Attacking Group Multicast Key Management Protocols Using Coral

Graham Steel[1],[2]   Alan Bundy[3]

School of Informatics,
University of Edinburgh,
Edinburgh, EH8 9LE, Scotland

Abstract

This paper describes the modelling of a two multicast group key management protocols in a first-order inductive model, and the discovery of previously unknown attacks on them by the automated inductive counterexample finder CORAL. These kinds of protocols had not been analysed in a scenario with an active intruder before. CORAL proved to be a suitable tool for a job because, unlike most automated tools for discovering attacks, it deals directly with an open-ended model where the number of agents and the roles they play are unbounded. Additionally, CORAL's model allows us to reason explicitly about lists of terms in a message, which proved to be essential for modelling the second protocol. In the course of the case studies, we also discuss other issues surrounding multicast protocol analysis, including identifying the goals of the protocol with respect to the intended trust model, modelling of the control conditions, which are considerably more complex than for standard two and three party protocols, and effective searching of the state space generated by the model, which has a much larger branching rate than for standard protocols.

Keywords: Security protocol analysis, group multicast key management

1 Introduction

In terms of analysing the standard corpus of two and three party protocols given in [7], the field of cryptographic security protocol analysis can be said to be saturated. Much research attention has now turned to trying to widen

---

[1] The work reported in this paper was carried out while the first author was supported by the European 'Calculemus' scheme as a visiting researcher at the University of Karlsruhe, Germany, and the University of Genova, Italy.

[2] Email: graham.steel@ed.ac.uk

[3] Email: bundy@ed.ac.uk

the scope of the techniques, e.g. to group protocols, [14,27,24]. The term 'group protocol' refers to a protocol (or suite of sub-protocols) for establishing a secure key between an unbounded number of agents. This may take the form of a single key-establishment run between the agents, or a series of requests to join and leave a group with associated key updates. Some protocols involve a trusted key server. Significant attacks on group protocols have appeared in the literature, but these have been discovered by patient pen and paper analysis, [22]. Attempts to extend automatic tools to analyse such protocols have had mixed results (see §2). Taghdiri and Jackson, [27], published an analysis of a protocol for group multicast key management proposed by Tanaka and Sato, [28]. They were able to find major flaws and proposed a new, improved version of the protocol. However, their model was rather weak: in particular, it did not include an active attacker. In this paper, we investigate their improved protocol using CORAL, [24], and show how it discovered two new attacks just as serious as the original ones. Additionally we look at the Iolus key management protocol, [18], which was analysed in Taghdiri's MSc thesis, [26], and thought to be secure. Again CORAL was able to find an attack.

To carry out the analysis, CORAL's model had to be developed to include state information not always included in the trace of messages exchanged, such as the current group key held by the key server, and the current time, expressed as the number of events that have taken place in the trace so far. The reasons for this will be made clear in the description of the case studies below.

In successfully modelling and attacking these protocols, we have given further evidence for the utility of the CORAL approach; though run times are typically slow, we can very quickly adapt the Horn-clause model to new kinds of protocol. Additionally, the attacks discovered are significantly longer than those typically found in the Clark-Jacob corpus, [7], and those found by CORAL before, [24]. To achieve really fast attack discovery times on these protocols, it should be possible to adapt purpose built protocol analysis tools, though significant work may be required. We examine this possibility in §5.

The paper is organised as follows: in §2, we review previous work on group protocol analysis. In §3 we describe the first case study, on Taghdiri and Jackson's improved version of the Tanaka-Sato protocol. The second case study, on the Iolus protocol, follows in §4. Throughout the case studies, we highlight some general issues about the modelling of multicast protocols. These points are summarised in §5, where we discuss the lessons learned and the problems that remain to be solved. §6 contains conclusions and plans for further work.

2   Background

The field of cryptographic security protocol analysis is now far too large to cover here. A recent survey can be found in [15]. 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, [6], as part of his development of the inductive method, [21]. Paulson was able to model it in the general case, i.e. for any number of participants, and prove some security properties for arbitrarily-sized groups. Paulson's method 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.

CORAL, [24], has been used to discover attacks on the Asokan–Ginzboorg protocol for key establishment in an ad-hoc wireless network, [2]. CORAL uses a first-order version of Paulson's protocol model, and searches for counterexamples to the security property under consideration, i.e. it searches for attacks when the protocol is flawed. Inference is carried out by an adapted version of the first-order theorem prover SPASS, [29], using the refutation complete 'proof by consistency' strategy, [8]. CORAL searches the infinite model of the system directly, with no parameterisation with respect to sessions, roles, size of the group etc. In addition to standard subsumption and tautology checking, a small set of domain-specific reduction rules are used to prune the search space. The use of an inductive model allowed the specification of the Asokan–Ginzboorg protocol for a group of unbounded size, which lead to the discovery of distinct attacks on groups of size two and then three while investigating the same security property.

