Formal Verification of Authentication-Type Properties of an Electronic Voting Protocol Using mCRL2

Hamid Reza Mahrooghi, Mohammad Hashem Haghighat, Rasool Jalili

Network Security Center
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran
{mahrooghi@ce., a_haghighat@ce., jalili@sharif.edu}

Having a doubtless election in the information technology era requires satisfaction and verification of security properties in electronic voting (e-voting) systems. This paper focuses on verification of authentication-type properties of an e-voting protocol. The well-known FOO92 e-voting protocol is analyzed, as a case study, against the uniqueness and eligibility properties and their satisfaction are verified. By means of an automated formal approach, the protocol is modelled in the mCRL2 language, which is a combination of the ACP process algebra language and abstract data types (ADT). Then, the eligibility and uniqueness properties as two authentication-type requirements are modelled in the modal µ-calculus. These are given to a combination of dedicated mCRL2 tools to verify the properties. Our research is valuable due to its direct modelling of authentication-type properties and their verification. The experiment can be easily generalized as a pattern for verification of similar protocols.

Electronic-voting protocols, Formal verification, mCRL2, Eligibility, Uniqueness.

1. INTRODUCTION

Electronic voting (e-voting) systems are aimed to rectify the problems traditional voting systems tackled with. Such an effort includes simplification of the voting process and achieving the whole process more efficiently and trustworthy. However, electronic nature of votes in such systems make them more vulnerable in terms of vote manipulation, impersonation, double-voting, missing of cast ballots, and so on; compared to the traditional approaches. Accordingly, researchers during the last three decades have investigated on different aspects of the security of e-voting systems. Of the interest of this paper, is a list of security requirements to be considered in designing an e-voting system. The most important requirements on the list include preservation of voters privacy, voter authentication, result verifiability, and fairness. Security related properties of e-voting systems can be classified into the following categories:

• **Authentication-Type properties;** which ensure that only the authenticated eligible voters cast their ballots. This category consists of eligibility (voting by only registered and authenticated voters) and uniqueness (voting only once).

• **Privacy-Type properties;** which guarantee the link between a voter and his vote is kept concealed. This category includes voter privacy (impossibility of determining who voted to whom), receipt-freeness (inability of gaining any knowledge by a voter to prove his vote’s content to a coercer or others) and coercion-resistance (inability of a coercer to force a voter to vote based on his wishes).

• **Accuracy-Type properties;** which provide the impression of accuracy through the possibility of verifying any objection. This category includes accuracy (only valid votes are considered in the result), individual verifiability (a voter is able to verify if his vote was correctly tallied), universal verifiability (the ability to verify if the election results is exactly the sum and distribution of all the cast ballots), open objection (the ability to object against the election results without revealing any vote), and fairness (inability of affecting the
remaining voters by revealing the intermediate results).

The negative political and social repercussions of incomplete and vulnerable e-voting systems design and implementation are so serious, especially when such systems are targeted to service for large communities or at the national level. Accordingly, verification of security properties is vital in all stages of the system development and operation. In several cases, it has been shown that an observational analysis cannot guarantee complete satisfaction of these properties (Rajabzadeh Assar, 2008). To overcome the dissatisfaction of observational analysis, formal methods can help. Using formal methods, the system is modelled in a mathematical form in order to provide the platform for a more precise checking.

Our objective in this paper is to investigate the verification of some security properties of an e-voting protocol. To this aim, process algebra performs a basic role. As a well-known and important formal method, process algebra models the behaviour of a system as a set of communicating subsystems, while providing modular specification to make specification of complex systems straightforward and understandable (Glabbeek and Vaandrager, 1989). Several formal languages and their corresponding tools are available, selection of one depends on the system and properties under study and verification. In particular, support of the language for data type definition is important for specification of complex behaviours. The mCRL2 language—an extension of µCRL—was developed to address this requirement. While several built-in data types exist in mCRL2, the language supports definition of new data types. We realized that mCRL2 is a well-adapted language to model security protocols such as e-voting protocols.

