Development of A Formal Security Model for Electronic Voting Systems

Katharina Bräunlich, Department of Information Systems Research, University of Koblenz-Landau, Koblenz, Germany

Rüdiger Grimm, Department of Information Systems Research, University of Koblenz-Landau, Koblenz, Germany

ABSTRACT

Trust that an electronic voting system realizes the security requirements in an adequate manner is an essential premise for electronic elections. Trust in a system can be achieved by controlling the system security. There are two ways to assure system security: One way is the evaluation and certification of the implementation’s security by neutral experts. Another way is the verification of the outcome by the users. Both approaches, verification and certification, should be combined to reasonably justify the voter’s trust in the electronic voting system. In this paper a formal security model with respect to the requirements of Fairness, Eligibility, Secrecy and Receipt-Freeness, Verifiability and Protection against Precipitation is given. This formal model helps to clarify and truly understand these requirements. Furthermore, it can be used for the evaluation and certification of online voting products according to the Common Criteria.

Keywords: Common Criteria, Development of a Formal Security Model, Electronic Voting, System Evaluation and Certification, Fairness, Eligibility, Receipt-Freeness, Secrecy

INTRODUCTION

Security is an elementary property of electronic voting systems. Security objectives for electronic voting were first collected in an informal way, for example by a European-wide accepted recommendation adopted by the Council of Europe (Council of Europe, 2004). Later the semi-formal method of the Common Criteria (CC) (Common Criteria, 2006) was used to specify a basic set of security requirements for online voting products (Volkamer & Vogt, 2008). Furthermore, the method KORA (Konkretisierung rechlicher Anforderungen/Concretisation of Legal Requirements) (Hammer, Pordesch, & Roßnagel, 1993) was used to derive security objectives for electronic voting from legal specifications (Bräunlich, Grimm, Richter, & Roßnagel, 2013).

DOI: 10.4018/jisp.2013040101
Besides their differences, there is common agreement that the following security objectives have to be granted by an electronic voting system:

- **Fairness:** Each voter can cast exactly one vote and no voter loses his voting right without having cast a vote.
- **Eligibility:** Only eligible voters who are unmistakably identified and authenticated are allowed to cast a vote that is stored in the ballot box.
- **Protection Against Precipitation:** The voter can abort his voting process at any time prior to the final casting of the vote without losing his right to vote and there is no limit on the number of corrections a voter can make to his vote until the final casting of the vote.
- **Secrecy and Receipt-Freeness:** The voter's voting decision must be kept secret (ballot secrecy) and must be enforced even against the voter himself (receipt-freeness). I.e. the voter does not gain any information that enables him to prove his voting decision to others. By hiding a voter's voting decision, the voter is protected against unlawful influence.

In the context of electronic voting, it is necessary that the voters have trust in the system security of the electronic voting system. One way to strengthen the trust of the voters in the electronic voting system is the assurance of system security by means of verifiability.

- **Verifiability:** To grant traceability of the whole voting procedure, so called end-to-end-verifiability (Adida & Neff, 2006), the essential steps of an election must be open for examination by the voter. The essential steps are the marking and encryption of the vote, the casting of the encrypted vote (=ballot) and the counting of the ballots from the ballot box, or more precisely:
  - **Cast-As-Intended:** The voter can ascertain himself that the ballot is correctly encrypted and represents his voting decision correctly. In order to protect ballot secrecy, cast-as-intended can only be verified by the voter himself. It can be verified by the voter only before the ballot has been cast into the ballot box. Because otherwise, this verification functionality could be used by an attacker to reveal the voter's voting decision.
  - **Recorded-as-Cast:** The voter can retrace that his ballot is correctly transmitted and stored in the ballot box. In order to protect receipt-freeness, recorded-as-cast has to be implemented such that the voter cannot prove his voting decision (to others). The verification of recorded-as-cast can take place directly after the ballot casting or after the voting phase.
  - **Counted-as-Recorded:** The voter as well as the broad public can verify the election result. I.e. it has to be verifiable that all ballots from the ballot box are counted-as-recorded.

Verifiability as described above is one way to assure system security and thus, to strengthen the trust of the voters in the electronic voting system. Another way is the certification of the electronic voting system. As discussed in Volkmann, Schryen, Langer, Schmidt, and Buchmann (2009), these approaches are not exclusive but should be combined in order to obtain the best possible result. Note that verifiability is a specific security function, which is subject to certification itself.

The Common Criteria for Information Technology Security Evaluation (CC) (Common Criteria, 2006) are a common standard for the evaluation and certification of IT-systems while the Common Evaluation Methodology (CEM) (Common Criteria, 2006) specifies the framework for the evaluation of such systems. An advantage of the CC is the structured and systematic specification of security requirements for an IT system in terms of Protection Profiles or Security Targets.
A Protection Profile (PP) specifies a product-independent description of security requirements in a group of products, e.g. online voting products in general. A Security Target (ST) represents a product-specific and thus, implementation-dependent specification of security requirements for a specific product, e.g. an implemented online voting system. The CC allows to evaluate (and then, to certify) a PP or the implemented security functions of a product against a specified ST. The ST may refer to a certified PP.

A PP is evaluated with respect to its completeness and consistence. It has to be proven that the identified threats, assumptions and security policies are completely covered by the security objectives of the system or its operational environment. Furthermore, the CC define seven Evaluation Assurance Level (EAL). These EAL indicate the evaluation depth from EAL 1 'functionally tested' to EAL 7 'formally verified and tested'. They are ordered hierarchically such that each EAL represents more assurance than all lower EALs. The increase in assurance is not only achieved by an increasing evaluation depth but also by an increasing scope of the evaluation, i.e. which documents are required and included in the evaluation (from software and documentation to a formal model). For instance, the CC Protection Profile 0037 (Volkamer & Vogt, 2008) defines a basic set of security requirements for online voting products and requires an evaluation according to EAL 2+ (structural tested). EAL 2+ seems to be sufficient for non-parliamentary elections. However, parliamentary elections demand an evaluation depth of EAL 5 or higher. A CC evaluation according to EAL 5 or higher claims for the application of formal methods (Common Criteria, 2006). It requires a formal proof that the security functions enforce the security properties (Mantel, Stephan, Ullmann, & Vogt, 2004).

In this paper a formal security model is provided. This security model covers the security objectives of Fairness, Eligibility, Secrecy and Receipt-Freeness, Verifiability and Protection against Precipitation. It helps to clarify and better understand these security objectives but can also be used for the evaluation of voting products against a CC Protection Profile according to EAL 5 or higher.

This paper is organized as follows: In the next section related work is discussed. In the section following, the formal basics are explained which are needed to formalize the given security objectives. In the fourth section the formal security model is presented. For better readability, the formalism in the third section and the section following is enriched with examples. In terms of the CC, the formal model in this paper addresses the PP and thus, the implementation-independent requirements. Due to the implementation-independent and universal character of the PP, the examples are chosen from different products/protocols. We will close in the final section with conclusions and future work.

Note, that the formal model in this paper is an aggregation of the formal sub models from Grimm, Hupf, and Volkamer (2010), Bräunlich and Grimm (2011) and Bräunlich and Grimm (2013). However, the synthesis of the formal sub models into the superior formal model in this paper is not simply a summary of previous work. Indeed the synthesis has caused changes in the security functions which enforce the security requirements (see [Rule 7] and [Rule 8] in definition 23). An additional proof is included that the given security objectives are realizable, i.e. that they are conflict-free (see the fourth section). Furthermore, this paper completes the proof that the given security objectives are enforceable (see fourth section).

RELATED WORK

The focus of this paper is the application of formal methods for the analysis and verification of electronic voting systems, especially in terms of a CC evaluation according to EAL 5 or higher. Thus, in the following we concentrate on related work concerning the formalization and verification of security requirements for electronic voting systems.
According to Mantel, Stephan, Ullmann, and Vogt (2004), it has to be proven in the course of a CC evaluation that

1. The security properties are conflict-free and thus, realizable,
2. The security functions enforce the security properties and
3. The evaluated product conforms to the formal model.

Compliance of the product with the formal model (3.) is verified by analyzing the product’s implementation of the security functions. In a state-oriented formalization these security functions are specified by a set of rules for allowed state transitions. In the course of a CC evaluation it then can be shown that the system to be evaluated implements the state transition rules in an adequate manner. Thus, a state-oriented formalism is better suitable for a CC product evaluation than a process-oriented formalism which does not explicitly specify the security functions by means of state transition rules.

Thus, related work will be analyzed according to the following criteria:

- Is the formalisation generic and thus, specifies implementation-independent requirements?
- Does the formalisation provide state transition rules which enable a CC evaluation?
- Which of the security objectives of Fairness, Eligibility, Secrecy and Receipt-Freeness, Verifiability and Protection against precipitation are already covered by the formalisation?

There exist several works concerning the definition and interpretation of receipt-freeness in the literature on electronic voting. Instead of many, here we refer to Langer, Volkamer, and Buchmann (2009). Furthermore, existing literature provides many publications about the development and/or analysis of concrete receipt-free electronic voting protocol as for example Moran and Naor (2006), Aditya, Lee, Boyd, and Dawson (2004), Bohli, Müller-Quadem and Röhrich (2007), Juels, Catalano, and Jakobsson (2005).

Also, there exists a number of works concerning the formal specification of privacy properties as for example Delaune, Kremer, and Ryan (2009), Jonker and Pang (2006), Jonker and de Vink (2006), Langer (2010). In Delaune, Kremer, and Ryan (2009), vote-privacy, receipt-freeness and coercion-resistance are formalized in the applied pi calculus by means of observational equivalence. In Jonker and Pieters (2006), a formal definition of receipt-freeness in epistemic logic by means of observational equivalence is given. Both approaches are process-oriented and thus, not suitable for a CC evaluation. In Langer (2010), a state-oriented view is given by describing security properties of privacy and receipt-freeness. A different, state-oriented formalization of receipts and receipt-freeness is provided in Jonker and de Vink (2006) by characterizing properties of receipts. Both approaches do not provide any functional rules. However, the definition of receipt-freeness in this paper is based on the latter (Jonker & de Vink, 2006).

The concept of verifiability or the development of voter verifiable voting schemes is subject to many publications in the academic literature as for example Adida and Neff (2006), Bohli, Müller-Quade, and Röhrich (2007), Benaloh (1987), Chaum, Ryan, and Schneider (2005), Popov and Hosp (2010), Ryan and Peacock (2005). However, only few works address the formalization of verifiability.

