Attacking a Protocol for Group Key Agreement by Refuting Incorrect Inductive Conjectures

Graham Steel, Alan Bundy, and Monika Maidl

School of Informatics,
University of Edinburgh,
Edinburgh, EH8 9LE, Scotland,
e-mail: g.j.steel@ed.ac.uk,{bundy,monika}@inf.ed.ac.uk
http://dream.dai.ed.ac.uk/graham

Abstract. Automated tools for finding attacks on flawed security protocols often struggle to deal with protocols for group key agreement. Systems designed for fixed 2 or 3 party protocols may not be able to model a group protocol, or its intended security properties. Frequently, such tools require an abstraction to a group of fixed size to be made before the automated analysis takes place. This can prejudice chances of finding attacks on the protocol. In this paper, we describe CORAL, our system for finding security protocol attacks by refuting incorrect inductive conjectures. We have used CORAL to model a group key protocol in a general way. By posing inductive conjectures about the trace of messages exchanged, we can investigate novel properties of the protocol, such as tolerance to disruption, and whether it results in agreement on a single key. This has allowed us to find three distinct novel attacks on groups of size two and three.

1 Introduction

The aim of cryptographic security protocols is to prescribe a way in which users can communicate securely over an insecure network. A protocol describes an exchange of messages in which the principals involved establish shared secrets, in order perhaps to communicate privately or to protect themselves from impersonators. These protocols are designed to be secure even in the presence of an active attacker, who may intercept or delay messages and send faked messages in order to gain access to secrets. Unsurprisingly, given this hostile operating environment, they have proved very hard to get right. What's more, protocol flaws are often quite subtle. New attacks are often found on protocols many years after they were first proposed.

In the last five years or so, there has been an explosion of interest in the idea of applying formal methods to the analysis of these protocols. Researchers have used techniques from model checking, term rewriting, theorem proving and logic programming amongst others, [19, 20, 27]. However, very few of these are able to analyse protocols for group key agreement, where an unbounded number of parties may be involved in a single round, [21, 27]. Significant attacks on such

protocols have appeared in the literature, but these have been discovered by hand, [28]. A problem for many automated approaches is that they can only attack concrete models of protocols, and so require the size of the group to be chosen in advance. This can prejudice the chances of discovering an attack. In this paper, we model such a protocol in a general way, without predetermining group size. The protocol in question is the Asokan–Ginzboorg protocol for key establishment in an ad-hoc Bluetooth network, [2]. Our formalism is a first-order version of the inductive model proposed by Paulson, [27]. The use of a first-order model allows us to search for counterexamples using automatic methods, which is not supported in Paulson's approach. We show how our counterexample finder for inductive conjectures, Coral, has been used to automatically discover three new attacks on the protocol. One requires the group to be of size two, and the other two require a group of size three or more. Coral refutes incorrect inductive conjectures using the 'proof by consistency' technique. Proof by consistency was originally developed as a method for automating inductive proofs in first-order logic, but has the property of being refutation complete, i.e. it is able to refute in finite time conjectures which are inconsistent with the set of hypotheses. Recently, Comon and Nieuwenhuis have drawn together and extended previous research to show how it may be more generally applied, [12]. Coral is the first full implementation of this technique, built on the theorem prover Spass, [37].

In the rest of the paper, we first briefly review previous work in security protocol analysis, refutation of incorrect conjectures and proof by consistency (§2). This explains the motivation for our development of Coral. The Coral system is described in §3, and Coral's protocol model in §4. We give a description of the Asokan–Ginzboorg protocol in §5. In §6, we explain how we modelled the Asokan–Ginzboorg protocol for a group of unbounded size. Then we show how we used Coral to discover three attacks on the protocol in §7. At the end of §7, we propose a new improved version of the protocol. In §8, we compare our results using Coral to other research on protocol analysis, paying particular attention to work on group protocol analysis. We suggest further work in §9, and summarise and conclude in §10.