Several significant attacks on the CLIQUES protocol suite, [3], were found by Pereira and Quisquater as a result of a programme of manual analysis, [22]. The CLIQUES protocol suite contains a number of protocols for establishment of groups, and groups within groups, each with their own secure key. Extensive use is made of Diffie-Hellman style exponentiation, [10]. 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 involved 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 use a key that he knows. 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, [16]. 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 the protocol, so a concrete abstraction of the protocol was made, 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, [14]. NPA was extended to handle the Diffie-Hellman exponentiation operation, but could not find the attacks described in [22].

The designers of the protocol specification language CAPSL, [9], have recently turned their attention to group protocols in the development of an extended language, MuCAPSL, [17]. After translating the GDH.2 protocol, [25], 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. The designers of the HLSPL protocol language, [23], have also been working on extending their coverage to group protocols, though no results of this work were available at the time of writing this paper.

Taghdiri and Jackson, [27], reported results from the modelling of a multicast key management scheme proposed by Tanaka and Sato, [28]. 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, [12], 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 the rest of this paper, we explain how we modelled and analysed both Taghdiri and Jackson's improved protocol, and the Iolus key management protocol treated in Taghdiri's MSc thesis (and considered to

be secure), [26], using our inductive counterexample discovery tool, CORAL. 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, [11].

3   The Tanaka–Sato/Taghdiri–Jackson Protocol

The protocol that Tanaka and Sato originally proposed, [28], 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, [27], 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. 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 A 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 A 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, because delays 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, [20]. His behaviour was formalised by Dolev and Yao, [11], and since then it has generally been accepted that the spy should also be able to pose as a legitimate agent, as for example in Lowe's famous attack, [13]. If anything, compared to a unicast protocol, it would seem even more likely that a multicast protocol would be subject to attack by an active intruder, as argued in [18]. 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 a valid long-term key.

Here is a description of the improved version of the protocol as described by Taghdiri and Jackson, and as modelled in this paper:

Joining the Group

$$1. \quad M_i \quad \rightarrow \quad S \quad : \quad \{| \text{ join } \}_{K_{M_i}}$$

$$2. \quad S \quad \rightarrow \quad M_i \quad : \quad \{| Ik_{M_i}, Gk(n)\}_{K_{M_i}}$$

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. $M_i \;\; \rightarrow \;\; S \;\;\; : \;\;\; \{\!|$ leave $\}_{Ik_{M_i}}$

2. $S \;\; \rightarrow \;\; M_i \;\;\; : \;\;\; \{\!|$ 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. The new group key is not distributed yet though. In fact, if another membership change occurs before a request for a key is received, it will never be distributed.

Sending a message

1. $M_i \;\; \rightarrow \;\; S \;\;\;\;\; : \;\;\; \{\!|$ send, $n\}_{Ik_{M_i}}$

2. $S \;\; \rightarrow \;\; M_i \;\;\; : \;\;\; \{\!| \; n', Gk(n')\}_{Ik_{M_i}}$

3. $M_i \;\; \rightarrow \;\; ALL \;\;\; : \;\;\; \{\!|$ 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. $M_j \;\; \rightarrow \;\; S \;\;\; : \;\;\; \{\!|$ 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.

Commentary

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 after presenting the attacks we discovered, in §3.2.

3.1   Modelling in CORAL

CORAL's model is a first-order version of Paulson's inductive model, [21]. We enforce strong typing on message elements by using a sorted signature, with unary functions acting as sort constraints. 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. Horn clauses define how a valid trace may be extended by honest agents as specified by the protocol. Further clauses model the knowledge the intruder can learn from previous messages in the trace, using the same *synth* and *analz* operators that Paulson uses. To search for attacks, we pose conjectures about security properties exactly as Paulson does. CORAL then searches for counterexamples, effectively by a backwards search, i.e. it goes from a state violating the security property backwards to see if there is a valid trace reaching that state. More details of our model and the operation of CORAL are available in [24].

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 group key agreement protocol, as it allowed us to create a general model for any group size, [24]. 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 is does not include a 'gets' event to model message reception, such as Bella later introduced to Paulson's model, [5]. 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. 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. We model multicast messages as $hello(T)$, where T is the counter value when the message was sent, thus ensuring all (honestly sent) messages are unique.