In Baskar, Ramanujam, and Suresh (2007) and Talbi, Morin, Tong, Bouhoula, and Mejri (2008) a formalization of individual and universal verifiability is applied to a concrete protocol, namely the protocol by Fujioka et al. (1992). But these two works do not provide a generic approach to verify security properties in the sense of a CC evaluation.

In Jonker and Pang (2011), a bulletin board for electronic voting systems is modelled. This formal framework is used to specify privacy issues like coercion-resistance and forced vote spoiling but it does not address verifiability.
Kremer, Ryan, and Smyth (2010) presents a formalization of verifiability in the applied pi calculus and thus, in a process-oriented formalism. Therefore, it is well suited for protocol analysis, but unsuitable for a CC evaluation.

Kremer and Ryan (2005) gives a definition of fairness and eligibility formalized in the applied pi calculus. Equally, Backes, Hritcu, and Maffei (2008) presents a formalization of eligibility, vote integrity and non-reusability in the applied pi calculus. Due to the process-oriented formalism, both are not suitable for a CC evaluation.

Novotny (2009) and Villafiorita, Wellemar, and Tiella (2009) formally define security properties that are applied to concrete electronic voting protocols. Thus, these works do not provide a generic approach to verify security properties in the sense of a CC evaluation.

To the best of our knowledge, there does not exist a formal model that satisfies all of the criteria above (Table 1). As mentioned above, the formal model in this paper is based on Grimm, Hupf, and Volkamer (2010), Bräunlich and Grimm (2011) and Bräunlich and Grimm (2013) and covers all of the security objectives of Fairness (One Voter, One Vote), Eligibility, Receipt-Freeness, Verifiability and protection against errors by haste. By choosing a state-oriented formalism, the formal model in this paper is suitable for the evaluation and certification of electronic voting products against the CC according to EAL5 or higher. The state transition rules in the formal model (see fourth section) represent the security functions to be evaluated in an electronic voting product.

**FORMAL BASICS**

The formalization in this paper requires some formal basics which are described in this section. This section is mainly a summary of the corresponding parts in Grimm, Hupf, and Volkamer (2010), Bräunlich and Grimm (2011) and Bräunlich and Grimm (2013).

<table>
<thead>
<tr>
<th></th>
<th>Generality</th>
<th>State-oriented Formalization with Transition Rules</th>
<th>Fairness</th>
<th>Eligibility</th>
<th>Receipt-Freeness</th>
<th>Verifiability</th>
<th>Protection against precipitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Delaune, Kremer, &amp; Ryan, 2009)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Jonker &amp; Pang, 2006)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Jonker &amp; de Vink, 2006)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Langer, 2010)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Baskar, Ramanujam, &amp; Suresh, 2007)</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>(Talbi et al., 2008)</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>(Jonker &amp; Pang, 2011)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Kremer, Ryan, &amp; Smyth, 2010)</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Kremer &amp; Ryan, 2005)</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Backes, Hritcu, &amp; Maffei, 2008)</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(Novotny, 2009)</td>
<td>-</td>
<td>-</td>
<td>+ (±)</td>
<td>+</td>
<td>(±)</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Agents and Their Communication

Let Agents be the set of all communication partners. In the context of electronic voting, eligible voters, attackers or the members of the election authority can be part of Agents. They communicate with each other by exchanging messages. Messages denote the set of all messages like, for example, startElection, stopElection or castBallot.

According to the Dolev-Yao model (Dolev & Yao, 1981), we assume that an attacker has full control of the public network. Thus, we differentiate between messages that are sent via public or private channels. The attacker can read any message which is exchanged via a public channel, as, for example, a plaintext message which is transmitted over an insecure network. However, private channels are not under the control of the attacker. The attacker cannot read privately exchanged messages as, for example, an encrypted message which is transmitted between the voting server and the voting client. The sending and receiving of a message is called an event.

Definition 1 (Events): Let Channel : = \{private, public\}. Then Events denotes the set of all possible events and is defined as:

\[\text{Events} \subseteq \pm \text{Messages} \times \text{Agents} \times \text{Agents} \times \text{Channel}\]

Let \(a, b \in \text{Agents}\) and \(m \in \text{Messages}\). A negative sign of a message \(m\) indicates that the associated event is being sent; a positive sign indicates the reception of an event. For instance, \(a(m:b)_{\text{pub, priv}}\) represents the event that \(a\) receives a message \(m\) from \(b\) via a public or a private channel. Accordingly, \(a(-m:b)_{\text{pub, priv}}\) specifies the event that \(a\) sends a message \(m\) to \(b\) via a public or a private channel.

Definition 2 (Projection): Let \(1 \leq k \leq n\). Then \(\pi_k\) denotes the set-theoretic projection of a Cartesian product of \(n\) sets on its \(k\)-th component.

Let \(x \in \text{Events}\). Then \(\pi_1(x)\) returns the message \(m\) of the associated event, \(\pi_2(x)\) returns \(a\) who is the sender of the message \(m\) if a negative sign is associated with \(m\) or the recipient otherwise, \(\pi_3(x)\) returns \(b\) who is the sender of the message \(m\) if a positive sign is associated with \(m\) or the recipient otherwise, and \(\pi_4(x)\) returns the type of the communication channel.

For the synchronization of events, we need a way to express that a message is observed by both sides. We assume that communication partners \(a, b, ..., \in \text{Agents}\) observe events that are correlated by a communication function \(\text{com}\) as defined below.

**Definition 3 (com):** Let Events be the set of all possible events, \(a, b \in \text{Agents}\), \(ch \in \{\text{public, private}\}\) and \(m \in \text{Messages}\), then we define:

\[
\text{com} : \text{Events} \rightarrow \text{Events}
\]

\[
\text{com}(a(m:b)^cb) := b(-m,a)^cb
\]

\[
\text{com}(a(-m:b)^cb) := b(m,a)^cb
\]

The function \(\text{com}\) is defined as in (Grimm, 2009) and maps the sending and receiving of a message on the corresponding event on the partner’s side.

Definition 4 (Lists): Let \(\mathcal{L}_q := \{ (q, \in Q) \}_{j \in \{1, \ldots, n\}} | k \in N \) be the set of all possible lists (= tuples) whose elements are members of the set \(Q\).

In this paper, we use lists to collect events. This enables the sequential operation on events and the unique identification of events within this list. Let \(L\) be a list of elements of a set \(Q\), \(q \in Q\) and \(n \in N\). Then we will use the following functions on lists:

- \(\text{set}(L)\) denotes the (unordered) set that consists of all elements of \(L\).
- \(L || q\) appends the element \(q\) at the end of the list \(L\).
- \(\text{head}(L)\) returns the last element of \(L\).
- \(\text{tail}(L)\) returns the rest of the list \(L\) without the last element.
• \( \text{tail}(L) \) is complementary to \( \text{tail} \) and returns the remaining list \( L \) without the first element of \( L \).
• \( \text{del}(L, i) \) returns a list where the \( i \)-th and all succeeding elements from \( L \) are removed.
• \( |L| \) returns the number of elements in \( L \).
• \( \text{pos}(L, q) \) returns the position of the last occurrence of the element \( q \) in the list \( L \).
• Let \( \text{filter} \) be a function on a list \( L \) of events and an agent \( a \in \text{Agents} \) that returns the list of all events in \( L \) that are sent or received by agent \( a \).
• Let \( \text{rmv} \) be a function on a list \( L \) of events and an agent \( a \in \text{Agents} \) that returns the list \( L \) where all events from \( L \) are removed which are sent to or received by \( a \).

Let \( op \) be a function on a list, as for example the appending of an element, and which returns another list. Then the \( k \)-times repetition of this operation is denoted as follows

\[
\text{op}^k(L) = \text{op}_k((\text{op}_{k-1}(\ldots(\text{op}_1(L))\ldots))
\]

**Reset and Cancel**

Protection against precipitation is a basic legal requirement well established in private and public law \[\text{[Ba06]}\]. It demands that a voter can abort his voting process at any time prior to the final casting of the vote without losing his right to vote and there is no limit on the number of corrections a voter can make to his vote until the final casting of the vote (see the first section). The abortion and/or correction of a voting process protect not only against precipitation, but they also protect the secrecy of voting against unwanted external events like the appearance of another person during the voting process. Thus, it is important for the support of the freedom of vote. In order to express the security objective of Protection against Precipitation we will propose two functions \( \text{reset} \) and \( \text{cancel} \) of a voting process. While the abortion of a voting process is correlated with the execution of \( \text{cancel} \), the correction of a vote is associated with the execution of \( \text{reset} \).

Let \( a, b \in \text{Agents} \). We assume in the following that \( a \) communicates with \( b \) only, while \( b \) communicates with \( a \) and other partners as well. In this formal model, \( a \) represents a voting client and \( b \) represents the voting server.

**Definition 5 (\text{reset})** Let \( a, b \in \text{Agents} \) and \( X_i \) be the list of events on the side of communication partner \( a \), i.e., \( \forall x \in X_i : \pi_a(x) = a \). Let \( Y_j \) denote the list of events on the side of communication partner \( b \). Let \( \text{sent}(X_i) \) denote the list of events that contains the send-events of \( X_i \) only, and let \( \text{received}(Y_j) \) denote the list of events that contains the received-events of \( Y_j \) only. Then we define as presented in Box 1.

By executing \( \text{reset} \), a voter can go back exactly one step in his voting process. Or more formally, if a communication partner \( a \) executes a \( \text{reset} \) then the last event \( x_i \in X_i \) which is sent

\[
\text{Box 1.}
\]

\[
X_i \parallel a(-\text{reset}: b) := \begin{cases} X_0 & \text{if } \text{set}(\text{sent}(X_i)) = \emptyset \\
\text{del}(X_i, l) & \text{else, where } l = \max\{n \in N \mid x_n \in \text{set}(\text{sent}(X_i))\} 
\end{cases}
\]

\[
Y_j \parallel b(\text{reset}: a) := \begin{cases} \text{del}(Y_j, k) \parallel \text{rmv}(\text{tail}^k(Y_j), a) & \text{if } C_2 \\
\text{rmv}(Y_j, a) & \text{else} 
\end{cases}
\]

with \( C_2 = \exists k > 0 : k = \max\{n \in N \mid y_n \in \text{set}(\text{received}(\text{filter}(Y_j, a)))\} \)
by a and all successive events to x, are deleted. If there is no event in X that is sent by a (i.e., X is empty or contains only received events), then X is set to its initial state.