2 Background

Security protocols were first proposed by Needham and Schroeder, [25]. The authors predicted that their protocols may be 'prone to extremely subtle errors that are unlikely to be detected in normal operation'. They turned out to be correct. Several attacks, i.e. sequences of messages leading to a breach in security, were found in subsequent years. Since then more protocols have been proposed, many of which also turned out to be flawed. In the last five years or so, the interest in the problem from formal methods researchers has greatly increased. The problem of deciding whether a protocol is secure or not is in general undecidable, [14], due to the unbounded number of agents and parallel runs of the protocol that must be considered, and the unbounded number of terms an intruder can generate. However, good results in terms of new attacks and

security guarantees have been achieved by a variety of methods, e.g. [19, 20, 27, 11]. Techniques based on model checking, term rewriting, theorem proving and logic programming each have their advantages and their advocates. For example, model checking approaches can find flaws very quickly, but can only be applied to finite (and typically very small) instances of the protocol, [19]. This means that if no attack is found, there may still be an attack upon a larger instance. Other methods can find guarantees of security quickly, but provide no help in finding attacks on flawed protocols, [11], or require the user to find and prove lemmas in order to reduce the problem to a tractable finite search space, [20]. Recently, dedicated tools for protocol analysis such as Athena have been built, combining techniques from model checking and theorem proving with a special purpose calculus and representation, [31]. Though user interaction is sometimes required to ensure termination, in general Athena's results are impressive. In terms of analysing the standard corpus of two and three party protocols given in [10], the field can now be said to be saturated. Most research attention has now turned to trying to widen the scope of the techniques, e.g. to more precise models of encryption, [23], 'second-level' protocols, [7], and group protocols, [21].

One method for protocol analysis that has proved very flexible is Paulson's inductive method, [27]. Protocols are formalised in typed higher-order logic as the set of all possible traces. Security properties can be proved by induction on traces, using the mechanized theorem prover Isabelle/HOL, [26]. The inductive method deals directly with the infinite state model, and assumes an arbitrary number of protocol participants, allowing properties of group protocols to be proved. However, proofs are tricky and require days or weeks of expert effort. Proof attempts may break down, and as Paulson notes, this can be hard to interpret. Perhaps further lemmas may need to be proved, or a generalisation made, or the conjecture may in fact be incorrect, indicating a flaw in the protocol. Coral was designed to automate the task of refuting incorrect inductive conjectures in such a scenario. Additionally, if a refutation is found, Coral provides a counterexample, giving the user the trace required to exploit the protocol flaw.

The refutation of incorrect inductive conjectures is a problem of general interest in the automated theorem proving community. Tools have been proposed by Protzen, [29], Reif, [30], and Ahrendt, [1]. Ahrendt's method works by constructing a set of clauses to send to a model generation prover, and is restricted to free datatypes. Protzen's technique progressively instantiates terms in the formula to be checked using the recursive definitions of the function symbols involved. Rief's method instantiates the formula with constructor terms, and uses simplifier rules in the theorem prover KIV to evaluate truth or falsehood. These techniques have found many small counterexamples, but are too naïve for a situation like protocol checking. They are designed for datatypes that can be easily enumerated, e.g. types in which any combination of constructors is a valid member of the type. In a protocol analysis scenario, this would tend to generate many non-valid traces, for example, traces in which an honest agent sends message 3 in a protocol without having received message 2. This could be accounted for in the specification by using a predicate to specify valid traces,

but given the complexity of the protocol analysis problem, it would seem too inefficient to keep generating invalid traces only to later reject them. A method more suited to inductive datatypes is required.

Proof by consistency was originally conceived by Musser, [24], as a method for proving inductive theorems by using a modified Knuth-Bendix completion procedure. The idea is to show that a conjecture is a theorem by proving consistency with the axioms in the intended semantics. It was developed by Bachmair, [4], Ganzinger and Stuber, [17], and Bouhoula and Rusinowitch, [9], amongst others. Interest waned as it seemed too hard to scale the technique up to proving larger conjectures. However, later versions of the technique did have the property of being *refutation complete*, that is able to detect false conjectures in finite time. Comon and Nieuwenhuis, [12], have shown that the previous techniques for proof by consistency can be generalised to the production of a first-order axiomatisation $\mathcal{A}$ of the minimal Herbrand model such that $\mathcal{A} \cup E \cup C$ is consistent if and only if conjecture C is an inductive consequence of axioms E. With $\mathcal{A}$ satisfying the properties they define as a *Normal I-Axiomatisation*, inductive proofs can be reduced to first-order consistency problems and so can be solved by any saturation based theorem prover. This allows all the techniques developed for improving the performance of automatic first-order provers, such as reduction rules, redundancy detection, and efficient memory management techniques, to be used to aid the search for a proof or refutation. We describe the method in the next section.

