HD-Sec: Holistic Design of Secure Systems on Capability Hardware

UNIVERSITY OF Southampton

Dana Dghaym, Robert Thorburn, Michael Butler, Thai Son Hoang, Leonardo Aniello, Vladimiro Sassone
University of Southampton, UK {d.dghaym, r.h.thorburn, m.j.butler, t.s.hoang, l.aniello, vsassone}@soton.ac.uk

HD-Sec Vision

Transformation of security-critical software development

- From an expensive iterative test-and-fix approach
  - To a correctness-by-construction (CxC) approach
- Where formal modelling, verification and model transformation tools guide the design of software from requirements to implementation

Objectives

1. Systematic approaches for elicitation and formal modelling of security requirements.
2. Reusable formal abstractions of data trust mechanisms.
3. High-level abstractions and model transformations.
4. Soundness of the high-level abstractions and model transformations.
5. Enhance the open-source Rodin toolchain to support our techniques.
6. Validate the resulting CxC toolchain on industrial-strength case studies.
   - Including a functioning prototype secure application designed using our CxC tools and running on capability hardware.

Case Study: Smart Ballot Box[1] (SBB)

- Key functions of the Smart Ballot Box (SBB):
  - Detect and decode a 2D barcode on a ballot paper.
  - Verify if the decoded contents of the ballot paper are from a Ballot Marking Device (BMD).
  - Valid ballot papers can be cast into the storage box of the SBB.
  - Valid ballot papers can be spoiled and ejected out of the SBB if the voter choose to spoil
  - their ballot.
  - Invalid ballot papers will be rejected by the SBB.
- Main security properties of the SBB:
  - **Confidentiality :** using encryption of the voter's choices.
  - **Integrity:** using message authentication to only accept valid ballots and reject invalid ballots.
  - **Availability:** is guaranteed by not preventing a voter from casting a valid ballot.

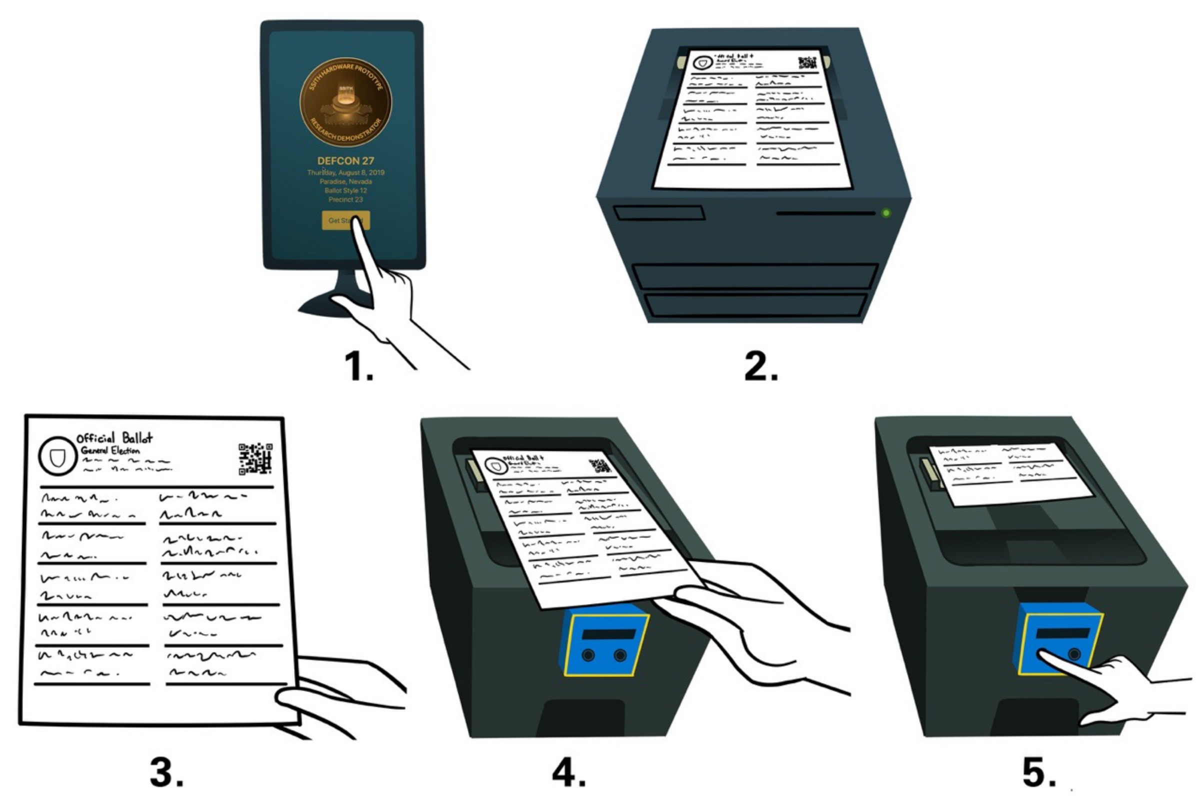


Fig 1: Workflow of Ballot Marking & Smart Ballot Box Accept of Ballot[1].

