Verifying Multicast-Based Security Protocols Using the Inductive Method

Jean E. Martina * and Lawrence C. Paulson

Computer Laboratory
University of Cambridge
William Gates Building
15 JJ Thomson Avenue
Cambridge - United Kingdom
CB30FD
Jean.Martina@cl.cam.ac.uk, lp15@cam.ac.uk

Abstract. Multicast, originally designed as an efficient way of broadcasting content, is increasingly used in security protocols. Multicast security protocols are difficult to verify using model checking because they typically involve a large number of participants and because of the exponentially growth of knowledge distribution. Multicast is a general way of representing message casting in protocol verification, with Unicast, Anycast and Broadcast as special cases. Using the inductive method of protocol verification and Isabelle/HOL, we have devised techniques for specifying multicast protocols and proving many of their essential properties. We show secrecy proofs for a mixed environment protocol.

1 Introduction

Multicast was initially advertised as a scheme for better network resources usage [24] and for maximising user experience when receiving content that could be replicated. Although Multicast is a reality today, its application remains limited by these initial assumptions. Multicast has seen increasing interest from the security community, initially with protocols for secure content delivery [10, 23] trying to address specific multicast problems and later in protocols that involve Byzantine Agreement [19] taking advantage of the message casting framework.

Multicast is versatile and can be seen as the basic building block of other common message casting frameworks. Once the complexity of specifying properties such as reliability can be addressed for a multicast framework, its application for other message casting frameworks is generally straightforward.

New security protocols are based on unicast, multicast, broadcast and a mixture of the modes. We can cite examples: protocols to assure secrecy on one-to-many communications [15], to guarantee authenticity in one-to-many communications [13, 26], and for key distribution in one-to-many communications [8, 16]. Some protocols deal with novel security goals such as Byzantine agreement [17, 27], multi-party computation [9] and digital-rights management [23].

* Supported by CAPES Foundation/Brazil on grant #4226-05-4

The verification process for such protocols must match the development done by designers. Some efforts were seen in the literature, but they generally struggle to cope: one-to-many message casting models inherently increase the size and complexity of the knowledge sets of peers as well as the size of the representation of the execution. Efforts are being made using model checking with sequential calculus approaches [1, 14] and theorem proving, in particular using the NPA approach from the Naval Research Laboratory in America [2] and using Graham Steel's CORAL [25].

The verification of security protocols using theorem provers in higher-order logic is due to Paulson [21]. He introduced the Inductive Method, where protocols are formalised in typed higher-order logic as being an inductively defined set of all possible execution traces. An execution trace is a list of all possible events in a given protocol. Events include the sending or receiving of messages. There is a Dolev-Yao intruder, or *Spy*, who processes messages using two operators called *analz* and *synth*. Operator *analz* represents all the terms that the attacker can learn by cumulatively decrypting messages, and *synth* represents all the messages he can compose with the knowledge he possesses.

Protocols are defined as inductive sets constructed over abstract definitions of the network and cryptographic primitives. Theorems about protocol properties are formalised as statements about the set of all possible traces, and are typically proven inductively. The computational model is an operational semantics, giving it great flexibility and admitting unlimited numbers of concurrent executions. This approach has been used to prove a series of classical protocols [20, 21] as well as some well-known industrial-grade ones, such as the SET online payment protocol, Kerberos and SSL/TLS [6, 7, 22].

The implementation aspects of what Steel did with Coral in the verification of multicast based security protocols confirms the great potential of the inductive method. Our aim, then, is to extend the inductive method to enable reasoning about the multicast-based events. Our main goals are to create a versatile event model that can encompass multicast and all the other variations of message casting that multicast can yield.

Our idea in this paper is to briefly review general multicast protocols (§2), and to motivate our augmentation of the inductive method with a Multicast-capable event theory. Then we will see our contributions in the extension of the inductive method towards having a fully capable multicast event theory (§3). We will also show an example of protocol verification in a mixed environment using the proposed message casting framework. We will conclude with an analysis of the new verification capabilities the inductive method has after these extensions.

2 General Multicast Protocols

Multicast aims to use the network layer efficiently by requiring the source to issue the packet only once even if it needs to be delivered to a large agent population. The network is in charge of doing the necessary replication and enabling the delivery of the payload to all the agents within the multicast group.

Normally, due to implementation constraints, multicast is unreliable. The usual application of transmitting data streams for content delivery [18] makes efficiency the sole objective. Other applications require multicast to behave as a network layer where lossless transmission, non-duplication of content and ordering need to be enforced. Security protocols are generally among these applications. To address that, we see the inception of a classification for multicast, dividing them in three categories.

1. *Unreliable multicast*: messages can be lost, permuted and multiply delivered. Some security protocols can cope with these properties.
2. *Reliable Multicast*: all honest members of the multicast group will receive the same message. The loss of messages is prevented, thanks to the usage of novel transport layers. Reliable Multicast schemes are the basis for Byzantine Agreement [19].
3. *Atomic Multicast* schemes furthermore enable honest members to have messages delivered in the order that they were sent, and each message only once. This category of multicast is very difficult to achieve and normally yields a security protocol in itself, since it requires the existence of some basic security properties, such as uniqueness and reliability.

