Modeling and analyzing the Corona-virus warning app with the Isabelle Infrastructure framework

Florian Kammüller and Bianca Lutz

Middlesex University London and
Technische Universität Berlin
`f.kammueller@mdx.ac.uk|bialut@gmail.com`

Abstract. We provide a model in the Isabelle Infrastructure framework of the recently published German Corona-virus warning app. The app supports breaking infection chains by informing users whether they have been in close contact to an infected person. The app has a decentralized architecture that supports anonymity of users. We provide a formal model of the existing app with the Isabelle Infrastructure framework to show up some natural attacks in a very abstract model. We then use the security refinement process of the Isabelle Infrastructure framework to highlight how the use of continuously changing ephemeral ids improves the anonymity.

1 Introduction

The German Chancellor Angela Merkel has strongly supported the publication of the mobile phone Corona warning app (CWA; [3]) by publicly proclaiming that the "Corona App deserves your trust" [1]. Many millions of mobile phone users in Germany have downloaded the app with 6 million on the first day. CWA is one amongst many similar applications that aim at the very important goal to "break infection chains" by providing timely information to users of whether they have been in close proximity to a person who tested positive for COVID-19.

The app was designed with great attention on privacy: a distributed architecture [4] has been adopted after a long and heated debate with supporters of a central architecture. The distributed architecture is based on a very clever application design whereby users' phones broadcast highly anonymized so called "Ephemeral IDs" at physical locations via the Bluetooth Low Energy Beacon protocol. The app saves those IDs of people in close proximity. When at a later date an infected person reports his infection to a central server, all clients are provided the means to reconstruct the IDs used by this particular person's device over the last 14 days. Therefore, any client can evaluate exposure risks locally and stored contact data has never to be shared.

The Isabelle Infrastructure framework [15] allows modeling and analyzing architecture and scenarios including physical and logical entities, actors, and policies within the interactive theorem prover Isabelle supported with temporal logic, Kripke structures, and attack trees. It has been applied for example to Insider analysis in airplanes [16], privacy in IoT healthcare [8], and recently also to blockchain protocols [14].

The technical advantage of modeling an application in the Isabelle Insider framework lies in (a) having explicit representations of infrastructures, actors and policies in a formal model that (b) allows additional automated verification of security properties with CTL, Kripke structures and Attack Trees within the interactive theorem prover Isabelle.

The app now in use in Germany has been developed based on a protocol proposed by the DP-3T project [5]: A sophisticated security concept conceived by experts in the field that has strong claims with regard to mathematical support [6] [p2]. However, there has been, as of yet, to our knowledge no formal verification been involved. Even if a "post-production" formal specification seems pointless, it allows to reveal weak points of the architecture, show that the measures that have been conceived are suitable to cover those weak points, or where and to what extend trade offs have to be made due to inherent vulnerabilities. Given that the protocol implemented by the framework CWA runs on resembles only the most basic of the three proposed by the DP-3T project, a formal verified model that stresses the impact or limits of certain security measures might help lend some more weight to appeals like [7] to adopt the more sophisticated protocols.

The contributions of this paper are (a) formal re-engineering of the Corona-virus warning app (b) providing an additional security and privacy analysis with interactive theorem proving certification of a novel view on the system architecture including actors, locations, and policies, (c) a formal definition of a security refinement process that allows to improve a system based on attacks found by the attack tree analysis and (d) an application of the refinement to improve security of the Corona-virus app specification.

In this paper, we first provide some background in Section 2: we give a detailed overview of the development history and relevant parts of the Corona-virus warning app (Section 2.1) We then introduce the Isabelle Infrastructure framework (Section 2.2). Next, we present our model (Section 3) and analysis of privacy and attacks (Section 3.4). The found attack on the first abstract specification motivates refinement. The formal definition of refinement for the Isabelle Infrastructure framework is introduced and illustrated on the Corona-virus warning app (Section 4) before drawing some conclusions (Section 5).

The formal model in the Isabelle insider framework is fully mechanized and proved in Isabelle (sources available [9]).

