1. Define Copland protocol in Coq specification
   * Many example phrases appear in src/Example\_Phrases.v
   * Easy way to add a phrase that fits into the code extraction workflow:
     + Add appropriate phrase parameter admits to src/Example\_Phrases\_Admits.v (e.g. P0, attest\_id, att\_tid, …)
     + Define phrase at the end of Example\_Phrases.v (using imported parameters from src/Example\_Phrases\_Admits.v)
2. Extract Coq protocol term to Cakeml
   * Add the newly-defined phrase from step above to src/Extraction\_Cvm\_Cake.v . The best place to add this is in the list of extracted terms called “term\_list” (other terms extracted include cert\_style, par\_mut\_p0, etc. )
   * Type “make” to rebuild the Coq sources (performs cakeml extraction upon successful compilaiton).
3. Bring extracted term over to am-cakeml
   * At the top-level of the am-cakeml repo, run the script ./do\_extract.sh
     + If your “COPLAND\_AVM\_DIR” environment variable is not set to point at your local Coq repo, an error will indicate this
   * Type “git status” in the am-cakeml repo to ensure your newly extracted phrase shows up as an unstaged source change (changes should be in i.e. “extracted/Example\_Phrases\*.cml”).
4. Fill in cakeml parameter stubs
   * If you try to compile the cakeml code right away, the compiler will complain about undefined variables for the new protocol parameter stubs.
   * You will need to add these stubs to i.e. “stubs/Example\_Phrases\_Admits.sml”.
     + Examples of these appear in this file
   * Recompile the am-cakeml to confirm absence of undefined variable errors
5. Prepare a directory with configuration files for running the new protocol
   * Create a new directory in “tests/DemoFiles” to hold config files
     + Easy way to do this: copy an existing directory from tests/DemoFiles (i.e. run: “mkdir tests/DemoFiles/NewProtocol; cp -r tests/DemoFiles/Cert/\* tests/DemoFiles/NewProtocol” )
     + Clean up extraneous files in new directory (tests/DemoFiles/NewProtocol/) – These will be auto-genned for the new protocol in subsequent steps
       - rm \*\_EXE
       - rm \*.json
       - rm \*Terms\*.sml
       - Should be left with files like this:
         * \*\_Am\_Lib\_\*.sml
         * \*\_PrivKey
6. Implement any newly-defined ASP stubs
   * Add asp function to its own file in “stubs/attestation\_asps
     + Use “cert\_asp\_stub” in stubs/attestation\_asps/cert\_ASP\_Stub.sml as a template
   * Update CMake config: “stubs/attestation\_asps/CMakeLists.txt”
     + Add stubs/attestation\_asps/<newStubFilel>.sml to the get\_files(attestation\_asps …) list.
   * Recompile the am-cakeml to ensure no type errors in the new stub function
7. Create new AM Library and link new ASP stubs
   * Copy an existing AM Library source file:
     + cp tests/DemoFiles/NewProtocol/\*\_Am\_Lib\_\*.sml

tests/DemoFiles/NewProtocol/\*\_Am\_Lib\_NewProtocol.sml

* + Edit this new AM Library to add a new mapping in the “aspMapping” value.
    - Mapping is from ASP ID (i.e. cert\_id) to ASP stub (i.e. cert\_asp\_stub)
    - Easy way: Copy the “cert\_id” mapping and just replace the ASP ID (cert\_id) and stub function (cert\_asp\_stub) with the new ones
    - NOTE: Be sure to add a comma before (but not after!) this new entry (mappings are lists of pairs under the hood).
  + OPTIONAL: If you want to support automatic appraisal of this new ASP, repeat steps 6) and 7) for its dual appraisal ASPs.
    - NOTE: Minor differences for appraisal stubs vs attestation stubs:
      * Update files in stubs/appraisal\_asps for step 6)
      * Edit the “appAspMapping” value in step 7)

