Towards Verifying Voter Privacy Through Unlinkability

Denis Butin, David T. Gray, Giampaolo Bella

Towards Verifying Voter Privacy Through Unlinkability

Denis Butin[1], David Gray[2], and Giampaolo Bella[3]

[1] Inria, Université de Lyon
INSA-Lyon, CITI-Inria, F-69621, Villeurbanne, France
denis.butin@inria.fr
[2] School of Computing, Dublin City University
Dublin, Ireland
david.gray@computing.dcu.ie
[3] Dipartimento di Matematica e Informatica, Università di Catania, Italy
Software Technology Research Laboratory, De Montfort University, UK
giamp@dmi.unict.it

Abstract. The increasing official use of security protocols for electronic voting deepens the need for their trustworthiness, hence for their formal verification. The impossibility of linking a voter to her vote, often called voter privacy or ballot secrecy, is the core property of many such protocols. Most existing work relies on equivalence statements in cryptographic extensions of process calculi. This paper provides the first theorem-proving based verification of voter privacy and overcomes some of the limitations inherent to process calculi-based analysis. Unlinkability between two pieces of information is specified as an extension to the Inductive Method for security protocol verification in Isabelle/HOL. New message operators for association extraction and synthesis are defined. Proving voter privacy demanded substantial effort and provided novel insights into both electronic voting protocols themselves and the analysed security goals. The central proof elements are described and shown to be reusable for different protocols with minimal interaction.

Keywords: E-voting, Trustworthy Voting System, Privacy, Security Protocols, Formal Methods

1 Introduction

The use of electronic voting (e-voting) for official elections is on the rise across the world. Security protocols claiming properties that protect voters and guarantee regular elections require formal scrutiny because of their sensitive nature. Voters are asked to trust, in particular, election officials regarding the handling of their votes. With e-voting, they are asked to trust a security protocol with special goals. One key goal of e-voting protocols is to hide the way a particular voter votes. Most recent efforts [17] to advance formal verification are based on process equivalence. Despite substantial progress, issues remain regarding simplification of protocol models or termination of supporting tools.

The benefits of specifying privacy in an interactive theorem prover have never been explored until now. Isabelle [15], a generic theorem prover, is flexible enough when used with higher-order logic to allow new classes of security properties to be analysed in the framework provided by the Inductive Method [3]. Its extensions for dealing with voter privacy are described and demonstrated on a classical protocol in the sequel of this manuscript. They required new proof techniques and lines of reasoning, whose development in turn demanded substantial effort. Nevertheless, their application to other protocols is expected to be straightforward, as has been the case for the confidentiality argument [14] for example, with most of the proof scripts adapted for new protocols without significant effort. Automated tools are ideal for checking conjectures about protocols quickly. However, the interactive nature of the Inductive Method pays back, also with e-voting, with a greater support to the analyst's understanding of the protocol entanglements than what automated tools offer today.

The most notable findings in this area stem from formalising the protocols with a process algebra and encoding the privacy properties by process equivalence [10]. As detailed below, process equivalence supports a notion of *indistinguishability* between two situations where a voter voted, respectively, for two different candidates. This implies that an observer cannot discern the two situations being formalised. In line with the operational semantics of the protocols specified by the Inductive Method, we develop an operational encoding of privacy based on *unlinkability* of voter with vote, focusing on the associations that an active attacker can derive from intercepting the protocol traffic. For example, if Alice sent her vote for Bob to the election administrator as a clear-text, then the attacker would build the association Alice-Bob.

However, actual protocol messages are complicated nestings of advanced cryptographic operations (which are still assumed to be reliable), so that the attacker's inspection is far from straightforward. This inspection is formalised by the innovative *association analyser* operator *aanalz* — naming is coherent with the existing lingo. Also, the attacker can intelligently merge associations when they have at least an element in common, similarly to an investigator relating Alice to a crime scene because she wears the same shoe size as that of a shoeprint in the scene. This merge is formalised by the innovative *association synthesiser* operator *asynth*. When it is impossible to build, by means of analysis and then synthesis, an association that features both voter and vote, then there is unlinkability of voter with vote, hence the protocol enforces voter privacy (about their vote). Conversely, the protocol violates voter privacy, irrespectively of how many other voters cast that vote.

An outline of the indistinguishability and unlinkability approaches to modelling privacy (§2) leads to our extensions to the Inductive Method to account for privacy specification and analysis (§3). These extensions are then demonstrated on a classical e-voting protocol known as FOO [12] through its inductive specification (§4) and verification of voter privacy (§5). Conclusions and future work end the manuscript (§6).

2 Modelling privacy

Voter privacy (also known as ballot secrecy) is generally [9,10] defined as follows: how a particular voter voted is not revealed to anyone. Votes may or may not be published at the end of an election, so it is not the confidentiality of the vote in itself that matters but its association with the voter who cast it. In other words, the way a voter votes should not be discoverable by anyone, even after vote count. A caveat on this definition is the exclusion of the corner case where all voters vote identically.

2.1 Indistinguishability

A common way of modelling privacy involves showing the indistinguishability between two situations:

1. Voter V_a votes x and Voter V_b votes y
2. Voter V_a votes y and Voter V_b votes x

Indistinguishability here means that when V_a and V_b swap their votes, no party (including trusted parties running the election) can tell situations 1 and 2 apart.