Formal Modelling of the SBB[2]

- Event-B: a refinement–based formal method for developing discrete transition systems.
- Formal system modelling
  - Security at the system level: interaction between different system components.
- Illustrate different possible security attacks.
- Derive component specification.
- Preserving availability properties through refinement.

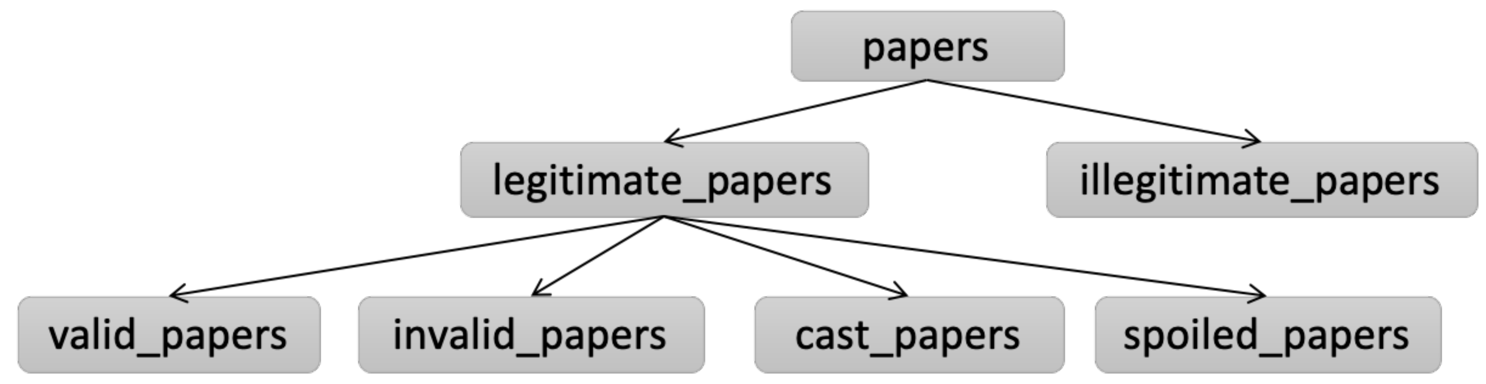


Fig 2: Different Types of ballot papers.

Refinement Strategy

1. Model an ideal voting system where only legitimate ballots can be cast.
2. Distinguish between the different types of ballot papers (Fig. 2) and model possible attackers' behaviour e.g., duplicate a valid ballot.
3. Introduce time and invalidate ballots with expired timestamp. Time can be also subject to malicious attacks e.g., delay the SBB clock.
4. Apply data refinement to introduce ballot encryption.
5. Introduce message authentication to ensure the legitimacy of the ballots.

2nd Refinement	3rd Refinement	4th Refinement
∀ paper · paper ∈ valid_papers ⇒ paper_time(paper) ≥ current_time − expiry_duration ∧ paper_voter(paper) ∉ paper_voter[casted_papers] ∧ (∀sp · sp ∈ spoiled_papers ⇒ paper_voter(paper) ≠ paper_voter(sp) ∨ paper_vote(paper) ≠ paper_vote(sp) ∨ paper_time(paper) ≠ paper_time(sp)) ∧ paper ∉ illegitimate_papers	∀ paper · paper ∈ valid_papers ⇒ paper_time(paper) ≥ current_time − expiry_duration ∧ paper_encrypted_ballot(paper) ∉ paper_encrypted_ballot[casted_papers] ∧ (∀sp · sp ∈ spoiled_papers ⇒ paper_encrypted_ballot(paper) ≠ paper_encrypted_ballot(sp) ∨ paper_time(paper) ≠ paper_time(sp)) ∧ paper ∉ illegitimate_papers	∀ paper · paper ∈ valid_papers ⇒ paper_time(paper) ≥ current_time − expiry_duration ∧ paper_encrypted_ballot(paper) ∉ paper_encrypted_ballot[casted_papers] ∧ (∀sp · sp ∈ spoiled_papers ⇒ paper_encrypted_ballot(paper) ≠ paper_encrypted_ballot(sp) ∨ paper_time(paper) ≠ paper_time(sp)) ∧ paper_mac(paper) = MACAlgorithm(paper_time(paper)↦paper_encrypted_ballot(paper) ↦ MACKey)

Preserving Availability Property by Proving Event Enabledness

Guards of the abstract Event ⇒ Guards of the Concrete Event

From Event-B to SPARK - Ada Implementation

1. Decompose the Event-B model to obtain the software related variables and events.
2. Introduce a new refinement where we Data refine the Event-B structures that cannot be represented in SPARK into their possible corresponding constructs in SPARK e.g., sets => arrays
3. Events in Event-B are translated into SPARK procedures where the event guards are represented as preconditions and the event actions are translated into post conditions.