3 The Coral System

Coral is an implementation of the Comon-Nieuwenhuis method for proof by consistency, [12], in the theorem prover Spass, [37]. There is only room for a summary of the technique and how it is implemented here. More details are available in [32].

The Comon–Nieuwenhuis method relies on a number of theoretical results, but informally, a proof attempt involves two parts. In the first part, we pursue a *fair induction derivation*. This is a restricted kind of saturation, [5], where we need only consider overlaps between axioms and conjectures, and produce inferences from an adapted superposition rule. In the second part, every clause in the induction derivation is checked for consistency against an *I-Axiomatisation*. This is typically a set of clauses sufficient for deciding inequality of ground terms. If all the consistency checks succeed, and the induction derivation procedure terminates, the theorem is proved. If any consistency check fails, then the conjecture is incorrect. Comon and Nieuwenhuis have shown refutation completeness for this system, i.e. any incorrect conjecture will be refuted in finite time, [12]. Since the induction derivation procedure may not terminate, we must carry out the consistency checks in parallel to retain refutation completeness. Coral uses a parallel architecture to achieve this, using a socket interface to send clauses for I-Axiomatisation checking to a parallel prover. For problems specified by a *reductive definition*, [12, p. 19], which includes most natural specifications of in-

ductive datatypes, this strategy offers marked performance improvements over a standard superposition strategy without losing refutation completeness. More details in [32].

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, [18]. When CORAL refutes a security conjecture of the form $\forall$trace.P(trace), it has proved in its superposition calculus that $\exists$trace.$\neg P$(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. We show how new attacks are discovered as counterexamples in §7.

4 Coral's Protocol 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 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. Our formalism is typed, though it is also possible to relax the types and search for type attacks. 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. Freshness is modelled by the *parts* operator: a nonce N is fresh with respect to a trace *trace* if $in(N, parts(trace)) = false$. This follows Paulson's model, [27, p. 12].

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. A distinct feature of our formalism is that the entire state of the system is encoded in the trace. Other models with similar semantics often encode information about the knowledge of principals and the intruder in separate predicates. The latter approach has advantages in terms of the time required to find attacks on standard 2 or 3 party protocols, but our approach allows us to add unbounded numbers of messages to the trace under a single first-order rule, which is the key feature required to model the Asokan–Ginzboorg protocol without predetermining group size.

The intruder has the usual capabilities specified by the Dolev-Yao model, [13], i.e. the ability to intercept all messages in the trace, and to break down and reassemble the messages he has seen. He can only open encrypted 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, xt, we define $analz(xt)$ to be the least set including xt closed under projection and decryption by known keys. This is accomplished by using exactly the same rules as the Paulson model, [27, p. 12]. Then we can define the terms the intruder may build, given a trace xt, as being members of the set $synth(analz(xt))$, where $synth(x)$ is the least set containing x, including

agent names and closed under pairing and encryption by known keys. The intruder may send anything in $synth(analz(xt))$ provided it matches the template of one of the messages in the protocol. This last feature is an optimisation to make searching for attacks more efficient, since the spy gains nothing from sending a message that no honest agent will respond to, [33]. We use this feature to define a domain specific redundancy rule for clauses: a clause is considered redundant if it specifies that the intruder must use a subterm which does not occur in any protocol message. By using the term indexing built into SPASS we can make this check extremely efficiently, resulting in a marked performance improvement. CORAL has a further redundancy rule which eliminates clauses with an unsatisfiable *parts* constraint, i.e. a *parts* literal where the variable that is supposed to represent a fresh nonce appears in the trace referred to by the *parts* operator. These clauses would of course be eliminated eventually by the axioms defining *parts*, but by eagerly pruning them, we save time.

For a specific protocol, we generally require one axiom per protocol message, each one having the interpretation, 'if xt is a trace containing message n addressed to agent xa, then xt may be extended by xa responding with message $n + 1$'. The tests that a suitable message n has been sent are known as *control conditions*. After the axioms specifying the protocol have been written down, a script automatically produces the clauses needed to model the intruder sending and receiving these protocol messages. These two sets of rules can be added to a standard set of axioms describing types, the member function, and the intruder to complete the protocol specification. It should be possibly to automatically generate the message axioms from a protocol specification language such as HLSPL, [6].