Formal analysis is often performed in cryptographic extensions of process calculi, with the applied pi calculus [2] being most typical. Automated tools such as ProVerif [7] or, more recently, AKiSs [9] can be used to assist with such analysis.

ProVerif is used to check protocols represented by processes modelled in the applied pi calculus. It does not restrict the number of protocol sessions. A stronger condition than observational equivalence between processes is checked. Since the validity criterion is an under-approximation, spurious attacks may be found in some cases. There is no risk for flawed protocols to be deemed correct, but correct protocols may be invalidated by the tool because of the approximation. Various approaches to checking voter privacy have been presented. Notably, Kremer and Ryan [13] presented an analysis with some manual parts. In 2008 [11], a fully automatic verification was done. However, a translation algorithm was used without formal proof of correctness. The next year, Delaune, Kremer and Ryan published a detailed analysis in which the number of voters is fixed, with a partially automated privacy proof [10]. New cryptographic primitives can be added easily to the tool via equational theories, but the resulting processes may not terminate in some cases.

AKiSs, the most recent automated tool able to check privacy automatically, is also based on equivalence properties. However, a new kind of cryptographic process calculus is used and a different type of process equivalence is checked, called trace equivalence. Under- and over-approximations of trace equivalence are used to detect flawed protocols and validate correct ones, respectively. The set of supported cryptographic primitives is broader than in ProVerif. For a specific class of processes, called *determinate*, a precise verification can be done. However, not all e-voting protocols fall in this class, in which case one of the approximations must be used. The number of sessions must be bounded as it has critical impact on the computational cost.

2.2 Unlinkability

In contrast to the indistinguishability modelling of privacy, an operational view reflects the natural threat model of an attacker monitoring all network traffic and using the data she can extract to associate a voter with a vote. An outline of this approach and comparison with the one based on indistinguishability first appeared in our recent position paper [8]. Initially, the attacker decomposes each individual message, and records all plaintexts and ciphertexts for which keys are available. She can also associate these with the intended recipient agent of the message. For each protocol event whereby an agent sends a message to another agent, this analysis gives the attacker a set of (components of) messages, namely an association. Moreover, if the communication channel is not anonymous, then the attacker can also extend the association just gathered by storing the identity of the sender.

However, it is not sufficient to inspect in isolation each of the messages sent in the traffic. A voter's identity V may appear near an element m that is later to be extracted again, this time in conjunction with the vote N_v. In this case, such a common element m provides the link between voter V and vote N_v. An attacker monitoring the network sees messages as discrete entities and can exploit the shared context of elements extracted from one given message. This process of combining sets of associations builds up an association synthesis. When all possible protocol scenarios are taken into account, establishing voter privacy boils down to inspecting the synthesised set for the presence of a voter's vote.

The only pieces of information that should not be treated as a possible link to synthesise new associations are those that can be linked to *all* voters, such as the name of the precise election officials that a protocol prescribes. Because their identities appear in each and every protocol session, using one of them as a link would lead to the synthesis of insignificant, that is, privacy-irrelevant, associations. For example, an investigator will not call up every human being as suspect of murder simply upon the basis that everyone could pull a trigger. We shall see that with the FOO protocol, the administrator and the collector are omnipresent, hence must be ruled out to synthesise significant associations.

Without setting bounds on the number of agents, sessions, or message nesting depth, the number of different associations that the attacker can synthesise is very large. Precisely, an unbounded number of associations can be derived by observing a full trace, due to the fact that its length is unbounded. This size limits the tool support that traditional finite-state search can offer. As experienced before with other goals [3], inductive reasoning bypasses the size constraints also with the analysis of associations.

3 Specifying unlinkability in Isabelle/HOL

3.1 Isabelle/HOL and the Inductive Method

Isabelle is a generic interactive theorem prover supporting many logics. The most commonly used one, HOL, allows formalisation and proof of predicates in

higher-order logic. Automated reasoning tools are available, but the user must still define the line of reasoning and guide the proving process. A file-based hierarchy of theories is available. All theorems from parent theories are available when those theories are imported in the current one.

The Inductive Method for security protocol verification was introduced in Paulson's paper [16] and later applied widely, notably to electronic payment [5], non-repudiation [6], certified e-mail [3] and multicast protocols [14]. Its central idea is the use of mathematical induction to model security protocols and their properties. The proofs are also done by induction. A specification of the standard Dolev-Yao threat model, common cryptographic primitives and their properties are provided. All seminal elements of the Inductive Method reside in three theory files. *Message* describes messages and agents, *Event* specifies network event datatypes and *Public* contains the lemmas relevant to cryptographic keys and initial states.

