Model of the Colorado Risk Limiting Audit (CORLA) tool

Joseph R. Kiniry and Daniel M. Zimmerman

July 2017

1 Introduction

This document contains a formal specification of the ColoradoRLA system/tool. It contains a detailed technical specification of the system, annotated with prose designed to help readers who are not expert in formal system specification understand the scope and interpretations of the elements of the specification. The details of the specification itself will guide deployment, management, maintenance and evolution of the system delivered to the Colorado Department of State. The prose annotations should help Colorado Department of State elections domain experts and technical staff to assess the completeness and correctness of the ColoradoRLA system/tool.

This document is written in a literate style using the PVS theorem prover. Writing system specifications in this fashion is called (formal) domain engineering. Our formal domain model is written in PVS's higher-order logic (HOL). Our informal domain model is written in the Free & Fair System Specification Language, or FAFESSL for short. FAFESSL is a daughter of Extended BON which was, in turn, a daughter of BON, the Business Object Notation.

See our Bibliography project at GitHub, `https://github.com/FreeAndFair/Bibliography`, for more information.

In order to relate PVS to FAFESSL, we must define a refinement relationship between their two type systems. Informally, that mapping is described in the following paragraph.

We map PVS public theories and their contents to FAFESSL constructs. The top-level corla theory maps to the FAFESSL system specification, theories map to clusters, (PVS) types map to (FAFESSL) types, and functions map to features. We use special comments to denote informal specifications which will be extracted using a shell script into a well-typed FAFESSL informal domain model specification.

As mentioned above, we write formal specifications using a literate style, a la Knuth. Doing so permits us to document our thinking and the system design, from domain modeling and engineering all of the way to formal specification and

verification, in a fashion that produces beautifully printed books and interactive hypertext.

In order to write literate PVS we use our old friend Adriaan de Groot's scripts, available at `http://www.cs.kun.nl/~adridg/research/PVS-literate.html`. Our formal domain model is also annotated with structured comments in precise natural language using a standard set of annotations. A shell script processes these annotations and generates a well-typed informal FAFESSL specification.

The FAFESSL annotations we use are as follows:

system the FAFESSL name of the system

cluster the FAFESSL name of a cluster

explanation the explanation for an artifact

description a short description of an artifact

indexing-CLAUSE a prefix for any indexing <CLAUSE>

These suffixes are commonly used:

author an author of an artifact

organization an organization responsible for an artifact

keywords a comma-separated list of keywords

created the creation date for an artifact

github the URL for the GitHub project containing an artifact

2 System Overview

The RLA Tool is being developed by Free & Fair for the Colorado Department of State (CDOS henceforth). The RLA Tool facilitates running a risk limiting audit across several jurisdictions. In the case of Colorado, it facilitates running risk limiting audits across all counties in the state simultaneously.

The Colorado election law pertaining to election audits is "TBD", and the rules pertaining to such are found in the "Revised Draft of Proposed Rules" whose latest version is labeled CCR 1505-1 and dated July 6, 2017. Section 25, Post-Election Audit, of that document is the salient portion of that document. We will call this document RDPR-6-Jul-2017 for the purpose of traceability in this specification.

This domain model is based upon that law, its rules, our contract, our bibliography of publications about risk-limiting audits, etc.

```
fafessl: THEORY
BEGIN
  TBD: bool = TRUE
END fafessl
```

3 Background Model Elements

It is necessary to summarize a number of background concepts from computer science in order to formalize the RLA Tool. The RLA Tool is a client-server system. The server is implemented in Java and runs on servers hosted by CDOS. The client is implemented in TypeScript and JavaScript and runs in modern web browsers.

As we are hosting the RLA Tool in CDOS's system, they are providing the hosting servers, network, and a number of services in support of the tools deployment and management.

A database is used to store all of the data relevant to audits. Databases contain tables that describe the information they contain. We need not formalize database elements in any more detail than what follows.

```
database: THEORY
BEGIN
  database_table: TYPE
  data: TYPE
  database: TYPE = setof[database_table]
  write: [database, database_table, data -> database]
  read: [database, database_table -> data]
END database
```

We also need to be able to talk about some pretty simple ideas from information systems. For example, many files in election systems are syntactically just lists of comma-separated values. Email is used for communication between officials and with the public. Files are used to store information in a persistent fashion. Consequently, we introduce a few types for these ideas.

```
information_systems: THEORY
BEGIN
  comma_separated_value: TYPE
  csv: TYPE = comma_separated_value
  email: TYPE
  email_address: TYPE
  file: TYPE
  network: TYPE
```

```
  web_browser: TYPE
  browser: TYPE = web_browser
  javascript_code: TYPE
END information_systems
```

Several kinds of servers are necessary to deploy this system. At the minimum, two of each of a web server, a time (NTP) server, a firewall, and a database server are necessary. Two servers of each class are required for redundancy to fulfill availability requirements. Preferably redundant servers would be hosted in separate facilities, with separate and independent power systems and networks.

An Election Night Reporting (ENR) server is mentioned repeatedly in the RDPR-6-Jul-2017, so we model it here.

```
server: DATATYPE
  WITH SUBTYPES web_server,
                time_server,
                firewall,
                database_server,
                enr_server
BEGIN
  IMPORTING information_systems, database, elections

  web_server?: make_web_server: web_server
  time_server?: make_time_server: time_server
  firewall?: make_filewall: firewall
  database_server?(databases: set[database]):
    make_database_server: database_server
  enr_server?: make_enr_server: enr_server
END server
```

Over time, as we need to introduce more background concepts, we will add them to this theory.

