Verifying System-level Security of a Smart Ballot Box

Dana Dghaym, Thai Son Hoang, Michael Butler, Runshan Hu, Leonardo Aniello, and Vladimiro Sassone

ECS, University of Southampton, Southampton, U.K.
{d.dghaym, t.s.hoang, m.j.butler, rs.hu, l.aniello, vsassone}@soton.ac.uk

Abstract. Event-B, a refinement-based formal modelling language, has traditionally focused on safety, but now increasingly finds a new role in developing secure systems. In this paper we take a fresh look at security and focus on what security means for the system rather than looking at detailed protocols. We use Event-B for proving security from an abstract view and refining it towards design details, focusing on the refinement of the availability property of the system. We define a general approach to guarantee the availability of events by ensuring the *non-strengthening* of their guards, taking into consideration their parameter types. We illustrate our approach using a smart ballot system, an integral part of modern voting systems.

Keywords: Event-B; Availability Property; System Security; Refinement; Voting System

1 Introduction

Event-B [1] is a refinement-based formal method for developing discrete transition systems. Data refinement is a standard technique in Event-B that requires relating the abstract variables with the concrete variables using gluing invariants. To ensure the refinement correctness, a guard strengthening (*GRD*) Proof Obligation (PO) is generated to verify that if a concrete event is enabled then its corresponding abstract event is also enabled.

In this paper, we investigate the application of refinement-based formal modelling in building a correct-by-construction secure system. Our focus is the refinement of the *availability property* of secure systems. The availability property of a refined event can be proved, if conditions under which event is enabled in an abstract machine are maintained in the refined machine.

We illustrate our approach using a smart ballot box case study. In this case study we build a secure system by gradually introducing the confidentiality and integrity properties of the voting system using encryption and message authentication respectively, while ensuring availability throughout the refinement process. In the smart ballot box system case study, confidentiality is ensured by having the voter's choices not visible to the system, while integrity is ensured by only accepting valid ballots and only rejecting invalid ballots. The availability of the system is guaranteed by not preventing a voter from casting a valid ballot. In this case study, we also model the intruder's behaviour and show how encryption and authentication can provide protection against an intruder behaviour.

We propose an extension to Event-B Proof Obligations (POs) that relates the enabledness of an event to availability of behaviour according to a new parameter type, which we call a *rigid* parameter. This PO will ensure the availability property of events is preserved by refinement.

The rest of the paper is structured as follows. Section 2 gives an overview of the smart ballot box case study. Section 3 introduces Event-B formal method. We propose a new PO to prove the availability of refined events in Section 4. In Section 5, we present our Event-B development of the case study across the different refinements. We discuss how model checking improved our development in Section 6. We compare our approach with other work in Section 7. Finally, we conclude and present our plans for future work in Section 8.

2 Case study: Smart Ballot Box

The main function of the Smart Ballot Box (SBB) [5] is to inspect a ballot paper by detecting a 2D barcode, decode it and evaluate if the decoded contents verifies the paper from a Ballot Marking Device (BMD). If the ballot is valid, then it can be cast into the storage box. Otherwise, the SBB rejects the paper, that will be ejected. The SBB does not conduct a full-scale analysis of the document, nor record the choices of the voters, nor tabulate the votes of the ballots it scans. The key function of the SBB is to ensure that only valid countable summary ballot documents that can be tabulated later are included in ballot boxes.

REQ 1 The BMD is used by a voter to make ballot choices and print a summary ballot. The ballot choices are not recorded by the BMD.

REQ 2 The barcode is created by the BMD as an authenticator so that the SBB can recognise a legitimate ballot.

REQ 3 The ballot barcode is formed of a timestamp followed by an encoding of the encrypted ballot and the Message Authentication Code (MAC).

REQ 4 All ballots with invalid or non-existent barcode are rejected by the SBB.

REQ 5 All ballots with a valid barcode are recognized by the SBB.

REQ 6 The user can decide whether to cast or spoil the given ballot. The ballot is deposited into the ballot box if the voter decides to cast their vote and subsequently all ballots with the same barcode will be considered invalid.

REQ 7 The user can decide to spoil a valid ballot. In such cases, the ballot is ejected and returned to the voter, and is subsequently considered invalid.

REQ 8 The ballot box shall reject a ballot with an expired barcode.

REQ 9 The authentication scheme is based on a MAC created with the AES standard encryption algorithm and a cryptographic key shared by the BMD and SBB. The SBB authenticates the ballot by recreating the MAC with the shared key and comparing the result with the MAC encoded in the barcode. If the two MAC values are equal the ballot is considered valid.

3 Background

Event-B [1] is a refinement-based formal method for system development. An Event-B model contains two parts: *contexts* for static data, and *machines* for dynamic behaviour specified by *variables* v, *invariant* predicates $I(v)$ that constrain the variables, and *events*. An event comprises a guard denoting its enabling condition and an action describing how the variables are modified when the event is executed. In general, an event e has the following form, where t are the event parameters, $G(t, v)$ is the event guard, and $v := E(t, v)$ is the action of the event.

$$\textbf{any}\ t\ \textbf{where}\ G(t,v)\ \textbf{then}\ v := E(t,v)\ \textbf{end}$$

Refinement in Event-B is reasoned event-wise where the behaviour of the concrete or refining machine conforms with the abstract machine. This is ensured by the refinement rules of guard strengthening and action simulation of the abstract events by their corresponding refining events. In addition to gluing invariants that relate the abstract and refined state of the models. In this case, given the abstract invariant $I(v)$ and gluing invariant $J(v,w)$ (where v and w are abstract and concrete variables, respectively), the GRD PO will check that the guards of the concrete event $H(q,v,w)$ are stronger than the guards of its corresponding abstract event $G(p,v)$ (where p and q are the abstract and concrete parameters, respectively) as follows:

$$I(v), J(v,w), H(q, v, w) \vdash \exists p \cdot G(p,v) \ ,$$

The GRD PO will ensure that if a concrete event is enabled, then its corresponding abstract event must be enabled. In Event-B, an event is enabled if there are parameter values that satisfy the guard of the event.

