paper, (Asokan and Ginzboorg, 2000). We have analysed the first of these. Completing analysis of the second protocol using CORAL would require some further work (see §4.6). Here is a description of the first Asokan–Ginzboorg protocol (which we will hereafter refer to as simply 'the Asokan–Ginzboorg protocol'). Let the group be of size n for some arbitrary $n \in \mathbb{N}, n \geq 2$. We write the members of the group as M_i, $1 \leq i \leq n$, with M_n acting as group leader.

1.	M_n	$\rightarrow$	ALL	:	$M_n, \{\!\|\ E\ |\!\}_P$	
2.	M_i	$\rightarrow$	M_n	:	$M_i, \{\!\|\ R_i, S_i\ |\!\}_E$	$i = 1, \ldots, n-1$
3.	M_n	$\rightarrow$	M_i	:	$\{\!\|\ \{Sj, j = 1, \ldots, n\}\ |\!\}_{R_i}$	$i = 1, \ldots, n-1$
4.	M_i	$\rightarrow$	M_n	:	$M_i, \{\!\|\ S_i, h(S_1, \ldots, S_n)\ |\!\}_K$	some i

What is happening here is:

1. M_n broadcasts a message containing a fresh public key, E, encrypted under the password, P, written on the whiteboard.

2. Every other participant M_i, for $i = 1, \ldots, n-1$, sends M_n a contribution to the final key, S_i, and a fresh symmetric key, R_i, encrypted under public key E.

3. Once M_n has a response from everyone in the room, she collects together the S_i in a package along with a contribution of her own (S_n) and sends out one message to each participant, containing this package $S_1, \ldots, S_n$ encrypted under the respective symmetric key R_i.

4. One participant M_i responds to M_n with the package he just received passed through a one way hash function $h()$ and encrypted under the new group key $K = f(S_1, \ldots, S_n)$, with f a commonly known function.

Asokan and Ginzboorg argue that it is sufficient for each group member to receive confirmation that one other member knows the key: everyone except M_n receives this confirmation in step 3. M_n gets confirmation from a random member of the group in step 4. Once this message is received, the protocol designers argue, agents $M_1, \ldots, M_n$ must all have the new key $K = f(S_1, \ldots, S_n)$. The protocol has three goals: the first is to ensure that a spy eavesdropping on Bluetooth communications from outside the room cannot obtain the key. Secondly, it aims to be secure against *disruption attacks*[3] by an attacker who can add fake messages, but not block or delay messages. Thirdly, it aims by means of contributory key establishment, to prevent a group of dishonest players from restricting the key to a certain range. We used these goals as the basis of our investigation described in §4.2.

4.1. Modelling the Asokan–Ginzboorg Protocol

Our methodology for modelling the Asokan–Ginzboorg protocol was the same as for fixed 2 or 3 principal protocols, i.e. to produce one rule for each protocol message describing how a trace may be extended by that message, taking into account the tests which an honest agent will apply. So for message 2, our rule expresses the fact that an agent will only send a message 2 if he has seen a message 1 in the trace, and will only use fresh numbers S_i and R_i in his message. However, for the Asokan–Ginzboorg protocol, message 3 posed a problem. For a group of n participants, $n-1$ message 3s will be sent out at once, each encrypted under a different key. Moreover, the group leader for the run must check that she has received $n-1$ message 2s. In order

[3] A disruption attack is an attack whereby a spy prevents the honest agents from completing a successful run of the protocol.

to model this without predetermining the size of the group, we needed to accommodate the possibility of different numbers of messages being added to the trace in one instant. This problem was solved by defining a new function called *all_msg2s_received*, which checks a trace to see if all message 2s have been sent for a particular group leader and number of participants. It returns a new trace containing the response prescribed by the protocol[4]. It works in all modes of instantiation, allowing us to use it to construct protocol runs for various sized groups while the refutation search is going on. We are taking advantage of two of CORAL's features here. The first is that we represent the trace as a variable which can later be instantiated to any number of messages by a single first-order rule - we don't have the notion of one transition in the model representing one message sent such as many model-checking approaches use. Additionally, we can define auxiliary functions like *all_msg2s_received* to check more complicated control conditions. Both features were necessary to create a realistic model of the protocol. All the details of the model are available via `http://homepages.inf.ed.ac.uk/gsteel/asokan-ginzboorg-model/`.

4.2. Attacking the Asokan–Ginzboorg Protocol

The protocol was tested against two attackers: one inside the room, and one outside. Different outcomes are considered successful for the different spies. Note that since we are analysing a wireless protocol, both spies are weaker than the standard Dolev-Yao spy – they cannot block messages from arriving. This is taken into account in our conjectures rather than by any change in the model, since following the Paulson model, we do not channel all messages through the spy, as for example the model used in (Basin et al., 2003) does.

Spy 1 - Outside the Room

This spy cannot see the whiteboard, so does not know the password. His objective is to effect a disruption attack. Since he cannot block messages or remove them from the airwaves, he must do this by adding messages in such a way as to disrupt the protocol run, e.g. by making honest participants accept keys that are not mutually shared.

[4] Note that this was only required for modelling *honest* agents sending message 3s, since they have to conform to the protocol. The intruder can send any combination of message 3s, no matter what message 2s have appeared in the trace. He is only constrained by what knowledge he can extract from previous messages in the trace, by the same rules as for regular protocols.

Spy 2 - In the Room

Asokan and Ginzboorg aimed, by means of the contributory key generation, to protect participants against conspiracy by a group of other participants trying to restrict the agreed key to a pre-chosen range. Other acts of dishonesty by agents in the room must therefore also be considered. We decided to test the protocol against a spy inside the room. This spy's capabilities differ from the first in that he knows the password written on the whiteboard, and is accepted as an honest agent. A disruption attack would be trivial for this spy (he could just refuse to send any messages), and he can learn the key just by taking part in the protocol, so instead his objective is to gain control of communication in the room, e.g. by making all participants agree on different keys that only he knows.