Using this model, CORAL has rediscovered several known attacks on two and three party security protocols including Needham-Schroeder, Neuman-Stubblebine and BAN Otway-Rees. This latter is significant since it requires an honest agent to generate two fresh nonces and to play the role of both the initiator and the responder, things which previous first-order models have not allowed, [36]. Run times on a Pentium IV Linux box vary from 17 seconds for Needham-Schroeder public key to 15 minutes for Otway-Rees. This is significantly slower than our competitors, e.g. [6, 31]. However, there are two rather more positive aspects to CORAL's performance. The first is that CORAL searches for a counterexample in the general model of a protocol, not just in a model involving particular principals being involved in particular sessions as specified by the user, as for example in [6]. The second is that although CORAL's backwards search proceeds quite slowly, primarily because of time spent doing subsumption checking for each new clause, this checking together with the domain-specific redundancy rules described above means that many states are eliminated as being redundant with respect to the states we have already explored. This leads to quite an efficient search, for example for the Needham-Schroeder public key protocol, CORAL generates 1452 clauses, keeps 610 after redundancy checking, and discovers the attack at the 411th clause it considers. This gave us the confidence

that CORAL would scale up to a group protocol whose model has a much larger branching rate, like the group protocol considered in this paper.

5 The Asokan–Ginzboorg Protocol

Asokan and Ginzboorg of the Nokia Research Centre have proposed an application level protocol for use with Bluetooth devices, [2]. 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. However, 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, [2]. We have analysed the first of these. Completing analysis of the second protocol using CORAL would require some further work (see §9). 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. $M_n \rightarrow \text{ALL}$	$: M_n, \{\!\!\{ E \}\!\!\}_P$	
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, $K = f(S_1, \ldots, S_n)$

What is happening here is:

1. M_n broadcasts a message containing a fresh public key, E, encrypted under the password, P, which she has 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 some member of the group in step 4. Once this final 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* – attacks whereby a spy prevents the honest agents from completing a successful run of the protocol – 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 §7.

6 Modelling the Asokan–Ginzboorg Protocol

In CORAL's model we reason about traces as variables, which may later be instantiated to any number of messages. This allows us to model the protocol generally with respect to the size of the group. The use of a general model means that we do not have to guess how many players are needed to achieve an attack – CORAL will search for attacks involving any number of participants. Of course, CORAL is more likely to find attacks with small number of participants first, since this will involve reasoning with smaller clauses.

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 control conditions. So for message 2, our rule express the fact that an agent can 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, $n-1$ different message 3s will be sent out at once. Moreover, the group leader for the run must check the control condition that she has received $n-1$ message 2s. This could not be modelled by a single first-order clause without predetermining the size of the group. This problem was solved by adding two more clauses to the model. We recursively define a new function 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[1]. 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. This shows an advantage of a theorem-prover based formalism – we have access to full first-order inference for control conditions like these in our protocol model. A full description of this function is available via `http://homepages.inf.ed.ac.uk/s9808756/asokan-ginzboorg-model/`.

[1] 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.

7 Attacking the Asokan–Ginzboorg Protocol

We decided to test the protocol against two attackers: one inside the room, and one outside. The spy outside tries to affect disruption attacks, and the spy inside tries to gain control of communication in the room, e.g. by making all participants agree on different keys that only he knows. As we are in a wireless scenario, both spies capabilities differ from the normal Dolev-Yao model in that they cannot prevent particular messages from being received, they can only insert additional fake messages of their own.

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. The following conjecture was used to check for disruption 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),
$\quad$ *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 $\wedge$
member(sent(MN,all,pair(principal(MN),encr(key(E),key(P))))) $\wedge$
,Trace)=true

%% 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 doesn't have RJ, and so can't
%% get the key from his message 3.
member(nonce(SJ),Package)=true $\wedge$
member(sent(XJ,MN,pair(principal(XJ),encr(pair(nonce(RJ),nonce(SJ)),
key(E)))),Trace)=false
$\rightarrow$

This conjecture expresses 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 (i.e. he cannot read his message 3). Note that conjecture is negative, i.e. it says that for all possible traces, no trace can have the combination of genuine messages 4,3 and 1 without a corresponding message 2. Note also that the conjecture is completely general is terms of the size of the group. When first run with this conjecture, CORAL produces the following counterexample for a group of size 2:

1. $M_2 \rightarrow \text{ALL} : M_2, \{\!|\ E\ |\!\}_P$
1′. $spy_{M_1} \rightarrow \text{ALL} : M_1, \{\!|\ E\ |\!\}_P$
2′. $M_2 \rightarrow M_1 \ : M_2, \{\!|\ R_2, S_2\ |\!\}_E$
2. $spy_{M_1} \rightarrow M_2 \ : M_1, \{\!|\ R_2, S_2\ |\!\}_E$
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. 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 §7.1 below). CORAL took about 2.5 hours to find the first attack, and about 3 hours to find the second. With these two attacks prevented, CORAL finds no further disruption attacks after three days run-time.