Event-B is supported by the Rodin Platform (Rodin) [2], an extensible toolkit which includes facilities for modelling, verifying the consistency of models using theorem proving and model checking techniques, and validating models with simulation-based approaches. In this paper we use both Event-B theorem proving and model checking using ProB [9] for the validation and verification of the SBB.

4 Rigid Events and Parameters

In this section, we first introduce the notion of enabledness with respect to a set of parameters (Section 4.1). Subsequently, we elaborate on the meaning of availability properties and introduce the notion of rigid events and parameters to capture availability properties (Section 4.2). We then address preservation of availability properties during refinement in Section 4.3.

4.1 Event Enabledness and Parameters

We extend the notion of event enabledness in Event-B to include the parameters. Given an event e with parameters p and q (both p and q can be a list of variables) and $\mathsf{G}(\mathsf{p}, \mathsf{q})$, the guard of e constraining p and q. We define $\mathsf{Enabled}_{\mathsf{p}}(\mathsf{e})$ as follows.

$$\mathsf{Enabled}_{\mathsf{p}}(\mathsf{e}) == \exists \mathsf{q} \cdot \mathsf{G}(\mathsf{p}, \mathsf{q})$$

In this definition, we explicitly specify the parameters in the enabledness condition to indicate their scope, i.e., the event e is enabled for all parameters p satisfying $\mathsf{Enabled}_{\mathsf{p}}(\mathsf{e})$. Notice that p appears freely in $\mathsf{Enabled}_{\mathsf{p}}(\mathsf{e})$.

4.2 Specifying Availability Properties with Rigid Events and Parameters

In Event-B, the availability of an event is determined by its enabledness condition, e.g., stating that *the event must be enabled under certain conditions*. As event guards can be strengthened during refinement, an event available in the abstraction might no longer be available in the refinement. As a result, we need a notation to specify which events' availability we are interested in, and treat them differently in the refinement. To signify that we are interested in the availability of event e, we say that event e is a '*rigid*' event, denoted as $[\mathsf{e}]$.

Of course, the availability of an event also needs to take into account its parameters. To indicate that we are interested in the availability of the event with respect to parameters p, we say that p are '*rigid*' parameters, denoted as $[\mathsf{p}]$. Note that only rigid events can have rigid parameters.

In general, consider a rigid event e with rigid parameters rp and other (non-rigid) parameters op of the following form:[1]

$$\begin{array}{l}\textbf{event}\ [\mathsf{e}] \\ \textbf{any}\ [\mathsf{rp}]\ \mathsf{op}\ \textbf{where}\ \mathsf{G}(\mathsf{rp}, \mathsf{op})\ \textbf{then} \ldots \textbf{end}\end{array}$$

The above rigid event means that the system satisfies the availability property stating that "event e must be enabled for any parameter rp satisfying $\mathsf{Enabled}_{\mathsf{rp}}$ (e)".

4.3 Refinement Preserving Availability Properties

We now discuss the refinement of rigid events preserving the associated availability properties. In a normal Event-B refinement, the guard of a refined event can be stronger than the abstract guard (and hence restrict the event enabledness). For a rigid event, on the other hand, we want to 'preserve' enabledness, that is do not strengthen in the refinement the conditions for event enabledness, so that the availability of the event is maintained by refinement.

[1] We use [ ] to distinguish rigid events and parameters from others.

The syntactic rules are (1) rigid events can only be refined by rigid events, and (2) the abstract rigid parameters must be retained in the concrete events.

Consider the following abstract rigid event [ae] and concrete rigid event [ce].

```
event [ae]
any [rp] oap where
  Ga(rp, oap, v)
then
  // abstract action
  ...
end
```

```
event [ce]
any [rp] ocp where
  Gc(rp, ocp, v, w)
then
  // concrete action
  ...
end
```

Here, [rp] represents the rigid parameters, oap and ocp are respectively the other abstract and concrete parameters. While v and w are the abstract and concrete variables.

To ensure that the availability properties are preserved through refinement, we must prove that the concrete event *does not* strengthen the enableness of the abstract event, taking into account the rigid parameters. We propose the following enabledness preservation PO (denoted as ENBL),

$$\mathsf{I}(\mathsf{v}), \mathsf{J}(\mathsf{v}, \mathsf{w}), \mathsf{Ga}(\mathsf{rp}, \mathsf{oap}, \mathsf{v}) \vdash \exists \mathsf{ocp} \cdot \mathsf{Gc}(\mathsf{rp}, \mathsf{ocp}, \mathsf{v}, \mathsf{w})\ ,$$

where $\mathsf{I}(\mathsf{v})$ and $\mathsf{J}(\mathsf{v}, \mathsf{w})$ are the abstract and concrete invariants.