4.3. Attacks by a Spy Outside the Room

Finding attacks in Coral results from finding counterexamples to security properties. These are formulated in a similar way to Paulson's method. We must formulate the required properties in terms of the set of possible traces of messages. When looking for disruption attacks, we assume that the disruption comes about as a result of one agent not being able to read the message 3 intended for him. This is because the leader can see how many people are in the room, and so will at the very least send out one message 3 to each other member of the group. The following conjecture was used to check for these kinds of attacks:

%% some honest XI has sent message 4, so has key *f(Package)*:
eqagent(XI,spy)=false $\wedge$
member(sent(XI,XK,pair(principal(XI),
$encr(pair(nonce(SI), h(Package)), f(Package)))), Trace) = true\wedge$

%% genuine messages 3 and 1 are in the trace to some agent XJ:
member(sent(MN,XJ,encr(Package,nonce(RJ))),Trace)=true
member(sent(MN,all,pair(principal(MN),encr(key(E),key(P)))),
Trace)=true$\wedge$

%% but XJ never sent a message 2 under public key E with
%%nonces SJ (which is in *Package*) and RJ (which the message
%% 3 meant for him was sent under). That means he
%% dosen't have RJ, and so can't get the key from his message 3.

member(nonce(SJ),Package)=true ∧
member(sent(XJ,MN,pair(principal(XJ),
encr(pair(nonce(RJ),nonce(SJ)),key(E)))),xtrace)=false →

Note that the final arrow (implication) with nothing on the right hand side indicates that the whole conjecture is negative, i.e. it is in the style of a Prolog query of conjoined negated goals to a database of clauses. This conjecture may look somewhat contrived, but it is a natural way of expressing that a run has been finished (i.e. a message 4 has been sent) and that there is some agent who does not now have the key. It states that for all possible traces, no trace can have the combination of genuine messages 4, 3 and 1 without a corresponding message 2. When first run with this conjecture, CORAL produced the following counterexample for a group of size 2:

1.	M_2	$\rightarrow$	ALL	:	$M_2, \{\!\|\ E\ |\!\}_P$
1′.	spy_{M_1}	$\rightarrow$	ALL	:	$M_1, \{\!\|\ E\ |\!\}_P$
2′.	M_2	$\rightarrow$	M_1	:	$M_2, \{\!\|\ R_2, S_2\ |\!\}_E$
2.	spy_{M_1}	$\rightarrow$	M_2	:	$M_1, \{\!\|\ R_2, S_2\ |\!\}_E$
3.	M_2	$\rightarrow$	M_1	:	$\{\!\|\ S_2', S_2\ |\!\}_{R_2}$
3′.	spy_{M_1}	$\rightarrow$	M_2	:	$\{\!\|\ S_2', S_2\ |\!\}_{R_2}$
4.′	M_2	$\rightarrow$	M_1	:	$M_2, \{\!\|\ S_2, h(S_2', S_2)\ |\!\}_{f(S_2', S_2)}$

At the end of the run, M_2 now accepts the key $f(S_2', S_2)$ as a valid group key, but it contains numbers known only to M_2, and not to M_1. The attack requires that the spy manages to send message 2 before the honest agent M_1 can send her reply. In a single hop network, there is a certain amount of luck involved – it might require M_1 to invoke the command to establish a key just after M_2 does. This is certainly possible, so we should alter the protocol to protect against it. Encrypting the agent identifier in message 1 stops the spy from sending the fake message 1', preventing the attack. However, when we made this correction to the protocol, and ran CORAL again with the same conjecture, CORAL found the following counterexample for a group of size 3:

1. $M_1 \rightarrow \text{ALL} : \{\!|\, M_1, E \,|\!\}_P$
2. $M_2 \rightarrow M_1 : M_2, \{\!|\, R_2, S_2 \,|\!\}_E$
2. $spy_{M_3} \rightarrow M_1 : M_3, \{\!|\, R_2, S_2 \,|\!\}_E$
3. $M_1 \rightarrow M_2 : \{\!|\, S_2, S_2, S_1 \,|\!\}_{R_2}$
3. $M_1 \rightarrow M_3 : \{\!|\, S_2, S_2, S_1 \,|\!\}_{R_2}$
4. $M_2 \rightarrow M_1 : M_2, \{\!|\, S_2, h(S_2, S_2, S_1) \,|\!\}_{f(S_2,S_2,S_1)}$

This is another disruption attack, where the spy eavesdrops on the first message 2 sent, and then fakes a message 2 from another member of the group. This results in the protocol run ending with only two of the three person group sharing the key. This attack can also be prevented by a small change to the protocol, this time by encrypting the agent identifier in message 2 (see §4.5 below). CORAL took about 1.5 hours to find the first attack, and about 20 hours to find the second. With these two attacks prevented, CORAL found no further disruption attacks, though this does not constitute a verification (see §8).

4.4. Attacks by a Spy Inside the Room

When considering a scenario where one of the agents inside the room is a spy, we decided to consider what might be possible when all the players in the room think they have agreed on a key, but they have in fact 'agreed' on different ones. What if the spy knows all these keys? He could filter all the information exchanged, perhaps making subtle but important changes to a document in a pre-defined way, such that the other agents in the room are none the wiser. We checked for these kinds of attacks by giving CORAL the following conjecture:

% we have distinct honest agents XI and XJ
eqagent(XI,spy)=false $\wedge$
eqagent(XJ,spy)=false $\wedge$
eqagent(XJ,XI)=false $\wedge$

% they both sent message 2s in response to the same message 1
member(sent(XI,MN,pair(principal(XI),
encr$(pair(nonce(RI), nonce(SI)), key(E))))$, *Trace*$) = true\wedge$
member(sent(XJ,MN,pair(principal(XJ),
encr$(pair(nonce(RJ), nonce(SJ)), key(E))))$, *Trace*$) = true\wedge$

% and received message 3s under the correct keys, RI and RJ
member(sent(MN,XI,encr(Package1,nonce(RI))),Trace)=true$\wedge$

member(sent(MN,XJ,encr(Package2,nonce(RJ))),Trace)=true∧

% but the packages they received were different
eq(Package1,Package2)=false →

Note again that the conjecture is negative, i.e. it states that there is no trace for which this combination of conditions can hold. CORAL refuted this property in about 3 hours, producing the counterexample trace:

1.	spy	$\rightarrow$	ALL	:	$spy, \{\!\|\ E\ |\!\}_P$
2.	M_1	$\rightarrow$	spy	:	$M_1, \{\!\|\ R_1, S_1\ |\!\}_E$
2.	M_2	$\rightarrow$	spy	:	$M_2, \{\!\|\ R_2, S_2\ |\!\}_E$
3.	spy	$\rightarrow$	M_1	:	$\{\!\|\ S_1, S_2, S_{spy}\ |\!\}_{R_1}$
3.	spy	$\rightarrow$	M_2	:	$\{\!\|\ S_1, S_2, S'_{spy}\ |\!\}_{R_2}$
4.	M_1	$\rightarrow$	spy	:	$M_1, \{\!\|\ S_1, h(S_1, S_2, S_{spy})\ |\!\}_{f(S_1,S_2,S_{spy})}$

This attack is just a standard protocol run for three participants, except that in the second message 3, the spy changes his contribution to the key (S'_{spy} in the place of S_{spy}). This means that M_1 accepts the key as $f(S_1, S_2, S_{spy})$, whereas M_2 accepts $f(S_1, S_2, S'_{spy})$, and both of these keys are known to the spy.