3 Inductive Method extensions for Multicast

The inductive method previously allowed only three formal events, which formalise the act of sending, receiving and noting a message. A *Says* event formalises the act of an agent sending a message to another agent. A *Gets* event, introduced by Bella [5], formalises the act of receiving a message. And finally, a *Notes* event formalises the act of an agent locally storing information for future use.

We explored different choices to the specification of Multicast events. In our first attempt, we tried specifying the Multicast communication as a series of Unicast communications from the sender to all recipients in the multicast group. This idea allowed us to use the existing datatype and associated formal theory, but it proved difficult to use, and it didn't properly distinguish between a multicast event and a series of unicast events. This approach didn't appear to be able to model the different types of Multicasts mentioned in Section 2.

Therefore we decided to extend datatype *event*, creating a primitive to represent a Multicast communication. This created the possibility of expressing Unicast, Broadcast and other casting frameworks using the Multicast primitive.

Definition 1 *Multicast event datatype definition*

```
datatype
  event = Says agent agens msg
        | Multicast agent "(agent list)" "( agent => msg)"
        | Gets agent msg
        | Notes agent msg
```

The new datatype of events retains the primitives of the original *event* datatype for the sake of compatibility. Our theory can replace the standard Events theory distributed with Isabelle/HOL. If Multicast is not used, the verification remains unchanged for the protocols in Isabelle/HOL's library.

We add to the datatype event a new primitive *Multicast*, where an agent sends a message to a multicast group. We represent the group by a list, which implies an ordering of the group members. The idea of representing the message as a function over agents comes from direct inspiration from the real Multicast communication and the necessity of implementing Anycast, as most security protocols require that. In a real scenario, a peer sends the same message to a group of receivers, and each receiver is capable of interpreting the message in different ways, sometimes depending on the knowledge he/she already has.

Another reason for letting each peer apply a function with his own parameters is that this creates their own view of the message. This is necessary for protocols where the message delivered to an agent is encrypted using that agent's key: the messages received by the agents in the multicast group are not identical. This will become clear below (3.1 and 3.2), where we extend the knowledge of peers based on their point of view inside or outside of the multicast group.

Although it seems contradictory to reject a formalisation based on Unicast to represent Multicast and implement a Multicast datatype capable of representing Unicast and Broadcast, this formalisation does not suffer the setbacks of the previous ideas. We also wanted to create a generic formalisation to corroborate our motivational idea that Unicast and Broadcast are extremes for Multicast, setting Multicast as the base primitive for verifying Security Protocols in the future.

In the next sections we will show the extensions made in how peers acquire knowledge (§3.1) and how we can have access to all derivations coming from the application of the function to each peer's point of view through the function *used* (§3.2).

3.1 Extending Peers' Knowledge set for Multicast Communications

Peer knowledge, formalised by the set *knows*, enables us to reason about key distribution as well as confidentiality. The original work by Paulson [21] did not take into account the knowledge each peer acquired. Its main concern was what the *Spy* was able to learn. The *knows* function represents how the knowledge of each peer — the Spy included — is expanded during the execution of the protocol, in accordance with Bella's goal availability principle [5]. Definition 2 shows our new specification for the *knows* function.

Definition 2 *Extended function representing Peer's knowledge under Multicast*

```
consts
  knows  :: "agent => event list => msg set"
primrec
  knows_Nil:   "knows A [] = initState A"
  knows_Cons:   "knows A (ev # evs) =
```

```
(if A = Spy then
 (case ev of
    Says A' B X => insert X (knows Spy evs)
 |  Multicast A' B XF =>  (XF ‘ set B) ∪  (knows Spy evs)
 |  Gets A' X => knows Spy evs
 |  Notes A' X  => if A' ∈ bad then insert X (knows Spy evs)
      else knows Spy evs)
 else (case ev of
    Says A' B X => if A'=A then insert X (knows A evs)
      else knows A evs
 |  Multicast A' B XF=> if A'=A then (XF ‘ set B) ∪ (knows A evs)
      else knows A evs
 |  Gets A' X    => if A'=A then insert X (knows A evs)
      else knows A evs
 |  Notes A' X    => if A'=A then insert X (knows A evs)
      else knows A evs))"
```

The function *knows* is now specified in four cases in two steps. In the base case, *knows_Nil*, the trace of events is empty and the knowledge of a peer is equal to its initial knowledge prior to the execution of the protocol. When we move to the case of a non-empty trace, we have two classes of peers (compromised and non-compromised) and four cases each.

The *Spy* is able to learn differently than other peers. When he sees an event *Says* sending X as a message, we extend his knowledge by inserting X to his knowledge set. When he sees an event *Multicast* from a peer to a multicast group casting XF we add to the knowledge of the *Spy* the image of the function XF over the set of peers in the multicast group. When he sees a *Gets* event he learns nothing, because he already learnt it at the sending event. When a peer learns a message X through the predicate *Notes*, the *Spy* will learn the message X if the peer is compromised.

When a peer sends a message X via a *Says* event, we extend his knowledge by inserting X. When the peer issues event *Multicast* to a multicast group casting XF, we add to his knowledge the image of the function XF over the set of peers in the multicast group. When he *Gets* a message X through the predicate *Gets* we insert X to his knows set. When he learns a message X through the predicate *Notes*, we insert X to his knows set.