Because of the nature of induction, both the number of protocol sessions and participating agents are unbounded. This allows detection of interleaving or replay attacks. The threat model is incarnated by a special agent called *Spy*, who sees all protocol messages, decrypts whatever she can, and participates actively by sending anything she can build from parts previously obtained. The Spy's capabilities are subsumed by an inductive rule called *Fake*:

$$\begin{aligned} &\mid \mathit{Fake}\colon [\![\mathit{evsf} \in \mathit{ns\_public};\ \ X \in \mathit{synth}\ (\mathit{analz}\ (\mathit{knows}\ \mathit{Spy}\ \mathit{evsf}))]\!] \\ &\qquad \Longrightarrow \mathit{Says}\ \mathit{Spy}\ B\ X\ \ \#\ \mathit{evsf} \in \mathit{ns\_public} \end{aligned}$$

Protocol steps are also modelled as inductive rules with pre- and postconditions. Security properties are proven by checking that inductive theorem statements hold over all possible network histories (*traces*).

Available network events are *Says*, *Gets* and *Notes*. The latter represents internal storage of a message by an agent. *Says A B X* represents the sending of message X by agent A to agent B. Delivery does not have to happen, but when it does, this is denoted using *Gets*.

The message operators are as follows:

- *analz* formalises the breaking-up of messages without cryptanalysis. Plaintext is only extracted when the relevant decryption key is part of the knowledge of the agent applying the operator.
- *parts* returns all message building blocks ; it can be seen as *analz* expanded with cryptanalysis.
- *synth* applied to a set of message returns the set of compound messages.

Asymmetric cryptography is available through functions *priEK* and *pubEK* for private and public encryption keys, and then *priSK* and *pubSK* for private and public signing keys. Each of them takes the proprietor agent as a parameter. A private key of a given operation mode is required to decrypt a message encrypted with the corresponding public key, and conversely.

Agent knowledge, formalised by the function *knows*, maps an agent and a list of network events to a set of messages: the knowledge that the agent can extract

from this trace. Agents already know some elements (those in *initState*) such as keys before a protocol even begins.

A hands-on, step by step guide to using the Inductive Method can be found in a recent paper [4].

3.2 Extensions for unlinkability

The analysis of associations requires a new message operator, *analzplus*. It is built on the traditional *analz* message operator, endowed with an external message set providing extra decryption keys:

```
inductive_set
  analzplus :: msg set ⇒ msg set ⇒ msg set
  for H :: msg set and ks :: msg set
  where
    Inj [intro,simp]: X ∈ H ⟹ X ∈ analzplus H ks
  | Fst:              {|X,Y|} ∈ analzplus H ks ⟹ X ∈ analzplus H ks
  | Snd:              {|X,Y|} ∈ analzplus H ks ⟹ Y ∈ analzplus H ks
  | Decrypt [dest]:   ⟦Crypt K X ∈ analzplus H ks; Key (invKey K) ∈ analzplus H ks⟧
                        ⟹ X ∈ analzplus H ks
  | Decrypt2 [dest]:  ⟦Crypt K X ∈ analzplus H ks; Key (invKey K) ∈ ks⟧
                        ⟹ X ∈ analzplus H ks
```

In particular, the new operator is useful to formalise everything, namely the set of all message components, that the attacker can extract from a single message sent in the traffic by hammering it with the entire knowledge she has acquired on an entire trace. For a message X and a trace *evs*, this set can be defined as *analzplus* $\{X\}$ (*analz*(*knows Spy evs*)).

Using *analzplus*, the message association analyser *aanalz* can be defined inductively. Only *Says* events influence it. Indeed, each *Gets* message reception event follow a message sending event *Says*, and *Notes* events correspond to private recording of data by agents:

```
primrec aanalz :: agent => event list => msg set set
where
  aanalz_Nil:   aanalz A [] = {}
| aanalz_Cons:
  aanalz A (ev # evs) =
   (if A = Spy then
    (case ev of
      Says A' B X ⇒
       (if A' ∈ bad  then aanalz Spy evs
        else if isAnms X
             then insert  ({Agent B} ∪ (analzplus {X} (analz(knows Spy evs))))
                          (aanalz Spy evs)
             else insert ({Agent B} ∪ {Agent A'} ∪
                          (analzplus {X} (analz(knows Spy evs)))) (aanalz Spy evs))
    | Gets A' X ⇒ aanalz Spy evs
    | Notes A' X ⇒ aanalz Spy evs)
    else aanalz A evs)
```

The definition indicates, among other aspects, that only the attacker can analyse associations. Also, she will neglect the associations created by compromised agents, thus including those that she may have created, by sending out specific messages. It can also be seen that the sender identity is extracted only for messages that are not sent anonymously. The *isAnms* predicate holds of messages with a specific form that we conventionally interpret to signify anonymity.

The association synthesiser *asynth* can be introduced now. Its definition is not tied to *aanalz*, but it will always be used in conjunction with it for our purposes. Specifically, we will examine the contents of the set *asynth (aanalz Spy evs)*, where *evs* is a generic protocol history. The *asynth* operator introduces a new association as the union of association sets that share a common element:

inductive_set
asynth :: *msg set set* ⇒ *msg set set*
for *as* :: *msg set set*
where
asynth_Build [*intro*]: ⟦*a1* ∈ *as*; *a2* ∈ *as*; *m* ∈ *a1*; *m* ∈ *a2*;
m ≠ *Agent Adm*; *m* ≠ *Agent Col*⟧
⟹ *a1* ∪ *a2* ∈ *asynth as*

As noted above, the definition insists that the common element is not a piece of information that can be linked to *all* voters — for instance, the name of election officials since they appear in every step. The version below can be used for protocols that define two election officials, here called *Adm* and *Col*, in line with the subsequent case study.