4.5. An Improved Version of the Protocol

As mentioned above, we can prevent the disruption attacks by encrypting the agent identifiers in messages 1 and 2. To prevent the attack by the spy inside the room, we can require message 4 to be broadcast to all participants, so that everyone can check they have agreed on the same key. CORAL did not find any attacks on this revised version, though of course this does not mean it is verified as being correct. Here is the revised Asokan–Ginzboorg protocol, with the changes highlighted:

1.	M_n	$\rightarrow$	ALL	:	$\{\!\|\ \boxed{M_n}\ , E\ |\!\}_P$
2.	M_i	$\rightarrow$	M_n	:	$\{\!\|\ \boxed{M_i}\ , R_i, S_i\ |\!\}_E, i = 1, \ldots, n-1$
3.	M_n	$\rightarrow$	M_i	:	$\{\!\|\ \{Sj, j = 1, \ldots, n\}\ |\!\}_{R_i}, i = 1, \ldots, n-1$
4.	M_i	$\rightarrow$	$\boxed{\text{ALL}}$	:	$M_i, \{\!\|\ S_i, h(S_1, \ldots, S_n)\ |\!\}_K$, some i.

4.6. The Second Asokan–Ginzboorg protocol

The second Asokan–Ginzboorg protocol, (Asokan and Ginzboorg, 2000, p. 9), requires Diffie-Hellman exponentiation to be modelled, which we have not yet attempted in Coral. We expect that we could add equations to the model to capture the relevant properties of the exponentiation operation, but the combinatorial blow-up might be too great without development of further heuristics. However, two things are clear from inspection of the second protocol: the first is that it is certainly susceptible to the kind of disruption attack we uncovered on the first protocol, since a spy outside the room could quite easily fake a message 1. The second is that an attack of the kind given in §4.4 is prevented by the final message being broadcast to all participants.

5. The Tanaka–Sato/Taghdiri–Jackson Protocol

Our second and third case studies deal with group key management protocols, i.e. protocols designed to maintain a secure key for a dynamic group of agents. Group key management is an area of increasing interest, driven by applications such as database access management, virtual conferencing, secure online broadcast, etc.. Both the protocols examined here assume the use of a trusted key server.

The first protocol was proposed as a result of some formal analysis by Taghdiri and Jackson, (Taghdiri and Jackson, 2003). They analysed a protocol originally proposed by Tanaka and Sato, (Tanaka and Sato, 2001), which was primarily concerned with minimising the burden of key updates in terms of network traffic and processor time. Two main design features were introduced for this purpose: the first was the division of the group into subgroups, each under the management of a key distribution server (KDS). The communication between the KDSs is assumed to be not only secure but also conducted under a reliable totally ordered multicast protocol (RTOMP). Taghdiri and Jackson modelled this by assuming that as soon as one KDS updates its key, all the other KDSs instantaneously update theirs, effectively reducing the model to a single server. In our model, we also restrict attention to a single server.

The second design feature of the protocol is that agents retain a list of keys rather than just one key. They discard an old key as invalid t units of time after having received a more up-to-date key, where t is set with respect to the delay in the network. Keys are distributed only when an agent sends a request to the server. An agent will make such a request when he wants to send a multicast message, or if he

receives a message encrypted under a key he doesn't know already. In both cases, he will send a message to the server giving the ID number of the newest key he has, and the server will send back all newer keys. Only the newest key is used for multicast broadcasting.

This retention of a list of keys was shown in Taghdiri and Jackson's analysis to lead to major security problems. The most serious attack involved members of the group accepting messages from a principal outside the group. A member of the group M_1 can simply broadcast a message from inside the group, leave, and then broadcast a message using the same key. Though the group key has been updated as a result of M_1 leaving, the other agents in the group will still accept the second message as valid, as they all have the old key. To counter this, Taghdiri and Jackson suggested changes to the protocol. Each agent should retain only the most recent key he has received, and upon receiving a multicast message, should contact the server to confirm that it is encrypted under the newest key. This may result in some message loss: delay in the network might mean that by the time a multicast message has been received, and a key request sent to the server, the group key has changed, but this was reckoned to be acceptable compared to the potential security breach.

It is the improved version of the protocol we have modelled and analysed using CORAL. In doing so we were aiming to address one major oversight of the Taghdiri–Jackson analysis, namely the lack of an active intruder in their model. An active intruder of some kind has been assumed since the very first security protocol paper, (Needham and Schroeder, 1978). His behaviour was formalised by Dolev and Yao, (Dolev and Yao, 1983), and since then it has generally been accepted that the spy should also be able to pose as a legitimate agent, for example in Lowe's famous attack, (Lowe, 1996). If anything, it would seem even more likely that a multicast protocol would be subject to attack by an active intruder compared to a unicast protocol, as argued in (Mittra, 1997). There are inherently more opportunities for interception of traffic, and the 'crowd' of principals would typically make it easier for an intruder to pose as another legitimate principal. Such protocols should therefore be subjected to analysis under the full Dolev-Yao attacker model, as is standard for unicast protocols.

The Tanaka-Sato protocol assumes the existence of a unicast authentication protocol that allows the server to establish an individual key (Ik) with a new member joining the group. This Ik is used to encrypt all communication between that member and the server. We model the underlying authentication protocol by assuming the existence of a long-term key shared by each valid potential member of the group with the KDS. Since we are looking for attacks on the protocol rather

than trying to verify it, we can easily justify this. We can simply take the attacks we discover and examine them to see if the specific way we implemented the authentication phase was exploited. The attacks described in this paper would be effective for any initial authentication protocol. Additionally, we make the standard assumption that the spy has access to one valid long-term key, in order to impersonate an honest agent.

Here is a description of the improved version of the protocol as described by Taghdiri and Jackson:

Joining the Group

1. $M_i \rightarrow S : \{\!|\ \text{join}\ |\!\}_{K_{M_i}}$
2. $S \rightarrow M_i : \{\!|\ Ik_{M_i}, Gk(n) |\!\}_{K_{M_i}}$

In message 1, M_i wants to join the group, so sends a join request under his long-term key K_{M_i}. The server generates a fresh individual key, $Ik_M i$, and a new group key $Gk(n)$. Each group key has a unique ID number n. The new individual key and group key are sent to the joining member in message 2.

Leaving the Group

1. $M_i \rightarrow S : \{\!|\text{leave}|\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i : \{\!|\text{ack.leave}|\!\}_{Ik_{M_i}}$

In message 1, M_i sends a request to leave encrypted under his individual key Ik. The server acknowledges the leave in message 2, and generates a new group key. This key is not distributed though, and if another membership change occurs before a request for a key is received, it will never be distributed.

Sending a message

1. $M_i \rightarrow S : \{\!|\text{send,}n|\!\}_{Ik_{M_i}}$
2. $S \rightarrow M_i : \{\!|\ n', Gk(n') |\!\}_{Ik_{M_i}}$
3. $M_i \rightarrow ALL : \{\!|\text{message}|\!\}_{Gk(n')}$

In message 1, agent M_i signals to the server that he would like to send a message by sending what the protocol designers call a 'sequence request' message together with the ID number of the newest key he has, n. The server checks that M_i is in the group, and then sends back the newest key $Gk(n')$. If no joins or leaves have occurred since M_i last received a key, it may be that $n = n'$, but this will not be the case in general. In message 3, agent M_i broadcasts his message to the group.

