High Level Modelling

February 25, 2020

1 Abstracting the implementation detail

1.1 Abstract Data Type

```
KeyId : Type (* Abstract Type *)
Fingerprint : Type (* Abstract Type *)
Identity : Type (* Abstract Type *)
Token : Type (* Abstract Type *)
(* Key is record data type *)
Key : Type = Record {
  keyId : KeyId
  fingerprint : Fingerprint
  identities : Identity set}
```

I am slightly confused between **sealed trait** and **case class**

- case class Identity(email: String)
- case class Token(uuid: UUID)

1.2 Maps as Function Type

```
keys : Fingerprint -> Key
uploaded : UToken -> Fingerprint
pending : VToken -> (Fingerprint, Identity)
confirmed : Identity -> Fingerprint
managed : MToken -> Fingerprint
```

1.3 Function Definition

```
byFingerprint (f : Fingerprint) :=
   ( In f (dom keys) -> Exists (v : Key), Some v /\ v = keys f) /\
   (~In f (dom keys) -> None)
```

```
byKeyId (keyId : KeyId) :=
  Exists (l : List[Key]), (forall (x : Key), In x l ->
     Exists (f : Fingerprint), keys f = x) /\
     (forall (f : Fingerprint), (keys f).keyId = keyId -> In (keys f) l)



upload (key : Key) :=
  Forall fingerprint key' uploaded', fingerprint = key.fingerprint ->
  (In key.fingerprint (dom keys) ->
        keys (key.fingerprint) = key  /\ Exists (token : UToken),
        fresh t  /\ keys' = keys [t := key.fingerprint]  /\
        uploaded' = uploaded [token := key.fingerprint]) /\
  (~In key.fingerprint (dom keys) -> keys' = key /\ uploaded' = uploaded)


This function mimics the for loop of requestVerify function.
uploadBunch (identites : Identity set)
  (fingerprint : Fingerprint) (f : Identity -> VToken)
  (pending : VToken -> (Fingerprint, Identity)) :=
  match identities with
  | [] => pending
  | h :: identities' =>
      fresh (f h) /\
      uploadBunch identities' fingerprint
                  f (pending [(f h) := (fingerprint, h)])

requestVerify (from : UToken) (identities : Identity set) :=
  Forall pending',
  (In from (dom uploaded) ->
      Forall fingerprint key, fingerprint = uploaded from ->
        key = keys fingerprint ->
        (Subset identiteis key.identities -> Exists (f : Identity -> VToken),
           pending' = uploadBunch identities fingerprint f pending) /\
        (~Subset s key.identities -> pending' = pending)) /\
 (~In from (dom uploaded) -> pending' = pending)



verify (token : VToken) :=
  Forall pending' confirmed',
  (In token (dom pending) -> Forall fingerprint identity,
       (fingerprint, identity) = pending token ->
```

```
          pending'   = deleteToken pending token /\
          confirmed' = confirmed [identity := fingerprint]) /\
  (~In token (dom pending) -> pending' = pending /\ confirmed' = confirmed)


requestManage (id : Identity) :=
  Forall managed',
  (In id (dom managed) -> Exists fingerprint token,
         fresh token /\  fingerprint = confirmed id /\
         managed' = managed [token := fingerprint]) /\
  (~In id (dom managed) -> managed' = managed)


requestManage (id : Identity) :=
   In id (dom confirmed) -> Exists token, fresh token /\
   fingerprint = confirmed id /\
   managed' = managed [token := fingerprint]



revoke (token : MToken) (identities : Identity set) :=
  Forall confirmed',
  (In token (dom managed) ->  Forall fingerprint key,
    fingerprint = managed token -> key = keys fingerprint ->
     ( Subset identities key.identities ->
            confirmed' = dubMinus confirmed identities) /\
     (~Subset identities key.identities ->
            confirmed' = confirmed)) /\
  (~In token (dom managed) -> confirmed' = confirmed)




ms -- ks
The map containing all mappings of ms except for any mapping with a key in ks.
dubMinus (ms ks) :=
  Exists tf, forall x, In x (dom tf) -> _
```