4 Elections

A number of general elections concepts are necessary to specify the RLA Tool. Some are generic to all elections, and others are specific to Colorado.

```
elections: THEORY
BEGIN
```

Voters are the most important concept in elections, as they are who elections are for, and who make the choices that determine the outcome of legitimate elections. Some voters are UOCAVA voters. What follows is one way to formalize such, simply stating that the type uocava_voter is a predicate on the type voter. Consequently, all voters are either UOCAVA or not.

```
person: TYPE
voter: TYPE FROM person
elector: TYPE = voter
uocava_voter: TYPE from voter
```

Candidates run for office to represent voters. A person can be a voter, a candidate, both, or neither.

```
candidate: TYPE FROM person
```

Elections focus on contests, each of which represents a set of choices that voters can make. Making a legal choice ranges enormously across the Earth, from marking a vote for a candidate by filling in a single bubble to enumerating a total order on all choices in a contest. Across a given ballot, each contest has a unique name.

```
contest: TYPE =
  [# name: string, description: string #] % , choices: choice #]
contest_name?: [contest -> string]
audited_contest: TYPE FROM contest
opportunistic_contest: TYPE FROM contest
full_hand_count_contest: TYPE FROM contest

contest_outcome: TYPE
election_outcome: TYPE = set[contest_outcome]
wrong_outcome: TYPE FROM election_outcome

reported_tabulation_outcome: TYPE FROM election_outcome

rla_tabulation_outcome: TYPE FROM election_outcome
```

Elections are defined across cohorts of voters arranged in any number of ways—geographic, political, professional, and more. In Colorado, elections are organized across counties and the entire state. Each county has a name and a county number, both of which are unique across the state.

```
county_id: TYPE = nat
county_name: TYPE = string
county: TYPE =
  [# name: county_name, id: county_id #]
state: TYPE = set[county]
nation: TYPE = set[state]
organization: TYPE

s: VAR state
```

```
  c1, c2: VAR county

  county_names_unique: AXIOM
  (FORALL s, c1, c2:
    member(c1, s) AND member(c2, s) AND
      c1`name = c2`name
    IMPLIES c1 = c2)
  county_numbers_unique: AXIOM
  (FORALL s, c1, c2:
    member(c1, s) AND member(c2, s) AND
      c1`id = c2`id
    IMPLIES c1 = c2)

  election_canvass: TYPE
  political_party: TYPE = string
  audit_center: TYPE
  ballot: TYPE % = [election, set[contest], ballot_id]
END elections
```

Elections come in many forms: public and private elections; national and local elections; etc. This RLA Tool focuses on state elections in Colorado, so we only model that one kind of election. Several other kinds of elections are mentioned as well.

Note that county elections are actually *multi*-county elections, where the typical case is that the set of counties is a singleton. In other words, in the general case, contests are multi-county contests, and county elections gather votes for contests from a single county.

```
election: DATATYPE
  WITH SUBTYPES county_election,
                state_election,
                national_election,
                private_election
BEGIN
  IMPORTING elections

  is_county_election?(s: set[county], c: set[contest]):
    make_county_election: county_election
  is_state_election?(s: state, c: set[contest]):
    make_state_election: state_election
  is_national_election?(n: nation, c: set[contest]):
    make_national_election: national_election
  is_private_election?(o: organization, c: set[contest]):
    make_private_election: private_election
END election
```

5 Ballots

The second basic concept of elections that we need to specify is the ballot. Ballots are the means by which voters indicate their preferences for candidates in contests and choices in ballot questions.

Ballots are stored by one of several possible means. Some ballot storage containers are secure, others are not. Some facilitate an easy means by which to find a particular ballots, others do not.

```
storage_container: DATATYPE
  WITH SUBTYPES box,
                bin,
                ballot_box
BEGIN
  IMPORTING elections

  box?(ballots:list[ballot]): make_box: box
  bin?(ballots:list[ballot]): make_bin: bin
  ballot_box?(ballots:list[ballot]): make_ballot_box: ballot_box


END storage_container

elections_equipment: THEORY
BEGIN
  IMPORTING storage_container

  voting_system: TYPE
  dominion_voting_system: TYPE FROM voting_system

  scanner: TYPE
  scanner_id_number: TYPE
  id_number?: [scanner -> scanner_id_number]
  vvpat: TYPE
  verified?: TYPE = pred[storage_container]
```

Voting system's logs are precisely defined in federal standards. Those standards are known as the "Voluntary Voting System Guidlines", or VVSG for short.

```
END elections_equipment
```

```
ballots: THEORY
BEGIN
  IMPORTING elections, elections_equipment, storage_container
```

Several of our concepts are simply numbers. For example, ballot identifiers (ids), batch numbers, and batch sizes are all just natural numbers (integers starting from zero).

```
  ballot_id: TYPE = nat
  batch_number: TYPE = nat
  batch_size: TYPE = nat
```

Ballots are marked by voters or ballot marking devices. A ballot mark is any kind of mark made on a ballot that is not found on a blank ballot. A voter_marking is any mark made by a voter. An ambiguous mark is a mark for which there is ambiguity in its interpretation (voter or machine). A stray mark is a mark that is outside of the legitimate regions of a ballot, such as hesitation marks outside of the mark regions for a contest or marks made by coffee spilled on a paper ballot.

```
  ballot_mark: TYPE
  voter_marking: TYPE FROM ballot_mark
  ambiguous_mark: TYPE FROM ballot_mark
  stray_mark: TYPE FROM ballot_mark
```

Ballots come in several varieties. All ballots are represented in a digital fashion (a scan, PDF, etc.), or are paper ballots, or both. Ballots are also classified based upon where they originate, such as delivery via mail, UOCAVA ballots, early ballots, etc. Ballots are also either tabulated or not. Note that none of these categories of ballots are mutually exclusive.

```
  digital_ballot: TYPE FROM ballot
  paper_ballot: TYPE FROM ballot
  mail_ballot: TYPE FROM ballot
  uocava_ballot: TYPE FROM ballot
  early_ballot: TYPE FROM ballot
  tabulated_ballot: TYPE FROM ballot
  provisional_ballot: TYPE FROM ballot
  property_owner_ballot: TYPE FROM ballot
  original_ballot: TYPE FROM ballot
  duplicated_ballot: TYPE FROM ballot
  non_voter_verifiable_ballot: TYPE FROM ballot
  voter_verifiable_ballot: TYPE FROM ballot
  phantom_ballot: TYPE FROM ballot
```