Receiving a message

1. $M_j \rightarrow S : \{\!|\text{read,}n|\!\}_{Ik_{M_j}}$
2. $S \rightarrow M_j : \{\!|\ Gk(n') |\!\}_{Ik_{M_j}}$

Suppose a multicast message has been broadcast, as in message 3 of the 'sending a message' fragment above. When another agent M_j receives the message, he first sends a request to the server for the newest key. He then receives the newest key $Gk(n')$, and will only accept the multicast message if it was encrypted under that key.

The revised protocol as proposed by Taghdiri and Jackson contains some redundancy as a result of their security improvements. For example, there is no reason for the server to send the key to a new member when he joins, since he is required to ask for a key update whenever he sends or receives a multicast message. Additionally, the sequence number sent in the request for a key update before sending a message also seems redundant. Previously, the server would have used it to decide which keys to send back, but in the revised version, the server only ever sends back the most recent key. It would be better to replace this with a nonce, as we argue in §5.2 after presenting the attacks we discovered.

5.1. Modelling TSTJ

A feature of our model was that all the information about the state of the system, i.e. the state and knowledge of all the principals involved, was stored in the trace and inferred from the trace each time it was needed. First-order rules can then be used to add an arbitrary number of messages to the trace in a single instant. This was particularly useful when we were modelling the Asokan–Ginzboorg protocol, as explained in §4. However, for the Taghdiri-Jackson protocol, this was not so helpful. Some information about the state of the system does not normally appear in the trace. For example, when an agent leaves the group, the server generates a new key, but this key does not appear in the trace. Additionally, our model follows Paulson's original design in that it does not include a 'gets' event to model message reception, such as Bella later introduced to Paulson's model, (Bella, 1999). The only events in the trace are 'sent' events, and in the presence of a Dolev-Yao attacker, these messages may never be received by their intended recipients. This makes it hard to work out who is legitimately in the group at a particular time, which is vital when modelling the control conditions, i.e. tests that honest agents apply before sending a protocol message. Even with a 'gets' event, a lot of digging through message trace would be required to determine group composition, and we would need to do this almost every time an agent sent a message, creating an enormous search problem. So, the model was changed to include some information about the state of the principals. The unary function $m()$ that was previously used to store just the message trace is now an arity

4 function storing the trace, a counter, the current group key stored by the server, and the composition of the group stored as a list of triples. The triples store the agents name, the individual key which he shares with the server for this session in the group, and the most recent group multicast key he has received. We define a boolean function *ingroup* on these lists of triples that determine whether or not a particular agent is in the group. A further change is our modelling of freshness. We used to use the *parts* operator as used by Paulson, but in our model for this protocol, we have a counter, and use this to model fresh values. Effectively, we are adopting a discrete-time model where each event in the trace, that is the sending of each messages, is assumed to take one unit of time. Whenever a fresh value is required, such as a fresh nonce, this is given the name $nonce(Tick)$ where $Tick$ is the current counter value. We model multicast messages as $hello(Tick)$, thus ensuring these are also unique. Our motivation for this was that so many fresh values have to be created in a typical scenario, for individual keys, group keys and multicast messages, that our checking of the *parts* literals would quickly slow down the search process. Being able to make these kinds of changes to our model to suit a slightly different kind of protocol is one of the advantages of our first-order logic, trace based approach. Changes like these would be much harder using a tool where strong assumptions about the kinds of protocol to be analysed are hard-wired, as we explain in §7.

Having chosen to use a counter-based model and discarded the *parts* operator, our third pruning rule described in §3.3 was made redundant. However, an analogous pruning rule was simple to add: if the counter variable occurs in term X in a literal $ingroup(X, Y, Z) = true$, then the clause is redundant, since this would require an agent at some point in the past, to have joined the group and obtained a group key, or individual key, that is only available now. A similar check is applied to $member(X, Y) = true$ literals. This rule proved to be highly effective, and could be generally applied to backwards search tools using a counter-based model.

As an illustration, in Figure 1, we give the clauses required for modelling the sub-protocol for the sending of multicast messages. Note that in a further change from our original model, we record the composition of the group at each point in time in the fourth argument of the *sent* constructor. This is important for making conjectures about security properties later on. Note also that *ingroup* is an arity 3 function, with the third argument returning a list of group members without the agent named in the first argument. This is used when agents leave the group or update their keys, as in the third clause in Figure 1. As for the Asokan–Ginzboorg protocol, it again proved useful to be able to add

a new function to the theory to deal with the control conditions for a group protocol. A further point to note is that we still infer some state information about principals from the trace, for example to decide if they should be expecting a key update message in the third clause.

```
%% SEND a message
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,key(Sq)),Group,Newgp)=true
→ m(cons(sent(Mi,server,encr(send(Sq),Ikey),Group),Trace),
   Group,Keysequence,s(Tick))=true

%% server gives key
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true ∧
member(sent(X,server,encr(send(Sq),Ikey),Tgroup),Trace)=true
→ m(cons(sent(server,Mi,encr(pair(key(Keysequence),send(Sq)),Ikey)
   ,Group),Trace),Group,keysequence,s(Tick))=true

%% agent broadcasts his message, updates his key
m(Trace,Group,Keysequence,Tick)=true ∧
ingroup(triple(principal(Mi),Ikey,Oldk),Group,Newgp)=true ∧
member(sent(X,Mi,encr(pair(key(Xk),send(Sq)),Ikey),Tg1),Trace)=true
∧
member(sent(Mi,server,encr(send(Sq),Ikey),Tg2),Trace)=true
→                    m(cons(sent(Mi,all,encr(hello(s(Tick)),key(Xk)),
cons(triple(principal(Mi),Ikey,key(Xk)),Newgp)),Trace),
   cons(triple(principal(Mi),Ikey,key(Xk)),Newgp),
   Keysequence,s(Tick))=true
```

Figure 1. Clauses for modelling the 'send' sub-protocol

5.2. Attacking the Protocol

In (Pereira and Quisquater, 2003), Pereira and Quisquater attempt to lay down a list of desirable security properties for fixed-network group protocols. They define *implicit key authentication*, that an outsider cannot learn the group key; two flavours of *perfect forward secrecy*, i.e. that the compromise of long-term keys does not compromise past session keys; and *resistance to known-key attacks*, i.e. that compromise of session keys does not lead to the loss of future session keys. However, the properties Taghdiri and Jackson found not to be satisfied by the original protocol design fall outside of this categorisation. Essentially, this is because we are analysing a protocol for managing a group key for an evolving group, not just establishing a key for a static one.