In general, more rigid parameters can be introduced during refinement. Notice that the ENBL proof obligation depends on the abstract rigid parameters, i.e., any rigid parameters *newly introduced* by a refinement will be treated as non-rigid parameters for the purpose of the current ENBL PO and will only be relevant for the ENBL PO in further refinement steps.

In Event-B an abstract event can be refined by a group of concrete events. In this case, the ENBL PO is generalised accordingly. Given an abstract rigid event [ae] and concrete rigid events [ce_1], [ce_2], .., [ce_n], the ENBL PO is as follows.

$$\begin{array}{l}\forall\, \mathsf{rp}, \mathsf{oap} \cdot \mathsf{Ga}(\mathsf{rp}, \mathsf{oap}) \Rightarrow \\ \quad \bigvee_i \left(\exists\, \mathsf{ocp}_i \cdot \mathsf{Gc}_i(\mathsf{rp}, \mathsf{ocp}_i)\right)\end{array}$$

In the formula above ocp_i and Gc_i ($\mathsf{i} \in 1 \,..\, \mathsf{n}$) are the other concrete parameters and guards of the corresponding concrete event ce_i.

5 SBB Systems Model in Event-B

In this section, we first present our refinement strategy and then show how we modelled the SBB system introduced in Section 2 using Event-B. We also illustrate the reasoning about availability properties using enabledness conditions based on the approach of Section 4.

5.1 Refinement Strategy

Our refinement plan consists of an abstract and four refinement levels. Although our focus is modelling the SBB, it is important to take into consideration how it interacts with other components of the system, in particular the BMD which generates the encrypted ballots for authorised voters only.

Abstract Model: We start by modelling an ideal voting system, where legitimate ballots are created for voters who have not voted before and only legitimate ballots are cast.

First Refinement: In this refinement we introduce the physical paper ballots and we distinguish between the different types of ballots according to Figure 1. We model possible attackers behaviour, where an attacker can produce illegitimate ballots or duplicate legitimate ballots which can invalidate legitimate ballots. Voters with valid ballots will have the option to either cast or spoil their ballots. Ballots which are spoiled, invalid or illegitimate will be ejected by the SBB.

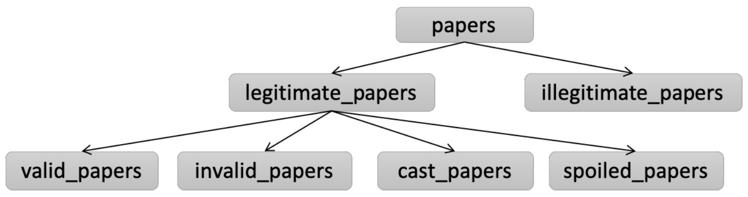

Fig. 1: Different types of paper ballots.

Second Refinement: We introduce time and invalidate ballots with expired timestamps. We assume synchronised clocks for both the SBB and BMD, but later we will show how time can be the subject of malicious attacks.

Third Refinement: We data refine the voter information and their votes by encrypting the ballots using a secret key.

Fourth Refinement: Ensuring the legitimacy of ballots is done through the Message Authentication Code (MAC), where we compare the computed MAC using the secret authentication key and the MAC in the ballot barcode.

5.2 Abstract Level: Modelling an Ideal Voting System

An ideal voting system is when each voter can have at most one vote. This is ensured by the typeof−ballots invariant, where ballots are defined as a partial function between VOTER and VOTE.

@typeof−ballots: $\mathsf{ballots} \in \mathsf{VOTER} \nrightarrow \mathsf{VOTE}$

Here, we only allow the creation of ballots for voters who were not issued any ballots previously. Casting of ballots is only applicable for ballots that are not invalidated, where the variable cast represents the set of cast ballots $\mathsf{cast} \subseteq \mathsf{ballots}$. Thus, we have a perfect secure voting system where we ensure that a voter can cast at most one valid ballot. At this level we only have three events create_ballot, invalidate_ballots and cast_ballot.

5.3 First Refinement: Introducing Physical Ballots and Possible Attacker Capabilities

At this level, we refine ballots and the cast ballots with the physical paper ballots (papers). These paper ballots are susceptible to attacks, e.g. duplicating paper ballots, making the voting system insecure. In this section we distinguish between the different types of papers using disjoint variables as shown in Figure 1. Ballot papers can be either legitimate or illegitimate, and legitimate papers can be in turn partitioned as valid, invalid, cast or spoiled which are modelled as disjoint sets as follows:

– legitimate_papers: Ballots created by BMD or copied from one created by BMD.
– illegitimate_papers: Ballots created by the attackers.
– valid_papers : Legitimate ballots that have not expired, and have not been cast or spoiled before.
– invalid_papers: Legitimate ballots but invalid either because expired or (a copy has been) already cast or spoiled.
– cast_papers: Legitimate and already cast.
– spoiled_papers: Legitimate ballots that are spoiled.

We define the events that lead to creating legitimate and illegitimate ballot papers and the events that invalidate legitimate ballots. All legitimate valid ballots are created by the BMD in the event BMD_issues_paper which refines the abstract event create_ballot. We also define three possible attacks: ATK_creates_paper creates an illegitimate ballot; in this case we assume the attacker does not know the secret keys. On the other hand, both attack events ATK_duplicates_valid_paper and ATK_duplicates_invalid_paper create legitimate ballots by duplicating existing valid and invalid ballots respectively; in this case the SBB will only accept one valid copy from the voter. The abstract event invalidate_ballots refers to the ballots that cannot be cast; here we refine it by two events papers_expired and spoil_valid_ballot where any ballots that expire or get spoiled cannot be cast anymore. The abstract event cast_ballot is refined by cast_paper where only valid_papers can be cast. We also introduce eject_paper where only invalid, spoiled or illegitimate ballots are ejected out of the SBB.