The FOO92 e-voting protocol (Fujioka et al., 1993), based on the digital blind signature approach, is selected as our case study in this paper, while our approach is not restricted to FOO92. The protocol is specified in the mCRL2 language. In order to verify the two authentication-type properties in the protocol, namely eligibility and uniqueness, the properties are specified in the modal µ-calculus and are verified using the mCRL2 toolset.

**Related work.** Over the past three decades, several formal methods have been developed for analyzing security protocols. Among them, a track has been focused on formal analysis of e-voting protocols based on process algebra and in particular applied pi-calculus. In (Kremer and Ryan, 2005), the authors modelled the FOO92 protocol using applied pi-calculus and studied the fairness and privacy properties of the protocol. They verified the eligibility property indirectly, via a challenged vote. Their further research in (Delaune et al., 2006a, Delaune et al., 2006b, Delaune et al., 2009) mainly focused on the privacy-type properties of some e-voting protocols and determining the relationship between verification of these properties. A formal analysis of the Internet-based e-voting protocol RIES, using µCRL, is also presented in (Maasbommel and Fokkink, 2007).

**Structure of the paper.** In section 2, the FOO92 protocol is described accompanying with description of major concepts in this category of e-voting protocols. Relevant concepts to formal methods for specification and verification of crypto-based protocols are given in section 3. The mCRL2 specification of FOO92 protocol with a rational intruder is presented in section 4. Section 5 includes our modelling of eligibility and uniqueness properties in the modal µ-calculus as well as the result of our model checking using the mCRL2 toolset. Section 6 draws some conclusions and discusses possible trends for future work.

## 2. THE FOO92 PROTOCOL

In most of the existing e-voting protocols, a set of components, stages, and concepts have been taken into account; so they are required to be considered in our analysis. These components and concepts are as follows.

A voter is a person who can cast his hidden vote freely according to the effective rules and regulations. The Registration Authority registers eligible voters before an election (according to relevant rules). This authority guarantees that only eligible voters cast their votes. The Collecting/Tallying Authority stores valid votes and counts them at the end of the election. The intruder is a subject who attempts to disrupt and challenge against the e-voting system's security. Almost all the e-voting protocols consist of the following three phases, while there might be some minor differences.

- The **Registration Phase**; in which the authorized people are registered for casting a vote.
- The **Voting Phase**; in which registered voters request their ballots from the respective authority on the election day. As the collecting authority authorizes the voters’ votes, a certificate indicating the eligibility of voters should be presented.
- The **Tallying Phase**; in which valid votes are counted and the results are published.

The Validation process is also involved in all the election phases; in the registration phase for voters authentication, in the voting phase for the vote validation; and in the tallying phase for
verifying that each eligible voter is casting his vote only once. Commitment is a special mechanism of being bound to some hidden value being sent to the other party. The hidden value can be revealed with the cooperation of the sender in a later phase, without the possibility of changing the hidden value. In an e-voting system, voters can send their committed vote to the respective authorities without disclosing the decommitment (open) key. At the end of the election, voters send their keys for opening and counting their votes. Several e-voting protocols, use the commitment scheme for satisfying some properties, such as fairness. However, a problem with this approach is the requirement of voters to participate in the tallying phase (Sampigethaya and Poovendran, 2006).

Anonymous channels can be used to provide anonymity of voters (not revealing the relationship between voters and their votes). Some e-voting protocols also use digital blind signature scheme (Chaum, 1984) to provide the voter-privacy in a simple, efficient, and flexible way to be implemented over the Internet. This approach allows signing a vote blindly without disclosing any information. FO092 is a well-known e-voting protocol, satisfying fairness, eligibility, privacy and individual verifiability properties, based on an informal analysis (Zuzana, 2002). The protocol consists of voters, an administrator who identifies eligible voters, and a collector who collects votes, counts, and publishes the result. Anonymous channels are assumed in the protocol for communication between voters and the collector authority. The protocol is described through three phases as follows:

In the first phase, voters receive their committed votes, signed by the administrator. To preserve voter’s privacy, the protocol uses blind signature, so the administrator does not get any information about his vote. Therefore:

- The voter V chooses a vote \( v \) and commits his vote through the commitment schema \( \xi \), the random key \( r \) and computing of \( x = \xi(v, r) \).
- V computes the message \( e = \chi(x, b) \) by the blinding function \( \chi \) and the blinding factor \( b \).
- V signs \( e, \sigma_v(e) \), and sends it together with his identity to the administrator A.
- A checks if V is an eligible voter, if this is his first vote, and if his signature is valid. In case these are all valid, A will sign V’s vote and sends it back to V.
- At this time, V has \( \sigma_A(e) \), his blinded committed vote signed with A. V unblinds \( \sigma_A(e) \) and obtains \( y = \sigma_A(x) \) (i.e. his own committed vote signed by the administrator).