During the specification of the *event* datatype and the extension of the function *knows*, we noticed some subtleties regarding the Multicast constructor and its reception, especially regarding knowledge gathering. Our initial design included an event *GetsMC* to deal with the reception of multicast messages. This seemed attractive since a Multicast message conveys information regarding the knowledge other peers in the multicast group may have acquired. Side-channel information leaking is inherent to Multicast. When we receive a Multicast message addressed to us and two others, we don't just learn the contents of the message, but also that the other two users may also have received their view of the message contents. This idea can be extended even further, since the other two peers know that we may know the information conveyed by the message.

This refinement of the function *knows* would add new possibilities of inferring other peers' knowledge. The extension of the function *knows* to deal with other peer's knowledge is very attractive to use with novel threat models. Examples of such new threat models are the Rational Attacker [3] where different attackers collude on the basis of cost/benefit decisions whether to follow or not to follow the protocol. The General Attacker [3], which drops the cost/benefit decision from the Rational Attacker. The General Attacker's differentiation from a Dolev-Yao is that each peer acts for his own sake. And the Multi-Attacker [4], where each principal behaves as a *Dolev-Yao* attacker, but they will never reveal their long-term secrets to other peers.

We dropped the idea: gathering this information is difficult in practice because multicast is supposed to hide group composition. This is even more problematic in an environment where message reception can not be guaranteed. Another issue was that we did not want to break backward compatibility by changing the shape of the function *knows* and re-implement all the other affected definitions to accommodate this change. Ultimately, making changes to accommodate these properties from Multicast communication would require a complete rethinking of the Dolev-Yao threat model. We formalise receiving a Multicast message via the usual *Gets* primitive.

3.2 Extending the Used Set for Multicast

The function *used* enables us to reason about freshness. Freshness is essential for reasoning about the unicity of certain messages. It is also a key compositional property for reasoning about key distribution. The *used* function forms the set of all message components that have already appeared in the event trace plus all the information all peers initiated the protocol run with. Definition 3 show the extended version of the function *used*, now encompassing the Multicast primitive.

Definition 3 *The used parts of messages, with Multicast*

```
consts
  used :: "event list => msg set"
primrec
 used_Nil:   "used []         = (UN B. parts (initState B))"
 used_Cons:  "used (ev # evs) =
              (case ev of
               Says A B X => parts {X} ∪ used evs
              |  Multicast A B XF => parts (XF ‘ set B) ∪ used evs
              |  Gets A X   => used evs
              |  Notes A X  => parts {X} ∪ used evs)"
```

The *used* function is specified recursively. The base case, *used_Nil*, is when our event trace is empty. It is defined by the union of the application of the function parts to the initial state of all peers.

The recursive case concerns the components used during trace construction. There are four cases, one for each event constructor. The first case concerns the

primitive *Says*, where the application of the function parts on the message X is joined with the used set of the remaining events. The second case concerns *Multicast*: we apply the function *parts* to the image of function XF over the set of peer in the multicast group B and join the result with the set of remaining events. The third case concerns the *Gets* primitive, where no action is taken since the parts are already considered used when sent. Finally the fourth case concerns the primitive *Notes*, which is similar to *Says*.

This extension of the *used* function is key for adding Multicast support to the inductive method. It has been kept stable since the first versions of the inductive method [21], since the additions made for encompassing message reception were only technical and did not change the set construction. Adding the Multicast primitive makes the set potentially bigger and more complex to reason about.

4 The Franklin-Reiter Sealed-Bid Auction Protocol

Franklin and Reiter [12] proposed in 1996 a protocol to enable the construction of a distributed trusted service capable of executing sealed-bid auctions by using threshold cryptography primitives and extended multicast properties. Their objective was to provide a sealed-bid auction service that is guaranteed to declare a winner, and also to collect payment from only that bidder, while guaranteeing that no bid was revealed before the agreed bid opening time. Moreover, that the system should be resilient to malfeasance of any auction house insider.

This protocol relies on *secret sharing*. A secret is divided into *shares*, which are distributed to a set of *trustees*. The secret is not intended to be known by any of the trustees. Let n be the number of trustees and let t be the *threshold*, the minimum number of shares required for the secret to be recovered. We call this n, t-sharing of the secret. Given at least t shares, the secret can be reconstructed. Collusion by fewer than t trustees does not yield any information about the secret.

The protocol also requires *verifiable signature sharing* [11], or simply $V\Sigma S$. This enables the holder of a digitally signed message to share the signature among a group of peers so that they can reconstruct the signature later, as with verifiable secret sharing. At the end of the sharing phase, the members can verify that they possess a valid share and that the signature can be reconstructed even if the original signer or some trustees are faulty. Faulty trustees gain no information regarding the original signature.

4.1 Protocol Description

The protocol proposed by Franklin and Reiter is constructed using n auction servers, of which t are assumed to operate faithfully. In its conception, the protocol is claimed to be Byzantine-failure secure.

A bidder submits a bid with the amount he wants to pay for the item by sharing a digital coin $(v_\$, \{|v_\$|\}_{Kr_{Bank}}, w_\$)$ with this value among all servers hosting the auction. To prevent the auction servers from cheating, the coin values

are split in different ways. The values of $v_\$$ and $w_\$$ are split using a standard secret sharing mechanism using our fault tolerance value of t as threshold. The signature of the face value for the coin $\{|v_\$|\}_{Kr_{Bank}}$ is shared using a $V\Sigma S$ signature sharing algorithm also using t as threshold.