4 Modelling the FOO protocol in the Inductive Method

The well-known Fujioka, Okamoto and Ohta (FOO) [12] protocol features two election officials called administrator and collector and involves bit commitments as well as blind signatures. The specification of its protocol steps follows after a description of some extensions.

Blind signatures are a cryptographic primitive often found in e-voting protocols, and in particular in the FOO protocol. We specify them for the first time in the Inductive Method, as an inductive rule in the protocol model. The Spy gains knowledge of a plain signature if she knows the corresponding blinded signature and blinding factor, modelled as a symmetric key:

| *Unblinding*:
⟦*evsb* ∈ *foo*; *Crypt* (*priSK V*) *BSBody* ∈ *analz* (*spies evsb*);
BSBody = *Crypt b* (*Crypt c* (*Nonce N*)); *b* ∈ *symKeys*; *Key b* ∈ *analz* (*spies evsb*)⟧
⟹ *Notes Spy* (*Crypt* (*priSK V*) (*Crypt c* (*Nonce N*))) # *evsb* ∈ *foo*

Anonymous channels are specified by defining a function to replace *Says* when needed. We are conventionally defining an anonymous message by means of a precise message format — the actual message is prepended with a constant number:

consts *anms* :: *nat*
definition *Anms* :: [*agent*, *agent*, *msg*] ⇒ *event* **where**
Anms A B X ≡ *Says A B* {|*Number anms*, *X*|}

Administrator and collector are introduced as translations *Adm* and *Col* of specific agents. We now turn to the actual protocol steps and their model.

4.1 FOO protocol steps and inductive protocol model

The FOO protocol features six phases that give rise to as many protocol steps and corresponding inductive rules.

1. *Preparation*: The voter V picks a vote N_v, builds N_{vc} using the commitment key c, and blinds this vote commitment using the blinding factor b. V then signs the blinded commitment and sends it to the administrator along with V's identity.

   | *EV1*:
   ⟦*evs1* ∈ *foo*; *V* ≠ *Adm*; *V* ≠ *Col*; *c* ∈ *symKeys*; *Key c* ∉ *used evs1*;
   b ∈ *symKeys*; *Key b* ∉ *used evs1*; *b* ≠*c*; *Nonce Nv* ∉ *used evs1*⟧
   ⟹ *Says V Adm* {|*Agent V*, *Crypt* (*priSK V*) (*Crypt b* (*Crypt c* (*Nonce Nv*)))|}
   # *Notes V* (*Key c*) # *Notes V* (*Key b*) # *evs1* ∈ *foo*

2. *Administration*: Upon reception of a signed, blinded commitment, the administrator opens it and checks that the quoted agent name is equal to the signer of the blind signature. If such is the case and the agent has not voted before, the administrator returns the message to V, now signed by the former. The administrator also records V's name.

   | *EV2*:
   ⟦*evs2* ∈ *foo*; *V* ≠ *Adm*; *V* ≠ *Col*; *Notes Adm* (*Agent V*) ∉ *set evs2*;
   Gets Adm {|*Agent V*, *Crypt* (*priSK V*) *BSBody*|} ∈ *set evs2*;
   BSBody = *Crypt P R*; ∀ *X Y*. *MPair X Y* ∉ *parts*{*BSBody*}⟧
   ⟹ *Says Adm V* (*Crypt* (*priSK Adm*) *BSBody*)
   # *Notes Adm* (*Agent V*) # *evs2* ∈ *foo*

3. *Voting*: If V obtained the administrator's reply, V unblinds it and sends the resulting plain signature to the collector over an anonymous channel.

   | *EV3*:
   ⟦*evs3* ∈ *foo*; *Says V Adm* {|*Agent V*,
   Crypt (*priSK V*) (*Crypt b* (*Crypt c* (*Nonce Nv*)))|} ∈ *set evs3*;
   Gets V (*Crypt* (*priSK Adm*) (*Crypt b* (*Crypt c* (*Nonce Nv*)))) ∈ *set evs3*⟧
   ⟹ *Anms V Col* (*Crypt* (*priSK Adm*) (*Crypt c* (*Nonce Nv*))) # *evs3* ∈ *foo*

4. *Collecting*: The collector checks the signature and publishes the enclosed vote commitment N_{vc} on a bulletin board, provided that it was not published before and that all votes have been received.

| *EV4*:
⟦*evs4* ∈ *foo*; *V* ≠ *Adm*; *V* ≠ *Col*; *Says Col Col CX* ∉ *set evs4*;
Gets Col {|*Number anms*, *Crypt* (*priSK Adm*) *CX*|} ∈ *set evs4*;
CX = *Crypt P R*; ∀ *X Y*. *MPair X Y* ∉ *parts*{*CX*}⟧
⟹ *Says Col Col CX* # *evs4* ∈ *foo*

5. *Opening*: Once N_{vc} has appeared on the bulletin board, V sends c over an anonymous channel so that N_v can be revealed.