In the second phase, as the main phase of voting;

- V sends \( y \) to the collector C via an anonymous channel.
- C checks the administrator’s signature on \( y \) and if it is valid, inserts \((l, x, y)\) as the l-th element of his memory list.

In the final phase of the protocol, voters reveal their random key \( r \) in order to enable C to open the votes and publish the results.

- C publishes the committed list in the form of \((l, x, y)\).
- V checks the existence of his committed vote in the list; then sends \( l \) and \( r \) via the anonymous channel to C.
- C opens the l-th element of the list using \( r \) and publishes the vote.

3. FORMAL ANALYSIS

Formal verification aims at presenting a mathematical proof of functional validity of a system. A formal approach provides a model (language) for system specification, a model (language) for expressing desired requirements, which need to be verified and an analytic technique to verify that the requirements are satisfied with the system specification. Process algebra is one of the well-known formal methods based on the process theory, which models the system behaviour as a set of communicating subsystems. Support of modularity in process algebra makes specification of complex systems manageable. Application of algebraic methods and rules makes computational operations over processes feasible. Process algebra is also considered as a method based on concurrency theory and usually has the basic operator for parallel composition. The \( \mu \text{CRL} \) language (Groote and Reniers, 2007) is to enrich the functionality and capability of process algebras languages such as CCS, CSP and ACP and to express the behaviour of real systems. The \( \mu \text{CRL} \) language is a combination of ACP language and ADT, with the aim of supporting data parameters in process communications and their effects on each other. We use mCRL2, which is a new version of the \( \mu \text{CRL} \) language in our modelling.

3.1. The mCRL2 Language

mCRL2 is a process algebra language which supports data types, a formal specification language with a rich syntax, and a strong toolset with visualization, simulation, and verification facilities (Blom et al., 2004).

Data type definition in mCRL2 is straightforward. Unlike the \( \mu \text{CRL} \) language which does not include any predefined data type, mCRL2 embeds the definition of some basic data types such as Boolean, Natural, Integer, List, Bag and Set. Basic
data definitions are not sufficient for convenient description of some protocols. Therefore, the language facilitates the definition of additional data types. As an example, a special binary data type can be defined as:

\[
\text{sort Sample} = \text{struct sample}(n: \text{Nat}, b: \text{Bool});
\]

\[
\text{map}
\]

\[
\text{eq} : \text{Sample} \ # \ \text{Sample} \rightarrow \text{Bool};
\]

\[
\text{first} : \text{Sample} \rightarrow \text{Nat};
\]

\[
\text{second} : \text{Sample} \rightarrow \text{Bool};
\]

\[
\text{var}
\]

\[
n, n' : \text{Nat};
\]

\[
b, b' : \text{Bool};
\]

\[
\text{eqn}
\]

\[
eq (\text{sample}(n, b), \text{sample}(n', b')) =
\]

\[
\text{if}(((n == n') && (b == b')), \text{true}, \text{false});
\]

\[
\text{first} (\text{sample}(n, b)) = n;
\]

\[
\text{second} (\text{sample}(n, b)) = b;
\]

in which, “Sample” is a binary data type whose first element is a Natural number and its second element is a Boolean value. Two data structures of this data type are equal (the “eq” equation), when their corresponding elements are equal. Using the first and the second equations, the first and second elements of this data type can be extracted, respectively.