Once the bidding phase finishes, the servers in agreement will reconstruct the values for $v_\$$ and $w_\$$ for all bids cast during the bidding phase and will independently determine the winner. For the winning bid, the auction servers will perform a $V\Sigma S$ verification to see if the bid is valid and the money can be collected by reconstructing $\{|v_\$|\}_{Kr_{Bank}}$. After this verification, the auction servers can award a token to the winning bidder to collect the item.

We found Franklin and Reiter's protocol description [11] unclear. The specification seemed to assume a lot of implicit calculations, in violation of protocol design principles, which could introduce some vulnerabilities. We summarise the Franklin-Reiter sealed-bid auction protocol in Figure 1.

1. $B \overset{A}{\rightsquigarrow} SG$:	$\lambda X.\{\|aid, \{\|S(B, v\$, w\$)_X, aid\|\}_{Kr_X},$ $V\Sigma S\_pub(\{\|v\$\|\}_{Kr_{Bnk}}),$ $\{\|V\Sigma S\_priv(\{\|v\$\|\}_{Kr_{Bnk}})_X\|\}_{Kr_X}\|\}$
2. $S_i \overset{A}{\rightsquigarrow} SG$:	$aid, close$
3. $S_i \overset{U}{\rightsquigarrow} SG$:	$\lambda XY.\{\|aid, S(Y, v\$, w\$)_X\|\}$
4. $S_i \overset{R}{\rightsquigarrow} SG$:	$\lambda X.aid, V\Sigma S\_stat(\{\|v\$\|\}_{Kr_{Bnk}})_X$
5. $S_i \rightarrow B$:	$aid, B, \{\|aid, B\|\}_{Kr_{S_i}}$

Fig. 1. Franklin-Reiter Sealed-Bid Auction Protocol

The bidder B, holding a digital coin $(v_\$, \{|v_\$|\}_{Kr_{Bank}}, w_\$)$, will issue an atomic multicast to the multicast group SG comprised of all participating auction servers. This multicast message starts with the auction identification token aid, followed by the n, t-sharing of the concatenation of his identity, the face value of the coin, and the freshness value for the coin. Each share is encrypted with the public key of each corresponding server in the multicast group SG. It is followed by the public $V\Sigma S$ of the Bank's signature to the coin's face $V\Sigma S\_pub(\{|v\$|\}_{Kr_{Bnk}})$, and the private $V\Sigma S$ n, t-shared to all members of SG. The $V\Sigma S$-private shares will be encrypted with the public key of each corresponding server in the multicast group SG.

Each auction server S_i in the multicast group SG will multicast the second message to the group. This message simply states the auction identification and the closing statement. After each auction server S_i has received at least t atomic multicasts stating the bidding phase is closed, no more bids are accepted. The inclusion of this message in the protocol is controversial for having no security intent, but the authors argue that due the implementation characteristics of atomic multicast, communication is authenticated within the multicast group.

Then, each auction server S_i will multicast to SG the auction identification *aid* and his shares $S(Y, v\$, w\$)_X$ composed of the concatenation of the bidder's identity, the face value of the coin and the freshness function of the coin. After the reception of t multicasts, a server can locally reconstruct each bid $Y, v\$, w\$$ and deterministically compute the winner.

With the winner locally determined, each auction server S_i will reliably multicast (again to SG) message four, which is composed of *aid* and the result of the $V\Sigma S$ verification of his share for the winning bid for the bank's signature to the coin's face value $v\$$. After the reception of t multicast messages, an auction server can locally decide whether the winning bid is valid.

Finally, each auction server S_i sends a unicast message to B, the winning bidder. The message is composed of *aid*, the bidder's identity and the signature of the concatenation of these values by the auction server S_i. A bidder can collect the won item if he possesses t tokens signed by different auction servers.

Although the protocol takes care of checking the bank's signature on the coin, coin reconstruction and deposit is out of scope for the protocol. By using an off-line digital cash scheme, the protocol provides a degree of anonymity against the detection of the spending by the Bank. The authors propose an anonymity scheme to protect the bidder's anonymity against the auction house.

4.2 Protocol Specification

The specification of the Franklin-Reiter Sealed-bid auction protocol starts, as usual, with the inductive definition of a constant naming the specification called *fr*. It is a set of possible traces (lists of events), representing the formal protocol model. The Isabelle/HOL specification is shown in Definition 4. We define the empty trace by the rule *Nil*, which sets the base of the induction. The next two rules represent the possibility of fake messages being sent by the Spy.

Definition 4 *Inductive definition of Franklin-Reiter Protocol - Basic Steps*

```
inductive_set fr :: "event list set"
where
  Nil: "[] ∈ fr"
| Fake: "[| evsf ∈ fr;
           X ∈ synth (analz (knows Spy evsf)) |]
           ==> Says Spy B X # evsf ∈ fr"
| Fakemc: "[| evsfmc ∈ fr;
              XF ∈ synth (analz (knows Spy evsfmc)) |]
    ==> Multicast Spy multicast_group (λC. XF) # evsfmc ∈ fr"
| Reception: "[| evsr ∈ fr; Says A B X ∈ set evsr |]
               ==> Gets B X # evsr ∈ fr"
| ReceptionMC: "[| evsrmc ∈ fr;
           Multicast A multicast_group (λC. XF ) ∈ set evsrmc;
           B ∈ set multicast_group |]
           ==> Gets B XF # evsrmc ∈ fr"
```

Although our initial specification took into account the reception of messages, it turned out to be inappropriate for the protocol's communication structure and objectives. The protocol provides little information to the bidder. Although our proofs do not use the message reception framework, we model reception events above for the sake of completeness.