If the voting server receives a voter’s reset, the voting server must undo the last step of this particular voter while the voting processes of the other voters must be kept by the voting server. Or more formally, if a communication partner b receives a reset then the last event y_j ∈ Y which is received by b from a is deleted as well as all successive events to y_j which are sent to or received from a by b. Note that all events successive to y_j which are exchanged with other communication partners are preserved in the state of communication partner b. If there is no event in Y that is received from a by b (i.e. Y_j is empty, doesn’t contain any events exchanged with a or contains only events sent to a), then all messages sent by b to a are deleted from the list Y_j, i.e. b is set to initial state with respect to a. All events that are exchanged with different communication partners are preserved.

**Definition 6 (Cancel):** Let a, b ∈ Agents and X_j be the list of events on the side of communication partner a, i.e., ∀x ∈ X_j : π(x) = a. And let Y_j be the list of events on the side of communication partner b, respectively. Then we define:

\[ X_j \parallel a(\text{-cancel : } b) := X_0 \]

\[ Y_j \parallel b(\text{cancel : } a) := \text{rmv}(Y_j, a) \]

With the execution of cancel a voter can restart his individual voting process. Then a is set back to his initial state with an empty event list X_j. On receiving a cancel message from a voter, the voting server must forget all events stimulated by this voter, but keep all events stimulated by other voters. Then all events sent to or received from a by b are eliminated from his event list.

According to definition 6 the following holds: Let k := \# sent(X_j) + 1 be one more than the number of all send-events in the list of events on the side of a, and let l := \# received(filter(Y_j, a)) + 1 be one more than all events that b has received from a, then

\[ X_j \parallel a(\text{-cancel : } b) = \]

\[ X_0 \parallel b(\text{-reset : } a) = \text{rmv}(Y_j, a) \]

The execution of cancel by a communication partner a can be expressed by the means of reset. A voter can repeat reset events back to the initial state, so that he can restart his individual voting process. Or more formally, a executes a(\text{-reset : } b) for each event sent by him, until there are no events left or only events that are received by a. By executing an additional reset, a is set to its initial state with empty event list X_j.

The execution of cancel on the partner’s side can be specified by the means of reset as well. The voting server b receives a reset for each event received from a. The remaining events are all either sent from b to a or are messages exchanged with other communication partners different from a. The remaining events sent to a are deleted by the execution of an additional reset.

The sending and receiving of a reset and cancel message must be carefully synchronized between voters and their voting server. Therefore, we assume that the system is available and works correctly. Thus, we assume secure communication channels in the following sense:

(SecCom1)

\[ \exists i \geq 0 : x \in \text{set}(X_j) \iff \exists j \geq 0 : \text{com}(x) \in \text{set}(Y_j) \]

iff a communication partner a exchanges a message x with b then there exists a state such that this message is observable on the partner’s side.
(SecCom2)
\[ \forall i > 0 : \forall x_n, x_m \in set(sent(X_i)) : pos(X_i, x_n) < pos(X_i, x_m) \iff \forall j \geq i : pos(Y_j, com(x_n)) < pos(Y_j, com(x_m)) \]

If a communication partner a sends two messages in a particular order then the communication partner b receives them in exactly that order.

SecCom1 can be realized by a confirmed message exchange service. SecCom2 can be realized by a confirmed message exchange service with consecutive message ordering numbers.

Theorem 1 (Synchronization Property of Reset): In a secure communication environment (i.e. SecCom1 and SecCom2 hold) the sending and receiving of reset events are well synchronized. Formally: \[ com(head(sent(X_i) \upharpoonright [a(-reset:b)))) = head(received(select_i(Y_j) \upharpoonright b(reset:a), a))) \]

The proof of this theorem is given in appendix I of this paper.

It has to be ensured that reset must not create or delete voting rights or casted votes. This is incorporated into our formal model by state transition rule [Rule 4] (see fourth section).

State of the Electronic Voting System

We now define the voting specific formalism. Let \( W_{total} \) be the set of all registered and eligible voters, where \( W_{total} \subseteq Agents \) holds. Remark that each member of \( W_{total} \) or \( Agents \) is considered as a potential attacker. Furthermore, let \( V \) be the set of all plaintext votes, representing the voting decisions of the voters. And let \( B \) be the set of all ballots, i.e. encrypted votes stored in the ballot box. We assume that the votes as well as the ballots are unique.

In this paper, we do not describe verifiability as an abstract property of a voting system. Instead we select a bulletin board as a concrete verification mechanism and specify verifiability as a security property of the bulletin board. A bulletin board is a public channel like, for example, a website (Benaloh, 1987). Information (e.g. ballots) is published on the bulletin board such that the voter and the public can verify the election. As the authors in Jonker and Pang (2011) we focus on the communication between the bulletin board and its communication partners and do not specify its internal behaviour. The bulletin board uses lists to collect the received messages. We assume that all messages received by the bulletin board are automatically published on the bulletin board and only authorized members of the election authority can send messages to the bulletin board. Once a message is published on the bulletin board, it cannot be deleted nor altered. The bulletin board is part of the electronic voting system. Thus, its formal definition is included in the definition of the state of the electronic voting system below.

Definition 7 (State of the Electronic Voting System): An electronic voting system is driven by events \( e_1, e_2, e_3, \ldots \) that carry the system through the states \( S_1, S_2, S_3, \ldots \). The state of the remote electronic voting system is a six-tuple \( S_i = (L_i, B_i, V_i, W_i, BB_i, Receipts_i) \) where

- \( L_i \) denotes the list of events \( e_1, e_2, \ldots, e_i \in Events \), observed by the remote electronic voting system.

\( L_i \in L_{Events} \) is defined as,
\[
L_i := \begin{cases} 
\emptyset & \text{if } i = 0 \\
(e_1, \ldots, e_i) & \text{otherwise} 
\end{cases}
\]

Initially, \( L_0 \) is the empty list. In the succeeding state, \( L_i \) contains the very event \( e_i \) which carried from state \( S_i \) to state \( S_{i+1} \). The event list \( L_i \) contains all events in the order they occurred.
• \( B \) denotes the set of the ballots (encrypted votes) in the ballot box.
• \( V \) denotes the set of plaintext votes which are used by the election authority for the tallying process.
• \( W \) denotes the set of eligible voters, i.e., the voters who have not cast their ballot, yet.
• \( BB := \{ \) denotes the list of messages that are published on the bulletin board with \( BB \in \mathcal{L}_{\text{message}} \). If the bulletin board receives a message via an event, the corresponding message will be appended to the end of the list. Or more formally: Let \( \pi := \text{head}(L) \) be the last event which carried to state \( S \) and let \( m = \pi(e) \) be the corresponding message that was exchanged by event \( e \). Then

\[
BB := \begin{cases} 
\varnothing & \text{if } t = 0 \\
BB \cup m & \text{if } (\pi, e) = BB \wedge \text{sign}(m) > 0 \\
BB & \text{otherwise} 
\end{cases}
\]

• \( \text{Receipts} \) denotes the set of exchanged messages and thus, potential receipts, according to definition 14 below.

The initial state is
\[
S_0 := (L_0 = \varnothing, B_0 = \varnothing, V_0 = \varnothing, W_0 = W^_{\text{total}}, BB_0 = \varnothing, \text{Receipts}_0 = \varnothing).
\]

**Authentification and Ballot Casting**

We now define the terms that identify a voter uniquely (cf. Jonker & de Vink, 2006).

**Definition 8 (.\( \mathcal{A} \)):** Let \( a \in W^_{\text{total}} \) and Messages\((a)\) the set of all messages a voter \( a \) can send because it contains a unique function only \( a \) is capable of, like a PIN or a message that is signed by a voter with his private key (digital signature). Then the set of authentification terms \( \mathcal{A}(a) \) for each voter is defined, such that

\[
\mathcal{A}(a) = \{ m \in \text{Messages}(a) | \forall b : a \neq b \Rightarrow m \notin \text{Messages}(b) \text{ and } a \text{ cannot repudiate the origin of } m \}
\]

Note that non-repudiation is not solely a technical property, but it depends on a socio-technical infrastructure like a trusted key distribution centre or a trusted public-key infrastructure.

Furthermore, we define the set of all authentification terms of all registered voters as:

\[
\mathcal{A} = \{ \mathcal{A}(a) | a \in W^_{\text{total}} \}
\]

**Lemma 1:** \( \mathcal{A} = \bigcup_{a \in W^}_{\text{total}} \mathcal{A}(a) \)

Lemma 1 states, that the set of all authentification terms \( \mathcal{A} \) is the direct sum of the authentification terms for each eligible voter. I.e., \( \mathcal{A} \) is the union of authentification terms for each eligible voter whereas the sets of authentification terms for each eligible voter are disjoint. According to definition 8, for all \( a, b \in \text{Agents} \) and all \( m \in \text{Messages} \):

\[
a \neq b \wedge m \in \mathcal{A}(a) \\
\Rightarrow m \notin \text{Messages}(b) \cup \mathcal{A}(b).
\]

This in turn implies that \( m \notin \mathcal{A}(b) \) holds as well.

**Definition 9 (auth):**

\[
\text{auth}(t) = \begin{cases} 
a & \text{if } \exists a \in W^_{\text{total}} : t \in \mathcal{A}(a) \\
\varepsilon & \text{otherwise, i.e. if } t = \varepsilon
\end{cases}
\]

\( \text{auth} \) returns the unique voter that created an authentification term \( t \). \( \text{auth} \) returns \( \varepsilon \) if the argument \( t \notin \mathcal{A} \), i.e., if \( t = \varepsilon \). The empty word \( \varepsilon \) is part of the domain of \( \text{auth} \) and is mapped to \( \varepsilon \). This is needed because \( \text{auth} \) will be composed with (applied to) other functions which might reveal \( \varepsilon \) as a value (see below).

**Lemma 2:** \( \text{auth} \) is well-defined.
It has to be proven, that \( \text{auth} \) maps every element \( t \in \mathcal{AT} \cup \{\varepsilon\} \) onto exactly one element \( a \in W_{\text{total}} \cup \{\varepsilon\} \). According to definition 9 \( \text{auth}(\varepsilon) = \varepsilon \). Let \( t \in \mathcal{AT} \). According to lemma 1, \( \mathcal{AT} \) is the direct sum of the authentication terms for each eligible voter. Thus, \( \exists a' \in W_{\text{total}} : t = \mathcal{AT}(a') \). Since the sets of authentication terms for each eligible voter are disjoint, \( \text{auth} \) is right-unique. Therefore, \( \text{auth} \) is well-defined.