Given the new variables paper_voter and paper_vote which are projection functions on papers to represent the voter's information and choices, the following invariants should hold. Invariant no_valid_double_voting_vote ensures that if two valid ballots exist for the same voter, then they are copies of each other. On the other hand, no_cast_double_voting_vote ensures that once a valid ballot is cast, the voter cannot have any other valid ballot papers. Invariants gluing_ballots and gluing_cast are gluing invariants that relate the paper ballot with the logical ballots and cast variables.[2]

$$
\begin{array}{l}
@\mathsf{no\_valid\_double\_voting\_vote}: \\
\quad \forall \mathsf{b1}, \mathsf{b2} \cdot \mathsf{b1} \in \mathsf{valid\_papers} \land \mathsf{b2} \in \mathsf{valid\_papers} \land \mathsf{paper\_voter}(\mathsf{b1}) = \mathsf{paper\_voter}(\\
\qquad \mathsf{b2}) \\
\qquad\qquad \Rightarrow \mathsf{paper\_vote}(\mathsf{b1}) = \mathsf{paper\_vote}(\mathsf{b2})
\end{array}
$$

$$
\begin{array}{l}
@\mathsf{no\_cast\_double\_voting\_vote}: \\
\quad \forall \mathsf{b} \cdot \mathsf{b} \in \mathsf{cast\_papers} \Rightarrow \mathsf{paper\_voter}(\mathsf{b}) \notin \mathsf{paper\_voter}[\mathsf{valid\_papers}]
\end{array}
$$

$$
\begin{array}{l}
@\mathsf{gluing{-}ballots}: \\
\quad \mathsf{ballots} = \{\mathsf{paper} \cdot \mathsf{paper} \in \mathsf{valid\_papers} \cup \mathsf{cast\_papers} \\
\qquad\qquad \mid \mathsf{paper\_voter}(\mathsf{paper}) \mapsto \mathsf{paper\_vote}(\mathsf{paper})\}
\end{array}
$$

$$
\begin{array}{l}
@\mathsf{gluing{-}cast}: \\
\quad \mathsf{cast} = \{\mathsf{paper} \cdot \mathsf{paper} \in \mathsf{cast\_papers} \mid \mathsf{paper\_voter}(\mathsf{paper}) \mapsto \mathsf{paper\_vote}(\mathsf{paper})\}
\end{array}
$$

At this level, we focus at the main security properties of the SBB:

1. Accept all valid ballots.
2. Reject invalid ballots.

These two goals have different purposes, the first one is concerned with the system availability, a key security property, where we need to make sure that valid ballots are not blocked from being cast. On the other hand, the combination of both goals will ensure the integrity of voting system. The second goal is expressed in the invariant

$$\mathsf{ejected\_papers} \subseteq (\mathsf{spoiled\_papers} \cup \mathsf{invalid\_papers} \cup \mathsf{illegitimate\_papers})$$

to ensure that only rejected ballots (invalid or illegitimate) and the ballots the user choose to spoil will be ejected out of the SBB. Refinement consistency POs will ensure that this invariant holds across the different refinement levels.

The availability property can be captured by the guard of the relevant events. In particular, we specify that cast_paper is a rigid event and its parameter paper is also rigid. Using the notation introduced in Section 4, the event cast_paper has the following form.

$$
\begin{array}{l}
\textbf{event}\ [\mathsf{cast\_paper}] \\
\textbf{any}\ [\mathsf{paper}]\ \textbf{where} \\
\quad @\mathsf{valid{-}paper}: \mathsf{paper} \in \mathsf{valid\_papers}
\end{array}
$$

[2] where [ ] is a relational image and { | } is a set comprehension. A concise summary of Event-B syntax is available at http://wiki.event-b.org/images/EventB-Summary.pdf.

```
then
  // actions for casting the ballot
  ...
end
```

The above event specifies that the cast_paper event must be enabled for any valid paper, i.e., satisfying the guard $\mathsf{paper} \in \mathsf{valid\_papers}$.

5.4 Second Refinement: Introducing Time and Availability of Events

In this refinement, we introduce Time where each ballot has a timestamp defined as $\mathsf{paper\_time} \in \mathsf{papers} \to \mathsf{TIME}$ and current_time demonstrates the progression of time. A ballot can expire after a certain time if it is not cast or spoiled, thus invalidating a legitimate ballot. Ballots issued in the future are also considered invalid. We introduce the clock_tick event to progress time which also refines papers_expired from Section 5.3.

By introducing time, now we have all the conditions to define a valid ballot both in terms of duplication and expiry of time stamp. We therefore refine for the guards of the events that depend on ballot validity, such as casting and ejecting ballots. These guards will ensure that ballots have not been cast or spoiled before, have not expired nor have they been issued by an illegitimate source.