The property vital to a key management protocol is that throughout the evolution of the group, agents currently outside the group should not be accepted as group members by the agents inside the group. We could perhaps call this *multicast group authenticity*. For this protocol, this property has two flavours: the first, which Taghdiri and Jackson call 'outsider can't read', implies that no agent outside the group should be able to read a message sent by a member of the group. The second, which they call 'outsider can't send', implies that members of the group should not accept as valid a message sent from outside the group.

Posing security conjectures in an inductive formalism requires some thought. We must translate an abstract idea of authenticity into a property expressed in terms of messages in the trace. For group protocol properties, our trace includes the composition of the group at each point, which becomes important now. For the property 'outsider can't read', we need to express the fact that when our dishonest agent is indeed outside the group, and a message is sent by an honest player under a key *Gk*, the spy does not know this key. So, We posed the property as a conjecture to CORAL in this form:

```
% For honest agent Mj
eqagent(Mj,spy)=false ∧

% There is a trace containing the sequence of
% messages for Mj broadcasting to the group under Gk
m(cons(sent(Mj,all,encr(hello(Y),Gk),Xgroup),
   cons(sent(X,Mj,encr(pair(Gk,send(Sq2)),Ikey),Xgroup),
   cons(sent(Mj,server,encr(send(Sq2),Ikey),Xgroup),
   Trace))),Group,Keyseq,Tick)=true ∧

% and the spy is not in the group
ingroup(triple(principal(spy),X3,X2),Xgroup,Newgp)=false ∧


% but the spy has the key Gk
in(Gk,analz(Trace)=true →
```

This conjecture is negative, i.e. it states there should be *no* trace *Trace* ending with the 3 messages specified in the first literal, with the spy outside the group, and with the message $hello(y)$ being sent under a key the spy knows (recall that $analz(X)$ is the set of terms the spy can learn from a trace X). The three final messages had to be specified together because otherwise CORAL (correctly) presents a counterexample trace in which the spy leaves the group between the server sending a key update out to Mj and Mj broadcasting his message. Then he can read the message quite legitimately, since he was in the group when

it was sent. Given the above conjecture, CORAL gives the following counterexample:

1. spy → server : $\{\!|\, spy \,|\!\}_{longtermK(spy)}$
2. server → spy : $\{\!|\, Ik(1), Gk(1) \,|\!\}_{longtermK(spy)}$
3. M_1 → server : $\{\!|\, M_1 \,|\!\}_{longtermK(M_1)}$
4. server → M_1 : $\{\!|\, Ik(3), Gk(2) \,|\!\}_{longtermK(M_1)}$
5. spy → server : $\{\!|\, send(1) \,|\!\}_{Ik(1)}$
6. server → spy : $\{\!|\, Gk(2), send(1) \,|\!\}_{Ik(1)}$
7. M_1 → server : $\{\!|\, send(2) \,|\!\}_{Ik(3)}$
8. server → M_1 : $\{\!|\, Gk(2), send(2) \,|\!\}_{Ik(3)}$
9. M_1 → all : $\{\!|\, hello(9) \,|\!\}_{Gk(2)}$
10. spy → server : $\{\!|\, leave \,|\!\}_{Ik(1)}$
11. server → spy : $\{\!|\, ackleave \,|\!\}_{Ik(1)}$
12. M_1 → server : $\{\!|\, send(2) \,|\!\}_{Ik(3)}$
13. spy → M_1 : $\{\!|\, Gk(2), send(2) \,|\!\}_{Ik(3)}$
14. M_1 → all : $\{\!|\, hello(14) \,|\!\}_{Gk(2)}$

This is an attack on the protocol which hinges on the spy sending a replayed key update message in message 13. Since the key may or may not have changed since she last saw it, agent M_1 will accept this key. The problem is that there is minimal freshness information sent in the request for a key (just the sequence number of the key an agent currently holds). Enclosing a fresh nonce inside the package sent to the server requesting a key update would blunt this attack. Having discovered this attack, we realised that there is a similar one whereby a spy can send a message from outside the group and have it accepted by an agent inside the group, thus breaking the multicast other group authenticity property, 'outsider can't send'. We gave an appropriate conjecture to CORAL for confirmation, and it discovered the following counterexample:

1. M_1 → server : $\{\!|\, M_1 \,|\!\}_{longtermK(M_1)}$
2. server → M_1 : $\{\!|\, Ik(1), Gk(1) \,|\!\}_{longtermK(M_1)}$
3. spy → server : $\{\!|\, spy \,|\!\}_{longtermK(spy)}$
4. server → spy : $\{\!|\, Ik(3), Gk(2) \,|\!\}_{longtermK(spy)}$
5. spy → server : $\{\!|\, read \,|\!\}_{Ik(3)}$
6. server → spy : $\{\!|\, Gk(2) \,|\!\}_{Ik(3)}$
7. M_1 → server : $\{\!|\, read \,|\!\}_{Ik(1)}$
8. server → M_1 : $\{\!|\, Gk(2) \,|\!\}_{Ik(1)}$

9. spy → server : $\{\!|\, leave \,|\!\}_{Ik(3)}$
10. server → spy : $\{\!|\, ackleave \,|\!\}_{Ik(3)}$
11. spy → all : $\{\!|\, hello(12) \,|\!\}_{Gk(2)}$
12. M_1 → server : $\{\!|\, read \,|\!\}_{Ik(1)}$
13. spy → M_1 : $\{\!|\, Gk(2) \,|\!\}_{Ik(1)}$

Like the previous attack, this is also a replay attack, in this case with the spy replaying message 8 in message 13, tricking agent M_1 into thinking that message 11 came from a legitimate member of the group. These replay attacks are more serious than those typically considered for fixed party protocols. A replay attack on a standard unicast protocol usually assumes that it would be possible for a spy to obtain a short-term key by cryptanalysis or some other means, and so it would constitute an attack on the protocol if he was able to force an agent to accept an old key. In the two attacks above, no cryptanalysis in necessary, since the spy can obtain some old keys by joining the group legitimately, and then leave before effecting the attack. However, even if we assume the spy does not have access to a valid long-term key, and so cannot join the group, these replay attacks are still dangerous. If the spy obtains a short-term group key by cryptanalysis, he can effect an attack without joining the group.

This attack can also be prevented by adding a fresh nonce to the request for a key, this time for reading a message, and including it in the reply from the server. A complete listing of the protocol model file is available at
`http://homepages.inf.ed.ac.uk/gsteel/tanaka-sato/`.