\( \text{auth} \) is only constructible for concrete electronic voting protocols. For example, if the authentication of a voting protocol is based on PIN-authentication, then \( \text{auth} \) corresponds to the comparison of a received PIN with a look-up-table. If the authentication is based on electronic signatures, then the authentication term \( t \) consists of the sender’s public key \( pk \) and a message \( m \), signed by the sender with his private key. In this case, \( \text{auth} \) complies with a cryptographic proof (that \( m \) was signed with the secret key appending to \( pk \)) and the identification of the sender by means of his public key \( pk \).

**Definition 10 (choice):**

\[
\text{choice} : W_{\text{total}} \cup \{\varepsilon\} \rightarrow V \cup \{\varepsilon\}
\]

\[
\text{choice}(a) = \begin{cases} 
   v & \text{if } a \text{ is the vote of voter } a \\
   \varepsilon & \text{if } a = \varepsilon \text{ or } a \text{ has not voted yet}
\end{cases}
\]

\( \text{choice} \) specifies how the voters vote. As soon as the voter has marked the voting form, \( \text{choice} \) returns the voter’s voting decision \( v \). Otherwise, it returns \( \varepsilon \).

In order to ensure vote secrecy, in practice, the function \( \text{choice} \) has to be restricted such that it is not available to any other person than the voter of this vote himself.

**Definition 11 (Encrypt, Decrypt):** Let \( \text{Params} \) be the set of all (public and private) keys.

\[
\begin{align*}
\text{encrypt} : & \quad V \times \text{Params} \rightarrow B \\
\text{decrypt} : & \quad B \times \text{Params} \rightarrow V
\end{align*}
\]

\( \text{encrypt} \) models the encryption of the votes and maps a plaintext vote \( v \in V \) and an encryption key \( p \) onto a ballot \( b \in B \). \( \text{decrypt} \) models the decryption of the ballots. To ensure ballot secrecy, we demand that all ballots have the same size, i.e., the ballots do not allow any conclusions about the number of marks and/or their positions on the vote. Furthermore, we assume that

\[
\text{decrypt} \circ \text{encrypt} = \text{id}_V
\]

and

\[
\text{encrypt} \circ \text{decrypt} = \text{id}_B
\]

hold for a proper selection of the parameters. In addition, we assume that without knowledge of the proper \( \text{Params} \) value, it is unfeasible to construct \( \text{encrypt} \) or \( \text{decrypt} \). We also assume that there exists exactly one parameter \( p \in \text{Params} \) which enables the decryption of a ballot.

In case of postal voting, \( \text{encrypt} \) corresponds to the inner envelope which hides the voter’s voting decision and which does not contain any link to the voter.

**Definition 12 (ballot):** Presented in Box 2.

\( \text{ballot} \) associates a voter with his ballot. If the voter has made his voting decision

**Box 2.**

\[
\begin{align*}
\text{ballot} : & \quad W_{\text{total}} \rightarrow B \cup \{\varepsilon\} \\
\text{ballot}(a) = & \begin{cases} 
   b & \text{if } \text{choice}(a) = \varepsilon \land \exists p \in \text{Params} : b = \text{encrypt}(	ext{choice}(a), p) \\
   \varepsilon & \text{otherwise}
\end{cases}
\end{align*}
\]

Copyright © 2013, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
Definition 13 (voter):

\[ \text{voter} : B \to W_{\text{total}} \cup \{\varepsilon\} \]
\[ \text{voter}(b) = \begin{cases} a & \text{if } b = \text{ballot}(a) \land b \not\in B_i \\ \varepsilon & \text{otherwise} \end{cases} \]

\(\text{voter}\) maps a ballot onto its producer. In order to ensure ballot secrecy, \(\text{voter}\) has to be restricted. Only ballots that are not yet cast into the ballot box (\(b \not\in B_i\)) can be mapped onto their producers. Otherwise, \(\text{voter}\) returns \(\varepsilon\). Thus, \(\text{voter}\) can only be applied to the ballot which is produced during the active state transition, namely \(b \in B_i \setminus B_i\). This complies with traditional paper-based elections, where ballots from the ballot box are not linkable to their producers.

Verification and Tallying

In the following we will define the set of messages that can potentially constitute a receipt.

Definition 14 (Receipts): Let \(\text{pubMsg}\) be a function on a list \(L\) of events that returns the list of all events in \(L\) that are exchanged via public channels. Then we define:

\[ \text{Receipts}_{\text{public}} := \text{set}(\text{pubMsg}(L)) \]
\[ \text{Receipts}_{\text{priv},a} := \text{set}(\text{filter}(L,a)) \]

Note that \(\text{Receipts}\) specify the set of messages that can potentially, but do not necessarily constitute a proof of the voter's voting decision. The property of receipt-freeness will be defined by definition 19 later in the fourth section as a secure state property.

Definition 15 (verifyEncryption): For every agent \(a \in \text{Agents}\) let (presented in Box 3), \(\text{verifyEncryption}_a\) represents the verification that a ballot encodes the voting decision correctly. To guarantee ballot secrecy, only the producer of a ballot can verify the encryption of his vote. In case that the voter \(a\) himself invokes \(\text{verifyEncryption}_a\), then it returns true if the ballot encodes the voter's voting decision correctly. Otherwise, \(\text{verifyEncryption}_a\) returns false. If the function is invoked by a person who is not the producer of that particular ballot (\(a \neq \text{voter}(b)\)), \(\text{verifyEncryption}_a\) returns \(\varepsilon\). In order to ensure receipt-freeness, \(\text{verifyEncryption}_a\) does not return the plaintext vote.
Though, implemented by an electronic voting system it must enable the voter to ascertain himself that the ballot is correctly encrypted. In case of Bingo Voting (Bohli, Müller-Quade & Röhrich, 2007) verifyEncrption_n complies with the check if the number shown by the random number generator is equivalent to the number on the ballot that is associated with the chosen candidate or party, respectively.

verifyEncryption_n is a mathematical function which returns a value independently from the moment of its invocation. Implemented in an electronic voting system the invocation of verifyEncryption_n has to be restricted, in order to protect the voter from coercion.

\[
\text{invoke}_{\text{verifyEncryption}}_n(S_i, b) \begin{cases} 
\text{if } (b \in B_i) \text{ then return } e; \\
\text{else return verifyEncryption}(b); 
\end{cases}
\]

If verifyEncryption_n is invoked in a state \(S_i\), in which a ballot \(b\) is already stored in the ballot box (\(b \in B_i\)), \(e\) is returned. If the ballot is not yet stored in the ballot box, verifyEncryption_n is successfully applied. As soon as a ballot is stored in the ballot box it can no longer be verified with respect to cast-as-intended. This complies with traditional paper-based elections where a ballot can only be unfolded and checked by the voter as long as it is not put into the ballot box.

**Definition 16 (tally):** Let \(k \in N\) be the number of choices (candidates or parties). And let \(2^k = P(V)\) be the power set of \(V\), i.e. the set of subsets of plaintext votes. Then tally is defined as follows:

\[
tally : 2^k \rightarrow N_0^k
\]

tally returns for a set of plaintext votes the number of marks for each choice.

In analogy to verifyEncryption, (see Definition 15), the access to the tally-function has to be controlled. I.e. implemented in an electronic voting system, it has to be restricted when tally can successfully be executed.

\[
\text{invoke}_{\text{tally}}(S_i) \begin{cases} 
\text{if } (B_i \models V_i) \text{ then return tally}(V_i); \\
\text{else return } e.
\end{cases}
\]

This access control ensures completeness of the tally. Completeness means that all ballots which are cast into the ballot box are actually used to calculate the election result. Thus, tally can only be invoked if all ballots from the ballot box are decrypted at that particular moment (\(B_i \models V_i\)). Otherwise, \(e\) is returned, which means that no election result is obtained. Note, that invoke_tally allows for intermediate results. If a legal system prohibits the calculation of intermediate results, the invocation of tally must be further restricted such that it can only be invoked after the official closing of the voting phase.

**FORMAL SECURITY MODEL**

In this section the formal sub models from (Grimm & Hupf & Volkamer, 2010), (Bräunlich & Grimm, 2011) and (Bräunlich & Grimm, 2013) are merged into one superior formal model. According to Grimm (2008), the formal security model in this work represents the security requirements in terms of secure states. The definition of secure states is not a plain of these formal sub models. The synthesis of these formal sub models has caused changes in the security functions. More precisely, the state transition rules concerning the enforcement of receipt-freeness (see [Rule 7] and [Rule 8] in definition 24) had to be adapted. Moreover, in order to complete the formal security model, the security theorem proves that the secure state properties are (a) realizable and (b) enforceable. The first part of the proof is a new contribution given in this work. Furthermore, the proof that the secure state properties are enforceable is completed in this paper.
Secure States

Now the security objectives Fairness, Eligibility, Verifiability and Secrecy and Receipt-Freeness are specified as secure states. Protection against Precipitation will not be met by a secure state property but by a state transition rule (see Rule 4 in definition 23).

For better readability, we present the secure state properties for each security objective one by one (see definitions 17-19 below). An electronic voting system is secure if and only if all security properties specified by 17-19 are fulfilled.

Note, that we now define secure states, but not (yet) their enforcement. Enforcement of state’s security will be supported by the allowed state transitions later in this paper.

Definition 17 (Fairness, Eligibility): A state \( S_i \) is a secure state with respect to Fairness and Eligibility if all of the following constraints hold:

\[
\begin{align*}
(A1) & \quad \forall b, b' \in B_i : \text{voter}(b) = \text{voter}(b') \Rightarrow b = b' \\
(A2) & \quad \forall w \in W_{\text{total}} \setminus W_i : \exists b \in B_i : \text{voter}(b) = w \\
(A3) & \quad \forall b \in B_i : \text{voter}(b) \in W_{\text{total}} \setminus W_i \\
(A4) & \quad W_i \subseteq W_{\text{total}} \\
(A5) & \quad \text{voter}(B_i) \subseteq W_{\text{total}}
\end{align*}
\]

According to (A1) no voter can cast more than one ballot into the ballot box. I.e. there is at most one ballot for each voter in the ballot box. (A2) states that a voter can loose his eligibility only by casting a ballot into the ballot box. I.e. if a registered voter is not eligible anymore, then he has already cast a ballot into the ballot box. And vice versa (A3) demands, that the ballot box only contains ballots from voters who are not eligible anymore. That means that by casting a ballot into the ballot box a voter looses necessarily his eligibility.