Ballots can be in various stages of processing, as implicitly mentioned in the C.R.S.

```
  verified_accepted?: pred[mail_ballot]
```

Each ballot has a single ballot style, which indicates (at least) the contests that are listed on that ballot. Ballot styles are usually encoded as natural numbers.

```
  ballot_style: TYPE = nat
  ballot_style?: [ballot -> ballot_style]

  ballot_contest: TYPE = contest
  option: TYPE = string
  choice: TYPE = set[option]

  batch: TYPE = set[ballot]
```

The county must secure and maintain in sealed ballot containers all tabulated ballots in the batches and order they are scanned. The county must maintain and document uninterrupted chain-of-custody for each ballot storage container. Sometimes ballots are processed, either manually or by machines, in such a way that an imprint is made on each ballot processed. By hand, a stamp and signature is sometimes used, e.g. Another example is that a scanner might automatically print a new ballot identifer on the corner of each ballot scanned. We call such a ballot an imprinted ballot.

```
  imprinted_ballot: TYPE FROM ballot
  ballot_certification: TYPE

END ballots
```

5.1 Ballot Manifests

While tabulating ballots, the county must maintain an accurate ballot manifest in a form approved by the Secretary of State. At a minimum, the ballot manifest must uniquely identify for each tabulated ballot the scanner on which the ballot is scanned, the ballot batch of which the ballot is a part, the number of ballots in the batch, and the storage container in which the ballot batch is stored after tabulation.

```
ballot_manifests: THEORY
BEGIN
  IMPORTING ballots, elections_equipment, information_systems

  ballot_manifest_info: TYPE =
```

```
    [county_id, scanner_id_number, batch_number, batch_size, storage_container]
  ballot_manifest: TYPE = set[ballot_manifest_info]
  verified?: pred[ballot_manifest]
  ballot_manifest_file: TYPE FROM file
  export_ballot_manifest: [voting_system -> ballot_manifest_file]

END ballot_manifests
```

The interpretation of the meaning of a voter's choice in a single ballot contest is either a legal vote, an overvote, or an undervote.

```
vote: DATATYPE
  WITH SUBTYPES well_formed_vote,
                overvote,
                undervote
BEGIN
  IMPORTING ballots

  well_formed_vote?(choices:set[choice]):
    make_well_formed_vote: well_formed_vote
  overvote?(choices:set[choice]): make_overvote: overvote
  undervote?(choices:set[choice]): make_undervote: undervote
END vote

ballot_interpretation: THEORY
BEGIN
  IMPORTING ballots, vote

  cast_vote_record: TYPE = set[set[choice]]

END ballot_interpretation

ballots_collections: THEORY
BEGIN
  IMPORTING ballots
```

Ballots are often ordered, such as after they are hand or machine sorted, when they are kept in storage that maintains order, or an order is induced upon them by a random shuffle.

```
  ballot_order: TYPE = sequence[ballot]
```

Sometimes, but not always, a total order is induced by ballot ids.

```
END ballots_collections
```

6 Instructions, Forms, and Reports

Elections have all kinds of different published instructions, forms, and reports. We enumerate a few here that are specific to Colorado, and the RLA Tool in particular.

```
instructions_forms_reports: THEORY
BEGIN
  IMPORTING information_systems, ballot_manifests, election, audits

  report: TYPE = [# name: string #]
  audit_investigation_report: TYPE =
    [# name: string, investigation_report: string #]
  empty_audit_investigation_report: audit_investigation_report

  audit_report: TYPE =
    [# name: string,
       election: election,
       audit_board: audit_board,
       county_administrator: county_administrator #]

  summary_results_report: TYPE =
    [overvotes: nat, undervotes: nat,
     blank_voted_contests: nat, valid_write_in_votes: nat]

  results_file: TYPE FROM file

  form: TYPE
  sos_audit_form: TYPE

  instructions: TYPE
  ballot_instructions: TYPE FROM instructions
  sos_voter_intent_guide: TYPE FROM instructions

  ballots_under_audit_instructions: TYPE =
    [ sequence[ballot], sequence[ballot_style],
      sequence[ballot_manifest_info] ]

END instructions_forms_reports
```

7 Roles

Within the RLA Tool, a person involved with the system can have a number of roles. Some of these roles are mutually exclusive for legal reasons.

```
roles: THEORY
BEGIN
  IMPORTING elections, fafessl

  audit_board_member: TYPE FROM person
  canvas_board: TYPE = set[person]

  colorado_department_of_state: TYPE
  cdos: TYPE = colorado_department_of_state

  sos: TYPE FROM person
  state?: [sos -> state]

  county_clerk: TYPE FROM person
  county?: [county_clerk -> county]

  administrator: TYPE FROM person
  audit_supervisor: TYPE FROM person
  county_administrator: TYPE FROM administrator
  state_administrator: TYPE FROM administrator




  system_administrator: TYPE FROM administrator

  candidates_cannot_be_administrators: AXIOM TBD

END roles

canvas_boards: THEORY
BEGIN
  IMPORTING roles, election

  canvas_board?: [election -> canvas_board]
END canvas_boards
```

8 Cryptography

We will add additional cryptographic primitives here as required.

```
cryptography: THEORY
BEGIN
  digest: TYPE = bvec[256]
  sha256: [size: nat, bvec[size] -> digest]
END cryptography
```

8.1 Randomness

High quality randomness is key to the legitimacy of any risk-limiting audit.

```
randomness: THEORY
BEGIN
  unit: TYPE
```

8.1.1 Random Seed

As stated in RDPR-6-Jul-2017 Section 25.2.2(K):