2 Background and related work

2.1 DP-3T and PEPP-PT

We are mainly concerned with the architecture and protocols proposed by the DP-3T (*Decentralized Privacy-Preserving Proximity Tracing*) project [5]. The main reason to focus on this particular family of protocols is the *Exposure Notification Framework* (ENF), jointly published by Apple and Google, that adopts core principles of the DP-3T proposal. This API is not only used in the German Corona-virus warning app but has the potential of being widely adopted in future app developments that might emerge due to the reach of players like Apple and Google.

There are, however, competing architectures noteworthy, namely protocols developed under the roof of the *Pan-European Privacy-Preserving Proximity Tracing* project (PEPP-PT) [18], e.g. PEPP-PT-ROBERT [19].

Neither DP-3T nor PEPP-PT are synonymous for just one single protocol. Each project endorses different protocols with unique properties in terms of privacy and data protection. Yet, on a higher level of abstraction, it seems feasible to distinguish two basic architectures: protocols as endorsed by PEPP-PT might be characterized as centralized architectures whereas DP-3T-inspired protocols follow a (more) decentralized approach.[1]

All DP-3T protocols roughly work the same: Upon installation, the app generates secret daily seeds to derive *Ephemeral IDs* (EphIDs) from them. EphIDs are generated locally with cryptographic methods and cannot be connected to one another but only reconstructed from the secret seed they were derived from.

During normal operation each client broadcasts its EphIDs via Bluetooth whilst scanning for EphIDs broadcasted by other devices in the vicinity. Collected EphIDs are stored locally along with associated meta-data such as signal attenuation and date. In DP-3T the contact information gathered is never shared but only evaluated locally.

If patients test positive for the Corona-virus, they are entitled to upload specific data to a central backend server. This data is aggregated by the backend server and redistributed to all clients regularly to provide the means for local risk scoring, i. e., determining whether collected EphIDs match those broadcasted by now-confirmed Corona-virus patients during the last, e. g., 14, days.

In the most simple (and insecure) protocol proposed by DP-3T this basically translates into publishing the daily seeds used to derive EphIDs from. The protocol implemented by ENF and, hence, CWA adopts this low-cost design [4]. DP-3T proposes two other, more sophisticated protocols that improve privacy and data protection properties to different degrees but are more costly to implement.

Figure 1 illustrates the basic system architecture along with some of the mitigation measures either implemented in CWA or proposed by DP-3T.

[1] DP-3T involves a central backend server. It is decentralized with regard to the collection and evaluation of contact information: In centralized architectures the server provides a risk scoring services, whereas decentralized approaches rely on local risk assessment and, thus, do not need to share contact information with the backend.

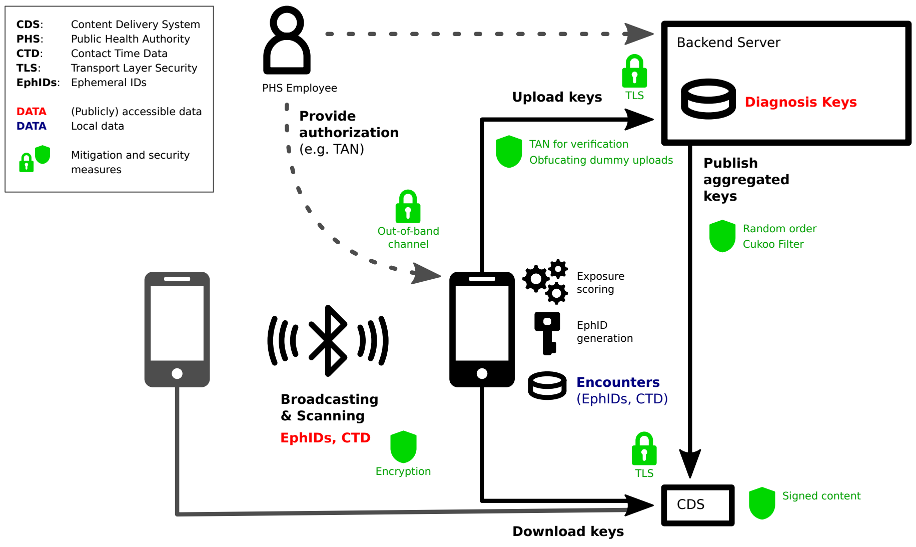

Fig. 1. Decentralized privacy-preserving proximity tracing protocol (as implemented by CWA)