(A4) addresses the eligibility and requires that only registered voters \( w \in W_{\text{total}} \) can cast a vote \( w \in W_i \). According to (A5) the ballot box only contains ballots from registered voters.

The five properties above are equivalent to the two following properties:

\[
\begin{align*}
[S1] & \quad \text{voter is an injective function} \\
[S2] & \quad W_{\text{total}} = W_i + \text{voter}(B_i)
\end{align*}
\]

\([S1]\) is equivalent to (A1) and \([S2]\) is equivalent to the other four properties \((A2)-(A5)\) above. Thereby, ‘+’ denotes the direct sum and means that both holds, \(W_i \cup \text{voter}(B_i) = W_{\text{total}}\) and \(W_i \cap \text{voter}(B_i) = \emptyset\).

Verifiability is a kind of “meta”-function which ensures the integrity of the election from beginning to end. In the context of electronic voting, it cannot be detached from ballot secrecy. Verifiability has to be provided without the violation of ballot secrecy. In this paper, we do not describe verifiability as an abstract property of a voting system. Instead we select a bulletin board as a concrete verification mechanism and specify verifiability as a security property of the bulletin board. In practice, information (e.g. ballots or plaintext votes) are published on a public channel as a bulletin board. To preserve ballot secrecy, only some parts of the data are published, while others are kept secret. By all means, it has to be ensured that the plaintext vote cannot be mapped onto its producer/voter (directly or indirectly/transitively). Nonetheless, the logical chain of the published data has to provide full coverage of the voting procedure and it has to exclude undetected manipulations with a probability bordering on certainty.

Definition 18 (Verifiability): Let \( \text{getBallot} \) be a function that extracts all ballots from a set of messages and \( \text{getVote} \) be a function that extracts all plaintext votes from a set of messages. These functions are used to filter information from the bulletin board. Then a state \( S_i \)
is a secure state with respect to verifiability iff all of the following constraints hold:

[S3] \( \forall b \in B_i \cdot \exists j \leq i \cdot \)
\[
\text{invoke\_verifyEncryption}_{\text{عت}(S_j, b)} = \text{true}
\]

[S4] \( B_i = \text{getBallot}(\text{set}(BB_i)) \)

[S5] \( V_i \subseteq \text{decrypt}(B_i) \)

[S6] \( V_i = \text{getVote}(\text{set}(BB_i)) \)

[S7] \( \forall i, j \in N \cdot b \in \text{getBallot}(\text{set}(BB_i)) \Rightarrow b \in \text{getBallot}(\text{set}(BB_{i+j})) \)

[S8] \( \text{tally}((\text{decrypt}(B_i)) = \text{tally}((\text{getVote}(\text{set}(BB_i))) \)

[S3] addresses the integrity of the ballot box. Thus, a state is secure with respect to verifiability only if all ballots from the ballot box (\( \forall b \in B_i \)) in some previous state (\( S_j \) with \( j \leq i \)) have been verifiable by their producers with respect to the encryption of the votes

\( \text{invoke\_verifyEncryption}_{\text{عت}(S_j, b)} = \text{true} \).

[S4] addresses the integrity and completeness of the ballots published on the bulletin board. It states that all ballots from the ballot box are published on the bulletin board. And vice versa, that no ballots are inserted nor manipulated on the bulletin board.

[S5] addresses the integrity of the decryption process and thus, the integrity of the plaintext votes used for the tally. It denotes that \( V_i \) is constructed only from encrypted votes from the ballot box \( B_i \). Thus, no vote stuffing takes place. It allows, however, that some ballots are not yet decrypted, but not vice versa.

[S6] addresses the integrity and completeness of the plaintext votes published on the bulletin board. It states that all plaintext votes correspond to votes from the bulletin board and the other way round, all published plaintext votes are actually contained in the set of plaintext votes. Thus, no plaintext vote has been deleted, inserted or manipulated.

[S7] ensures that once a ballot is published on the bulletin board it cannot be removed. Note that this constraint does not address the ballots in the ballot box directly. This definition takes into account that the content of the bulletin board is publicly accessible while the content of the ballot box may not. However, in a secure state according to constraint [S5] the set of ballots in the ballot box equals the set of ballots published on the bulletin board.

[S8] addresses the completeness of the tally. i.e. the result obtained from the ballots which are cast into the ballot box equals the result obtained from the plaintext votes which are published on the bulletin board.

Note that the constraints [S3] - [S6] do not guarantee the completeness of the tally. Without further restrictions, ballots which have been cast into the ballot box could be deleted and thus, incomplete election results could be obtained. Nonetheless, the electronic voting system would remain in a secure state according to [S3] - [S6], as long as the corresponding ballot would be deleted from the bulletin board as well. Therefore, [S7] requires that once a ballot is published on the bulletin board, it cannot be deleted. Furthermore, [S3] - [S7] alone would allow possible that ballots from the ballot box are not included in the tally. Therefore, [S8] demands that all ballots from the ballot box are actually counted for the election result. A state is secure with respect to verifiability if and only if all constraints [S3] - [S8] hold.

**Lemma 3:** Let \( S_i \) be a secure state, then

\( \text{blind}(V_i) \subseteq B_i \) holds as well.

According to definition 17 above, \( V_i \subseteq \text{unblind}(B_i) \) holds. Furthermore,
(encrypt o decrypt)(B_i) = B_i holds due to definition 11. Thus, lemma 3 holds as well.

The definition of receipts and receipt-freeness is based upon the properties of receipts from (Jonker & de Vink, 2006). In a state $S_i$, a receipt $r \in Receipts_i$ violates the property of receipt-freeness if for a voter $a \in W_{total}$ all three of the following “attack properties” (Q1), (Q2) and (Q3) are true. I.e. if the receipt $r$ identifies the voter $a$ unambiguously (Q1) and proves the voting decision of the voter $a$ (Q2) and proves that the voter $a$ has cast his ballot (Q3). Or more formally:

\begin{align*}
(Q1) & \quad getVoter(r) \in AT(a) \\
(Q2) & \quad getVote(r) = choice(a) \\
(Q3) & \quad getBallot(r) \in B_i
\end{align*}

Vote secrecy is given because once a ballot is cast into the ballot box there is no link back to either the voter or the vote. This leads to the following definition of secure states with respect to secrecy and receipt-freeness.

**Definition 19 (Secrecy and Receipt-Freeness):** A state $S_i$ is secure with respect to secrecy and receipt-freeness iff

\[ S_9 \quad \forall r \in Receipts_i : \text{getVote}(r) = \text{choice}(\text{auth}(\text{getVoter}(r))) \Rightarrow \text{getBallot}(r) \notin B_i \]

In a receipt-free state $S_i$ for all receipts $r \in Receipts_i$ holds: If the receipt $r$ establishes a link between a voter and his voting decision

\[ (\text{getVote}(r) = \text{choice}(\text{auth}(\text{getVoter}(r)))) \]

then this voter has not cast his ballot into the ballot box ($\text{getBallot}(r) \notin B_i$). Thus, a receipt $r$ either establishes the link between a voter and his voting decision or proves that he has cast his ballot into the ballot box, but not both at the same time.

Note that a state is secure if all constraints \([S1]-[S9]\) hold and that the initial state is secure with respect to the definition of secure states above.

**Allowed State Transitions**

The rules for allowed state transitions represent the system’s behaviour and thus, the security functions that need to be implemented by the electronic voting system. For a better readability, we present the rules for allowed state transitions one by one.

**Definition 20 ([Rule 1], no Ballot is Cast, no Ballot is Decrypted):**

\[ B_{i+1} = B_i \land V_{i+1} = V_i \land W_{i+1} = W_i \land BB_{i+1} = BB_i \]

This represents a state transition during which no ballot is cast into the ballot box nor decrypted. For example, this rule allows for the verification of the election or for the closing of the polling phase.

**Definition 21 ([Rule 2], One Ballot is Cast, no Ballot is Decrypted):**

\begin{align*}
V_{i+1} & = V_i \land (\exists b \in B_{i+1} : B_{i+1} = B_i \cup \{b\}) \\
\land BB_{i+1} & = BB_i \land b \land voter(b) \in W_i \\
\land W_{i+1} & = W_i \setminus \{voter(b)\} \\
\land \text{invokeVerifyEncryption}_{\text{voter}(b)}(S_i, b) & = \text{true}
\end{align*}

This rule models a state transition during which a ballot is cast into the ballot box. In order to ensure that each eligible voter can cast exactly one vote, only eligible voters ($voter(b) \in W_i$) can cast a ballot into the ballot box ($B_{i+1} = B_i \cup \{b\}$) and then are eliminated.
from the list of eligible voters \((W_{i+1} = W_i \setminus \{\text{voter}(b)\})\). In addition, a ballot can only be cast into the ballot box if the voter was able to verify the correct encryption of his vote \((\text{invoke\_verify\_Encryption}(\text{voter}(b), S, b) = \text{true})\). Furthermore, this ballot is published on the bulletin board \((BB_{i+1} = BB_i \parallel b)\).

In this formal model, ballot casting and its publication on the bulletin board are defined as a closed transaction. i.e. it is assumed that both, storage and publication of a ballot happen simultaneously during one atomic state transition. In practice, storage and publication of a ballot may be executed consecutively. However, then it has to be ensured that the execution of these two steps are performed directly one after the other and that they cannot be interrupted nor aborted.

**Definition 22 (Rule 3), no Ballot is Cast, one Ballot is Decrypted:**

\[
B_{i+1} = B_i \land \exists v \in V_{i+1} \setminus V_i : V_{i+1} = V_i \cup \{v\} \\
\land BB_{i+1} = BB_i \parallel v \land \exists p \in \text{Params} : \text{encrypt}(v, p) \in B_i
\]

Typically this rule is applied after the voting phase is closed. It represents a state transition during which a ballot is decrypted. The decrypted ballot is added to the set of plaintext votes \((V_{i+1} = V_i \cup \{v\})\) and published on the bulletin board \((BB_{i+1} = BB_i \parallel v)\). However, plaintext votes are only obtained from the ballot box \((\text{blind}(v, p) \in B_i)\). Moreover, because \text{encrypt} and \text{decrypt} are injective functions, a new ballot must be taken for every invocation of \text{decrypt}. Otherwise, the obtained plaintext vote \(v\) was already in \(V_i\) and thus, \(V_{i+1} \setminus V_i\) empty. Therefore, no votes are inserted, double counted or manipulated.