6. The Iolus Protocol

Our final case study is the Iolus protocol, (Mittra, 1997). The main difference between the Iolus protocol and the Taghdiri-Jackson version of the Tanaka-Sato protocol is that Iolus eagerly distributes new keys, whereas Tanaka-Sato distributes keys only on demand. Here is the protocol:

Joining the Group

1. M_i → S : $\{\!|\, \text{join} \,|\!\}_{K_{M_i}}$
2. S → M_i : $\{\!|\, Ik_{M_i}, Gk(n') \,|\!\}_{K_{M_i}}$
3. S → ALL : $\{\!|\, Gk(n') \,|\!\}_{Gk(n)}$

Members join the Iolus protocol in the same way as for Tanaka-Sato, i.e. by use of a pairwise authentication protocol that we model with

the use of a long-term key. The server generates a fresh individual key, $Ik_M i$, and a new group key with ID n', $Gk(n')$. In message 2, the group key is sent to the new member, and in message 3, it is sent to the old members of the group under the old group key, $Gk(n)$.

Leaving the Group

1. $M_i \rightarrow S : \{\!|\text{leave}|\!\}_{Ik_{M_i}}$
2. $S \rightarrow ALL : [\{\!|\, Gk(n')|\!\}_{Ik_{M_j}} \ldots] \, \forall j \neq i, M_j \in \text{group}$

When a member M_i leaves, a new key $Gk_{n'}$ is generated and sent to each member in the form of a broadcast list. The list contains the new group key encrypted under the pairwise session key of each member still in the group (the key cannot be broadcast under the old group key, because this would give it away to the leaving member).

Sending a message

1. $M_i \rightarrow ALL : \{\!|\text{message}|\!\}_{Gk(n)}$

The message is simply broadcast under the current group key.

6.1. Modelling Iolus

No changes were required to the framework of the Coral model to formalise the Iolus protocol. Though the protocol distributes new keys eagerly rather than lazily, the operations are similar to those used in the Tanaka-Sato protocol. The most complex part of the model concerns the second message of the 'leave' sub protocol. Here we must model the generation of a list for an arbitrary group. This is a straightforward task in our first-order model. We define a recursive function *rekey* that generates an appropriate rekeying message for a given group and given fresh group key. This function works correctly in all modes of instantiation, i.e. given a rekeying message it will return an appropriate group and key. This is important in our backwards search process. The use of an auxiliary *rekey* function is similar to our use of the *all_msg2s_received* function in our model of the Asokan–Ginzboorg protocol (see §4.1). Being able to make these kinds of recursive calculations seems (perhaps unsurprisingly) to be important in modelling group protocols, and our first-order inductive model is well suited to them.

6.2. Attacking Iolus

Having posed security conjectures for the Tanaka-Sato/Taghdiri-Jackson protocol above, formulating an appropriate conjecture for the Iolus protocol was easier. Again, we are investigating multicast group authenticity, i.e. that those outside the group cannot successfully impersonate

those inside the group, either by sending or receiving messages. However, since keys are supplied eagerly, we do not have the same division of this property into 'outside can't read' and 'outsider can't send' conjectures. Instead we have to look at the possible ways keys can be updated. The desirable property is: when a key update to key *Gk* is accepted by a member of a group, and that group does not contain the spy, then the spy should not know the key *Gk*. We formulate this property with respect to the updates sent after a group member leaves like this:

```
% For honest agent Mj
eqagent(Mj,spy)=false ∧
% Mj accepts an update to key Gk
m(cons(sent(X,all,cons(encr(Gk,Ikey),Rest),
   cons(triple(Mj,Ikey,OldGk),Restgp)),Trace),
   Group,Xk,Tick)=true ∧
% the spy is not in the group
ingroup(triple(principal(spy),Y,Z),Restgp,Newgp)=false ∧
% but he knows the key
in(Gk,analz(Trace))=true →.
```

Again this is a negative conjecture suggesting that for the protocol to be secure, no trace should exist where the spy knows a key *Gk* accepted by group member *Mj* when the spy is outside the group. CORAL finds the following counterexample:

1. $M_1 \rightarrow \text{server} : \{\!|\, M_1 \,|\!\}_{longtermK(M_1)}$
2. $\text{server} \rightarrow \text{all} : \{\!|\, Gk(1) \,|\!\}_{Gk(X8)}$
3. $\text{server} \rightarrow M_1 : \{\!|\, Ik(2), Gk(1) \,|\!\}_{longtermK(M_1)}$
4. $\text{spy} \rightarrow \text{server} : \{\!|\, spy \,|\!\}_{longtermK(spy)}$
5. $\text{server} \rightarrow \text{all} : \{\!|\, Gk(2) \,|\!\}_{Gk(1)}$
6. $\text{server} \rightarrow \text{spy} : \{\!|\, Ik(5), Gk(2) \,|\!\}_{longtermK(spy)}$
7. $M_2 \rightarrow \text{server} : \{\!|\, M_2 \,|\!\}_{longtermK(M_2)}$
8. $\text{server} \rightarrow \text{all} : \{\!|\, Gk(3) \,|\!\}_{Gk(2)}$
9. $\text{server} \rightarrow M_2 : \{\!|\, Ik(8), Gk(3) \,|\!\}_{longtermK(M_2)}$
10. $M_1 \rightarrow \text{server} : \{\!|\, leave \,|\!\}_{Ik(2)}$
11. $\text{server} \rightarrow \text{all} : [\{\!|\, Gk(4) \,|\!\}_{Ik(8)}, \{\!|\, Gk(4) \,|\!\}_{Ik(5)}]$
12. $\text{spy} \rightarrow \text{server} : \{\!|\, leave \,|\!\}_{Ik(5)}$
13. $\text{server} \rightarrow \text{all} : [\{\!|\, Gk(5) \,|\!\}_{Ik(8)}]$
14. $\text{spy} \rightarrow \text{all} : [\{\!|\, Gk(4) \,|\!\}_{Ik(8)}, \{\!|\, Gk(4) \,|\!\}_{Ik(5)}]$

Again this is a replay attack. In message 14, the spy replays a key update originally sent in message 11, while he was still in the group.

So, the spy knows the group key $Gk(4)$ even though he is no longer a group member. Note for this attack to work, it is necessary for two honest agents to join the group as well as the spy, whereas only one was required for the attacks in §5, and note further that CORAL has discovered this for itself - there is no pre-setting of the number of agents. Preventing this attack is not as straightforward as it was for the Tanaka-Sato/Taghdiri-Jackson protocol because the key updates are unsolicited, so there is no opportunity for the agents and the server to exchange a nonce. The only way to protect against replays would seem to be to include a timestamp inside the encrypted packages sent in key updates, both when agents join and leave the group. This would require all group members to have at least loosely synchronised clocks, and would further require a decision in advance about the lifetime of group keys, and the expected amount of delay in the network. However, this limitation seems inescapable for such protocols. In the presence of a Dolev-Yao spy it is insufficient, for example, to include the last key in the key update message as freshness information, since the spy may prevent this message from being received, and then re-send it once he has left the group.

A complete listing of the Iolus protocol model file is available at `http://homepages.inf.ed.ac.uk/gsteel/iolus/`.