2.2 Isabelle Infrastructure framework

The Isabelle Infrastructure is built in the interactive generic theorem prover Isabelle/HOL [17]. As a framework, it supports formalization and proof of systems with actors and policies. It originally emerged from verification of insider threat scenarios but it soon became clear that the theoretical concepts, like temporal logic combined with Kripke structures and a generic notion of state transitions were very suitable to be combined with attack trees into a formal security engineering process [2] and framework [11]. Figure 2 gives an overview of the Isabelle Infrastructure framework with its layers of object-logics – each level below embeds the one above showing the novel contribution of this paper in blue on the top.

Kripke structures, CTL, and Attack Trees The Isabelle framework has now after various case studies become a general framework for the state-based security analysis of infrastructures with policies and actors. Temporal logic and Kripke structures build the foundation. Meta-theoretical results have been established to show equivalence between attack trees and CTL statements [8]. This foundation provides a generic notion of state transition on which attack trees and temporal logic can be used to express properties. The main notions used in this paper are:

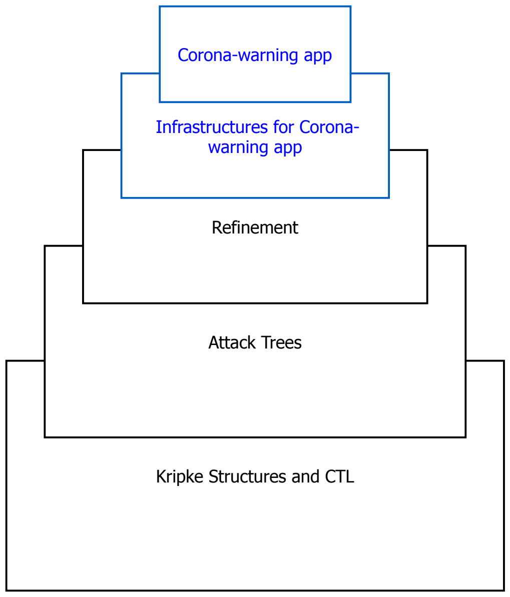
Corona-warning app
Infrastructures for Corona-warning app
Refinement
Attack Trees
Kripke Structures and CTL

Fig. 2. Generic Isabelle Infrastructure framework applied to Corona-warning app.

– Kripke structures and state transitions:
  Using a generic state transition relation $\mapsto$, Kripke structures are defined as a set of states reachable by $\mapsto$ from an initial state set, for example

  $\texttt{Kripke \{t. } \exists \texttt{ i} \in \texttt{I. i} \rightarrow^* \texttt{t\} I}$

– CTL statements:
  For example, we can write

  $\texttt{K} \vdash \texttt{ EF s}$

  to express that in Kripke structure `K` there is a path on which the property `s` (a set of states) will eventually hold.
– Attack trees:
  The datatype of attack trees has three constructors: $\oplus_\vee$ creates or-trees and $\oplus_\wedge$ creates and-trees. And-attack trees $l \oplus_\wedge^s$ and or-attack trees $l \oplus_\vee^s$ consist of a list of sub-attacks – again attack trees. The third constructor creates a base attack as a pair of state sets written $\mathcal{N}_{(\texttt{I,s})}$. For example, a two step and-attack leading from state set `I` via `si` to `s` is expressed as

  $\vdash [\mathcal{N}_{(\texttt{I,si})}, \mathcal{N}_{(\texttt{si,s})}] \oplus_\wedge^{(\texttt{I,s})}$

– Attack tree refinement, validity and adequacy:
  Attack trees have their own refinement (not to be mixed up with the system refinement presented in this paper as introduced in the next section). An

abstract attack tree may be refined by spelling out the attack steps until a valid attack is reached:
`⊢A :: (σ:: state) attree)`.
The validity is defined constructively (code is generated from it) and its adequacy with respect to a formal semantics in CTL is proved and can be used to facilitate actual application verification as demonstrated her in the stepwise system refinements.