As well as the ballot casting, the decryption of a ballot and its publication are modelled as a closed transaction. If the decryption of a ballot and the publication of the corresponding plaintext vote are not realized simultaneously in an electronic voting system, it has to be ensured that once a ballot is decrypted its publication on the bulletin board cannot be interrupted nor aborted.

**Definition 23 (Rule 4), reset:** Let \(T_i\) be the list of events observed by the voting client \(a\) before the execution of \(\text{reset}\), and let \(T_{i+1}\) be the list of events observed by \(a\) after the execution of \(\text{reset}\), and let \(l := T_{i+1}\) be the length of the list \(T_{i+1}\). Let \(T = \text{tail}(T_i)\) be the list of reverted events. Furthermore, let \(\text{permitted}(S_i \xrightarrow{b \circ a \circ \text{reset}} S_{i+1})\) denote a permitted state transition according to definition 25. Then for the execution of \(\text{reset}\), or more formally \(l_{i+1} = a(-\text{reset}; b)^{\text{end}, \text{prev}}\), the following holds:

\[
\{ a \in W_i \cap W_{i+1} \} \land \forall 1 \leq j \leq T : \text{permitted}(S_i \xrightarrow{r_{i+1} \circ \text{reset}} S_{i+1})
\]

This rule represents a state transition during which a voter reverts a number of \(|T|\) state transitions. In order to avoid multiple ballot casting, the voter has not yet cast his vote, both, before and after the execution of \(\text{reset}\) \((a \in W_i \cap W_{i+1})\). Furthermore, all reverted state transitions had to be permitted as well.

In addition to the state transition rules above, the following state transition rules \([\text{Rule 5]} - [\text{Rule 8}]\) are needed to enforce receipt-freeness. \([\text{Rule 5]} - [\text{Rule 8}]\) are defined such that the system remains in a receipt-free state according to definition 19. i.e. at least one of the attack properties \((Q1), (Q2)\) or \((Q3)\) gets false.

The point of the secure state property of receipt-freeness is that it must hold for all receipts \(r \in \text{Receipts}\). In practice, no state transition is able to enforce receipt-freeness during an active state transition for all receipts that have been produced in the course of an
election. However, it is indeed feasible to check receipt-freeness with exactly the one receipt \( r \in \text{Receipts}_{i+1} \setminus \text{Receipts}_i \) that is produced during the active state transition. Thus, receipt-freeness holds for all receipts \( r \in \text{Receipts}_{i+1} \), if only it is enforced with the receipts one by one during their production. The formal proof of that is given with the security theorem.

**Definition 24 ([Rule 5]–[Rule 8], Enforcement of Receipt-Freeness):** Presented in Box 4. [Rule 5] represents a state transition during which no new receipt is generated.

If a state transition takes place according to [Rule 6] then there is no further restricting rule. According to this state transition rule, the current receipt allows observers to read the vote and identify the voter, but the ballot is not cast into the ballot box. This rule allows for correcting and reverting a vote.

[Rule 7] allows the generation of a receipt \( r \) which can be mapped onto a ballot from the ballot box. However, in this case the new receipt \( r \) can either be mapped onto its voter or onto its corresponding vote, but not onto both, voter and vote. For instance, the public can observe a voter to cast an encrypted vote into the ballot box but cannot read the vote or the public can monitor the decryption process but cannot link ballots or votes to their producers. [Rule 7] holds for ballots that are already in the ballot box (\( \text{getBallot}(r) \in B_i \cup B_{i+1} \)) and for ballots that are part of the active transition (\( \text{getBallot}(r) \in B_{i+1} \setminus B_i \)).

Note that in this paper there is no distinction between public and private receipts as in (Bränulich & Grimm, 2011). This formal model here is based on the assumption that every agent can potentially read every message. Thus, the formal model in this paper is more restrictive with respect to receipt-freeness than the original formal model from (Bränulich & Grimm, 2011). However, the rules for allowed state transitions which enforce the receipt-freeness become simpler. Or more precisely, the “old” rules (RF2) and (RF4) can be merged to a “new” rule [Rule 7].

The integration of the formal sub models from (Grimm & Hupf & Volkamer, 2010; Bränulich & Grimm, 2011; Bränulich & Grimm, 2013) necessitate the introduction of the additional state transition rule [Rule 8]. The formal sub model of verifiability in (Bränulich & Grimm, 2013) distinguishes between ballots and (plaintext) votes, while the formal sub model of receipt-freeness from (Bränulich & Grimm, 2011) does not make this differentiation. This has to be taken into account while merging these formal sub models. Thus, [Rule 8] enforces receipt-freeness during the decryption process. If the receipt \( r \) of the current state transition refers to a casted plaintext vote (\( \text{getVote}(r) \in V_{i+1} \)), then the receipt does not identify the corresponding voter (\( \text{getVoter}(r) = \varepsilon \)). I.e. the link between ballot and (plaintext) vote has to be eliminated during the decryption process. This complies with traditional postal voting where the outer envelope identifies the voter and is used to check

---

**Box 4. Definition 24**

\[
\begin{align*}
\text{[Rule 5]} & \quad \text{Receipts}_{i+1} = \text{Receipts}_i \\
\text{[Rule 6]} & \quad \exists r : \{ r \} = \text{Receipts}_{i+1} \setminus \text{Receipts}_i \land \text{getBallot}(r) \notin B_{i+1} \\
\text{[Rule 7]} & \quad \exists r : \{ r \} = \text{Receipts}_{i+1} \setminus \text{Receipts}_i \land \text{getBallot}(r) \in B_{i+1} \land \\
& \quad \left( \text{getVoter}(r) = \varepsilon \lor \text{getVote}(r) = \varepsilon \right) \\
\text{[Rule 8]} & \quad \exists r : \{ r \} = \text{Receipts}_{i+1} \setminus \text{Receipts}_i \land \text{getVote}(r) \in V_{i+1} \land \text{getVoter}(r) = \varepsilon
\end{align*}
\]

Copyright © 2013, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
the voter’s eligibility. Then the outer envelope is removed and the ballot (=plaintext vote in blank, inner envelope) is cast into the ballot box. During this step, the link between the ballot and its voter (and in consequence the link between the plaintext vote and its voter) is destroyed.

Note that the secure state property of secrecy and receipt-freeness (see definition 19) does not have to be adapted. All plaintext votes are obtained from ballots in the ballot box (as formally specified by \(S5\) in definition 18). Thus, for all plaintext votes \(Q2\) and \(Q3\) are true. Then necessarily \(Q1\) has to be false. This case is already covered by the definition of receipt-freeness in definition 19.

While \([\text{Rule 5}]-[\text{Rule 8}]\) enforce receipt-freeness, \([\text{Rule 1}]-[\text{Rule 4}]\) control the execution of the voting procedure. Thus, \([\text{Rule 1}]-[\text{Rule 4}]\) and \([\text{Rule 5}]-[\text{Rule 8}]\) have to be combined as specified in the following definition of permitted state transitions in order to guarantee all security objectives.

**Definition 25 (Permitted):** A state transition from state \(S_i\) to state \(S_{i+1}\) triggered by event \(t_{i+1}\) is permitted,

\[
\text{permitted}(S_i \xrightarrow{t_{i+1}} S_{i+1}),
\]

if one of the following combinations of rules is true:

- \([\text{Rule 1}]\) and \([\text{Rule 5}]\) or \([\text{Rule 6}]\),
- \([\text{Rule 2}]\) and \([\text{Rule 7}]\),
- \([\text{Rule 3}]\) and \([\text{Rule 8}]\), or
- \([\text{Rule 4}]\) and \([\text{Rule 5}]\) or \([\text{Rule 6}]\).

Table 2 summarizes the possibilities how the state transition rules can be combined in order to present a permitted state transition.

Not all possible combinations are permitted. For instance, \([\text{Rule 1}]\) cannot be applied with \([\text{Rule 7}]\) or \([\text{Rule 8}]\). \([\text{Rule 7}]\) implies that a new ballot was cast into the ballot box during the active state transition. This directly contradicts \([\text{Rule 1}]\) \(B_{i+1} = B_i\). \([\text{Rule 8}]\) implies that a new plaintext vote was obtained which also contradicts \([\text{Rule 1}]\) \(V_{i+1} = V_i\).

**Security Theorem**

In order to enable the implementation of an electronic voting system in compliance with the formal model, the security properties of the formal model have to be (a) realizable (i.e. they have to be conflict-free) and (b) enforceable (i.e. there exist state transition rules which preserve the secure state properties).

The latter will be covered by the security theorem (see theorem 2 below). The conflict-freeness of the secure state properties will be discussed now.

Two constraints for secure states are conflict-free

1. If these two constraints address different objects and thus, cannot contradict each other, or
2. If the fulfilment of these two constraints are independent from each other and thus, these two constraints do not interfere.

The secure state properties \([S4]\) and \([S6]\) are an example for the first constraint. \([S4]\) demands the equality between the ballots in the

<table>
<thead>
<tr>
<th></th>
<th>[Rule 5]</th>
<th>[Rule 6]</th>
<th>[Rule 7]</th>
<th>[Rule 8]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Rule 1]</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>[Rule 2]</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>[Rule 3]</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>[Rule 4]</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Copyright © 2013, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
ballot box and on the bulletin board (\( B_i = \text{getBallot(set(BB_i)}) \)), while security property \([S6]\) requires the equality between (plaintext) votes in \( V_i \) and on the bulletin board (\( \text{getVote(set(BB_i)}) \)). Thus, the security properties \([S4]\) and \([S6]\) address different objects and do not interfere. I.e. \([S4]\) and \([S6]\) are conflict-free.

The secure state properties \([S5]\) and \([S6]\) are an example for the second constraint. Both security properties address one common object, namely \( V_i \). While \([S5]\) demands that all votes in \( V_i \) are obtained from the ballot box \( B_i \) only, \([S6]\) requires that all votes in \( V_i \) are published on the bulletin board. The fulfillment of one of these constraints does not affect the fulfillment of the other. Thus, there is no conflict between \([S5]\) and \([S6]\).

Table 3 summarizes the relationship between the secure state properties \([S1]-[S9]\) and states a reason for the conflict-freeness according to the two options above (different objects (1) or independence (2)). The conflict-relation is commutative and irreflexive. These entries are shaded gray in Table 3.

Table 3 shows that the secure state properties \([S1]-[S9]\) do not contradict themselves and thus, are realizable in an electronic voting system.