| *EV5*:
⟦*evs5* ∈ *foo*; *Says V Adm* {|*Agent V*,
Crypt (*priSK V*) (*Crypt b* (*Crypt c* (*Nonce Nv*)))|} ∈ *set evs5*;
Gets Col (*Crypt c* (*Nonce Nv*)) ∈ *set evs5*; *Key c* ∈ *analz* (*knows V evs5*);
c ∉ *range shrK*; *c* ∈ *symKeys*⟧
⟹ *Anms V Col* (*Key c*) # *evs5* ∈ *foo*

6. *Counting*: Upon reception of V's key, the collector publishes N_v on the condition that the key be identical to c.

| *EV6*:
⟦*evs6* ∈ *foo*; *Gets Col* {|*Number anms*, *Key c*|} ∈ *set evs6*;
Gets Col (*Crypt c* (*Nonce Nv*)) ∈ *set evs6*;
Says Col Col (*Nonce Nv*) ∉ *set evs6*⟧
⟹ *Says Col Col* (*Nonce Nv*) # *evs6* ∈ *foo*

5 Proving voter privacy for FOO

5.1 Main results

The following theorem, *foo_V_privacy_asynth*, is the culmination of the entire proof process and states that the FOO protocol guarantees voter privacy to all honest voters that started the protocol. More precisely, assume that the regular, honest voter V sent the administrator a message in line with the first step of the protocol, containing a blinded commitment on the vote *Nv*. Also assume that this very vote is in the message set of association syntheses. Then the name of V is not in that set:

theorem *foo_V_privacy_asynth*:
⟦*Says V Adm* {|*Agent V*, *Crypt* (*priSK V*) (*Crypt b* (*Crypt c* (*Nonce Nv*)))|} ∈ *set evs;*
a ∈ (*asynth* (*aanalz Spy evs*));
Nonce Nv ∈ *a*; *V* ∉ *bad*; *V* ≠ *Adm*; *V* ≠ *Col*; *evs* ∈ *foo*⟧
⟹ *Agent V* ∉ *a*

Before turning to the proof itself, we focus on the most important proof elements, which are mainly results about associations.

A fundamental result is *foo_V_privacy_aanalz*, which looks similar to the *foo_V_privacy_asynth* theorem. However, whereas the latter is a statement about *asynth*, hence about association synthesis, the former only considers *aanalz*, that is associations arising from individual messages. Whenever an honest voter performed the first step of the protocol, the voter's identity and vote cannot be found in the same association:

theorem *foo_V_privacy_aanalz*:
$[\![Says\ V\ Adm\ \{\!|Agent\ V, Crypt\ (priSK\ V)\ (Crypt\ b\ (Crypt\ c\ (Nonce\ Nv)))|\!\}\in set\ evs;$
$a \in (aanalz\ Spy\ evs);\ Nonce\ Nv \in a;\ V \notin bad;\ evs \in foo]\!]$
$\Longrightarrow Agent\ V \notin a$

The lemma called *asynth_insert* is a direct consequence of the definition of *asynth* quoted in 3.2. By introducing the various cases that an application of *asynth* may imply, it provides a useful rewrite rule for expressions involving the operator name:

lemma *asynth_insert*:
$a \in asynth(insert\ a1\ as) \Longrightarrow$
$(a{=}a1 \vee$
$a \in asynth\ as \vee$
$(\exists\, a2\ m.\ a2 \in as \wedge a = a1 \cup a2 \wedge m \in a1 \wedge m \in a2 \wedge$
$m \neq Agent\ Adm \wedge m \neq Agent\ Col))$

The next three theorems allow more precise reasoning about messages that contain encryption. They are all concerned with the situation where a message yields an association set containing at least one ciphertext. They are necessary for dealing with situations where protocol messages are not completely specified. For instance, an agent may have to transmit an encrypted commitment without even being able to check that the commitment is actually about a vote. In those situations, protocol step specification must model agents' limited knowledge when dealing with sealed messages. However, even when the complete contents of a ciphertext is not known, a number of scenarios can be distinguished. Various encryption key values and partial knowledge of the ciphertext contents lead to contradictions. Possible configurations are therefore made explicit in the following results.

Lemma *aanalz_PR* states constraints about the possible forms of a generic ciphertext appearing in any association. Its conclusion is expressed as a conjunction between two predicates that are themselves disjunctions. The first conjunct relates to the presence of an agent name in the association. If the name of the collector appears in the association and any nonce (a vote) is an atomic component of R, then no agents that are both honest and different from the collector can also be in a. The second conjunct states that if any nonce is part of the association, then the Spy must be able to decrypt the ciphertext and no agent name can be an atomic component of R:

lemma *aanalz_PR*:
$[\![a \in aanalz\ Spy\ evs;\ Crypt\ P\ R \in a;\ evs \in foo]\!] \Longrightarrow$
$(Agent\ Col \notin a \vee$
$(Agent\ V \in a \longrightarrow V \in bad \vee V = Col) \vee$
$(Nonce\ Nv \notin parts\ \{R\})) \wedge$
$((Nonce\ Nv \notin a) \vee$
$(Key\ (invKey\ P) \in analz\ (spies\ evs) \wedge Agent\ V \notin parts\ \{R\}))$