**Algebraic Process Specification** in mCRL2 is achieved using the defined data types, actions, and composition of actions. Actions can be composed using the choice, conditional, parallel, and alternative operators. The core of a mCRL2 description is its Proc section where the system’s behaviour is described. This description is as:

\[
\text{Proc } X(x_1 : s_1, \ldots, x_n : s_n) = P
\]

where \(X\) is a process name, \(x_i\) is a variable name, \(s_i\) is the data type of \(x_i\), and \(P\) is a process expression describing \(X\). A process expression is formed based on the following expressions:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P, Q, := Id)</td>
<td>an action</td>
</tr>
<tr>
<td>(Id(DataPar))</td>
<td>an action with one or more parameters</td>
</tr>
<tr>
<td>(delta)</td>
<td>dead lock</td>
</tr>
<tr>
<td>(tau)</td>
<td>internal action</td>
</tr>
<tr>
<td>(P \parallel Q)</td>
<td>parallel composition</td>
</tr>
<tr>
<td>(P</td>
<td>Q)</td>
</tr>
<tr>
<td>(P + Q)</td>
<td>nondeterministic choice</td>
</tr>
<tr>
<td>(c \rightarrow P)</td>
<td>conditional (or if-then)</td>
</tr>
<tr>
<td>(sum(IdDecl,P))</td>
<td>summation</td>
</tr>
<tr>
<td>(allow(SetOfActions,P))</td>
<td>allowing</td>
</tr>
<tr>
<td>(hide(SetOfActions,P))</td>
<td>hiding</td>
</tr>
<tr>
<td>(comm(SetOfActions,P))</td>
<td>communication</td>
</tr>
</tbody>
</table>

The sum operator \(\sum_{d \in D} P(d)\) is a generalisation of the choice operator. The notation \(P(d)\) is used to stress that \(d\) can occur in the process \(P\). Where \(P + Q\) allows a choice among processes \(P\) and \(Q\), \(\sum_{d \in D} P(d)\) allows to choose any \(P(d)\) for some value \(d\) from \(D\). If \(D\) is finite, then the sum operator can be expressed using the choice. For example, if in case \(d\) is Boolean (Groote and Reniers, 2008):

\[
\sum_{d \in \{true, false\}} P(d) = P(true) + P(false)
\]

The mCRL2 Toolset is a combination of several tools. The mcrllps tool receives a protocol descriptions based on the mCRL2 language as input and automatically converts it into the Linear Process Specification (LPS) form. This LPS format is the basis for other components of this toolset. The lps2lts tool generates the LTS graph from the LPS description format. The simulated behaviour of the protocol can be traceable using the xsim tool. The visualization of the LTS graph can be shown using the ltsview and ltsgraph tools. The description of desired property, which is based on the modal \(\mu\)-calculus, and the LPS form are given to both the lps2pbes and pbes2bool tools as inputs and the validity of the property is checked and reported.

### 3.2. Modelling of FOO92 in mCRL2

Our experience of using mCRL2 to model the FOO92 protocol is discussed in this section. The modelled properties in the modal \(\mu\)-calculus are also verified using the mCRL2 toolset. Based on our introduction of the protocol, the modelled processes are Voter, Administrator, Collector, and Intruder w.r.t. the agents of protocol. Due to our goal of verifying the authentication-type properties, only one voter and one intruder are considered. Then it is verified whether the intruder can vote without authorization or the corrupted voter can vote more than once.

#### 3.2.1. Data Types Definitions

In our model, as partially shown below, commitment is modelled using the functions commit and open. The sign function is used to model digital signature of votes and this signature can be checked through the checksign function. To model the blind signature, two functions blind and unblind are added. The getpk and pair functions extract an agent’s public key using the agent’s identity and private key, respectively. Finally, the inc function is used to compute final result by incrementing the respective candidate’s number of votes.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>map checksign/3</td>
<td>(<em>check vote's sign/</em>)</td>
</tr>
<tr>
<td>map blind/2</td>
<td>(<em>blind vote</em>)</td>
</tr>
<tr>
<td>map unblind/1</td>
<td>(<em>unblind vote</em>)</td>
</tr>
<tr>
<td>map open/2</td>
<td>(<em>open committed vote</em>)</td>
</tr>
<tr>
<td>map commit/2</td>
<td>(<em>commit vote</em>)</td>
</tr>
<tr>
<td>map sign/2</td>
<td>(<em>sign vote</em>)</td>
</tr>
<tr>
<td>map getpk/1</td>
<td>(<em>get public key from id/</em>)</td>
</tr>
<tr>
<td>map pair/1</td>
<td>(<em>get public key from secret key/</em>)</td>
</tr>
</tbody>
</table>
3.2.2. The Intruder Process