It still has to be proven that these secure state properties are enforceable. This proof will be given with the security theorem. The security theorem presents the formal proof that a permitted state transition always carries a secure state into another secure state and thus, that the secure state properties are enforceable in an electronic voting system.

**Theorem 2 (Security Theorem):** If the state \( S_i \) of a voting system is secure and if the state transition \( t_{i+1} \) is permitted, then the succeeding state \( S_{i+1} \) is also secure.

**Proof:** A state is secure, if all constraints \([S1]-[S9]\) as specified by definitions 17-19 are fulfilled.

A state transition is permitted if it follows a combination of \([\text{Rule 1}]-[\text{Rule 4}]\) and \([\text{Rule 5}]-[\text{Rule 7}]\) as specified in definition 25. Let \( S_i \) be secure and \( t_{i+1} \) be permitted, then it has to be proven that \( S_{i+1} \) is secure and thus, \([S1]-[S9]\) are fulfilled in \( S_{i+1} \).

1. Let \( t_{i+1} \) be a state transition according to \([\text{Rule 1}]\): Then no ballot is cast (\( B_{i+1} = B_i \)), no vote is obtained (\( V_{i+1} = V_i \)) nor is any information published on the bulletin board.

---

**Table 3. Summary of the conflict-relation between the security properties \([S1]-[S9]\)**

<table>
<thead>
<tr>
<th></th>
<th>([S1])</th>
<th>([S2])</th>
<th>([S3])</th>
<th>([S4])</th>
<th>([S5])</th>
<th>([S6])</th>
<th>([S7])</th>
<th>([S8])</th>
<th>([S9])</th>
</tr>
</thead>
<tbody>
<tr>
<td>([S1])</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S2])</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S3])</td>
<td>(1)</td>
<td>(1)</td>
<td>(2)</td>
<td>(1)</td>
<td>(1)</td>
<td>(2)</td>
<td>(2)</td>
<td>(2)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S4])</td>
<td></td>
<td>(2)</td>
<td>(2)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S5])</td>
<td></td>
<td></td>
<td>(2)</td>
<td>(1)</td>
<td>(2)</td>
<td>(2)</td>
<td>(2)</td>
<td>(2)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S6])</td>
<td></td>
<td></td>
<td></td>
<td>(2)</td>
<td>(1)</td>
<td>(2)</td>
<td>(2)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S7])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1)</td>
<td>(2)</td>
<td>(2)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S8])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
<td>(1)</td>
</tr>
<tr>
<td>([S9])</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(1)</td>
</tr>
</tbody>
</table>
Thus, \( S_{i+1} \) inherits the security properties \([S1]-[S8]\) from \( S_i \). [Rule 1] has to be applied in combination with [Rule 5] or [Rule 6].

a. If it is applied in combination with [Rule 5], then no new receipt is produced during the active state transition. Because \([S9]\) was true in \( S_i \), \( S_{i+1} \) inherits the property of receipt-freeness \([S9]\) from \( S_i \).

b. If it is applied in combination with [Rule 6], then exactly one new receipt \( r \) is produced during the active state transition. Because \( S_i \) was receipt-free, it is sufficient to prove that the new receipt \( r \) does not violate the secure state property \([S9]\). \( r \) does not refer to a ballot which is actually cast into the ballot box. Thus, attack property \([Q3]\) gets falsified and state \( S_{i+1} \) is receipt-free according to \([S9]\).

2. Let \( t_{i+1} \) be a state transition according to [Rule 2]. Then the voting phase is still active and exactly one ballot \( b \) is cast into the ballot box and published on the bulletin board during state transition \( t_{i+1} \). [Rule 2] preserves the secure state properties \([S1]\) and \([S2]\) as proven in (made anonymous for the reviewing process, 2010) and the secure state properties \([S3]-[S8]\) as shown in (made anonymous for the reviewing process, 2013). It still has to be proven that the property of receipt-freeness is preserved. [Rule 2] is applied in combination with [Rule 7]. According to [Rule 7], exactly one new receipt \( r \) is produced during the active state transition. Because \( S_i \) was receipt-free, it is sufficient to prove that this new receipt \( r \) does not violate the secure state property \([S9]\). According to [Rule 7], \( r \) may refer to a ballot from the ballot box. But in this case, \( r \) does either reveal the voter or the voting decision, but not both on the same time. Thus, one of the attack properties \((Q1\text{-}\text{reveal\ voter})\) or \((Q2\text{-}\text{reveal voting decision})\) gets falsified and the state \( S_{i+1} \) is receipt-free according to \([S9]\).

3. Let \( S_i \) be secure and \( t_{i+1} \) be a state transition according to [Rule 3]. Then exactly one ballot is decrypted and the obtained vote \( v \) is published on the bulletin board during state transition \( t_{i+1} \). [Rule 3] preserves the secure state properties \([S3]-[S8]\) as shown in (made anonymous for the reviewing process, 2013). It still has to be proven that the properties \([S1]\), \([S2]\) and \([S9]\) are preserved. voter maps a ballot on the voter of this particular ballot. Because \( S_i \) was secure, voter was injective on \( B_i \). According to [Rule 3] no ballot was cast during the active state transition \( (B_{i+1} = B_i) \). Thus, voter is injective on \( B_{i+1} \) as well. According to [Rule 3] neither the ballot box, the set of eligible voters nor the set of registered voters is changed. Thus, \( S_{i+1} \) simply inherits the security property \([S2]\) from \( S_i \).

[Rule 3] is always applied in combination with [Rule 8]. According to [Rule 8], exactly one new receipt \( r \) is produced during the active state transition. Because \( S_i \) was receipt-free, it is sufficient to prove that this new receipt \( r \) does not violate the secure state property \([S9]\). According to [Rule 8], \( r \) may refer to a casted plaintext vote but does not identify the corresponding voter. Thus \((Q1\text{-}\text{reveal\ voter})\) gets falsified and the state \( S_{i+1} \) is receipt-free according to \([S9]\).

4. Let \( S_i \) be secure and \( t_{i+1} \) be a state transition according to [Rule 4]. The following Lemma helps to facilitate the proof that [Rule 4] preserves the secure state properties.
Lemma 4: If a state $S_i$ is secure and permitted $\langle S_{i-1}, t_{i-1}, S_i \rangle$, then $S_{i-1}$ was a secure state.

The proof of Lemma 4 is given in appendix 2 of this paper.

Note that due to this Lemma it is not possible that a permitted state transition transfers an insecure state into a secure state. Thus, the formal model in this work does not allow to fix an insecure system state by means of an allowed state transition. As a consequence, an insecure system state can only be fixed by means of an illegal state transition. For example, let there be two ballots from the same voter in the ballot box in state $S_i$. Thus, state $S_i$ is insecure because $[\{S1\}]$ does not hold in $S_i$ (see $[A11]$ in definition 17). This insecure situation can only be fixed by removing these two (or optional only one of these two) ballots from the ballot box and the bulletin board. In case that the decryption process has already been started, the corresponding plaintext votes have to be removed from the ballot box and bulletin board as well. Even though this state transition results in a secure state, it does not follow one of the rules for permitted

Given Lemma 4, the proof of the security theorem with respect to $[\text{Rule 4}]$ is trivial: If $t_{i+1}$ follows $[\text{Rule 4}]$ and $S_i$ was secure, then all reverted state transitions were permitted according to $[\text{Rule 4}]$, and hence $S_{i+1}$ is a secure state according to the Lemma 4 above.

This completes the proof that a permitted state transition always carries a secure state into another secure state.

CONCLUSION AND FUTURE WORK

In this paper a formal IT security model with respect to the requirements of Fairness, Eligibility, Secrecy and Receipt-Freeness, Verifiability and Protection against Precipitation is specified. The security properties are specified by means of secure states and enforced by rules for allowed state transitions. The security theorem proves that the security properties of the formal model are (a) realizable, i.e. conflict-free, and (b) enforceable, i.e. by following the rules for allowed state transitions the system remains in secure states.

The formal IT security model in this paper can be seen as a step towards the evaluation of online voting products against the CC according to EAL5 or higher. The CC evaluation according to EAL5 or higher demands the application of formal methods. According to Mantel, Stephan, Ullmann, and Vogt (2004), this evaluation depth includes the verification that (1.) the security requirements are realizable, (2.) the security functions enforce the security requirements and (3.) the electronic voting system complies with the formal model. The first and second parts of the proof are presented in this paper (see fourth section). The third part of the proof addresses the compliance of a concrete electronic voting product with the formal model. This can be (manually or automatically) done by analyzing an electronic voting system whether or not it implements the security functions specified in our formal model by the rules for allowed state transitions. Thus, our next research step will be a proof of concept by conducting a case study concerning a concrete voting protocol.

ACKNOWLEDGMENT

The research of this paper is funded by the Deutsche Forschungsgemeinschaft (DFG) under the project 'Modellierung von Internetwahlen II' (ModlWa II).

REFERENCES


Fujikawa, A., Okamoto, T., & Ohta, K. (1992). A practical secret voting scheme for large scale elections (pp. 244-251). AUSCRYPT.


Katharina Brümlich is a Research Fellow at the Research Group 'IT Risk Management' in the Department of Information Systems Research at the University of Koblenz-Landau. She conducts research on foundations and applications of IT security, with a focus on electronic voting. Currently she works on the project 'Modellierung von Internetwahlen II' (ModlWa II) which is funded by the Deutsche Forschungsgemeinschaft (DFG).

Rüdiger Grimm is Professor and head of the Research Group 'IT Risk Management' in the Department of Information Systems Research at the University of Koblenz-Landau. He teaches and conducts research on foundations and applications of IT security.
APPENDIX 1

Proof of Theorem 1

The following theorem has to be proven:

**Theorem 1 (Synchronisation Property of Reset):** In a secure communication environment (i.e. SecCom1 and SecCom2 hold) the sending and receiving of reset events are well synchronized. Formally: $\text{com}(\text{head}(\text{sent}(X_i \parallel a(-\text{reset}:b)))) = \text{head}(\text{received}(\text{filter}(Y_j \parallel b(\text{reset}:a),a)))$.