> The Secretary of State will convene a public meeting on the tenth day after election day to establish a random seed for use with the Secretary of State's RLA tool's pseudo-random number generator based on Philip Stark's online tool, *Pseudo-random Number Generator using SHA-256*. This material is incorporated by reference in the election rules and does not include later amendments or editions. The following material incorporated by reference is posted on the Secretary of State website and available for review by the public during regular business hours at the colorado secretary of state's office: pseudo-random number generator using SHA-256 available at `https://www.stat.berkeley.edu/~stark/java/html/sha256rand.htm`. The Secretary of State will give public notice of the meeting at least seven calendar days in advance. The seed is a number consisting of at least 20 digits, and each digit will be selected in order by sequential rolls of a 10-sided die. The Secretary of State will randomly select members of the public who attend the meeting to take turns rolling the die, and designate one or more staff members to take turns rolling the die in the event that no members of the public attend the meeting. The Secretary of State will publish the seed on the audit center immediately after it is established.

```
  seed: TYPE = {n: nat | 9999999999999999999 < n}
  rng: TYPE = [unit -> nat]
  seeded_prng: TYPE = [seed -> rng]
END randomness
```

9 Audits

Election audits come in many forms. The two main kinds of audits we focus on in this system are ballot polling audits and comparison audits, both of which are risk-limiting audits.

```
audits: THEORY
BEGIN
  IMPORTING roles

  audit: TYPE
```

Audits are run by audit boards, whose members come from various constituancies and have various roles. Deciding who is on an audit board is also usually a matter of law, policy, and history.

```
  audit_board: TYPE = set[audit_board_member]
  audit_board_size: AXIOM TBD
  audit_board_members_political_parties_disjoint: AXIOM TBD
```

These two terms are used but underdefined at this time. When the C.R.S. or CO law is further refined to explain audit investigations (i.e., the process by which a mismatch between the human adjudication and the machine interpretation is resolved), then we will have clarity on the notions of "audit investigation" and "audit progress".

```
  audit_investigation: TYPE
  audit_progress: TYPE
```

The count of the total number of ballots to audit varies across audit types. For example, historically in Colorado random audits must audit 500 ballots or 5% of all ballots, whichever is smaller.

```
  ballots_to_audit: TYPE = nat
```

We list various other terms relating to audits here. They will be defined and refined in later versions of the domain model.

```
  contest_margin: TYPE = nat
  margin: TYPE = real

  digital_ballot_adjudication: TYPE
  manual_ballot_adjudication: TYPE



  diluted_margin: TYPE FROM margin



  margin_overstatement: TYPE = nat



  margin_understatement: TYPE = nat

  random_audit: TYPE
END audits
```

10 RLAs

Audits ideally come after all the votes are tabulated, canvassed and reconciled. In Colorado, however, since the certification deadline comes shortly after the last date for voters to cure signature verification problems, etc., there is not always enough time to do them sequentially. An updated vote count may be released at the end of the tabulation and canvass, and the risk limit of the audit needs to apply to the updated outcome.

Given these constraints, it is generally best to audit conservatively. For example, we could assume that any late-tabulation ballots are cast for the losers. As discussed in [BanuelosEtAl12], counties should add a batch of phantom ballots to the manifest, one for each possible late-tabulation ballot. Another possibility, if it turns out that not enough phantom ballots were added, would be to use more flexible Bayes audit techniques to do a followup audit after the late-tabulation ballots and tabulations are available; however, that capability is not implemented at this time.

```
rlas: THEORY
BEGIN
  IMPORTING elections

  risk_limit: TYPE = {n : nonneg_real | n <= 100}
```

```
  risk_limiting_audit: TYPE
  RLA: TYPE = risk_limiting_audit

  ballot_polling_audit: TYPE FROM risk_limiting_audit

  comparison_audit: TYPE FROM risk_limiting_audit
```

These are some terms of the art that we will more carefully define as the model is refined.

```
  discrepency: TYPE
  random: TYPE
  sample_size: TYPE = nat

  number_of_ballots_to_audit: [audited_contest -> nat]

END rlas
```

11 Cast Vote Records

Cast vote records, also known as CVRs, are the digital interpretations of paper ballot records by a computer. They frequently, but not always, contain an interpretation of all voter choicdes in all contests on a ballot. CVRs are sometimes syntactically written as comma-separated values, and other times in plain English. Some CVRs use a election-specific encoding scheme to represent choices (e.g., a '1' means "John Doe"); others use plain English.

CDOS requirements mandate that CVRs are exported and uploaded to the RLA back-end as CSV files by county officials using their voting systems and the RLA Tool, respectively.

```
cast_vote_records: THEORY
BEGIN
  IMPORTING vote, information_systems, elections_equipment

  cvr: TYPE = set[vote]
  verified?: pred[cvr]
  as_csv: [cvr -> csv]
  cvr_file: TYPE FROM file
  export_cvr: [voting_system -> cvr_file]

  cvr_number: TYPE = nat
```

```
END cast_vote_records
```

12 User Interfaces

The user interfaces of the system are the visible interactive parts of the application. There are three different user interfaces in the RLA Tool. The precise names of these interfaces are still under discussion; one is for CDOS personnel responsible for the audit at the state-level, one is for county personnel at the county-level, and one is for audit board members.

```
user_interface: DATATYPE
  WITH SUBTYPES uploading_interface,
                cvr_uploading_interface,
                county_auditing_interface,
                audit_adjudication_interface,
                public_interface
BEGIN
  uploading_interface?:
    make_uploading_interface: uploading_interface
  cvr_uploading_interface?:
    make_cvr_uploading_interface: cvr_uploading_interface
  county_auditing_interface?:
    make_county_auditing_interface: county_auditing_interface
  audit_adjudication_interface?:
    make_audit_adjudication_interface: audit_adjudication_interface
  public_interface?:
    make_public_interface: public_interface
END user_interface
```