The cast_paper event, introduced at the previous level, is a rigid event with the rigid parameter paper. The refinement of the cast_paper event is as follow.

```
event [cast_paper] refines cast_paper
any [paper] where
 @typeof−paper: $paper \in papers$
 @not−already−expired: $paper\_time(paper) \geq current\_time - expiry\_duration$
 @copy−not−already−cast: $paper\_voter(paper) \notin paper\_voter[cast\_papers]$
 @copy−not−already−spoiled:
  ($\forall sp \cdot sp \in spoiled\_papers \Rightarrow$
            $paper\_voter(paper) \neq paper\_voter(sp) \lor$
            $paper\_vote(paper) \neq paper\_vote(sp) \lor$
            $paper\_time(paper) \neq paper\_time(sp)$
  )
 @not−illegitimate−paper: $paper \notin illegitimate\_papers$
then
  // cast the paper
  ...
end
```

This gives rise to the ENBL PO at this refinement. Because ENBL PO is not yet supported by Rodin, we encode this ENBL PO as the following theorem (accept−valid−paper) in our model.

```
theorem @accept−valid−paper: // Encoding of the ENBL PO for cast_paper
    event
```

```
  $\forall \mathsf{paper} \cdot$                    // Universally quantified over abstract rigid
      variables
  $\mathsf{paper} \in \mathsf{valid\_papers}$        // Guard of the abstract event
$\Rightarrow$
  /* Guards of the concrete event */
  $\mathsf{paper} \in \mathsf{papers} \land$
  $\mathsf{paper\_time}(\mathsf{paper}) \geq \mathsf{current\_time} - \mathsf{expiry\_duration} \land$
  $\mathsf{paper\_voter}(\mathsf{paper}) \notin \mathsf{paper\_voter}[\mathsf{cast\_papers}] \land$
  $(\forall \mathsf{sp} \cdot \mathsf{sp} \in \mathsf{spoiled\_papers} \Rightarrow$
      $\mathsf{paper\_voter}(\mathsf{paper}) \neq \mathsf{paper\_voter}(\mathsf{sp}) \lor$
      $\mathsf{paper\_vote}(\mathsf{paper}) \neq \mathsf{paper\_vote}(\mathsf{sp}) \lor$
      $\mathsf{paper\_time}(\mathsf{paper}) \neq \mathsf{paper\_time}(\mathsf{sp})$
  $) \land$
  $\mathsf{paper} \notin \mathsf{illegitimate\_papers}$
```

Notice that there are no non-rigid parameters for cast_paper event, i.e., the right-hand side of the implication is not existentially quantified. The same applies to spoil_valid_paper because the voter should have the option to either cast or spoil their valid ballot paper, so both events have the same guards or enabling conditions and the theorem will apply to both events.

As mentioned earlier, if a ballot has been cast or spoiled before, it will be considered invalid. However, there is a difference here between the two cases, a voter with a spoiled ballot can be issued another paper if they present a physical proof to the BMD of the spoiled ballot, whereas a voter with a cast ballot cannot. Hence the difference between the guards for checking whether a paper has been cast or spoiled. In the case of a spoiled ballot another ballot belonging to the same voter can be considered valid if it at least has a different time stamp. Using model checking it was possible to discover such difference and add this assumption in the form of guard to BMD_issues_paper.

Further to the typing invariants related to the new timing variables, the following invariants describe the difference between valid and invalid ballots in regards to time and double voting and can ensure that theorem (accept−valid−paper) above is true. The last invariant valid−and−spoiled−papers−disjoint was actually discovered by attempting to prove that there is no valid ballot which is an exact copy of a spoiled ballot. A voter with a spoiled ballot can get a new legitimate ballot, but the new ballot will have at least a different time stamp (paper_time).

```
// The valid ballot papers must not have the future time stamps.
@no−future−valid−papers:
 $\forall \mathsf{b} \cdot \mathsf{b} \in \mathsf{valid\_papers} \Rightarrow \mathsf{paper\_time}(\mathsf{b}) \leq \mathsf{current\_time}$

// The valid ballot papers must not expired
@no−expiry−valid−papers:
 $\forall \mathsf{b} \cdot \mathsf{b} \in \mathsf{valid\_papers} \setminus \mathsf{cast\_papers}$
              $\Rightarrow \mathsf{current\_time} - \mathsf{expiry\_duration} \leq \mathsf{paper\_time}(\mathsf{b})$

// For two different valid ballot papers, if it is for the same voter then
// they must have the same time stamp, i.e., they are copies of each other.
@no_valid_double_voting_time:
 $\forall b1, b2 \cdot$
   $b1 \in valid\_papers \land b2 \in valid\_papers$
        $\land b1 \neq b2 \land paper\_voter(b1) = paper\_voter(b2)$
           $\Rightarrow paper\_time(b1) = paper\_time(b2)$

// Any invalid paper will either be expired, or a copy has been cast or a
// copy has been spoiled.
@expired−or−cast−or−spoiled−copy−invalid_papers:
 $\forall paper \cdot paper \in invalid\_papers \Rightarrow$
    $current\_time - expiry\_duration > paper\_time(paper)$
  $\lor paper\_voter(paper) \in paper\_voter[cast\_papers]$
  $\lor (\exists sp \cdot sp \in spoiled\_papers$
           $\land paper\_voter(paper) = paper\_voter(sp)$
           $\land paper\_vote(paper) = paper\_vote(sp)$
           $\land paper\_time(paper) = paper\_time(sp)$
)

@valid−and−spoiled−papers−disjoint:
 $\forall vp, sp \cdot vp \in valid\_papers \land sp \in spoiled\_papers \Rightarrow$
             $paper\_voter(vp) \neq paper\_voter(sp)$
           $\lor paper\_vote(vp) \neq paper\_vote(sp)$
           $\lor paper\_time(vp) \neq paper\_time(sp)$
```