7. Evaluation

We have seen in these case studies the re-emergence in new protocols of oversights made by the designers of the first security protocols. The situations these protocols are designed to deal with are sufficiently different from the normal 2 or 3 party scenario to make the old mistakes hard to spot. For example, we used to consider protocols that were just played through once and then finished, whereas the group key management protocols allow users to join, leave and re-join as many times as they please. The oversights in the protocol design seem to be a direct consequence of the extra complexity introduced by these temporal considerations. In the case of the Asokan–Ginzboorg wireless protocol, the property of 'resistance to disruption' is a relatively new idea. The attacks we discovered show that it is more difficult to achieve than the designers expected. All this is evidence for the value of formal methods in the process of protocol design, and in particular, for the value of CORAL in the analysis of group protocols. We have seen the value of a model that does not require us to pre-set the number of agents involved in a group - only two agents were required for the attacks on the Tanaka-Sato/Taghdiri-Jackson protocol, but three are necessary for

the Iolus protocol attack. More significantly, CORAL has also to discover the scenario for the attacks on the group management protocols, i.e. who will join, who will leave, when they will send a message etc.. This is a more difficult problem than the mere interleaving of sessions normally dealt with by protocol analysers. Tools requiring parameterisation of the scenario, e.g. (Basin et al., 2003; Chevalier and Vigneron, 2002), would need some development before they could perform this search, since the key updates sent by the server vary depending on the composition of the group.

There were two weaker aspects to CORAL's performance: one was the difficulty of posing conjectures. For the first case study, it took several attempts to pose the security property in such a way that counterexamples really were attacks. This is particularly annoying when it takes several hours to get the counterexample. Some of the difficulty comes from the fact that is not trivial to translate the kinds of authenticity properties required of unicast protocols to group protocol situations. As we saw in §5.2, the properties outlined by Pereira and Quisquater for group key establishment protocols are also unsuitable for a dynamic group situation. For example, the property of perfect forward secrecy, where the compromise of long-term secrets does not lead to the compromise of past sessions, does not hold for either of the protocols analysed in this paper, since they both assume an authentication protocol using long-term secrets is used to set up the pairwise keys for communication with the server. Compromise of these would lead to the compromise of all session keys from the period when the compromised agent was in the group. It is clear the designers of these multicast key management protocols have not aimed to provide this kind of perfect forward secrecy. Instead, we have identified *multicast group authenticity* as the key property, i.e. throughout the evolution of the group, keys used by group members must be known only to other group members. Having analysed the first protocol, it was much easier to form the property required for the second. There has been some work on automating the process of formulating conjectures in the inductive model, (Monroy and Carrillo, 2003), and this can perhaps be adapted to formulate properties for group key management protocols.

The second weaker aspect of CORAL's performance was the run times. It may be possible to improve on this by inspecting the output closely and looking for more heuristics to add to those in §3.3. However, the search spaces created by CORAL's open model are always going to be large, particularly when dealing with group protocols, so we perhaps should not expect too much.

At the time of writing, CORAL is the only tool that finds attacks on group protocols automatically in a general model, with no restric-

tions on agents and roles. Meadows' NPA has been used to attack the GDOI protocol, but only after restricting the model to one agent and a server, (Meadows and Syverson, 2001). We have seen that Taghdiri and Jackson's model was inadequate, because it did not include any kind of active intruder, (Taghdiri and Jackson, 2003). It would not be easy to make adaptations to most tools for fixed-party protocol analysis in order to make them more suitable for group protocol analysis. For example, the Athena tool has achieved impressive results on fixed party protocols, (Song et al., 2001). Athena uses an extension of the strand space model, (Fábrega et al., 1999). One of Athena's requirements is that a so-called semi-bundle should be finite. However, this would be impossible for semi-bundles representing the protocols analysed here. A strand for the Asokan–Ginzboorg protocol may contain an arbitrary number of message 3s, and strands for the group key management protocols must allow agents to join and leave an arbitrary number of times. Another leading tool, OFMC, (Basin et al., 2003), is constrained to a one to one relationship between transitions and messages sent. Additionally, it requires the user to choose finite roles for the agents to play before the analysis can start. These restrictions would prevent the modelling of group protocols in an open scenario. You could parameterise the scenario with respect to the number of agents, but in its current implementation, OFMC would also require you to parameterise exactly when each agent would join, leave, and send messages, because the key updates would have to be worked out in advance to correspond to the composition of the group at the time. Of course, it may be possible to make changes to the tools to overcome these difficulties, but it would involve some significant work. Our case studies highlight the areas that would need attention.

8. Conclusions

We have presented Coral, our system for refuting incorrect inductive conjectures, and shown 6 new attacks it has been used to discover. We were pleased with the way Coral performed on these protocols. Firstly, the use of an inductive model meant we didn't have to make fundamental changes to our modelling strategy to accommodate an unbounded number of agents, an open-ended protocol, lists of keys in messages, an unspecified scenario in terms of joining and leaving the group, etc. Secondly, modelling at in first-order logic level in a theorem prover meant making the adaptations required to deal with complex control conditions, for example to store and manipulate a list of current group members, was straightforward. Thirdly, in the second

and third case studies, CORAL was able to discover attacks requiring a long trace of messages to be sent, indicating it has scaled up well, despite exploring a model without any pre-setting of the number of agents, joins and leaves etc.

A major area of future development is improving CORAL's ability to verify correct protocols. At present, CORAL can only complete verifications when restrictions are applied to the model, such as restricting the number of agents or the length of the trace, which does not offer an advantage over model checking type approaches. Our inductive framework does offer the possibility of completing proofs in an unrestricted scenario, but work will be needed to improve the redundancy checking of newly generating clauses, possibly by using critics, (Walsh, 1996), or by developing domain specific redundancy tactics.

In other future work, we are using CORAL to refute incorrect conjectures in other areas of formal verification. We have carried out some experiments using CORAL to suggest counterexamples to buggy functional programs written by undergraduate students. This looks like a promising area of application. Another area of interest is the discovery of attacks on the APIs of the crypto-processors used in hardware security modules, such as are used in cash machines and automated payment devices. We aim eventually to use CORAL to find attacks that require brute force guess complexity considerations to be taken into account.

References

Asokan, N. and P. Ginzboorg: 2000, 'Key-Agreement in Ad-hoc Networks'. *Computer Communications* **23**(17), 1627–1637.

Ateniese, G., M. Steiner, and G. Tsudik: 2000, 'New Multiparty Authentication Services and Key Agreement Protocols'. *IEEE Journal on Selected Areas in Communications* **18**(4), 628–639.

Bachmair, L. and H. Ganzinger: 1990, 'Completion of First-Order Clauses with Equality by Strict Superposition (Extended Abstract)'. In: *Proceedings 2nd International CTRS Workshop*. Montreal, Canada, pp. 162–180.

Bachmair, L. and H. Ganzinger: 1991, 'Perfect Model Semantics for Logic Programs with Equality'. In: *Logic Programming, Proceedings of the Eigth International Conference*. Paris, France, pp. 645–659, MIT Press.