Having chosen to use a counter-based model, we introduced a new pruning rule to CORAL: if the counter variable occurs in term X inside an antecedent 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 eliminates a lot of unreachable states from the search. This rule could be generally applied to backward searching tools using a tick 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 used in [24], 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 the 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. A further point to note is that we still infer state information about principals from the trace, for example to decide if they should be expecting a key update message in the third clause. Our new model is something of a hybrid between a Paulson style trace model and a state-based model like that used, for example, in [4].

%% *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

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

3.2 *Attacking the Protocol*

In [22], Pereira and Quisquater attempt to lay down a list of desirable security properties for 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 $\wedge$
> *% 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 $\wedge$
> *% and the spy is not in the group*
> ingroup(triple(principal(spy),X3,X2),Xgroup,Newgp)=false $\wedge$
> *% but the spy has the key Gk*
> in(Gk,analz(Trace)=true $\rightarrow$

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 ($analz(X)$ 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) finds a rather trivial attack where 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 counterexample in Figure 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 a 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

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

Fig. 2. First attack on the Tanaka-Sato/Taghdiri-Jackson Protocol

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 group authenticity property, 'outsider can't send'. We gave an appropriate conjecture to CORAL for confirmation, and it discovered the counterexample in Figure 3. Like the previous attack, this is also a replay attack, in this case with the spy replaying message 8 in message 13, tricking agent *a* into thinking that message 11 came from a legitimate member of the group. Replay attacks are in general more serious in the context of a group key management protocol. A replay attack on a standard unicast protocol typically 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.

$$1. \qquad a \;\rightarrow\; \text{server} \;:\; \{\!|\, a \,|\!\}_{longtermK(a)}$$

$$2. \quad \text{server} \;\rightarrow\; a \qquad\; :\; \{\!|\, ik(1), Gk(1) \,|\!\}_{longtermK(a)}$$

$$3. \quad\; spy \;\rightarrow\; \text{server} \;:\; \{\!|\, spy \,|\!\}_{longtermK(spy)}$$

$$4. \quad \text{server} \;\rightarrow\; spy \qquad :\; \{\!|\, ik(3), Gk(2) \,|\!\}_{longtermK(spy)}$$

$$5. \quad\; spy \;\rightarrow\; \text{server} \;:\; \{\!|\, read \,|\!\}_{ik(3)}$$

$$6. \quad \text{server} \;\rightarrow\; spy \qquad :\; \{\!|\, Gk(2) \,|\!\}_{ik(3)}$$

$$7. \qquad a \;\rightarrow\; \text{server} \;:\; \{\!|\, read \,|\!\}_{ik(1)}$$

$$8. \quad \text{server} \;\rightarrow\; a \qquad\; :\; \{\!|\, Gk(2) \,|\!\}_{ik(1)}$$

$$9. \quad\; spy \;\rightarrow\; \text{server} \;:\; \{\!|\, leave \,|\!\}_{ik(3)}$$

$$10. \quad \text{server} \;\rightarrow\; spy \qquad :\; \{\!|\, ackleave \,|\!\}_{ik(3)}$$

$$11. \quad\; spy \;\rightarrow\; \text{all} \qquad :\; \{\!|\, hello(12) \,|\!\}_{Gk(2)}$$

$$12. \qquad a \;\rightarrow\; \text{server} \;:\; \{\!|\, read \,|\!\}_{ik(1)}$$

$$13. \quad\; spy \;\rightarrow\; a \qquad\; :\; \{\!|\, Gk(2) \,|\!\}_{ik(1)}$$

Fig. 3. Second attack on the Tanaka-Sato/Taghdiri-Jackson Protocol

The second 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/s9808756/tanaka-sato/.

4 The Iolus Protocol

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, i.e. as and when members of the group want to send or read messages.

Joining the Group

$$1. \quad M_i \;\rightarrow\; S \qquad :\; \{\!|\, \text{join} \,|\!\}_{K_{M_i}}$$

$$2. \quad S \;\rightarrow\; M_i \qquad :\; \{\!|\, Ik_{M_i}, Gk_{n'} \,|\!\}_{K_{M_i}}$$

$$3. \quad S \;\rightarrow\; \text{ALL} \;\;:\; \{\!|\, Gk_{n'} \,|\!\}_{Gk_n}$$

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. $M_i \rightarrow S \quad : \quad \{| \text{ leave } \}_{Ik_{M_i}}$

2. $S \rightarrow ALL \quad : \quad [\ \{ | \ \ Gk_{n'} \} \ |_{Ik_{M_j}} \ \ \ldots] \ \forall j \ \neq i, M_j \in \text{group}$

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

The message is simply broadcast under the current group key.