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 have in fact agreed on different keys. If the spy knows all these keys, he could filter all the information exchanged, perhaps making changes to documents, such that the other agents in the room are none the wiser. The only change to CORAL's model for this scenario is to allow the spy to read the password on the whiteboard. We then checked for non-matching key 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 $\wedge$

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

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. $spy \rightarrow \text{ALL} : spy, \{\!|\ E\ |\!\}_P$
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 first message 3, the spy switches in a number of his own (S'_{spy} in the place of S_2). This means that M_1 accepts the key as $f(S_{spy}, S_1, S'_{spy})$, whereas M_2 accepts $f(S_{spy}, S_1, S_2)$, and both of these keys are known to the spy.

7.1 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. Here is the revised Asokan–Ginzboorg protocol, with boxes highlighting the changes:

1. $M_n \rightarrow \text{ALL} \ : \{\!|\ \boxed{M_n}\ , E\ |\!\}_P$
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.

8 Related Work

A protocol for group key management has also been modelled by Taghdiri and Jackson, [34]. In this case the protocol under consideration was the Pull-Based Asynchronous Rekeying Framework, [35]. The Taghdiri-Jackson model is general in terms of the number of members of the group, but does not model a malicious intruder trying to break the protocol. Instead, they just investigate correctness properties of the protocol. One correctness property is found not to hold, revealing a flaw whereby a member of the group may accept a message from an ex-member of the group as being current. However, though potentially serious, this flaw is somewhat trivial to spot, and is evident from a description of the rather weak protocol. Taghdiri and Jackson propose a fix for the protocol. However, without the modelling of a wilfully malicious member, who can eavesdrop on protocol traffic and construct fake messages, they failed to produce a secure protocol. We have recently modelled the improved protocol in CORAL and discovered attacks which are just as serious as the two they discovered. Details will appear in a future publication.

The most successful attempt at group protocol analysis in terms of finding new attacks was [28], where the case study was the CLIQUES protocol suite, [3]. Pereira and Quisquater discovered a number of new attacks, using a pen-and-paper approach and borrowing some ideas from the strand space model, [15]. Their attacks were quite subtle, involving properties of the Diffie-Hellman exponentiation operation widely used in the CLIQUES suite. They also involved the spy doing some quite imaginative things, like joining the group, leaving, and then forcing the remaining members to accept a compromised key. Interestingly, some of their attacks required the group to be of size four or more, backing up the case for taking a general approach in terms of group size. Pereira and Quisquater showed the value of by-hand analysis taking algebraic properties of cryptographic functions into account, but only when undertaken by experts.

Meadows made an attempt to extend the NRL protocol analysis tool to make it suitable for analysing group protocols, [21]. Again the CLIQUES protocols were used as an example. However, the NRL tool was not able to rediscover the attacks Pereira and Quisquater had discovered, because of the intricate series of actions the spy has to perform to effect the attack. The NRL tool is tied to quite constrained notions of secrecy and authenticity, which may be where the problem lay. It would be interesting to see whether these kinds of attacks could be found automatically by Coral. We hope that the very flexible inductive model used might mean that these attacks are within Coral's scope. However, some work would be required to model the associative and commutative properties of the exponentiation operation used. Modelling these properties is a topic which has recently started to attract more research interest [23]

Tools like Athena, [31], and the On-The-Fly Model Checker (OFMC), [6], can discover attacks on standard 2 and 3 party protocols much faster than Coral, thanks to their special purpose representations and inference algorithms. However, both would have trouble modelling the Asokan–Ginzboorg protocol and similar group protocols in a general way, without setting the group size. Athena uses an extension of the strand space model, [15]. One of Athena's requirements is that a so-called semi-bundle should be finite. However, this would be impossible for semi-bundles in an Asokan–Ginzboorg protocol strand space, since a strand may contain an unbounded number of message 3s. The OFMC is constrained to a one to one relationship between transitions and messages sent. This would seem to make it impossible to model an arbitrary number of message 3s being added to the trace without some adaptation of the tool.