The first message of the bid casting phase is specified below. Some of its preconditions are merely technical.

Definition 5 *Inductive definition of Franklin-Reiter Protocol : Bid Casting*

```
| FR1: "[| evs1 ∈ fr; Nonce w ∉ used evs1; w ≠ close;
           w ∉ sessionIDs; w ∉ shares;
           aid ∈ sessionIDs; v ≠ close; v ∉ sessionIDs;
           v ∉ shares; Nonce v ∉ used evs1;
           Multicast S multicast_group (λC. {|
                     Nonce aid, Number close|}) ∉ set evs1 |]
==> Multicast B multicast_group (λC. {|Nonce aid,
     Crypt (pubK C) ( {|Nonce (share (nat t, multicast_group, C) {|
           Agent B, Nonce v, Nonce w|}), Nonce aid|}),
     Nonce (pub_share (nat t, multicast_group, C)
           (signOnly (priSK Bank) (Nonce v))),
     Crypt (pubK C)(Nonce (priv_share (nat t, multicast_group, C)
          (signOnly (priSK Bank) (Nonce v))) )|})
# Notes B {|Nonce aid, Nonce w, Nonce v|} # evs1 ∈ fr"
```

Definition 5 starts with inductive rule *FR1*, stating that the trace *evs1* belongs to the inductive set *fr*, the nonce w and the nonce v were not used in this trace before, are not equal to the constant *close*, does not belong to *sessionIDs* and does not belong to *shares*. We also require that the auction identifier *aid* belongs to *sessionIDs* and that message two (closing the auction) has not appeared in the trace. If these preconditions are met, we extend the trace of events *evs1* belonging to *fr* with message one.

Message one is a *Multicast* event from the bidder B to the multicast group of auction servers. Its payload has the nonce *aid* as the session identifier, followed by the bidder B's identity concatenated with the digital coin v and nonce w. The bidder's identity, coin description and coin function are shared to the multicast's destinations with a threshold t. Each share is encrypted with the intended destination's key creating the sharing token

share (nat t, multicast_group, C) {|*Agent B, Nonce v, Nonce w*|},

which is encrypted. Then the coin's digital signature's public share is included using a verifiable secret sharing scheme to the same group and threshold parameters as the initial sharing scheme. The public part

pub_share (nat t, multicast_group, C)(signOnly (priSK Bank)(Nonce v))

is directly included in the message. The private share

priv_share (nat t, multicast_group, C)(signOnly (priSK Bank) (Nonce v))

is encrypted with the intended destination's key. Formally, a share is modelled as a nonce, because it is a large integer that is infeasible to guess.

To finish the *FR1* specification, we formalise B's knowledge of his bid through a *Notes* event. This is needed because *parts* and *used* do not know about *share*. B's knowledge of the unique identifiers *aid*, w and v is necessary for some proofs.

Message two is specified as rule *FR2* in our specification of the Franklin-Reiter protocol. The preconditions are that *evs2* is a valid trace of events for the protocol, that *aid* belongs to the set of *sessionIDs*, the auction server S belongs to the multicast group and that we have a bid (an instance of message one in the trace). If these preconditions are met, we extend the trace *evs2* with a multicast from the S to the multicast group containing *aid* and the command *close* as shown in Definition 6.

Definition 6 *Inductive definition of Franklin-Reiter Protocol : Bid Closure*

```
| FR2: "[|evs2 ∈ fr; S ∈ set multicast_group; aid ∈ sessionIDs;
           Multicast B multicast_group (λC. {|Nonce aid,
           Crypt (pubK C) (
                {|Nonce (share (nat t, multicast_group, C) {|
                Agent B, Nonce v, Nonce w|}), Nonce aid|}),
           Nonce (pub_share (nat t, multicast_group, C)
                (signOnly (priSK Bank) (Nonce v))),
           Crypt (pubK C )(
                Nonce ( priv_share (nat t, multicast_group, C)
                (signOnly (priSK Bank) (Nonce v))))|})
                ∈ set evs2 |]
==> Multicast S multicast_group (λC.{|Nonce aid, Number close|})
    # evs2 ∈ fr"
```

Again in message two we had to take some decisions regarding our specification. Similarly to message one, we did not implement any trigger and assumed the multicast as being atomic. But here the authors impose a very strong requirement, atomic multicast where multicasts from within the group are authenticated. Our concept of atomic multicast is weaker. We took this specification decision although it could allow attacks that the authors would reject on the basis of their assumptions.

Once no more bids can be cast in the auction and the servers agree that the auction is closed, we proceed to a phase where we verify which bidder cast the best bid. In the Franklin-Reiter protocol this is done by message three, which is specified by our inductive rule *FR3* as shown in Definition 7.