To verify the security properties of a protocol, we should consider all the processes in a non-secure environment, with the presence of the adversary. According to the Dolev-Yao (Dolev and Yao, 1983) adversary model, an intruder is specified as a process having the following capabilities:

C1. Communicating with the other processes.
C2. Eavesdropping all messages being exchanged among processes.
C3. Opening and saving messages which are encrypted with its public key. It may save parts of the message in its knowledge-base.
C4. Saving messages in order to do replay them, even if it cannot open them.
C5. Forging messages using any combination of its knowledge-base.

As depicted in Figure 1, the intruder mediates all the communication between each sender and each receiver and so we model any communications via the intruder. This means that sending a message from the voter to the administrator is replaced by sending the message from the voter to the intruder and then sending it to the administrator. In the other words, instead of the action “snd1 V_to_A”, we consider “snd1 V_to_I” and “snd1 I_to_A” (1: step#, V: voter, A: admin, I: intruder). It means that sending a message from the voter to the administrator is changed to the sending the message from voter to the intruder and then sending it from the intruder to the administrator. Therefore, five main steps of the FOO92 protocol are modelled as the following synchronized actions via the intruder.

```
map inc1 (*increment final result*)

unblind(sign(blind(cv,b), admin_key), b) =
  sign(cv, admin_key);
checksign(sign(bcv,sk),bcv1.pk) =
  if(pair(sk) == pk, eq(bcv,bcv1), false);
```

According to the capabilities C1-C5 and the above actions A1-A10, the intruder process is defined as follows. Each part of this process refers to these capabilities and actions through some C#-A# notations in the inline comments. IN fact, that segment of the intruder behaviour has been modelled in order to provide the C# capabilities by doing the A# actions. For example, “Attack on the eligibility property” is a part of the intruder process related to C4 and C5 capabilities. It is modelled by actions A2 and A3 where the intruder is sender or receiver (x-I-x).

![Figure 1: Communications via an intruder](image)

```
A1. snd1 V_to_I | rcv1 I_from_V = com1 Vl
A2. snd1 I_to_A | rcv1 A_from_I = com1 IA
A3. snd2 A_to_I | rcv2 I_from_A = com2 AI
A4. snd2 I_to_V | rcv2 V_from_I = com2 IV
A5. snd3 V_to_I | rcv3 I_from_V = com3 VI
A6. snd3 I_to_C | rcv3 C_from_I = com3 IC
A7. snd4 C | rcv4 I = com4 C
A8. snd_4 I | rcv4 V = com4 I
A9. snd5 V_to_I | rcv5 I_from_V = com5 VI
A10. snd5 I_to_C | rcv5 C_from_I = com5 IC

Proc Intruder(id, intruder_id, sk, key, sk:Set(key),
  vid:Set(voter_id),
  vsbc_vote:Set(signed_blinded_committed_vote),
  bc_vote:Set(blinded_committed_vote),
  asbc_vote:Set(signed_blinded_committed_vote),
  asc_vote:Set(signed_committed_vote),
  c_vote:Set(committed_vote),
  l_set:Set(Nat),
  r_set:Set(Nat)) =
  (* two random numbers r, b are known as
   the bit commitment key and the blinding factor*)

/* Attack on the eligibility property:
   C4,C5-A2,A3 */

A1. snd1 I_to_A(id, sign(commit(chosen_vote,r),r), sk),
  commit(chosen_vote,r),
  \sum_{svbc: signed_blinded_committed-vote: rcv2 I_from A}(svbc),
  checksign(svbc,bcv,admin_pk) ->
  commitCP(getpk(id)).
Intruder(id, sk, vid, vsbc_vote, bc_vote,
  asbc_vote, asc_vote,c vote, l_set+{l}, r_set) +

/* Attack on the uniqueness property:
   C4,C5-A2,A3,A6,A7,A10 */

/* Collusion corrupted voter with Intruder*/
```