After a county administrator attempts to upload artifacts to the RLA system's server, one of several different messages is shown. Each is self-explanatory in this domain model, and each must be used in a scenario of the system.

```
upload_system_message: DATATYPE
  WITH SUBTYPES upload_successful,
                hash_wrong,
                file_type_wrong,
                data_transmission_interrupted,
                too_late
BEGIN
  upload_successful?: make_upload_successful: upload_successful
```

```
  hash_wrong?: make_hash_wrong: hash_wrong
  file_type_wrong?: make_file_type_wrong: file_type_wrong
  data_transmission_interrupted?: make_data_transmission_interrupted:
    data_transmission_interrupted
  too_late?: make_too_late: too_late
END upload_system_message
```

Authentication attempts can result in two different kinds of message: either a person authenticated successfully, or they did not.

```
authentication_message: DATATYPE
  WITH SUBTYPES successful_authentication,
                unsuccessful_authentication
BEGIN
  successful_authentication?: make_successful_authentication:
    successful_authentication
  unsuccessful_authentication?: make_unsuccessful_authentication:
    unsuccessful_authentication
END authentication_message
```

13 Dashboards

The system has several dashboards that are the main user interfaces to the system. State officials use the state-wide dashboard; county officials use the county dashboard; audit board members use the audit board dashboard; and the general public uses the public dashboard.

```
dashboard: DATATYPE
  WITH SUBTYPES department_of_state_dashboard,
                county_dashboard,
                audit_board_dashboard,
                public_dashboard
BEGIN
  department_of_state_dashboard?: make_department_of_state_dashboard: department_of_state_da
  county_dashboard?: make_county_dashboard: county_dashboard
  audit_board_dashboard?: make_audit_board_dashboard: audit_board_dashboard
  public_dashboard?: make_public_dashboard: public_dashboard
END dashboard
```

13.1 State-wide Dashboard

The status of uploaded data will be summarized in a state-wide dashboard, along with information on which counties have not yet uploaded their CVRs, and uploads that have formatting or content issues. The status of data, and results as audits are performed, will be provided for each contest to be audited.

```
department_of_state_dashboard: THEORY
BEGIN
  IMPORTING dashboard, elections, rlas, fafessl,
            instructions_forms_reports
```

CDOS staff, after authenticating to the state-wide dashboard, can see the status of the entire election. Various static information about the election is displayed along with dynamic information, the type and content of which is dependent upon the current audit stage.

```
  audit_stage: TYPE =
    { initial, authenticated, risk_limits_set,
      contests_to_audit_identified, random_seed_published,
      ballot_order_defined, audit_ongoing, audit_complete,
      audit_results_published }
```

No later than 30 days before Election Day, the Secretary of State will establish and publish on the audit center the risk limit(s) that will apply in RLAs for that election. The Secretary of State may establish different risk limits for comparison audits and ballot polling audits, but in no event will the risk limit exceed five percent.

```
  establish_risk_limit_comparison_audits:
    [department_of_state_dashboard, {r: risk_limit | r <= 5} -> department_of_state_dashboa
  establish_risk_limit_ballot_polling_audits:
    [department_of_state_dashboard, {r: risk_limit | r <= 5} -> department_of_state_dashboa

  county_status: TYPE =
    { no_data, cvrs_uploaded_successfully, error_in_uploaded_data }
  upload_status: [department_of_state_dashboard -> set[county_status]]
```

13.1.1 Selection of Audited Contests

No later than 5:00 PM MT on the Friday after Election Day, the Secretary of State will select for audit at least one statewide contest, and for each county at least one countywide contest. The Secretary of State will select other ballot contests for audit if in any particular election there is no statewide contest

or a countywide contest in any county. The Secretary of State will publish a complete list of all audited contests on the audit center. The Secretary of State will consider the following factors in determining which contests to audit:

1. The closeness of the reported tabulation outcome of the contests;
2. The geographical scope of the contests;
3. Any cause for concern regarding the accuracy of the reported tabulation outcome of the contests;
4. Any benefits that may result from opportunistically auditing certain contests; and
5. The ability of the county clerks to complete the audit before the canvass deadline.

```
audit_reason: TYPE =
  { state_wide_contest, county_wide_contest, close_contest,
    geographical_scope, concern_regarding_accuracy,
    opportunitic_benefits, county_clerk_ability }

select_contest_for_comparison_audit:
  [contest, audit_reason -> audited_contest]
at_least_one_statewide_contest: AXIOM TBD
at_least_one_countywide_contest_per_county: AXIOM TBD

select_contest_for_opportunistic_audit:
  [contest, audit_reason -> opportunistic_contest]
```

Ballots can be randomly selected for audit in two ways, either by:

1. permuting all ballots and auditing a prefix of ballots (thereby auditing ballots "with no replacement"); or
2. randomly selecting ballots ballot-by-ballot (thereby auditing ballots "with replacement" as the same ballot may be audited multiple times).

```
ballot_permutation:
  [set[ballot] -> list[ballot]]
random_list_of_ballots:
  [set[ballot] -> sequence[ballot]]

print_ballots_under_audit_list:
  [sequence[ballot] -> ballots_under_audit_instructions]
```

13.2 Random Selection of Ballots for Audit

The Secretary of State will randomly select the individual ballots to audit. The Secretary of State will use a pseudo-random number generator with the seed established under subsection (H) of this rule to identify individual ballots as reflected in the county ballot manifests. The Secretary of State will notify each county of, and publish on the audit center, the randomly selected ballots that each county must audit no later than 11:59 PM MT on the tenth day after Election Day.

```
  public_ballots_to_audit:
    [department_of_state_dashboard, list[ballot] -> department_of_state_dashboard]
```

The Secretary of State can indicate that a contest must be a full hand count contest.

```
  designate_full_hand_count_contest:
    [department_of_state_dashboard, contest -> department_of_state_dashboard]

END department_of_state_dashboard
```

13.3 County Dashboard

The County dashboard is used by county officials to communicate with the Secretary of State for the purpose of planning and executing risk-limiting audits.

```
county_dashboard: THEORY
BEGIN
  IMPORTING dashboard, information_systems, upload_system_message,
            election, roles, cast_vote_records, ballot_manifests,
            cryptography, server
```

Some of the generation information contained in the county dashboard is stipulated by C.R.S.

```
  general_static_information: string


  establish_audit_board:
    [county_dashboard, set[elector] -> county_dashboard]
```

13.3.1 Verification of Election Report Artifacts

To prepare for uploading of artifacts to the Secretary of State, counties conducting a comparison audit must verify several properties (all of which are discussed in section 16). After verifying those properties, counties must generate a digest of the CVR file using a hash designated by the Secretary of State.