Proof:

Given the two assumptions (SecCom1) and (SecCom2). Furthermore, we denote $C_j$: $\text{set}(\text{sent}(X_i)) \neq \emptyset$, i.e. $a$ hasn’t sent anything so far, and $C_i$: $\text{set}(\text{received}(\text{filter}(Y_j,a))) \neq \emptyset$, i.e. $b$ hasn’t received any message from $a$. According to definition 5 of reset, the following four possibilities exist:

1. Neither $C_1$ nor $C_2$ holds.
   Then $X_i \parallel a(-\text{reset}:b) = \emptyset$ and $\text{filter}(Y_j \parallel b(\text{reset}:a)) = \emptyset$. Obviously, Theorem 1 is true.
2. $C_i$ does not hold, but $C_2$ holds.
   This directly contradicts assumption (SecCom1). If there is no message sent by communication partner $a$ then there can’t be any message received from $a$ by $b$.
3. $C_1$ holds and $C_2$ does not hold.
   This is a direct contradiction to (SecCom1) as well. If there is no message received by $b$ from $a$ then there can’t be any message sent from $a$ to $b$.
4. $C_1$ and $C_2$ hold
   Let $x_i$ be the last event sent by $a$ before executing reset. Due to (SecCom2) $\text{head}(\text{received}(\text{filter}(Y_j,a))) = \text{com}(x_i)$ holds. On the side of communication partner $a$, the event $x_i$ and all successive events to $x_i$ are eliminated during the execution of reset. On the side of communication partner $b$, the event $\text{com}(x_i)$ and all successive events to $\text{com}(x_i)$ that are exchanged with the communication partner $a$ are eliminated during the execution of reset. All events successive to event $x_i$ that are exchanged with different communication partners are preserved.

   If $\text{set}(\text{sent}(X_i \parallel a(-\text{reset}:b))) = \emptyset$ holds, then due to (SecCom1) $\text{set}(\text{received}(\text{filter}(Y_j \parallel b(\text{reset}:a),a))) = \emptyset$ holds as well. Thus, Theorem 1 holds.

   Assume $\text{sent}(X_i \parallel a(-\text{reset}:b)) \neq \emptyset$ and let $x_m = \text{head}(\text{sent}(X_i \parallel a(-\text{reset}:b)))$ be the last sent event after the execution of reset. Given precondition (SecCom1) there exists a state on the partner’s side such that $\text{com}(x_m) \in Y_j \parallel b(\text{reset}:a)$. Furthermore, in accordance to (SecCom2) $\text{com}(\text{head}(\text{sent}(X_i \parallel a(-\text{reset}:b)))) = \text{head}(\text{received}(\text{filter}(Y_j \parallel b(\text{reset}:a),a)))$ holds.
APPENDIX 2

Proof of Lemma 4

The following Lemma has to be proven:

**Lemma 4:** If a state $S_i$ is secure and permitted($S_{i-1} \xrightarrow{t_i} S_i$), then $S_{i-1}$ was a secure state.

Proof.

Note, that once a receipt is produced it cannot be eliminated. Or more formally, $\text{Receipts}_{i-1} \subseteq \text{Receipts}_i$. This simplifies the proof of this Lemma with respect to secrecy and receipt-freeness. If $S_i$ [S9] holds in $S_i$, then [S9] holds in $S_{i-1}$ as well. Therefore, the following proof is restricted to the secure state properties [S1]-[S8].

If $S_i$ is a secure state and $t_i$ was a permitted state transition, then the state transition $t_i$ was performed according to definition 25 which reduces the proof to the following three cases:

1. If $t_i$ was a state transition according to [Rule 1], then no ballot was cast ($B_{i-1} = B_i$), no ballot was decrypted ($V_{i-1} = V_i$) nor was any information published on the bulletin board ($BB_{i-1} = BB_i$). Because $S_i$ is secure, $S_{i-1}$ was secure as well (i.e. [S1]-[S8] hold).

2. If $t_i$ was a state transition according to [Rule 2], then the voting phase is still active and exactly one ballot $b$ was cast into the ballot box and published on the bulletin board during $t_i$. In (made anonymous for the reviewing process, 2010) it has already been proven, that [S1] and [S2] held in $S_{i-1}$. It still has to be proven that [S3]-[S8] held in $S_{i-1}$.

   a. Because $S_i$ is secure, [S3] holds in $S_i$. I.e. every ballot from the ballot box has been verifiable in some previous state with respect to its encrypting. Because $B_{i-1} \subseteq B_i$ holds, [S3] was true in $S_{i-1}$ as well.

   b. The secure state property [S4] states that all ballots from the ballot box are published on the bulletin board. According to [Rule 2], the following holds: $B_i = \text{getBallot(set}(BB_i)) = \text{getBallot(set}(BB_{i-1} \parallel b))$

   \[ = \text{getBallot(set}(BB_{i-1})) \cup \{b\} = B_{i-1} \cup \{b\} \]

   Given the last equality, [S4] obviously held in $S_{i-1}$ as well.

   c. If $t_i$ was a state transition according to [Rule 2], then at least one ballot $b$ from the ballot box is not yet decrypted, namely the one which was cast during state transition $t_i$. Thus, $V_i \subseteq \text{decrypt}(B_i)$ holds. According to [Rule 2], $V_i = V_{i-1} \subseteq \text{decrypt}(B_{i-1}) = \text{decrypt}(B_i \setminus \{b\})$. Thus, [S5] was true in $S_{i-1}$. 

   d. During state transition $t_i$ no ballot was decrypted, thus $V_i = V_{i-1}$ as well as \( \text{getVote(set}(BB_i)) = \text{getVote(set}(BB_{i-1})) \) hold and thus, [S6] was true in $S_{i-1}$.

   e. It has to be proven that all ballots which are published on the bulletin board in state $S_{i-1}$ are still on the bulletin board in state $S_i$. According to [Rule 2], \( \text{getBallot(set}(BB_i)) = \text{getBallot(set}(BB_{i-1})) \cup \{b\} \supset \text{getBallot(set}(BB_{i-1})) \). Thus, [S7] held in $S_{i-1}$.
f. In order to guarantee completeness of the tally, the tally-function returns a result only if all ballots from the ballot box are decrypted at that particular moment. Otherwise, the empty word is returned. Thus, the following two cases, have to be distinguished:

i. \( |B_{i-1} \models V_i | \) holds in \( S_i \). In this case \( |B_{i-1} \models B_i | - 1 = |B_i \models V_i | \models V_{i-1} | \) holds as well. Thus, tally returns the empty word and \( [S8] \) was true in \( S_{i-1} \).

ii. \( |B_i \models V_i | \) holds in \( S_i \). Since \( S_i \) is secure, \( V_i \subseteq decrypt(B_i) \) holds. Thus, \( |B_i \models |V_i | \). Then, there are again two possible options:

\[
|B_{i-1} \models |V_{i-1} | \text{ held such that } \text{tally returned the empty word and }[S8] \text{ was true in } S_{i-1}. \text{ Or,} \\
|B_{i-1} \models |V_{i-1} | \text{ held. Then } \text{tally}(decrypt(B_{i-1})) = \text{tally}(V_{i-1}) = \text{tally}(getVote(set(BB_{i-1}))).
\]

The first equality holds due to \([S5] \), the second equality due to \([S6] \) (which both have already been proven above).

3. Let \( S_i \) be secure. If \( t_i \) was a state transition according to \([\text{Rule 3}] \) then exactly one ballot was decrypted and published on the bulletin board.

Because \( S_i \) is a secure state, voter is injective on \( B_i \). According to \([\text{Rule 3}] \) \( B_{i-1} = B_i \).

Thus, voter was injective on \( B_{i-1} \) as well. I.e., \([S1] \) was true in \( S_i \).

According to \([\text{Rule 3}] \) neither the ballot box, the set of eligible voters nor the set of registered voters was changed. If \([S2] \) holds in \( S_i \), \([S2] \) was true in \( S_{i-1} \) as well.

According to \([\text{Rule 3}] \) no ballot was cast \( (B_{i-1} = B_i) \) nor published on the bulletin board \( (getBallot(set(BB_{i-1})) = getBallot(set(BB_i))) \) during \( t_i \). Because \( S_i \) is secure and thus, \([S3] \), \([S3] \) holds in \( S_i \), it held in \( S_{i-1} \) as well.

According to \([\text{Rule 3}] \) no ballot was cast into the ballot box during \( t_i \) \( (B_{i-1} = B_i) \). One plaintext vote \( v \) has been published on the bulletin board during \( t_i \) \( (getVote(set(BB_i)) = getVote(set(BB_{i-1}) \mid v)) \), but the ballots on the bulletin board remain unchanged \( getBallot(set(BB_{i-1})) = getBallot(set(BB_i)) \). Thus, if \([S4] \) holds in \( S_i \), \([S4] \) held in \( S_{i-1} \) as well.

Because \( S_i \) is secure, \( V_i \subseteq decrypt(B_i) \) holds. Thus, \( V_{i-1} = V_i \setminus \{v\} \subseteq decrypt(B_{i-1}) = decrypt(B_i) \) holds even more. Thus, \([S5] \) was true in \( S_{i-1} \).

The secure state property \([S6] \) states that all plaintext votes are actually published on the bulletin board. According to \([\text{Rule 3}] \), the following holds:

\[
V_i = getVote(set(BB_i)) = getVote(set(BB_{i-1}) \mid v)) = getVote(set(BB_{i-1}) \cup \{v\}) \cup \{v\} = V_{i-1} \cup \{v\}.
\]

Given the last equality, \([S6] \) obviously held in \( S_{i-1} \) as well.

It has to be proven that all ballots which are published on the bulletin board in state \( S_{i-1} \) are still on the bulletin board in state \( S_i \). According to \([\text{Rule 3}] \), \( getBallot(set(BB_{i-1})) = getBallot(set(BB_i)) \). Thus, \([S7] \) held in \( S_{i-1} \).

Let \( t_i \) be a state transition according to \([\text{Rule 3}] \). Then there are two possible options.
Either all ballots are decrypted in state $S_i$. Then $|B_i| = |V_i|$ holds. This in turn implies, that $|B_{i-1}| = |B_i| = |V_{i-1}| = |V_i| - 1$ holds. Thus, tally returns $\epsilon$ such that [S8] was satisfied in $S_{i-1}$.

Otherwise, $|B_i| = |V_i|$ hold in $S_i$. Since $S_i$ is secure, $V_i \subseteq decrypt(B_i)$ holds. Thus, $|B_i| > |V_i|$. Due to [Rule 3], $|B_{i-1}| = |B_i| > |V_{i-1}| = |V_i| - 1$ holds even more. Thus, tally returns $\epsilon$ such that [S8] was satisfied in $S_{i-1}$.

This completes the proof of Lemma 4.