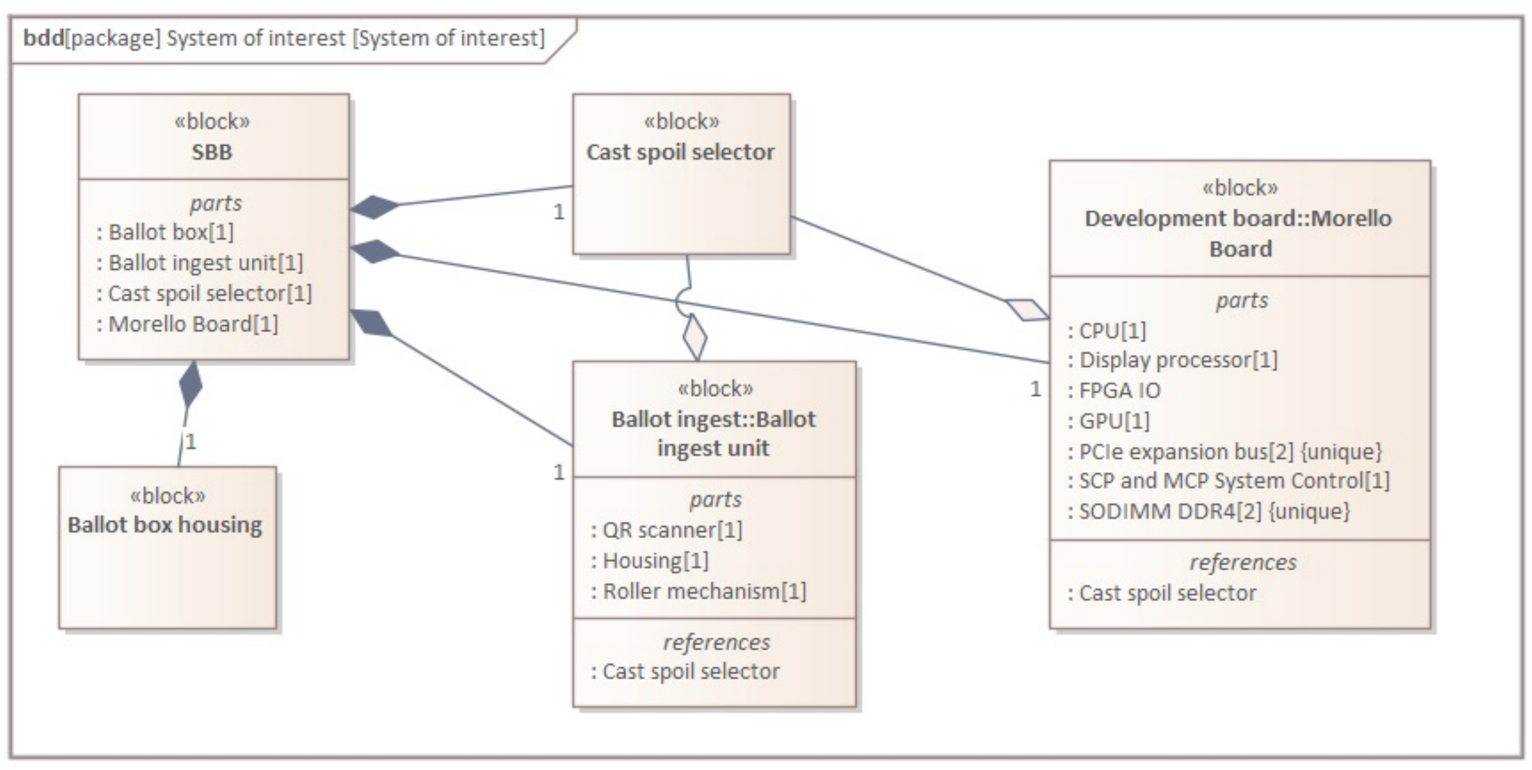
```
procedure cast(paper : in  barcode) with
  Global => (Proof_In => (MACKEY, spoiled_arr, curr_time),
  In_Out => (cast_arr, vote_count)),
  Pre => ( already_cast(paper) = False
             and then already_spoiled(paper) = False
             and then valid_time (paper) = True
             and then validate_barcode(paper) = True
             and then vote_count < Max_Votes),
  Post => (already_cast(paper)
            and vote_count = vote_count' old + 1);
```

The Morello Fixed Virtualisation Platform (FPV)

- Testing and development in preparation of hardware
- Currently supports CheriBSD, Android, and Linux
- Linux development with capability pointers

Proceeding to hardware testing

- Integrating physical Morello board and test rig
- Testing functionality, cybersecurity, and physical security



References

[1] Galois and Free & Fair. The BESSPIN Voting System (2019).
[2] D. Dghaym, T.S. Hoang, M. Butler, R. Hu, L. Aniello, V. Sassone (2021) Verifying System-level Security of a Smart Ballot Box. In ABZ 2021- 8th International Conference on rigorous State Based Methods.