Basin, D., S. Mödersheim, and L. Viganò: 2003, 'An On-The-Fly Model-Checker for Security Protocol Analysis'. In: *Proceedings of the 2003 European Symposium on Research in Computer Security*. pp. 253–270. Extended version available as Technical Report 404, ETH Zurich.

Bella, G.: 1999, 'Message Reception in the Inductive Approach'. Technical Report 460, Computer Laboratory, University of Cambridge.

Bull, J. and D. Otway: 1997, 'The authentication protocol'. Technical Report DRA/CIS3/PROJ/CORBA/SC/1/CSM/436-04/0.5b, DERA, Malvern, UK.

Chevalier, Y. and L. Vigneron: 2002, 'Automated Unbounded Verification of Security Protocols'. In: E. Brinksma and K. Larsen (eds.): *Computer Aided Verification, 14th International Conference*, Vol. 2404 of *Lecture Notes in Computer Science*. Copenhagen, Denmark, pp. 324–337, Springer.

Clark, J. and J. Jacob: 1997, 'A Survey of Authentication Protocol Literature: Version 1.0'. Available via `http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz`.

Comon, H. and R. Nieuwenhuis: 2000, 'Induction = I-Axiomatization + First-Order Consistency'. *Information and Computation* **159**(1-2), 151–186.

Denker, G. and J. Millen: 2000, 'CAPSL integrated protocol environment'. In: *DARPA Information Survivability Conference and Exposition*, Vol. 1. pp. 207–221.

Diffie, W. and M. Helman: 1976, 'New directions in cryptography.'. *IEEE Transactions on Information Theory* **22**(6), 644–654.

Dolev, D. and A. Yao: 1983, 'On the security of public key protocols'. *IEEE Transactions in Information Theory* **2**(29), 198–208.

Fábrega, F., J. Herzog, and G. J.: 1999, 'Strand Spaces: Proving Security Protocols Correct'. *Journal of Computer Security* **7**, 191–230.

Green, C.: 1969, 'Theorem Proving by Resolution as a Basis for Question-Answering Systems'. In: B. Meltzer and D. Michie (eds.): *Machine Intelligence*, Vol. 4. pp. 183–208, Edinburgh University Press.

Jackson, D.: 2002, 'Alloy: a lightweight object modelling notation'. *ACM Transactions on Software Engineering and Methodology (TOSEM)* **11**(2), 256–290.

Lowe, G.: 1996, 'Breaking and Fixing the Needham Schroeder Public-Key Protocol using FDR'. In: *Proceedings of TACAS*, Vol. 1055. pp. 147–166, Springer Verlag.

Meadows, C.: 2000, 'Extending Formal Cryptographic Protocol Analysis Techniques for Group Protocols and Low-Level Cryptographic Primitives'. In: P. Degano (ed.): *Proceedings of the First Workshop on Issues in the Theory of Security*. Geneva, Switzerland, pp. 87–92.

Meadows, C.: 2003, 'Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends'. *IEEE Journal on Selected Areas in Communication* **21**(1), 44–54.

Meadows, C. and P. Syverson: 2001, 'Formalizing GDOI group key management requirements in NPATRL'. In: *ACM Conference on Computer and Communications Security 2001*. pp. 235–244.

Millen, J. and G. Denker: 2003, 'MuCAPSL'. In: *DISCEX III, DARPA Information Survivability Conference and Exposition*. pp. 238–249.

Mittra, S.: 1997, 'Iolus: A Framework for Scalable Secure Multicasting'. In: *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*. Cannes, France, pp. 277–288.

Monroy, R. and M. Carrillo: 2003, 'On Automating the Formulation of Security Goals under the Inductive Approach'. In: M. H. Hamza (ed.): *Applied Informatics*. pp. 1020–1025, IASTED/ACTA Press.

Musser, D.: 1980, 'On Proving Inductive Properties of Abstract Data Types'. In: *Proceedings 7th ACM Symp. on Principles of Programming Languages*. pp. 154–162, ACM.

Needham, R. and M. Schroeder: 1978, 'Using Encryption for Authentication in Large Networks of Computers'. *Communications of the ACM* **21**(12), 993–999.

Paulson, L.: 1998, 'The Inductive Approach to Verifying Cryptographic Protocols'. *Journal of Computer Security* **6**, 85–128.

Pereira, O. and J.-J. Quisquater: 2003, 'Some attacks upon authenticated group key agreement protocols'. *Journal of Computer Security* **11**(4), 555–580. Special Issue: 14th Computer Security Foundations Workshop (CSFW14).

Song, D., S. Berezin, and A. Perrig: 2001, 'Athena: A Novel Approach to Efficient Automatic Security Protocol Analysis'. *Journal of Computer Security* **9**(1/2), 47–74.

Steel, G.: 2004, 'Discovering Attacks on Security Protocols by Refuting Incorrect Inductive Conjectures'. Ph.D. thesis, University of Edinburgh. Electronic copy available on request from the author: `graham.steel@ed.ac.uk`.

Steel, G., A. Bundy, and M. Maidl: 2004, 'Attacking a Protocol for Group Key Agreement by Refuting Incorrect Inductive Conjectures'. In: D. Basin and M. Rusinowitch (eds.): *Proceedings of the International Joint Conference on Automated Reasoning*. Cork, Ireland, pp. 137–151, Springer-Verlag Heidelberg.

Steiner, M., G. Tsudik, and M. Waidner: 1996, 'Diffie-Hellman key distribution extended to group communication'. In: *Proc. 3rd ACM Conference on Computer and Communications Security (CCS' 96)*. pp. 31–37.

Syverson, P., C. Meadows, and I. Cerversato: 2000, 'Dolev-Yao is no better than Machiavelli'. In: P. Degano (ed.): *Proceedings of the First Workshop on Issues in the Theory of Security*. Geneva, Switzerland, pp. 87–92.

Taghdiri, M.: 2002, 'Lightweight Modelling and Automatic Analysis of Multicast Key Management Schemes.'. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Taghdiri, M. and D. Jackson: 2003, 'A Lightweight Formal Analysis of a Multicast Key management Scheme'. In: *Proceedings of Formal Techniques of Networked and Distributed Systems - FORTE 2003*. Berlin, pp. 240–256, Springer.

Tanaka, S. and F. Sato: 2001, 'A Key Distribution and Rekeying Framework With Totally Ordered Multicast Protocols'. In: *Proceedings of the 15th International Conference on Information Networking*. pp. 831–838.

Walsh, T.: 1996, 'A Divergence Critic for Inductive Proof'. *Journal of Artificial Intelligence Research* **4**, 209–235.

Weidenbach, C.: 2001, 'Combining Superposition, Sorts and Splitting'. In: A. Robinson and A. Voronkov (eds.): *Handbook of Automated Reasoning*, Vol. II. Elsevier Science, Chapt. 27, pp. 1965–2013.