5.5 Third Refinement: Ballot Encryption

At this level we introduce encryption so tat the SBB will not be able to access the voters information. Consequently, we apply data refinement to replace the variables paper_vote and paper_voter with the encrypted ballot. The following invariants describe ballot encryption and include gluing invariants to relate the new variable paper_encrypted_ballot with the disappearing variables paper_voter and paper_vote.

```
@typeof−paper_encrypted_ballot:
 $paper\_encrypted\_ballot \in papers \rightarrow CYPHER\_TEXT$

@gluing−legitimate−papers:
 $\forall paper \cdot paper \in legitimate\_papers$
            $\Rightarrow EncryptionAlgorithm($
                    $paper\_voter(paper) \mapsto paper\_vote(paper) \mapsto EncryptionKey$
              $) = paper\_encrypted\_ballot(paper)$

@encrypted−ballot−disjoint−cast−valid:
 $\forall p \cdot p \in legitimate\_papers$
        $\land paper\_encrypted\_ballot(p) \notin paper\_encrypted\_ballot[cast\_papers]$
```

$$\begin{aligned}
&\land (\forall \mathsf{sp} \cdot \mathsf{sp} \in \mathsf{spoiled\_papers} \Rightarrow \\
&\qquad\qquad \mathsf{paper\_encrypted\_ballot}(\mathsf{p}) \neq \mathsf{paper\_encrypted\_ballot}(\mathsf{sp}) \\
&\qquad\quad \lor \mathsf{paper\_time}(\mathsf{p}) \neq \mathsf{paper\_time}(\mathsf{sp})) \\
&\land \mathsf{paper\_time}(\mathsf{p}) \geq \mathsf{current\_time} - \mathsf{expiry\_duration} \\
&\qquad\qquad \Rightarrow \mathsf{paper\_voter}(\mathsf{p}) \notin \mathsf{paper\_voter}[\mathsf{cast\_papers}]
\end{aligned}$$

@gluing_encryption_voter:

$$\begin{aligned}
&\forall \mathsf{p1}, \mathsf{p2} \cdot \mathsf{p1} \in \mathsf{legitimate\_papers} \land \mathsf{p2} \in \mathsf{legitimate\_papers} \\
&\qquad\quad \land \mathsf{paper\_encrypted\_ballot}(\mathsf{p1}) = \mathsf{paper\_encrypted\_ballot}(\mathsf{p2}) \\
&\qquad\qquad \Rightarrow \mathsf{paper\_voter}(\mathsf{p1}) = \mathsf{paper\_voter}(\mathsf{p2})
\end{aligned}$$

@inv_not_already_cast:

$$\begin{aligned}
&\forall \mathsf{p} \cdot \mathsf{p} \in \mathsf{valid\_papers} \\
&\qquad \Rightarrow \mathsf{paper\_encrypted\_ballot}(\mathsf{p}) \notin \mathsf{paper\_encrypted\_ballot}\,[\mathsf{cast\_papers}]
\end{aligned}$$

As the SBB cannot access the voter's information on the ballot directly, the guards for cast_paper need to be refined. As a result, the ENBL PO needs to be proved for the refinement. In particular, the part of the ENBL PO related to the refinement is shown below.

theorem @accept−valid−paper:

$$\begin{aligned}
&\forall \mathsf{paper} \cdot \ldots \land \\
&\quad (\forall \mathsf{sp} \cdot \mathsf{sp} \in \mathsf{spoiled\_papers} \Rightarrow \\
&\qquad\quad \mathsf{paper\_voter}(\mathsf{paper}) \neq \mathsf{paper\_voter}(\mathsf{sp}) \lor \mathsf{paper\_vote}(\mathsf{paper}) \neq \mathsf{paper\_vote}(\\
&\qquad \mathsf{sp}) \\
&\qquad\quad \lor \mathsf{paper\_time}(\mathsf{paper}) \neq \mathsf{paper\_time}(\mathsf{sp})\ ) \land \ldots \\
&\qquad \Rightarrow \ldots \land \\
&\qquad\ (\forall \mathsf{sp} \cdot \mathsf{sp} \in \mathsf{spoiled\_papers} \Rightarrow \\
&\qquad\quad \mathsf{paper\_encrypted\_ballot}(\mathsf{paper}) \neq \mathsf{paper\_encrypted\_ballot}(\mathsf{sp}) \\
&\qquad\quad \lor \mathsf{paper\_time}(\mathsf{paper}) \neq \mathsf{paper\_time}(\mathsf{sp})\ ) \land \ldots
\end{aligned}$$

Notice that this PO is discharged trivially due to the property of the encryption function.

5.6 Fourth Refinement: Ballot Authentication

The purpose of ballot authentication is to protect against malicious intruder behaviour, where an intruder tries to cast a ballot not issued by its only legitimate source, BMD. These attacks are introduced earlier in Section 5.3 and specified as the attacker events. We introduce MAC to check the legitimacy of the ballot. Therefore, all the event guards checking for ballot legitimacy will be replaced by an equality check of the MAC paper_mac with the calculated MAC. The MAC can be calculated using the MACAlgorithm which requires the secret MACKey.