Then, lemma *aanalz_AdmPR_V_Nparts* relates to the specific case when a ciphertext signed by the administrator is in an association. It establishes a disjunction: either no nonce is an atomic component of the ciphertext's body, or

the Spy cannot open the ciphertext inside the signature, or there is no regular, honest agent name in the association:

lemma *aanalz_AdmPR_V_Nparts*:
$[\![a \in aanalz\ Spy\ evs;\ Crypt\ (priSK\ Adm)\ (Crypt\ P\ R) \in a;\ evs \in foo]\!]$
$\Longrightarrow Nonce\ Nv \notin parts\ \{R\} \vee$
$Key\ (invKey\ P) \notin analz\ (knows\ Spy\ evs) \vee$
$(Agent\ V \in a \longrightarrow V \in bad \vee V = Adm \vee V = Col)$

Finally, lemma *aanalz_Adm* is still about associations containing a ciphertext. Like *foo_V_privacy_aanalz*, it binds the variables involved in a version of the first step of the protocol. Assume an association contains the name of an honest agent who already sent a message corresponding to step one. Also assume it contains a ciphertext *Crypt P R*, and that the nonce from step one is in *parts* of *R*. If the name of the collector is absent from the association, then the following conclusions hold:

- If P is neither the signing key of the voter mentioned in the precondition nor the signing key of the administrator, then it must be the blinding factor;
- If P is the administrator's signing key, then the body of the ciphertext is exactly the body of the message signed by the voter in the bound first message:

lemma *aanalz_Adm*:
$[\![Says\ V\ Adm\ \{\!|Agent\ V, Crypt\ (priSK\ V)\ (Crypt\ b\ (Crypt\ c\ (Nonce\ Nv)))|\!\} \in set\ evs;$
$a \in aanalz\ Spy\ evs;\ Agent\ Col \notin a;\ Agent\ V \in a;\ V \notin bad;$
$Crypt\ P\ R \in a;\ Nonce\ Nv \in parts\ \{R\};\ evs \in foo]\!]$
$\Longrightarrow (P = priSK\ V \vee P = priSK\ Adm \vee P = b) \wedge$
$(P \neq priSK\ Adm \vee R = Crypt\ b\ (Crypt\ c\ (Nonce\ Nv)))$

5.2 Proof of the main theorem

Proving privacy by *foo_V_privacy_asynth* is done, as usual in the Inductive Method, by induction on the protocol model. Every protocol step generates a subgoal. When all subgoals are closed, the theorem is proven. Developing the proof required considerable effort. After eliminating redundancies and streamlining, it was reduced to about 170 steps. It will be shown that despite its length, the proving strategy is general, hence reusable for different protocols.

Induction and simplification leaves us with seven subgoals: the six protocols steps, plus *Fake*. The *Fake* is closed thanks to the classical reasoner *blast*. Its proof is simple because messages sent by dishonest agents do not yield associations. Intuitively, the goal of the Spy is to extract plausible associations, not make up new ones. However, it keeps its traditional Dolev-Yao attacker role and influences all usual theorems proven for the protocol ; those are used in the privacy proof.

The subgoal arising from *EV1* is first simplified by remarking that fresh keys (the blinding factor and commitment key) can never be known to the Spy — they cannot yet be in the set *analz (knows Spy evs1)*. We then perform a

case split about the agent *Va* involved in the version of the first protocol step generated by this subgoal. If *Va* is dishonest (a member of the *bad* set), then the message it sent yields no new association and the subgoal concludes thanks to the inductive hypothesis. If *Va* is an honest agent, we must apply, for the first time, *asynth_insert*. This lemma is of constant use throughout the proof because it allows us to split the *asynth* set. For instance, this stage of the proof features the following precondition:

a ∈ *asynth (insert*
{{|*Agent Va, Crypt (priSK Va) (Crypt ba (Crypt ca (Nonce Nva)))*|},
Agent Va, Crypt (priSK Va) (Crypt ba (Crypt ca (Nonce Nva))),
Agent Va, Agent Adm, Crypt ba (Crypt ca (Nonce Nva))}
(aanalz Spy evs1))

Let us call X the set such that *a* ∈ *asynth (insert X (aanalz Spy evs1))*. Applying *asynth_insert* leaves us with three possibilities:

1. $a = X$.
2. *a* ∈ *asynth (aanalz Spy evs1)*.
3. There exists *a2* in *aanalz Spy evs1* and an element m such that a is the union of *a2* and X and m is both in X and in *a2*.

The inductive hypothesis tells us that *Nv* is in a and X contains no nonces, so the first disjunct is excluded. The second disjunct is eliminated thanks to the lemma *nv_fresh_a2*, not quoted here, which states that fresh nonces do not appear in association syntheses.

If a is a union, more precision is required. First, if the agent V from the inductive hypothesis and the agent *Va* introduced by the induction are different, then $V \notin X$ and therefore V must be in *a2*. Since *Nv* is also in *a2*, *foo_V_privacy_aanalz* leads us to a contradiction.