(k in sk && id in vid) ->

/**/
3.2.3. Initial State Declaration

The Init process models the environment of the protocol execution. It determines initial assumptions (about the hidden or allowed actions), synchronized actions, and the parallel compositions of processes with their initial parameters. It is coded as follows.

```
Init
  hide(commitCP, eligibleCP, saveCP),
  allow(com1 Va, com2 Av, com3 VC, com4 Cv,
        com5 VC, commitCP, eligibleCP, saveCP),
  comm([snd1 V to I | rcv1 I from V -> com1 VI,
        snd1 I to A | rcv1 A from I -> com1 IA,
        snd2 A to I | rcv2 I from A -> com2 AI,
        snd2 I to V | rcv2 V from I -> com2 IV,
        snd3 V to I | rcv3 I from V -> com3 VI,
        snd3 I to C | rcv3 C from I -> com3 IC,
        snd4 C | rcv4 I -> com4 C,
        snd4 I | rcv4 V -> com4 I,
        snd5 V to I | rcv5 I from V -> com5 VI,
        snd5 I to C | rcv5 C from I -> com5 IC],
  Voter(v1, sk1) || Administrator(pk1) || Collector(0) || Intruder(i1, ski, { }, { }, { }, { }, { }, { }, { }, {}));
```

The commitCP, eligibleCP and saveCP actions are defined to help in verifying the desired properties (eligibility and uniqueness). In addition, communication actions are specified using the term "comm".

The number of voters is not important in our analysis due to the type of verified properties (authentication). So, without loss of generality, we model the protocol with one voter in order to prevent the state space explosion problem. The Voter, Administrator, Collector and Intruder processes are initialized by their parameters including:

- identities of administrator, voter, and candidates
- the public and private keys of the voter and the administrator
- the set of eligible voters
- blinding factors and random numbers

Now, we should verify whether the intruder can vote without authentication or the corrupted voter can vote more than once.

3.2.4. The Voter Process

The Voter process is described as follows.
Proc Voter(idi:voter_id, ski:key) =
1. snd1_V_to_I(idi, 
   sign(blind(commit(chosen_vote,r),b),ski), 
   blind(commit(chosen_vote,r),b)).
2. rcv2_V_from_I(sbcv).
3. checksign(sbcv,bcv,admin_pk) ->
4. (commitCP(getpk(idi)).
5. snd3_V_to_I(unblind(sbcv,b)).
   ∑ l:Nat ∑ cv:committed-vote.rcv4_V(l,cv).
6. snd5_V_to_I(l,r));

It is worth mentioning that based on our assumption of the intruder mediation, all messages from the voter process are sent toward the intruder, and all received messages have been sent from the intruder.

Two random numbers $r$ and $b$ are known as the blind factor and the vote commitment key, respectively. The voter blinds his signed vote and sends it to the administrator and requests a signature on it (snd1_V_to_I in line 1). When the voter receives administrator’s response (rcv2_V_from_I in line 2), he verifies the signature (checksign in line 3), eliminates the blind factor of his vote, and sends it to the collector (snd3_V_to_I in line 4). As soon as the list of committed votes is published, the voter receives his index (rcv4_V in line 5) and sends his commitment key “r” to the collector at the end of the election for decommitment (snd5_V_to_I in line 6).

3.2.5. The Administrator Process

The Administrator process, as shown below, checks the received message (signed and blinded vote) from the voter and signs his message only once, if the voter is eligible.

Proc Administrator(eligible_voter_list:Set(voter_id)) =
   ∑ sbcv:signed-blinded-committed-vote ∑ id:voter-id.
   rcv1_A_from_I(id,sbcv,bcv).
   checksign(sbcv,getpk(id)) ->
      "* Voters Authentication *
      ((eligibleCP(getpk(id)).
      snd2_A_to_I(admin_sign(sbcv),bcv).
      Administrator(eligible_voter_list-{id}))
   <> Administrator(eligible_voter_list));

3.2.6. The Collector Process

The Collector process consists of some steps in accordance with the FOO92 protocol as follows.