@typeof−paper_mac: $\mathsf{paper\_mac} \in \mathsf{papers} \rightarrow \mathsf{MAC}$

@mac−illegitimate_papers: $\forall \mathsf{paper} \cdot \mathsf{paper} \in \mathsf{illegitimate\_papers} \Rightarrow$
$\mathsf{paper\_mac}(\mathsf{paper}) \neq \mathsf{MACAlgorithm}($

$$\mathsf{paper\_time}(\mathsf{paper}) \mapsto \mathsf{paper\_encrypted\_ballot}(\mathsf{paper}) \mapsto \mathsf{MACKey})$$

$$\begin{array}{l}
\mathsf{@mac{-}legitimate\_papers}: \\
\forall \mathsf{paper} \cdot \mathsf{paper} \in \mathsf{legitimate\_papers} \Rightarrow \\
\quad \mathsf{paper\_mac}(\mathsf{paper}) = \mathsf{MACAlgorithm}(\\
\quad\quad \mathsf{paper\_time}(\mathsf{paper}) \mapsto \mathsf{paper\_encrypted\_ballot}(\mathsf{paper}) \mapsto \mathsf{MACKey})
\end{array}$$

The invariants mac−illegitimate_papers and mac−legitimate_papers define the difference between legitimate and illegitimate ballots in relation to MAC. As a consequence the cast_papers guards are refined to check for MAC equality and the ENBL PO is generated (similarly to the previous section).

In ATK_creates_paper, we assume the attacker does not know MACKey and will create illegitimate paper ballots. However, if an attacker compromises key, they will be able to generate ballots that are accepted by the SBB. It is therefore crucial to ensure the secrecy of this key. This could be achieved, for example, using secure hardware which can provide memory protection to the secret keys.

6 Debugging Models using Model Checking

In this section, we discuss the use of model checking to help with debugging our model. We first discuss the analysis of refinement consistency of the rigid events in Section 6.1. Subsequently, we analyse the attack on the clocks of the BMD and SBB in Section 6.2.

6.1 Consistency of the Refinement of the Rigid Events

As presented in the previous section, the consistency of the refinement of the rigid event cast_paper is captured as the theorem accept−valid−paper. The theorem states that a valid paper will be the one that is not yet expired, the voter on the ballot has not yet voted, and a copy of the paper has not yet been spoiled. In our initial model the theorem could not be discharged automatically. ProB Model checker shows a trace that can violate the theorem as follows.

```
INITIALISATION
BMD_issues_paper(PAPER1, VOTER1, VOTE7)
spoil_valid_paper(PAPER1)
BMD_issues_paper(PAPER2, VOTER1, VOTE7)
```

The trace shows a scenario where a VOTER1 got a ballot PAPER1 with VOTE7 from the BMD, subsequently spoils the paper, and get another ballot PAPER2 with the same choice VOTE7. At this point, since PAPER2 is a valid paper, but it has the same information as the spoiled paper PAPER1 (including the time stamp), PAPER2 cannot be cast. An important assumption that is missing from our model is that the papers PAPER1 and PAPER2 must have a different time stamps. As a result, we strengthen the guard of BMD_issue_paper to add the assumption

$$\begin{array}{l}
\mathsf{@no{-}clash{-}spoiled{-}papers:} \\
\quad \forall \mathsf{sp} \cdot \mathsf{sp} \in \mathsf{spoiled\_papers} \Rightarrow \\
\qquad \mathsf{voter} \neq \mathsf{paper\_voter(sp)} \lor \mathsf{vote} \neq \mathsf{paper\_vote(sp)} \lor \mathsf{current\_time} \neq \\
\qquad \mathsf{paper\_time(sp)}
\end{array}$$

Furthermore, we add an invariant to state the relationship between valid papers and spoiled papers.

$$\begin{array}{l}
\mathsf{@valid{-}and{-}spoiled{-}papers{-}disjoint:} \\
\quad \forall \mathsf{vp, sp} \cdot \mathsf{vp} \in \mathsf{valid\_papers} \land \mathsf{sp} \in \mathsf{spoiled\_papers} \Rightarrow \\
\qquad\qquad \mathsf{paper\_voter(vp)} \neq \mathsf{paper\_voter(sp)} \lor \mathsf{paper\_vote(vp)} \neq \mathsf{paper\_vote(sp)} \\
\qquad \lor\ \mathsf{paper\_time(vp)} \neq \mathsf{paper\_time(sp)}
\end{array}$$

Given the above invariant, the theorem for proving the consistency of the refinement of the rigid event is proved automatically.

6.2 Attacks on the Clocks

In Section 5.4, we use one global clock and consider that both the SBB and BMD are always synchronised with current_time. However, this might not be the case and there can be some attacks on the clocks that can lead to accepting invalid paper ballots or rejecting valid paper ballots.

To model such attacks, in addition to current_time in Section 5.4, we introduce two clock variables BMD_time and SBB_time that are synchronised with the global clock current_time. All the invariants related to time will remain the same based on current_time, while the BMD events such as BMD_issues_paper will use the BMD_time. Similarly, the SBB events, cast_paper and eject_paper, will use the SBB_time. Therefore, to prove the invariants related to paper_time, both clocks should be equal to current_time. The equality invariants cannot be proved, if we introduce attacker events that can advance or delay the SBB and BMD clocks.

We use the ProB model checker to show how the clock attacks can invalidate the two main requirements of the case study. To help ProB automatically generate a counter example: We restrict ProB to one refinement level. We remove the invariants related to equality of time and relating the validity of ballots and time. Then, we copy over the ejected_papers type invariant that ensure that only invalid, illegitimate or spoiled ballots are ejected from previous refinement level. The model checker generates the following counter example: ⟨attacker_advance_bmd_clock(3), BMD_issues_paper(PAPER1, VOTER3, VOTE1), eject_paper(PAPER1)⟩

This trace will violate the ejected_papers type invariant copied from the first refinement, because $\mathsf{PAPER1} \in \mathsf{valid\_papers}$. Therefore, using ProB, we have shown how an attack on the BMD clock can lead to rejecting a valid ballot that has not been spoiled. Other clock attacks can be demonstrated in a similar way.