4.1   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 an appropriate key update 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, [24]. Being able to make these kinds of recursive calculations seems (perhaps unsurprisingly) to be important in modelling group protocols, and our first-order logic model is well suited to them.

4.2   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 $\wedge$
> % *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 $\wedge$
> % *the spy is not in the group*
> ingroup(triple(principal(spy),Y,Z),Restgp,Newgp)=false $\wedge$
> % *but he knows the key*
> in(Gk,analz(Trace))=true $\rightarrow$

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 counterexample in Figure 4 [4]. 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 previous attacks, 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

---

[4] In CORAL's inductive model, we have an arbitrary and unbounded number of agents, so the first agent is called a, the second $s(a)$, and so on.

1.	a	$\rightarrow$	server	$:$
2.	server	$\rightarrow$	all	$:$
3.	server	$\rightarrow$	a	$:$
4.	spy	$\rightarrow$	server	$:$
5.	server	$\rightarrow$	all	$:$
6.	server	$\rightarrow$	spy	$:$
7.	s(a)	$\rightarrow$	server	$:$
8.	server	$\rightarrow$	all	$:$
9.	server	$\rightarrow$	s(a)	$:$
10.	a	$\rightarrow$	server	$:$
11.	server	$\rightarrow$	all	$:$
12.	spy	$\rightarrow$	server	$:$
13.	server	$\rightarrow$	all	$:$
14.	spy	$\rightarrow$	all	$:$

1. $\quad a \rightarrow server \quad : \quad \{| a |\}_{longtermK(a)}$

2. $\quad server \rightarrow all \quad : \quad \{| Gk(1) |\}_{Gk(X8)}$

3. $\quad server \rightarrow a \quad : \quad \{| ik(2), Gk(1) |\}_{longtermK(a)}$

4. $\quad spy \rightarrow server \quad : \quad \{| spy |\}_{longtermK(spy)}$

5. $\quad server \rightarrow all \quad : \quad \{| Gk(2) |\}_{Gk(1)}$

6. $\quad server \rightarrow spy \quad : \quad \{| ik(5), Gk(2) |\}_{longtermK(spy)}$

7. $\quad s(a) \rightarrow server \quad : \quad \{| s(a) |\}_{longtermK(s(a))}$

8. $\quad server \rightarrow all \quad : \quad \{| Gk(3) |\}_{Gk(2)}$

9. $\quad server \rightarrow s(a) \quad : \quad \{| ik(8), Gk(3) |\}_{longtermK(s(a))}$

10. $\quad a \rightarrow server \quad : \quad \{| leave |\}_{ik(2)}$

11. $\quad server \rightarrow all \quad : \quad [\{| Gk(4) |\}_{ik(8)}, \{| Gk(4) |\}_{ik(5)}]$

12. $\quad spy \rightarrow server \quad : \quad \{| leave |\}_{ik(5)}$

13. $\quad server \rightarrow all \quad : \quad [\{| Gk(5) |\}_{ik(8)}]$

14. $\quad spy \rightarrow all \quad : \quad [\{| Gk(4) |\}_{ik(8)}, \{| Gk(4) |\}_{ik(5)}]$

Fig. 4. Attack on the Iolus Protocol

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/s9808756/iolus/.

5   Lessons from the Case Studies

We have seen in these case studies the re-emergence in new protocols of oversights made by the designers of the first security protocols. As we argued in §3.2, replay attacks like the ones we discovered are even more serious in the context of a multicast key management protocol, and do not even require the

spy to have full Dolev-Yao capabilities - he need only be able to replay an old message, he does not need to stop a message from being received or break messages apart. Our fix for the Iolus protocol requires timestamps to be used in the protocol, which is not wholly desirable, but seems the only solution for multicast protocols that eagerly update keys. For on-demand rekeying such as is used in the Tanaka-Sato protocol, a nonce exchange can be used.

In terms of automated group protocol analysis, we have seen that CORAL's inductive model, a first-order version of Paulson's, is well suited to modelling group key management protocols. The protocols involve an unbounded number of agents, each of whom may play a role in different sub-protocols an unbounded number of times. CORAL's inductive model handles this naturally. Again 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. CORAL discovered this for itself.