13.3.2 Digest Generation

After verifying the accuracy of the CVR export, the county must apply a hash value to the CVR export file using the hash value utility provided by the Secretary of State.

Note that this feature is implemented by a tool provided by the Secretary of State. We have not been asked to produce such a tool, though one could compute the hash of a file locally in a web browser, so this could be part of our system. From an assurance standpoint it is a better idea to use a completely separate tool, developed independently from us, to perform this hashing.

```
generate_cvr_digest: [cvr_file -> digest]
```

Each county performing a comparison audit must upload a hash (digest) of its ballot manifest to the RLA Tool.

```
generate_ballot_manifest_digest: [ballot_manifest_file -> digest]
```

13.3.3 Comparison Audit Uploads

Each county conducting a comparison audit must upload:

1. its verified and hashed ballot manifest to the RLA Tool;

   ```
   county_upload_verified_ballot_manifest:
     [county_dashboard, ballot_manifest_file, digest ->
      [county_dashboard, email, upload_system_message]]
   ```

2. its verified and hashed CVR export to the RLA Tool; and

   ```
   upload_verified_cvrs:
     [county_dashboard, cvr_file, digest ->
      [county_dashboard, email, upload_system_message]]
   ```

3. its RLA tabulation results export to the Secretary of State's election night reporting system.

```
upload_tabulation_results:
  [enr_server, rla_tabulation_outcome -> enr_server]
```

13.3.4 Ballot Polling Audit Uploads

Mo later than 11:59 PM MT on the ninth day after Election Day, each county conducting a ballot polling audit must upload:

1. its verified and hashed ballot manifest to the RLA tool; and
2. its RLA tabulation results export to the Secretary of State's election night reporting system.

Note that both of these uploads are facilitated by the functions defined above, as their types do not mandate a particular kind of audit on the county.

```
END county_dashboard
```

13.4 Audit Board Dashboard

There are a number of assumptions that the RLA Tool audit board dashboard makes with respect to the C.R.S. It also facilitates the comparison audit.

```
audit_board_dashboard: THEORY
BEGIN
  IMPORTING dashboard, ballots, storage_container, audits,
            cast_vote_records, instructions_forms_reports
```

The audit board must locate and retrieve from the appropriate storage container each randomly selected ballot. The audit board must verify that the seals on the appropriate storage containers are those recorded on the applicable chain-of-custody logs.

```
  ballots_to_audit_to_storage_container_list:
    [list[ballot] -> list[storage_container]]
  verify_all_seals_on_storage_containers:
    [list[storage_container] -> list[storage_container]]
```

The audit board must examine each randomly selected ballot or VVPAT and report the voter markings or choices using the RLA Tool or other means specified by the Secretary of State.

```
  next_ballot_for_audit:
    [audit_board_dashboard ->
     [ballot_manifest_info, audit_board_dashboard]]

  report_markings:
    [audit_board_dashboard, ballot_manifest_info,
     list[ballot_mark] -> audit_board_dashboard]
```

```
report_ballot_not_found:
  [audit_board_dashboard, phantom_ballot -> audit_board_dashboard]
```

If a ballot does not have a voter-verifiable paper ballot associated with it then the Audit Board reports the lack of voter-verifiable paper ballot.

```
report_ballot_has_no_voter_verifiable_paper_record:
  [audit_board_dashboard, non_voter_verifiable_ballot ->
   audit_board_dashboard]
```

> If supported by the county's voting system, the audit board may refer to the digital image of the audited ballot captured by the voting system in order to confirm it has retrieved the correct ballot randomly selected for audit. If the scanned ballot was duplicated prior to tabulation, the audit board must also retrieve and compare the markings on the original ballot. The audit board must complete its reports of all ballots randomly selected for audit no later than 5:00 PM MT one business day before the canvass deadline.
>
> The audit board must interpret voter markings on ballots selected for audit in accordance with the Secretary of State's voter intent guide.
>
> To the extent applicable, the Secretary of State will compare the audit board's reports of the audited ballots to the corresponding CVRs and post the results of the comparison and any margin overstatements or understatements on the audit center.

```
compare_reported_markings_to_cvr:
  [ballot, cvr, list[ballot_mark] ->
   [margin_overstatement, margin_understatement]]
```

The RLA will continue until the risk limit for for each audited contests is met or until a full hand count results. If the county audit reports reflect that the risk limit has not been satisfied in an audited contest, the Secretary of State will randomly select additional ballots for audit. We presume at the moment that if errors are made during the auditing process, we should capture information in the RLA Tool about those errors and their mitigation and resolution.

```
submit_audit_investigation_report:
  [audit_board_dashboard, audit_investigation_report ->
   audit_board_dashboard]

submit_audit_report:
  [audit_board_dashboard, audit_report -> audit_board_dashboard]

signoff_intermediate_audit_report:
```

```
    [audit_board_dashboard, audit_report -> audit_board_dashboard]

END audit_board_dashboard
```

13.5 Public Dashboard

```
public_dashboard: THEORY
BEGIN
  IMPORTING dashboard
```

We are currently proposing that the following set of data and reports be included on the public dashboard:

1. Target and Current Risk Limits, by Contest
2. Audit Board names by County
3. County Ballot Manifests, CVRs and Hashes (status & download links)
4. Seed for randomization
5. Ballot Order
6. List of Audit Rounds (number of ballots, status by County, download links)
7. Link to Final Audit Report

```
END public_dashboard
```

14 Authentication

Authentication is currently underspecified in our system design as we do not yet have information from CDOS on the nature and kind of their mandated two-factor authentication system. Consequently, we have only modeled the necessary concepts and features of any two-factor authentication system. Included in this model are the ideas of usernames, passwords, other authentication factors mentioned in CDOS documents (such as biometrics and physical tokens like smartcards and one-time authentication code books), etc.