Definition 7 *Inductive definition of Franklin-Reiter Protocol : Bid Opening*

```
| FR3: "[| evs3 ∈ fr ; S ∈ set multicast_group ; S ∉ bad;
           aid ∈ sessionIDs; (B, v, w): bids aid evs3;
           Multicast S multicast_group (λC.
                {| Nonce aid, Number close|}) ∈ set evs3 |]
==> Notes S {|Nonce aid, Nonce w, Nonce v|} #
    Multicast S multicast_group (λC. {| Nonce aid,
```

```
    Nonce (share (nat t, multicast_group, S)
          {|Agent B, Nonce v, Nonce w|})|}) # evs3 ∈ fr"
```

The preconditions for rule *FR3* are that the trace of events *evs3* is part of our inductive specification for the protocol, that the auction server S is not compromised, that the auction identifier *aid* is a session identification, that the triple *(B, v, w)* represents a valid bid for the auction *aid* in the events trace *evs3* and that the auction was closed by the existence of message two in the trace. If these preconditions are met we extend the trace *evs3* with two events. A first event *Notes* to the auction server S the values for *aid*, v and w followed by a *Multicast* event from the auction server S to the multicast group running the auction service. The payload of this message is composed of *aid*, plus the share the server holds for the bid being broadcasted.

Once again, we have made some specification decisions. In rule *FR3* we extend the trace by a *Multicast* event and then we extend it again with a *Notes* events. The latter is done for the sake of giving the servers following the protocol the knowledge regarding the values of the bids they opened.

Message four is represented in our specification by rule *FR4*, as shown on Definition 8. Its preconditions are that the trace of events *evs4* is valid, that *aid* is in the set of *sessionIDs*, that the triple *(B, v, w)* is a valid bid within the auction *aid* in the trace *evs4* and that the nonces w and v are in the knowledge of the auction server S. If these preconditions are met, we extend the trace of events *evs4* by adding an event *Notes* to the auction server S for the bank's digital signature for the coin's face value. Then we add a *Multicast* event from server S to the multicast group of servers running the auction containing *aid* and his share of the digital signature for the coin of the winning bid.

Definition 8 *Inductive definition of Franklin-Reiter Protocol : Bid Collection*

```
| FR4: "[| evs4 ∈ fr; S ∈ set multicast_group ;
          aid ∈ sessionIDs; (B, v, w): bids aid evs4; S ∉ bad;
          Nonce w ∈ knows S evs4; Nonce v ∈ knows S evs4; |]
==> Notes S (signOnly (priSK Bank) (Nonce v)) #
    Multicast S multicast_group (λC. {| Nonce aid,
     Nonce ( priv_share (nat t, multicast_group, S)
             (signOnly (priSK Bank) (Nonce v)))|}) # evs4 ∈ fr"
```

As with *FR3*, we have a double extension of the trace: a multicast and a note that they were able to validate the coin. Here we again simplified the original protocol. Instead of implementing the verifiable signature sharing proposed by the authors, we decided to implement a reconstruction using the casting of the private shares to the multicast group members.

This specification choice clearly weakened the protocol, since at this point one server can collude with the attacker and deposit the coin for himself. But our choice is justified beyond the plain simplification of the verification process, since the guarantees the original protocol yields are just that the coin may be reconstructable at a later time, and not that it is indeed reconstructable, since more than $n - t$ server can be corrupted after the protocol run.

Our focus here is the demonstration of the suitability of the multicast theory, not of the full verification of the Franklin-Reiter protocol. We sought only to verify the secrecy of the bids before closure time. This was enough to validate the multicast events theory. For this modest objective, we can make do with a simple version of rule *FR4*.

With the winning bid known and with its digital coin payable, we can now deliver to the winner the tokens he needs to collect the item. Message five is a unicast from each one of the servers that concluded the protocol execution to the winning bidder with the winner declaration token. We specify message five from the Franklin-Reiter protocol as our inductive rule *FR5*. It starts with the precondition that *evs5* is a valid trace, that *aid* is in the set of *sessionIDs*, that the triple *(B, v, w)* is a valid bid within the auction *aid* and that the nonces *w*, *v* and the digital signature for the coin's face value *(signOnly (priSK Bank) (Nonce v))* are known by the auction server, *S*. If these preconditions are met, the auction server sends the winning bidder a message consisting of *aid*, the winner's identifier *B*, and the same items signed using the server's private key.

Definition 9 *Inductive definition of Franklin-Reiter Protocol : Winner Declaration*

```
| FR5: "[|evs5 ∈ fr; S ∈ set multicast_group; aid ∈ sessionIDs;
          S ∉ bad; w ∉ sessionIDs;
          w ∉ shares; (B, v, w): bids aid evs5;
          Nonce w ∈ knows S evs5; Nonce v ∈ knows S evs5;
          (signOnly (priSK Bank) (Nonce v)) ∈ knows S evs5|]
==> Says S B {| Nonce aid, Agent B,
          sign (priSK S) {| Nonce aid, Agent B |}|} # evs5 ∈ fr"
```

With the specification of message five we complete the protocol description. Note that the specification of rule *FR5* deliberately takes some steps to test our multicast specification and the distribution of knowledge within the protocol. First, the pre-conditions to the firing of *FR5* are based on the knowledge of the auction server *S* acquired during the previous phases and the contents of the bid set. Second, we deliberately did not represent the unicast method using the *Multicast* event to be able to test the integration of our specification with the original one in the inductive method.

Here we took similar specification decisions as we did in the previous messages. But note that the trigger required for collecting the item in this message is left off-protocol by the authors, which makes it difficult to verify the ability of collecting the item by the winner.