9 Further Work

Our work with Coral is ongoing. We do not intend to compete for speed of attack finding on a large corpus of standard protocols, since other approaches are better suited to this. Instead we intend to focusing on the kind of flexible or group protocols that are not easy to attack using a model checking approach. One goal is to develop a corpus of group protocols and attacks like the Clark-Jacob corpus for standard protocols, [10]. We also intend to write a converter to allow protocols

to be given to Coral in an existing protocol specification language like HLSPL, [6], or MuCAPSL, [22]. This would ease further comparison between Coral and other tools. It would be very interesting to see if the Asokan–Ginzboorg protocol analysed here could be specified in such a language without choosing a group size, and if not, what features we should have to add to these languages.

As Coral is built on Spass, a theorem prover capable of equational reasoning, we should be able to reason about some simple algebraic properties of the cryptosystems underlying protocols, such as Diffie-Hellman type operations. This would allow us to analyse the second Asokan–Ginzboorg protocol, which is quite different to the first and seems not to be susceptible to the same attacks, and also the CLIQUES protocols mentioned above. The main task here would be to devise a way of modelling the low-level cryptographic operations in such a way that the essential properties are captured, but without going into too fine a degree of detail which would make automated analysis infeasible.

Apart from the small number of domain specific redundancy rules described in §4, Coral is a general counterexample finding tool for inductive theories. We would like to explore other situations where such a tool may be of use, such as the detection of false lemmas and generalisations in an inductive theorem prover.

A longer term aim is to adapt Coral to be able to analyse security APIs of cryptographic hardware modules, such as are used for electronic point-of-sale devices and automated teller machines. Serious flaws have been found in these in recent years, [8]. Some of these attacks involve reducing the complexity of guessing a secret value by brute force search using some information leaked by the API as the result of some unexpected sequence of commands. These kinds of attacks are beyond the scope of current protocol analysis tools, and would require measures of guess complexity to be considered along with the usual protocol traces. We could perhaps accomplish this in Coral by attaching complexity literals to each clause in the same way that we currently attach answer literals.

10 Conclusions

We have presented Coral, our system for refuting incorrect inductive conjectures, and shown three new attacks on the Asokan–Ginzboorg protocol that it found. An advantage of looking for attacks on group protocols with Coral is that we can model the protocol generally and so look for attacks on any size of group. The distinct attacks discovered on groups of size two and three demonstrate the efficacy of this approach. A model checking type approach would have required us to fix the size of the group in advance. Additionally, refuting conjectures about traces in an inductive formalism allows us to look for unconventional attacks, e.g. where two members of the group share different keys with the intruder which they both believe to be the group key.

In future we intend to attack more group protocols, particularly wireless group protocols. We aim eventually to be able to find attacks on security APIs that require brute force guess complexity considerations to be taken into account.

References

1. W. Ahrendt. Deductive search for errors in free data type specifications using model generation. In A. Voronkov, editor, *18th Conference on Automated Deduction*, volume 2392 of *LNCS*, pages 211–225. Springer, 2002.
2. N. Asokan and P. Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.
3. G. Ateniese, M. Steiner, and G. Tsudik. New multiparty authentication services and key agreement protocols. *IEEE Journal on Selected Areas in Communications*, 18(4):628–639, April 2000.
4. L. Bachmair. *Canonical Equational Proofs*. Birkhauser, 1991.
5. L. Bachmair and H. Ganzinger. Completion of First-order clauses with equality by strict superposition (extended abstract). In *Proceedings 2nd International CTRS Workshop*, pages 162–180, Montreal, Canada, 1990.
6. D. Basin, S. Mödersheim, and L. Viganò. An on-the-fly model-checker for security protocol analysis. In *Proceedings of the 2003 European Symposium on Research in Computer Security*, pages 253–270, 2003. Extended version available as Technical Report 404, ETH Zurich.
7. G. Bella. Verifying second-level security protocols. In D. Basin and B. Wolff, editors, *Theorem Proving in Higher Order Logics*, volume 2758 of *LNCS*, pages 352–366. Springer, 2003.
8. M. Bond and R. Anderson. API level attacks on embedded systems. *IEEE Computer Magazine*, pages 67–75, October 2001.
9. A. Bouhoula and M. Rusinowitch. Implicit induction in conditional theories. *Journal of Automated Reasoning*, 14(2):189–235, 1995.
10. J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0. Available via `http://www.cs.york.ac.uk/jac/papers/drareview.ps.gz`, 1997.
11. E. Cohen. TAPS a first-order verifier for cryptographic protocols. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop*, pages 144–158, Cambridge, England, July 2000.
12. H. Comon and R. Nieuwenhuis. Induction = I-Axiomatization + First-Order Consistency. *Information and Computation*, 159(1-2):151–186, May/June 2000.
13. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions in Information Theory*, 2(29):198–208, March 1983.
14. N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In N. Heintze and E. Clarke, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy*, July 1999. Electronic proceedings available at `http://www.cs.bell-labs.com/who/nch/fmsp99/program.html`.
15. F. Fábrega, J. Herzog, and Guttman J. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7:191–230, 1999.
16. H. Ganzinger, editor. *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction*, volume 1632 of *LNAI*, Trento, Italy, July 1999. Springer-Verlag.
17. H. Ganzinger and J. Stuber. *Informatik — Festschrift zum 60. Geburtstag von Günter Hotz*, chapter Inductive theorem proving by consistency for first-order clauses, pages 441–462. Teubner Verlag, 1992.
18. C. Green. Theorem proving by resolution as a basis for question-answering systems. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, pages 183–208. Edinburgh University Press, 1969.