Given that CDOS is handling two-factor authentication, it is unclear if they want any additional features such as a password reset.

```
authentication: THEORY
BEGIN
  IMPORTING roles, dashboard, election, dashboard,
            information_systems, upload_system_message, ballots,
            cryptography, cast_vote_records

  credential: TYPE
  username: TYPE FROM credential
  password: TYPE FROM credential
  complex_enough?: [password -> bool]
  biometric: TYPE FROM credential
  physical_token: TYPE FROM credential

  authentication: TYPE FROM [person, set[credential]]
  two_factor_authentication: TYPE =
    [[username, password], [physical_token + biometric]]
```

In order to obtain a new, valid credential, some authority must issue credentials to a specific person.

```
  issue_credential: TYPE =
    [cdos, state_election, person -> [person, two_factor_authentication]]
```

What are the credentials that have been issued to this person?

```
  credential?: [person -> two_factor_authentication]
```

Is this person authenticated?

```
  authenticated: [person, two_factor_authentication -> bool]
```

What follows are several features used to authenticate various roles to their respective dashboards.

Each dashboard has an abstract state machine that captures the dashboard's workflow and consequntly identifies which features are visible to its users at various stages. Those abstract state machines and features are modeled in the dashboard modules below. We focus only on authentication here.

```
  authenticate_county_administrator:
    [county_dashboard, county_administrator,
     two_factor_authentication -> bool]
  authenticate_state_administrator:
    [department_of_state_dashboard, state_administrator,
     two_factor_authentication -> bool]
END authentication
```

15 System Architecture

As mentioned early in this chapter, the RLA Tool is a client-server system. As usual for these kinds of systems, the server part of this architectural style is known as the back-end and the client part as the front-end.

```
system_architecture_component: DATATYPE
  WITH SUBTYPES back_end,
                front_end
BEGIN
  IMPORTING information_systems, database, server

  back_end?(servers: set[server],
            networks: set[network],
            databases: set[database]): make_back_end: back_end
  front_end?(web_browser: web_browser,
             code: javascript_code): make_front_end: front_end
END system_architecture_component
```

The RLA system focuses on a single election and has a back-end and a front-end.

```
rla_tool: DATATYPE
BEGIN
  IMPORTING election, system_architecture_component

  rla_tool?(election: state_election,
            front_end: front_end,
            back_end: back_end): make_rla_tool
END rla_tool
```

The system architecture consists of:

1. several servers of different kinds deployed and configured in a redundant fashion as described elsewhere;
2. several databases whose tables and data are transactionally identical (this means that after each transaction completes, all databases are guaranteed to never witness to a client an inconsistent state relative to that transaction);
3. a JavaScript-based front-end whose code comes only from the system's web servers; and

4. HTTPS-based connections between the front-end and the back-end over which the web application transmits data, including new HTML pages, style sheets, data input by the web browser user, etc.

```
system_architecture: THEORY
BEGIN
  IMPORTING system_architecture_component, server, rla_tool

  rla: VAR rla_tool
  be: VAR back_end
  fe: VAR front_end
  db1, db2: VAR database_server

  make_rla_tool: [back_end, front_end -> rla_tool]

  javascript_code?: [front_end -> javascript_code]
  code_origins?: [javascript_code -> web_server]
```

The front-end, which is written in TypeScript and JavaScript, is provided to a web browser client directly from the web server.

```
  browser_code_origins: AXIOM
    (FORALL (rla):
      member(code_origins?(code(fe)), servers(be))
      WHERE fe = front_end(rla), be = back_end(rla))

  transactionally_synchronized?: [set[database] -> bool]
```

The back-end consists of two or more servers, two or more networks, and two or more databases. At least two of each is necessary because the system must have redundancy to be fault tolerant and not have a single point of failure.

```
  redundancy: AXIOM
    (FORALL (be): 2 <= card(servers(be))
              AND 2 <= card(networks(be))
              AND 2 <= card(databases(be)))
```

Multiple databases need multiple synchronized database servers among the back-end's servers. Note that our model is simplified here insofar as we are presuming that all databases across synchronized database servers are synchronized. It is certainly possible to deploy synchronized databases in a fashion that does not fulfill this requirement, but for all databases relevant to this system, this requirement does hold.

```
  database_redundancy: AXIOM
    (FORALL (be): EXISTS (db1, db2):
```

```
      member(db1, servers(be)) AND member(db2, servers(be))
  AND db1 /= db2
  AND transactionally_synchronized?(union(databases(db1),databases(db2))))
END system_architecture
```

16 System Assumptions

This system architecture includes a number of explicit assumptions derived from C.R.S.

```
system_assumptions: THEORY
BEGIN
  IMPORTING election, fafessl

  e: VAR state_election
```

According to the section entitled "CVR Export Verification", counties conducting a comparison audit must verify that:

1. The number of individual CVRs in its CVR export equals the aggregate number of ballots reflected in the county's ballot manifest as of the ninth day after election day;

   ```
   CVR_count_equals_ballot_manifest_count: AXIOM TBD
   ```

2. The number of individual CVRs in its CVR export equals the number of ballots tabulated as reflected in the summary results report for the RLA tabulation;