4.3 General Validity Proofs

We stress that the proofs we show here are not the complete set we verified. They are shown to exemplify the suitability of the multicast event theory in dealing with knowledge distribution and with mixed environments. We focused on verifying the secrecy of the bids as a way of showing that our multicast specification is capable of representing real problems.

We will start by looking to Lemma 1 (bid_secrecy), which concerns the secrecy of v, the coin's face value. Concomitantly we have two similar lemmas for w and $\{|v|\}_{Kr_{Bank}}$. This lemma states that if an event with syntax of the one yielded by rule *FR1* is in the trace of events, and the bidder B is not colluding with the *Spy* and the *Spy* is not in the multicast group of auction servers, then coin's face value v is not in the knowledge of the *Spy*.

Lemma 1. *bid_secrecy*

```
[| Multicast B multicast_group (λC. {| Nonce aid,
      Crypt (pubK C)({|Nonce (share (nat t, multicast_group, C)
                 {|Agent B, Nonce v, Nonce w|}), Nonce aid|}),
      Nonce ( pub_share (nat t, multicast_group, C)
                 (signOnly (priSK Bank) (Nonce v))),
      Crypt (pubK C)(Nonce(priv_share (nat t, multicast_group, C)
                 (signOnly (priSK Bank)(Nonce v))))|}))∈ set evs)
B ∉ bad; Spy ∉ set multicast_group; evs ∈ fr |]
==> Nonce v ∉ analz (knows Spy evs)
```

Proving Lemma 1 (bid_secrecy) is difficult, as usual for secrecy properties. We start by using the usual proof method for secrecy lemmas. We are left with seven subgoals, representing the two *Fake* rules and the five protocol steps. The Fake rule can be proven appealing to fact Fake_analz_eq, which states that $X \in$ *synth(analz H)* $\Rightarrow$ *synth (analz (insert X H))* = *synth (analz H)*. The subgoals regarding messages two to five are proven appealing to the fact that $v \notin$ *sessionIDs* and to the function congruence rule to eliminate the λ expression in the multicast events. Proving the subgoal for message one involves applying the tactic *auto* augmented with the destruction rule for function congruence as usual for multicast messages. This yields seven new subgoals. The first one is proven resorting to the fact that $v \notin$ *sessionIDs*. Another two of these subgoals are proven appealing to *analz_into_parts* and *shares_shares*, which are simple facts. The next two subgoals are proven by appealing to *Multicast_imp_in_parts_spies* and the fact that if v is inside a share it belongs to the set *used*. The final two subgoals are proven appealing to *analz_keyfree_into_Un* which isolate key material in the *analz* set and the fact that v is not in *sessionIDs*.

4.4 General Re-Interpretation of Security Goals under Multicast

Extending the Inductive method to accommodate a Multicast communication primitive requires a re-interpretation of how the method works and how some security goals should be understood. The first change is the modification of the idea of trace of events. Prior to the introduction of the *Multicast* primitive, we had the idea of a linear trace and a linear expansion of the *knows* and *used* set.

Interpreting now the relation between trace expansion and the *knows* and *used* set expansions, we see the linearity in both. The *knows* set for peer A is initialised with his *a priori* knowledge: his own shared key and private keys as well as all the public keys. After the sending of the first message onto the trace, by the definition of *knows*, the knowledge of peer A will be extended by the

message's payload. And the same will happen for any *Notes* event. After the extensions proposed by Bella [5], the knowledge set of a peer is also extended by a message reception event *Gets*. The recipient of a message from the network will insert its payload onto his knowledge set. A similar procedure happens to the extension of the *used* set. To conclude, for every message event in the trace we expect a linear expansion of the sets *knows* and *used* by the size of the payload of the *Says* event.

If we look closer to the relation happening in a one-to-many communication style we see that this linearity is lost. When we cast a Multicast event in the trace we are not extending the *knows* and *used* sets by exactly the payload of the message, but by the application of the payload function over the list of agents in the multicast group. This inherently changes the shape of the trace construction since the we also do not have this conveying of knowledge between two peers only. The inductive method is capable of coping with this exponential growth in the size of the trace information, which clearly corroborates our choice in using it as the testbed for our verification efforts.

Another important re-interpretation concerns the goal of Secrecy, due to the leakage of information to peers by the usage of side-channels inherent to multicast. As mentioned earlier, with the exception of Multicast being used in an Anycast mode of operation, we can argue that there is a side channel leaking information that could be formalised within the method. Although the method is not prepared today to make use of this information, with the expansion of it to encompass newer threat models, this can be important to represent knowledge that could help us to protect or prepare better retaliation attacks.

5 Final Remarks

We have extended the inductive method to enable it to reason about non-Unicast message casting frameworks. Based on the assumption that other message casting methods are special cases for multicast, we built a theory for representing multicast communication. The point of building such a theory was to create a more flexible infrastructure for the inductive method to represent new classes of protocols. We also experiments with our proposed design to show its backwards compatibility and its novel verification capabilities, which are not shown here.

We expect to see more demand for protocol verification methods capable of being extended, capable of representing the interesting subtleties of ever new designs. Extensibility becomes a key issue to this field because of the stability it has already assumed. No major theoretical breakthrough happened in the last decade. The community must work to broaden the scope of verification methods to cover an ever growing set of security protocol types.