19. G. Lowe. Breaking and fixing the Needham Schroeder public-key protocol using FDR. In *Proceedings of TACAS*, volume 1055, pages 147–166. Springer Verlag, 1996.
20. C. Meadows. The NRL protocol analyzer: An overview. *Journal of Logic Programming*, 26(2):113–131, 1996.
21. C. Meadows. Extending formal cryptographic protocol analysis techniques for group protocols and low-level cryptographic primitives. In P. Degano, editor, *Proceedings of the First Workshop on Issues in the Theory of Security*, pages 87–92, Geneva, Switzerland, July 2000.
22. J. Millen and G. Denker. MuCAPSL. In *DISCEX III, DARPA Information Survivability Conference and Exposition*, pages 238–249. IEEE Computer Society, 2003.
23. J. Millen and V. Shmatikov. Symbolic protocol analysis with products and Diffie-Hellman exponentiation. In *16th IEEE Computer Security Foundations Workshop*, pages 47–61, 2003.
24. D. Musser. On proving inductive properties of abstract data types. In *Proceedings 7th ACM Symp. on Principles of Programming Languages*, pages 154–162. ACM, 1980.
25. R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.
26. L.C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer, 1994.
27. L.C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.
28. O. Pereira and J.-J. Quisquater. Some attacks upon authenticated group key agreement protocols. *Journal of Computer Security*, 11(4):555–580, 2003. Special Issue: 14th Computer Security Foundations Workshop (CSFW14).
29. M. Protzen. Disproving conjectures. In D. Kapur, editor, *11th Conference on Automated Deduction*, pages 340–354, Saratoga Springs, NY, USA, June 1992. Published as Springer Lecture Notes in Artificial Intelligence, No 607.
30. W. Reif, G. Schellhorn, and A. Thums. Flaw detection in formal specifications. In *IJCAR'01*, pages 642–657, 2001.
31. D. Song, S. Berezin, and A. Perrig. Athena: A novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.
32. G. Steel, A. Bundy, and E. Denney. Finding counterexamples to inductive conjectures and discovering security protocol attacks. In *Proceedings of the Foundations of Computer Security Workshop, FLoC'02*, 2002. Also available as Informatics Research Report 141 `http://www.inf.ed.ac.uk/publications/report/0141.html`.
33. P. Syverson, C. Meadows, and I. Cerversato. Dolev-Yao is no better than machiavelli. In P. Degano, editor, *Proceedings of the First Workshop on Issues in the Theory of Security*, pages 87–92, Geneva, Switzerland, 2000.
34. M. Tagdhiri and D. Jackson. A lightweight formal analysis of a multicast key management scheme. In *Proceedings of Formal Techniques of Networked and Distributed Systems - FORTE 2003*, LNCS, pages 240–256, Berlin, 2003. Springer.
35. S. Tanaka and F. Sato. A key distribution and rekeying framework with totally ordered multicast protocols. In *Proceedings of the 15th International Conference on Information Networking*, pages 831–838, 2001.
36. C. Weidenbach. Towards an automatic analysis of security protocols in first-order logic. In Ganzinger [16], pages 314–328.
37. C. Weidenbach et al. System description: SPASS version 1.0.0. In Ganzinger [16], pages 378–382.