7 Related Work

In this paper we have modelled a smart ballot box of a secure voting system. We use Event-B to analyse the system level security focusing on the refinement of the availability property, whereas most security verification tools such as [10, 11] consider the verification of security protocols.

In [6], the authors also use a correct-by-construction approach using Event-B to model a secure e-voting system. The authors focus on the recording and the tallying phases to ensure the verifiability of the system using a decomposition pattern and a contextualisation technique. Our case study focuses on the smart ballot box which only allows the casting of valid encrypted ballots. The encrypted ballots in the SBB can in turn be used for rapid digital tabulation (tallying) and to provide an evidence-based auditing for the tabulation process. In this model we do not handle tabulation, but we are considering the extension of our models to include tabulation as a future work. Similarly, in [4], the authors focus on the verifiability of a peered web bulletin board for publishing the evidence of voting and tallying using Event-B.

In order to prove the availability of casting ballots through refinement, we prove the enabledness preservation of the events. In [12], the authors use enabledness preservation in conjunction with non-divergence to prove the liveness of an Event-B model. In their case enabledeness preservation can have two notions, in the weakest notion, the enabledness preservation states if one of the events in the abstraction is enabled then one or more events in the refinement are also enabled. While, the strongest notion states if an abstract event is enabled then either the refining event is enabled or one of the new events are enabled. Even their strongest notion of enabledness preservation is still weaker than our definition of enabledness preservation which requires proving the non-strengthening of the guards of the rigid event. In [7], the authors use enabledness proofs to ensure the “refinement equivalence” of external events in the shared-variable decomposition of Event-B models. However, their proposed POs are similar to the standard Event-B POs. The idea of enabledness preservation has also been considered in the formal method ASM [3], in the concepts of ground model and refinements where the abstract and concrete guards are equivalent.

8 Conclusions and Future Work

In this paper, we have shown how the availability property of an event can be ensured through refinement by preserving the enabledness of its corresponding refined events. Such property relates to the parameter type where some parameters are considered rigid and should be preserved through refinement. We provide a general PO (ENBL) that can be applied to any event with rigid parameters. We apply ENBL PO by defining a theorem to ensure the availability of casting valid ballots.

In the future, we will focus on the semantics model to justify the soundness of the rigid property of events, we can possibly explore failure semantics. We will also look at how introducing new events in refinement can affect the ENBL PO. Finally, we plan to provide tool support for the enabledness preservation PO in Rodin. This can be done by extending the CamilleX [8] textual framework. In the CamilleX textual editor, the modeller will identify the rigid parameters of the event, and the enabledness preservation theorems will be automatically added to the Event-B generated machine.

Acknowledgement: This work is supported by the HD-Sec project, which was funded by the Digital Security by Design (DSbD) Programme delivered by UKRI to support the DSbD ecosystem.
We would like to thank Joseph Kiniry and Daniel Zimmerman from Galois for providing details of and insights into the case study.

References

1. J-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.
2. J-R. Abrial, M. Butler, S. Hallerstede, T. S. Hoang, F. Mehta, and L. Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *STTT*, 12(6):447–466, 2010.
3. Egon Börger. *The ASM Ground Model Method as a Foundation of Requirements Engineering*, pages 145–160. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.
4. C. Culnane and S. Schneider. A peered bulletin board for robust use in verifiable voting systems. In *2014 IEEE 27th Computer Security Foundations Symposium (CSF)*, pages 169–183, Los Alamitos, CA, USA, jul 2014. IEEE Computer Society.
5. Galois and Free & Fair. The BESSPIN Voting System. `https://github.com/GaloisInc/BESSPIN-Voting-System-Demonstrator-2019`, May 2019. Accessed: 2021-02-02.
6. J. P. Gibson, S. Kherroubi, and D. Méry. Applying a Dependency Mechanism for Voting Protocol Models Using Event-B. In *Formal Techniques for Distributed Objects, Components, and Systems*, pages 124–138. Springer International Publishing, 2017.
7. S. Hallerstede and T. S. Hoang. Refinement of decomposed models by interface instantiation. *Science of Computer Programming*, 94:144–163, 2014. Abstract State Machines, Alloy, B, VDM, and Z.
8. T. S. Hoang and D. Dghaym. Event-B and Rodin Wiki: CamilleX. `http://wiki.event-b.org/index.php/CamilleX`, 2018. Accessed Feb 2021.
9. M. Leuschel and M. Butler. ProB: A model checker for B. In *International Symposium of Formal Methods Europe*, pages 855–874. Springer, 2003.
10. G. Lowe. Casper: a compiler for the analysis of security protocols. In *Proceedings 10th Computer Security Foundations Workshop*, pages 18–30, 1997.
11. B. Schmidt, S. Meier, C. Cremers, and D. Basin. Automated analysis of diffie-hellman protocols and advanced security properties. In *2012 IEEE 25th Computer Security Foundations Symposium*, pages 78–94, 2012.
12. D. Yadav and M. Butler. Verification of liveness properties in distributed systems. In *Contemporary Computing*, pages 625–636, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.