1. Prepare tmux script
   * Copy an existing bash script from tests/ to use as a template
     + i.e. “cp tests/manifest\_demo\_cert.sh tests/manifest\_demo\_<newProtocol>.sh”
   * Edit this file to configure and run the new protocol
     + Edit “DEMO\_FILES” variable to point to the new configuration directory:
       - i.e. …Cert 🡪 …NewProtocol
     + Add an appropriate number of Formal Manifest pointers (for how many AM servers are involved in the new protocol)
       - Update SERVER\_P\*\_FORM\_MAN variables and maintain naming convention of “FormalManifest\_P\*.json”
       - NOTE: If support for a dedicated appraisal server is required, include a formal manifest for place P3
     + Update pointer to new AM Library (created in step 7) in SERVER\_AM\_LIB variable
     + Add appropriate number of Server executable names to variables: SERVER\_P\*\_EXE\_NAME
     + Repeat above for the CLIENT variables
       - NOTE: it is usually fine to re-use server variables for this:
         * CLIENT\_FORM\_MAN=$SERVER\_P0\_FORM\_MAN
         * CLIENT\_AM\_LIB=$SERVER\_AM\_LIB
     + Rename CLIENT\_TERM\_FILE\_JSON variable
       - $DEMO\_FILES/ClientCvmTerm<blah>.json 🡪 $DEMO\_FILES/ClientCvmTermNewProtocol.json
     + Change MANGEN\_TERMS\_FILE and MANGEN\_EVS\_FILE variables similarly
     + Add/Remove an appropriate number of “$MAN\_COMP -s -o …” lines to compile the servers
     + Add/Remove an appropriate number of “$BUILT\_SERVER\_AM\_P\*” variables to name the server executables.
     + Same for the “$BUILT\_CLIENT\_AM\_\*” client variables
     + What remains is to set up the tmux windows and start the AM executables.
       - Depending on how many AMs are involved, you may need to Add/Remove “tmux split-window -v ‘bash -I’” lines to create enough windows in the tmux terminal session.
       - Similarly, you’ll need to start each AM by giving a “tmux send-keys -t n …” command, with increasing numbers of n starting at 0.
2. Provision Manifest Generator executable
   * Configure example protocol variables in stubs/ManGenConfig.sml
     + Add variable pointing to newly extracted Copland phrase (extracted to extracted/Example\_Phrases.sml in step 3)
       - val <newProtocol>\_phrase = <newProtocol>
     + Add variable that is a list of Coq pairs (of the “(Coq\_Term \* Coq\_Plc) prod” Coq extracted type). This tells the manifest generator to support execution of the new term at the given top-level place
       - val <newProtocol>\_phrases = [(Coq\_pair <newProtocol>\_phrase coq\_P0)]
   * Configure appraiser evidence variables in stubs/ManGenConfig.sml
     + Add variable that computes the expected evidence type resulting from execution of the new protocol (via the “eval” evidence semantics function)
       - val appraiser\_evidence\_<newProtocol> = eval <newProtocol>\_phrase coq\_P0 (Coq\_nn 0)
       - NOTE: At the moment, all protocols are assumed to start at top-level place P0 and use a nonce (Coq\_nn 0) as initial evidence
     + Add variable that is a list of Coq pairs “(Coq\_Evidence \* Coq\_Plc) prod”
       - val ets\_<newProtocol> = [(Coq\_pair appraiser\_evidence\_<newProtocol>, coq\_P0), (Coq\_pair appraiser\_evidence\_<newProtocol>, coq\_P3)]
       - NOTE: The second entry in this list (for coq\_P3) is because P3 has been configured as a designated “appraisal server” for the current demo scripts.
   * Edit ManifestGenerator main source file (apps/ManifestGenerator/ManifestGenerator.sml)
     + Search for “if(provisioningBool)”
     + Edit the plcTerms and plcEts variables in the “then” branch to point to the protocol and evidence variables defined in stubs/ManGenConfig.sml
       - val plcTerms = ManGenConfig.<newProtocol>\_phrases
       - val plcEts = ManGenConfig.ets\_<newProtocol>
   * Add provisioning CLI flag (-p) to Manifest Generator executable command in tests/manifest\_demo\_<newProtocol>.sh
     + $MAN\_GEN … -p
3. Provision Client AM

* Add client phrase to stubs/ManGenConfig.sml (this should already be there as the phrase from the Manifest Generator provisioning)
* Edit Client AM main source file (apps/ManifestCompiler/Client.sml)
  + Search for “if(provisioningBool)”
  + Edit “provisioningPhrase” variable
    - val provisioningPhrase = ManGenConfig.<newProtocol>\_phrase
* Add provisioning CLI flag (-p) to Client AM executable command in tests/manifest\_demo\_<newProtocol>.sh
  + “$BUILT\_CLIENT\_AM\_\* … -p”

1. Run demo script to provision Manifest Generator and Client AM executables
   * Navigate to tests/ (cd tests/)
   * Run the new script: ./manifest\_demo\_<newProtocol>.sh
   * Check that this generated 3 new json congiruration files in tests/DemoFiles/<newProtocol>/ :
     + For the Manifest Generator:
       - ServerCvmTerms<newProtocol>.json
       - ServerPlcEvs<newProtocol>.json
     + For the Client AM:
       - ClientCvmTerms<newProtocol>.json
2. Remove the -p CLI args for Manifest Generator and Client executable commands in tests/manifest\_demo\_<newProtocol>.sh script
3. Run the demo script again