Otherwise, $V = Va$. If *Nv* and *Nva* are equal, *Nv* must be fresh like *Nva*. The auxiliary lemma *aanalz_traffic*, according to which elements in associations which are not agent names must have appeared in the traffic, solves this case (fresh elements never appeared in network traffic). On the other hand, if $Nv \neq Nva$, the element in common m can be any of the elements in X. We appeal to another lemma, *association_Nv*. It is specifically tailored for this subgoal, used only here, and shows that an association containing a nonce can not contain also any of the possibilities for m listed here except for V. Together with *foo_V_privacy_aanalz*, that takes care precisely of the case $m = V$, this solves the subgoal.

The use of *asynth_insert* to split the association synthesis is a technique used for all subgoals of the theorem. It turns out that the third disjunct generates the bulk of the proving work for the remaining subgoals. We will therefore focus on it. It requires taking a close look at the structure of sets in *aanalz*.

Subgoals arising from protocol steps two and four are much larger than the other ones because of the generic specification of the steps. For instance, in the second protocol step, the administrator has received a signed ciphertext from the voter. The administrator can extract the ciphertext from the signature, but has no means in general to look inside. We only assume that it is possible to

know that the ciphertext contains no more than one atomic component, by inspecting its length. However, the precise nature of the plaintext is unknown in general and this generality in the specification of the inductive step explains the additional complexity of the proof. It is necessary if the precondition is to be realistic. Likewise, in step four, the collector receives a signed ciphertext that he cannot open in general. The concrete consequence in terms of association syntheses is that potential common elements m are not listed explicitly in the goal preconditions. Instead of belonging to a finite set of bound variables, only partial information is known about them. For instance, we may only know that an element m can be deduced from some ciphertext via *analzplus*. By contrast, for non-generic protocol steps, we obtain an explicit set and the proof is much easier.

A number of results about elements in *aanalz* are available, such as *aanalz_PR* and *aanalz_Adm*. These theorems are stated with weak premises and offer a number of conclusions as disjunctions. The most systematic proof strategy is therefore to perform case splits about the ciphertext contained in *aanalz*. As this is done, one can reason more precisely about encryption keys and plaintexts until a contradiction is reached thanks to the aforementioned results. One crucial distinction is whether the name of the collector appears in the association. If such is the case, the elements in *aanalz* arose from the collection step *EV4*. Conversely, if *Agent Col* $\notin$ *a2*, the association set in the precondition was generated by another protocol step. The encryption key P from the ciphertext *Crypt P R*, assumed to be in an association, is then compared in turn to the voter's signing key, the administrator's signing key, and to the blinding factor. Contradictions are reached in every case. The value of the payload R is also compared with the voter's blinded vote commitment *Crypt b (Crypt c (Nonce Nv))*. Those different situations obviously refer to various ciphertext values naturally generated by the protocol steps. In essence, the proving strategy amounts to zooming in sufficiently into the various possible association configurations to uncover contradictions that are not apparent at a more general level.

The outline of this proving strategy is not dependent of a given protocol. Let us recall the important steps:

1. For every subgoal, split the association syntheses set *asynth* using *asynth_insert*.
2. The subgoals arising from explicit protocol steps are straightforward to close because the set of potential common elements m becomes explicit as well.
3. For more general subgoals, case splits about the possible values of initially generic ciphertexts are combined with lemmas describing their structure in associations in a systematic way.

5.3 Proof of the supporting theorems

Rather than describing the full proof of every theorem required for the privacy one, we focus on *aanalz_PR* due to space constraints. It is of constant use in the privacy proof, appearing in it eleven times, and its proof exemplifies the kind of reasoning required for the other supporting theorems. Recall its statement

from earlier (5.1) ; it constrains the form of elements of *aanalz* that contain a ciphertext.

As expected, complications arise again from the generic steps, namely *EV2* and *EV4*. As the other subgoals are easier to prove, let us concentrate on *EV2*, as the proof for *EV4* is similar.

We require the following subsidiary results in addition to standard lemmas from the existing Inductive Method framework:

- *analzplus_into_parts*: Elements in the set *analzplus X ks* (recall *ks* is the external key set) are in *parts X*.
- *no_pairs*: If a message contains only atomic components and already contains an agent name in its *parts* set, a number of other elements cannot be in the *parts* set.
- *analzplus_Nv*: Assume an *analzplus Q (analz H)* set contains a ciphertext and a nonce. If *parts Q* contains only atomic components, then the decryption key of the ciphertext must be in *analz (insert Q H)*.

The case where the administrator (*Adm*) is dishonest is closed easily. Else, we must distinguish cases on the basis of the origin of the association in the inductive hypothesis. The first possibility is that it was generated by the *EV2* message introduced by the induction. In that case, the key *P* in the *aanalz_PR* theorem statement could either be the administrator's signing key, or the encryption key of the ciphertext that he signed. A third possibility is that the entire *Crypt P R* is embedded deeper in the signed ciphertext. Let us examine each possibility in turn.

In the first case, we must show that an agent name (either the collector or a regular voter) and a nonce cannot be present in *analzplus R* at the same time. This is shown by combining *analzplus_into_parts* and *no_pairs*. In the second case, the ciphertext *Crypt P R* from the theorem precondition is exactly the ciphertext *Crypt Pa Ra* signed by the administrator in this version of the second protocol step. Disentangling the precondition conjunction leads to the same scenario and an additional one that entails proving that if the inverse of key *P* is known to the attacker in the first place, it is all the more known to her after getting hold of *R*.