Proc Collector(votes:Set(committed_vote)) =
   (all-voters-vote) ->
   ∑ scv:signed-committed-vote
   ∑ id:voter-id.rcv3_C_from_I(scv).
   checksign(scv,admin_pk) ->
      (saveCP.
      collector(votes+{committed_vte_list(unsign(scv),l)})
   <> collector(votes)
   <> broadcast_list:
   Proc broadcast_list(votes) =
      (votes-not-empty) ->
      ∑ cv: committed-vote ∑ l:Nat.snd4_C(l,cv) <>
      counting:
      Proc counting =
         ∑ l: Nat ∑ r: Nat.rcv5_C_from_I(l,r);

At the third phase of election, the collector publishes the list of all the votes that have been received and then waits to receive each voters decommitment (open) key. Finally, it opens and tallies the votes.

3.3. Analysis

The LPS format of the FOO92 protocol specification is made using the mcrl22lps tool and its LTS graph is then generated using the lps2lts tool. A simulated running of the protocol based on this graph can be traceable using the xsim tool. The visualization of the LTS graph is shown in Figure 2, using the ltsview tool.

As mentioned earlier, the three actions “commitCP”, “eligibleCP”, and “saveCP” are defined as checkpoints embedded in the protocol specification. Through tracing of the checkpoints, verification of the eligibility and uniqueness properties is made feasible. Accordingly, we model the properties in the modal μ-calculus, as follows.

3.3.1. Eligibility

The eligibility requirement influences the protocol to disallow un-registered people to vote. The Administrator process authenticates each voter and generates the corresponding checkpoint. This means that eligibility can be modelled as a successful authentication and generation of a valid checkpoint by Administrator. In fact, in concurrent execution of the processes, execution of any “commitCP” action for each public key without respective prior “eligibleCP” or accepting and saving a non-authenticated vote, (in other words) indicates an attack. Therefore, we model this property in the modal μ-calculus as follows.

\[ (\neg (eligibleCP(pk1)) \land \neg commitCP(pk1)) \land \neg (saveCP) \] (1)
This statement ensures that the set of states, where action "eligibleCP" is not observed before action "commitCP", should be empty (i.e. authentication of voters must not be bypassed). The eligibility term (1) and the LPS format of the protocol description, were given to both the lps2pbes and pbes2bool tools as inputs. Finally the validity of this property was proved.

3.3.2. Uniqueness

The uniqueness requirement enforces that each voter can vote only once. Only one "commitCP" and one "saveCP" action should be done for each voter based on the checkpoints and specification of the Collector process. This means that uniqueness is violated in two ways. First, if a voter can acquire the administrator to sign more than once, then the "commitCP" action is occurred more than once for the respective public key. Second, if the collector accepts and saves a vote more than once, then the "saveCP" action is done more than once. Accordingly, the specification of this property in the modal $\mu$-calculus is:

\[
\text{true}*.\text{commitCP}(pk1).\text{true}*.\text{commitCP}(pk1)\text{false} \&\& \text{true}*.\text{saveCP}.\text{true}*.\text{saveCP}\text{false}. \quad (2)
\]

The term (2) states that the set of states in which the "commitCP" action is done more than once per individual public keys, must be empty. The empty set implies that the voters cannot send more than one vote to the collector authority. This property was proved in the same way as the eligibility property using the toolset.

4. CONCLUSIONS AND FUTURE WORK

We modelled the VO92 e-voting protocol using the mCRL2 language and proved its eligibility and uniqueness properties. The properties were specified in the modal $\mu$-calculus and verified using mCRL2 toolset. As future direction, investigating our current experience on the other well-known e-voting protocols as well as modelling and verification of other properties, such as verifiability, are of special interest. Alternative approaches, such as combination methods of theorem proving and model checking, can also be considered as future work.

5. ACKNOWLEDGEMENT

The authors wish to thank Dr. Seyyed MohammadReza Mousavi from Eindhoven University of Technology for his valuable comments and corrections. This work is partially supported by Iranian Telecom Research Center (ITRC).

6. REFERENCES