Instantiation of Framework The formal model of the Corona-warning app uses the Isabelle Infrastructure framework instantiating it by reusing its concept of *actors* for users and smartphones whereby locations correspond to physical locations. The Ephemeral IDs, their sending and change is added to Infrastructures by slightly adapting the basic state type of infrastructure graphs and accordingly the semantic rules for the actions move, get, and put. The details of the newly adapted Infrastructure are presented in Section 3. Technically, an Isabelle theory file `Infrastructure.thy` builds on top of the theories for Kripke structures and CTL (`MC.thy`), attack trees (`AT.thy`), and security refinement (`Refinement.thy`). Thus all these concepts can be used to specify the formal model for the Corona-virus warning app, express relevant and interesting properties, and conduct interactive proofs (with the full support of the powerful and highly automated proof support of Isabelle). All Isabelle sources are available online [12].

Refinement An additional feature that has been integrated into the Isabelle Infrastructure framework motivated by security engineering formal specifications for IoT healthcare system is an extension of the formal specification process introducing refinement of Kripke structures [11,15]. It refines a system model based on a formal definition of a combination of trace refinement and structural refinement (or datatype refinement). The definition allows to prove property preservation results crucial for an iterative development process. The refinements of the system specification can be interleaved with attack analysis while security properties can be proved in Isabelle. In each iteration security qualities are accumulated while continuously attack trees scrutinize the design. One of the contributions of this paper is to explore different concepts of refinement: the formal expression of refinement, enables to pin down (i.e. exemplify) different concepts of refinement (data refinement, action refinement, trace refinement (aka spec refinement) and combinations thereof with concrete attack scenarios.

3 Modeling and analyzing Corona-warning app

3.1 Infrastructures, Policies, and Actors

The Isabelle Infrastructure framework supports the representation of infrastructures as graphs with actors and policies attached to nodes. These infrastructures are the *states* of the Kripke structure.

The transition between states is triggered by non-parametrized actions `get`, `move`, and `put` executed by actors. Actors are given by an abstract type `actor` and a function `Actor` that creates elements of that type from identities (of type `string` written `''s''` in Isabelle). Actors are contained in an infrastructure graph type `igraph` constructed by its constructor `Lgraph`.

```
 datatype igraph =
        Lgraph (location × location)set
                location ⇒ identity set
                identity ⇒ (string set × string set × efid)
                location ⇒ string × (dlm × data) set
                location ⇒ efid set
                actor ⇒ location ⇒ (identity × efid) set
```

In the current application of the framework to the Corona-warning app case study, this graph contains a set of location pairs representing the topology of the infrastructure as a graph of nodes and a functionthat assigns a set of actor identities to each node (location) in the graph. The third component of an `igraph` assigns the credentials to each actor: a triple-valued function whose first range component is a set describing the credentials in the possession of an actor and the second component is a set defining the roles the actor can take on; most prominently the third component is the `efid` assigned to the actor. This is initially just a natural number but will be refined to actually represent lists of "Ephemeral" Ids later when refining the specification.

```
 datatype efid = Efid nat
```

The fourth component of the type `igraph` assigns security labelled data to locations, a feature not used in the current application. The second to last component assigns the set of efids of all currently present smart phones to each location of the graph. The last component finally denotes the knowledge set of each actor for each location: a set of pairs of actors and potential ids.

Corresponding projection functions for each of the components of an infrastructure graph are provided; they are named `gra` for the actual set of pairs of locations, `agra` for the actor map, `cgra` for the credentials, `lgra` for the data at that location, `egra` for the assignment of current efids to locations, and `kgra` for the knowledge set for each actor for each location.