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 is §3.2, 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. The Iolus protocol quite trivially does not satisfy the property of resistance to known key attacks. New session keys are always distributed under old session keys, so the compromise of one will lead to the loss of all subsequent keys. It is clear the designers of these multicast key management protocols have not aimed to provide these properties. 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. This proves to be quite a tricky property to express formally, and not just in CORAL, as we learnt from our experiments with the AVISPA SATMC tool (see below). 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, [19], and this could perhaps be adapted to formulate properties for group key management protocols.

The second weaker aspect of CORAL's performance was the run times (up to 3.5 hours to find the second attack). This seems to be because the mixed trace based and state based nature of the model led CORAL to explore a lot of unreachable states. This we hope to address with a revised model storing more state information. We are working on an interface to the intermediate format used in the AVISPA project (see for example [4]). Thus we hope to take advantage of the efforts being made to extend HLSPL to group protocols to facilitate further testing of CORAL. With the help of L. Compagna, we have also been experimenting with one of the current AVISPA back-ends, the SATMC tool, [1], to see if these protocols can be modelled and the same attacks found. Some significant work will be needed to model the protocols in a completely general way, without restrictions on the number of agents and sub-protocol roles. Also, it seems that modelling key update in the Iolus protocol will require the ability to reason explicitly about lists. We have found that posing group security conjectures in the AVISPA intermediate format is also difficult. In particular, it is not straightforward to express multicast group authenticity. Not being able to reason about the order in which events took place, and who was in the group at the time, further complicates the issue. However, with appropriate extensions to the intermediate format and some more work, we expect be able to rediscover the attacks in much faster times.

6   Conclusions

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 open-ended protocol with an unbounded number of agents, joins, leaves, messages sent and received etc. Secondly, modelling at the first-order Horn clause level in a theorem prover meant making the adaptations required to store and manipulate a list of current group members was just an evening's work. Thirdly, 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.

There are dozens of protocols for this scenario in the literature, most of which have received little or no formal attention. Our experience so far suggests that they are likely to be vulnerable to attack. Since the field of standard protocol analysis is already mature, this would seem to be a great opportunity

for the automated reasoning/formal methods community to solve an outstanding problem and prove the worth of their methods. Two pre-requisites for this are the extension of protocol specification languages to group protocols, which is already underway, and the establishment of a corpus of group protocols together with attacks found or verification results achieved. This we are beginning to do: see http://homepages.inf.ed.ac.uk/s9808756/group-protocol-corpus/.

References

[1] A. Armando, L. Compagna, and P. Ganty. Sat-based model-checking of security protocols using planning graph analysis. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 875–893. Springer, 2003.

[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] 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.

[5] G. Bella. Message reception in the inductive approach. Technical Report 460, Computer Laboratory, University of Cambridge, 1999.

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

[7] 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.

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

[9] G. Denker and J. Millen. CAPSL integrated protocol environment. In *DARPA Information Survivability Conference and Exposition*, volume 1, pages 207–221, 2000.

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

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

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

[13] 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.

[14] 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.

[15] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *IEEE Journal on Selected Areas in Communication*, 21(1):44–54, January 2003.

[16] C. Meadows and P. Syverson. Formalizing GDOI group key management requirements in NPATRL. In *ACM Conference on Computer and Communications Security 2001*, pages 235–244, 2001.

[17] J. Millen and G. Denker. MuCAPSL. In *DISCEX III, DARPA Information Survivability Conference and Exposition*, pages 238–249. EEE Computer Society, 2003.

[18] S. Mittra. Iolus: A framework for scalable secure multicasting. In *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 277–288, Cannes, France, September 1997.

[19] R. Monroy and M. Carrillo. On automating the formulation of security goals under the inductive approach. In M. H. Hamza, editor, *Applied Informatics*, pages 1020–1025. IASTED/ACTA Press, 2003.

[20] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

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

[22] 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).

[23] AVISPA Project. Deliverable 2.1: The high-level protocol specification language. Available from http://www.avispa-project.org/delivs/2.1.

[24] G. Steel, A. Bundy, and M. Maidl. Attacking a protocol for group key agreement by refuting incorrect inductive conjectures. In D. Basin and M. Rusinowitch, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, Cork, Ireland, July 2004. To appear.

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

[26] M. Taghdiri. Lightweight modelling and automatic analysis of multicast key management schemes. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, December 2002.

[27] M. Taghdiri 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.

[28] 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.

[29] C. Weidenbach et al. System description: SPASS version 1.0.0. In H. Ganzinger, editor, *Automated Deduction – CADE-16, 16th International Conference on Automated Deduction*, LNAI 1632, pages 378–382, Trento, Italy, July 1999. Springer-Verlag.