On the aspects of our direct contributions to protocol verification, we envisage the verification of election protocols as being the next big step. With the setup for supporting Multicast, Anycast and Broadcast it is possible for a whole new family of such protocols to be verified by the Inductive Method. This will include

the investigation of new security goals such as Anonymity, which such voting protocols require.

References

1. Anastasi, G., Bartoli, A., Francesco, N.D., Santone, A.: Efficient verification of a multicast protocol for mobile computing. Comput. J 44(1), 21–30 (2001)
2. Archer, M.: Proving correctness of the basic TESLA multicast stream authentication protocol with TAME*. In: Workshop on Issues in the Theory of Security. Portland, OR (2002)
3. Arsac, W., Bella, G., Chantry, X., Compagna, L.: Validating security protocols under the general attacker. In: Degano, P., Viganò, L. (eds.) ARSPA-WITS. Lecture Notes in Computer Science, vol. 5511, pp. 34–51. Springer (2009), http://dx.doi.org/10.1007/978-3-642-03459-6
4. Arsac, W., Bella, G., Chantry, X., Compagna, L.: Multi-attacker protocol validation. Journal of Automated Reasoning 45, 1–36 (2010)
5. Bella, G.: Formal Correctness of Security Protocols. Information Security and Cryptography, Springer (2007)
6. Bella, G., Paulson, L.C.: Kerberos version IV: Inductive analysis of the secrecy goals. Lecture Notes in Computer Science 1485 (1998)
7. Bella, G., Paulson, L.C., Massacci, F.: The verification of an industrial payment protocol: the set purchase phase. In: 9th ACM conference on Computer and communications security. pp. 12–20. ACM Press, New York, NY, USA (2002)
8. Blundo, C., De Santis, A., Vaccaro, U., Herzberg, A., Kutten, S., Yong, M.: Perfectly secure key distribution for dynamic conferences. Inf. Comput. 146(1), 1–23 (1998)
9. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T.P., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) Financial Cryptography. Lecture Notes in Computer Science, vol. 5628, pp. 325–343. Springer (2009), http://dx.doi.org/10.1007/978-3-642-03549-4
10. Brown, I., Perkins, C., Crowcroft, J.: Watercasting: Distributed watermarking of multicast media. In: Rizzo, L., Fdida, S. (eds.) Networked Group Communication. Lecture Notes in Computer Science, vol. 1736, pp. 286–300. Springer (1999)
11. Franklin, M.K., Reiter, M.K.: Verifiable signature sharing. In: Advances in Cryptology (1995)
12. Franklin, M.K., Reiter, M.K.: The design and implementation of a secure auction service. IEEE Transactions on Software Engineering 22(5), 302–312 (1996)
13. Gennaro, R., Rohatgi, P.: How to sign digital streams. Inf. Comput. 165(1), 100–116 (2001)
14. Gorrieri, R., Martinelli, F., Petrocchi, M.: Formal models and analysis of secure multicast in wired and wireless networks. J. Autom. Reasoning 41(3-4), 325–364 (2008), http://dx.doi.org/10.1007/s10817-008-9112-7
15. Hardjono, T., Weis, B.: The Multicast Group Security Architecture. RFC 3740 (Informational) (March 2004), http://www.ietf.org/rfc/rfc3740.txt
16. Harney, H., Muckenhirn, C.: RFC 2094: Group key management protocol (GKMP) architecture (Jul 1997), ftp://ftp.internic.net/rfc/rfc2094.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc2094.txt, status: EXPERIMENTAL.

17. Huang, D., Medhi, D.: A byzantine resilient multi-path key establishment scheme and its robustness analysis for sensor networks. In: 19th International Parallel and Distributed Processing Symposium. Denver, CO, USA (Apr 2005)
18. Kreibich, J.A.: The mbone: the internet's other backbone. Crossroads 2(1), 5–7 (1995)
19. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems 4 (1982)
20. Paulson, L.C.: Mechanized proofs for a recursive authentication protocol. In: Proceedings of The 10th Computer Security Foundations Workshop. IEEE Computer Society Press (1997)
21. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security 6, 85–128 (1998)
22. Paulson, L.C.: Inductive analysis of the internet protocol tls. ACM Trans. Inf. Syst. Secur. 2(3), 332–351 (1999)
23. Pinto, A., Ricardo, M.: Smiz - secure multicast iptv with efficient support for video channel zapping. In: Proceedings of the NAEC 2008, Networking and Electronic Commerce Research Conference 2008. Lake Garda, Italy (September 2008)
24. Quinn, B., Almeroth, K.: IP Multicast Applications: Challenges and Solutions. RFC 3170 (Informational) (Sep 2001), http://www.ietf.org/rfc/rfc3170.txt
25. Steel, G., Bundy, A.: Attacking group multicast key management protocols using coral. Electr. Notes Theor. Comput. Sci 125(1), 125–144 (2005), http://dx.doi.org/10.1016/j.entcs.2004.05.023
26. Wong, C.K., Lam, S.S.: Digital signatures for flows and multicasts. In: IEEE/ACM Transactions on Networking. pp. 502–513 (1998)
27. Zhou, L., Schneider, F.B., Van Renesse, R.: Coca: A secure distributed online certification authority. ACM Trans. Comput. Syst. 20(4), 329–368 (2002)