If *Crypt P R* is embedded in the ciphertext generated by the administrator, we must perform a few additional case splits but the line of reasoning is the same, with the additional use of *analzplus_Nv*.

Even though specifying the possible forms of elements in *aanalz* requires inspecting a number of scenarios, the proving process is straightforward once some crucial building blocks are established. Notably, the three subsidiary results we listed earlier (*analzplus_into_parts* and so on) are stated without any reference to the FOO protocol — they are protocol-independent and can be reused directly. A submission of our Isabelle theories to the online Archive of Formal Proofs [1] is being prepared.

6 Conclusion

We have presented the first interactive theorem proving-based analysis of voter privacy. It offers an independent and complementary means of investigation to consolidated work based on process equivalence, ultimately contributing to the trustworthiness of voting systems. Privacy is modelled as an unlinkability property between a voter and her ballot. Extensions to the Inductive Method are implemented in Isabelle/HOL to specify associations between elements and combinations of associations that share a common element.

The initial proof development effort was significant, but a coherent line of reasoning emerges from the proof. This general strategy and a number of protocol-independent results about the new operators support the case of re-usability for other e-voting protocols. Interactive proofs entail a level of clarity about protocol scenarios that is unavailable from automatic tools. The inductive nature of our specification eliminates termination issues or inherent size limitations. While the benefits of automated tools are clear, our approach sheds a complementary light on voter privacy by its operational view.

A more general version of the *asynth* operator, allowing unbounded association synthesis, is needed. Other privacy-type properties such as receipt-freeness and coercion-resistance ought to be specified in the Inductive Method. Additionally, e-voting protocols that are not amenable to analysis in the process equivalence model must be studied in our framework to investigate its domain of applicability. We would also like to program some of the recurring proof steps as ML tactics.

Acknowledgement This research was supported in part by the Science Fundation Ireland (SFI) grant 08/RFP/CMS1347.

References

1. In G. Klein, T. Nipkow, and L. Paulson, editors, *The Archive of Formal Proofs*. `http://afp.sf.net`.
2. M. Abadi and C. Fournet. Mobile Values, New Names, and Secure Communication. In *Proc. of the 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115. ACM Press, 2001.
3. G. Bella. *Formal Correctness of Security Protocols*. Information Security and Cryptography. Springer, 2007.
4. G. Bella. Inductive study of confidentiality: for everyone. *Formal Aspects of Computing*, pages 1–34, 2012.
5. G. Bella, F. Massacci, L. C. Paulson, and P. Tramontano. Formal Verification of Cardholder Registration in SET. In F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, editors, *Proc. of the 6th European Symposium on Research in Computer Security (ESORICS'00)*, LNCS 1895, pages 159–174. Springer, 2000.
6. G. Bella and L. C. Paulson. Mechanical Proofs about a Non-Repudiation Protocol. In R. J. Boulton and P. B. Jackson, editors, *Proc. of the 14th International Conference on Theorem Proving in Higher Order Logics (TPHOLs'01)*, LNCS 2152, pages 91–104. Springer, 2001.

7. B. Blanchet. An Efficient Cryptographic Protocol Verifier Based on Prolog Rules. In *Proc. of the 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96. IEEE Press, 1998.
8. D. Butin and G. Bella. Verifying Privacy by Little Interaction and No Process Equivalence. In *SECRYPT*, pages 251–256. SciTePress, 2012.
9. R. Chadha, Ştefan Ciobâcă, and S. Kremer. Automated Verification of Equivalence Properties of Cryptographic Protocols. In H. Seidl, editor, *ESOP*, LNCS 7211, pages 108–127. Springer, 2012.
10. S. Delaune, S. Kremer, and M. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, 2009.
11. S. Delaune, M. Ryan, and B. Smyth. Automatic verification of privacy properties in the applied pi calculus. *Syntax*, 263/2008:263-278, 2008.
12. A. Fujioka, T. Okamoto, and K. Ohta. A Practical Secret Voting Scheme for Large Scale Elections. In *Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques: Advances in Cryptology (ASIACRYPT'92)*, pages 244–251. Springer-Verlag, 1993.
13. S. Kremer and M. Ryan. Analysis of an Electronic Voting Protocol in the Applied Pi Calculus. In *In Proc. 14th European Symposium On Programming (ESOP'05), LNCS 3444*, pages 186–200. Springer, 2005.
14. J. E. Martina and L. C. Paulson. Verifying Multicast-Based Security Protocols Using the Inductive Method. In *Workshop on Formal Methods and Cryptography (CryptoForma 2011)*, 2011.
15. L. C. Paulson. *Isabelle: A Generic Theorem Prover*. LNCS 828. Springer, 1994.
16. L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6:85–128, 1998.
17. M. Ryan. Keynote: Analysing security properties of electronic voting systems. In Ú. Erlingsson, R. Wieringa, and N. Zannone, editors, *ESSoS*, volume 6542 of *LNCS*, pages 1–14. Springer, 2011.