   ```
   CVR_count_equals_summary_results_report_count: AXIOM TBD
   ```

3. The number of individual CVRs in its CVR export equals the number of in-person ballots issued plus the number of mail ballots in verified-accepted stage in SCORE, plus the number of provisional ballots and property owner ballots included in the RLA tabulation, if any; and

   ```
   CVR_count_equals_aggregate_count_over_ballot_kinds: AXIOM TBD
   ```

4. The vote totals for all choices in all ballot contests in the CVR export equals the vote totals in the summary results report for the RLA tabulation.

```
CVR_vote_totals_equals_summary_results_vote_totals: AXIOM TBD
```

```
END system_assumptions
```

17 System Logging

The system must log a variety of events. One reason to log information is to help understand how the system is operating, fix bugs post-facto, understand how users are using the system, etc. Another, reason to log in this context of this system is to provide an indelible record of administrator and auditor actions so that any audit can be "replayed" by any third party.

```
logging: THEORY
BEGIN
  log: TYPE = sequence[string]
  chain_of_custody_log: TYPE FROM log
  rla_tool_log: TYPE FROM log
END logging
```

18 Data Model

A single, mirrored relational database is used to store all persistent information for the RLA Tool. In order to define the data model of the system, we need to:

1. identify the *core concepts* that must be persistent in the system
2. identify the *relationships* that must be persistent in the system
3. derive from these pieces of information:
   (a) the necessary *tables* that *collect concepts*,
   (b) the *indexing concept* of each table, and
   (c) definitions of the appropriate *joins* that realize relationships

```
data_model: THEORY
BEGIN
  IMPORTING fafessl,
            database,
            information_systems,
            roles,
            rla_tool,
            authentication,
            rlas,
            department_of_state_dashboard,
            cast_vote_records

  be: VAR back_end
  db: VAR database
```

The first set of data persistently stored relates to authentication. Some of this data will be stored in the two-factor authentication system provided by CDOS. All other data must be stored in the RLA Tool database. In general, each entry in the authentication table is simply an administrator and two-factor authentication credential pair.

```
  authentication_table: TYPE =
    sequence[[administrator, two_factor_authentication]]
```

From the State's point of view, the State-wide dashboard permits State administrators to establish risk limits for each contest, specify which kinds of audits are used for which races and in which counties, etc. Most of the dynamic data available to State administrators is provided via County administrators using the County dashboard.

```
  state_status_table: TYPE =
    sequence[[# county: county,
                general_status: county_status,
                audit_status: audit_stage,
                date: string #]]
```

There is also some background information that relates to the RLA overall, much of which is relevant to the State-wide dashboard. In particular, geographic (such as the identity of the State and its Counties) and political (registered political parties) information must be stored, as must information about each election under audit.

Notice that these tables are simply a flattening of the election datatype. This is a standard pattern in mapping HOL specifications to relational data models.

```
geography_table: TYPE =
  sequence[[nation, state, county]]
political_party_table: TYPE =
  sequence[[political_party]]
national_contest_table: TYPE =
  sequence[[nation, contest]]
state_contest_table: TYPE =
  sequence[[state, contest]]
county_contest_table: TYPE =
  sequence[[county, contest]]

public_meeting_to_determine_seed_table: TYPE =
  [# location: string, date: string #]
rla_information_table: TYPE =
  sequence[[RLA, audit_reason, contest, risk_limit]]
ballots_to_audit_list_table: TYPE =
  sequence[[contest, sequence[ballot_manifest_info]]]
```

County administator actions create and update several kinds of data. For example, they define who is on their audit boards, they generate digests of files critical to the audit (principly, ballot manifests and CVRs), they upload those files, etc.

For general county information, we currently model the data as property/value pairs in a record. For example, we imagine that each county will want to display who their County Clerk is, etc.

```
county_general_information_table: TYPE =
  sequence[[# property: string, value: string #]]
audit_board_table: TYPE =
  sequence[[county, audit_board_member]]
ballot_manifest_digest_table: TYPE =
  sequence[[# county: county,
              ballot_manifest_file_name: string,
              digest: string,
              data: ballot_manifest_file #]]
cvr_digest_table: TYPE =
  sequence[[# county: county,
              cvr_file_name: string,
              digest: string,
              data: cvr_file #]]
ballot_manifest_table: TYPE =
  sequence[[county, ballot_manifest_info]]
```

Cast vote records uploaded by counties contain CVRs of three kinds: CVRs for local (county) contests, CVRs for contests that span counties, and CVRs for

contests that span the whole state. These latter two cases are identical from a modeling point of view, as the whole state is simply a spanning contest over all counties.

We can store and organize this information in several different ways. First, we can simply store the individual CVRs uploaded by a given county in a table.

```
cvr_table: TYPE =
  sequence[[county, cvr]]
```

But we also need to store all CVRs for each contest, aggregating all CVRs across the upload from all counties in which the contest was on the ballot.

```
contest_cvr_table: TYPE =
  sequence[[contest, cvr]]
```

Finally, the tabulation results are uploaded by the counties, and is the key information necessary in order to run the RLA algorithm.

```
tabulation_result_table: TYPE =
  sequence[[contest, rla_tabulation_outcome]]
```

While running the RLA, the Audit Board uploads information about each ballot they audit. Also, if any investigations are made and remedied during the audit, that information must be stored as well. Finally, after an audit is complete an audit report is uploaded and stored.

```
ballots_under_audit_table: TYPE =
  sequence[[ballot, ballot_manifest_info, list[ballot_mark],
            cvr, margin_overstatement, margin_understatement]]
auditing_investigations_table: TYPE =
  sequence[[county, audit_investigation_report]]
audit_reports_table: TYPE =
  sequence[[county, audit_report]]
rla_summary_results_table: TYPE =
  sequence[[contest, summary_results_report]]
```

All of the information on the public dashboard is derived from the above data.

```
END data_model
```