In the Corona-virus warning app, the initial values for the `igraph` components use two locations `pub` and `shop` to define the following constants (we omit the data map component `ex_locs`).

```
ex_loc_ass ≡ (λ x. if x = pub then {''Alice'', ''Bob'', ''Eve''}
                else (if x = shop then {''Charly'', ''David''}
                else {}))
ex_creds ≡ (λ x. if x = ''Alice'' then ({}, {}, Efid 1) else
              (if x = ''Bob'' then  ({},{}, Efid 2) else
              (if x = ''Charly'' then ({},{}, Efid 3) else
              (if x = ''David'' then ({},{}, Efid 4) else
              (if x = ''Eve'' then ({},{}, Efid 5)
               else ({},{},Efid 0))))))
ex_efids ≡ (λ x. if x = pub then {Efid 1, Efid 2, Efid 5}
                  else (if x = shop then {Efid 3, Efid 4} else {}))
ex_knos ≡ (λ x. (if x = Actor ''Eve'' then (λ l. {} else (λ l. {})))
```

These components are wrapped up into the following `igraph`.

```
ex_graph ≡
      Lgraph {(pub, shop)} ex_loc_ass ex_creds ex_locs ex_efids ex_knos
```

Infrastructures are given by the following datatype that contains an infrastructure graph of type `igraph` and a policy given by a function that assigns local policies over a graph to all locations of the graph.

```
datatype infrastructure = Infrastructure igraph
                                          [igraph, location] ⇒ policy set
```

There are projection functions `graphI` and `delta` when applied to an infrastructure return the graph and the local policies, respectively. The function `local_policies` gives the policy for each location `x` over an infrastructure graph `G` as a pair: the first element of this pair is a function specifying the actors `y` that are entitled to perform the actions specified in the set which is the second element of that pair. The local policies definition for the Corona-warning app, simply permits all actions to all actors in both locations.

```
  local_policies G ≡
         (λ x. if x = pub then  {(λ y. True, {get,move,put})}
          else (if x = shop then {(λ y. True, {get,move,put})} else {}))
```

For the Corona-warning app, the initial infrastructure contains the graph `ex_graph` with its two locations pub and shop and is then wrapped up with the local policies into the infrastructure `Corona_scenario` that represents the "initial" state for the Kripke structure.

```
 Corona_scenario ≡ Infrastructure  ex_graph local_policies
```

3.2 Policies, privacy, and behaviour

Policies specify the expected behaviour of actors of an infrastructure. They are given by pairs of predicates (conditions) and sets of (enabled) actions. They are defined by the `enables` predicate: an actor `h` is enabled to perform an action `a` in infrastructure `I`, at location `l` if there exists a pair `(p,e)` in the local policy of `l` (`delta I l` projects to the local policy) such that the action `a` is a member of the action set `e` and the policy predicate `p` holds for actor `h`.

```
enables I l h a ≡ ∃ (p,e) ∈ delta I l. a ∈ e ∧ p h
```

The Privacy protection goal is to avoid deanonymisation. That is, an attacker should not be able to disambiguate the set of pairs of real Ids and EphIds. This is abstractly expressed in the predicate identifiable.

```
identifiable eid A ≡ is_singleton{(Id,Eid). (Id, Eid) ∈ A ∧ Eid = eid}
```

The predicate identifiable is used to express as the global policy 'Eve cannot deanonymize an Ephemeral Id `eid` using the gathered knowledge':

```
  global_policy I eid ≡
             ¬(identifiable eid
                ((⋂ (kgra(graphI I)(Actor ''Eve'')`(nodes(graphI I))))
                 - {(x,y). x = ''Eve''}))
```

3.3 Infrastructure state transition

The state transition relation uses the syntactic infix notation `I` $\rightarrow_i$ `I'` to denote that infrastructures `I` and `I'` are in this relation. To give an impression of this definition, we show first the rule defining the state transition for the action get. Initially, this rule expresses that an actor that resides at a location `l` (`a` $@_G$ `l`) and is enabled by the local policy in this location to "get" can combine all ids at the current location (contained in `egra G l`) with all actors at the current location (contained in `agra G l`) and add this set of pairs to his knowledge set `kgra G` using the function update `f(l := n)` redefining the function `f` for the input `l` to have now the new value `n`.

```
 get_data: G = graphI I ⟹ a @_G l ⟹ enables I l (Actor a) get ⟹
   I' = Infrastructure
          (Lgraph (gra G)(agra G)(cgra G)(lgra G)(egra G)
            ((kgra G)((Actor a) := ((kgra G (Actor a))(l:=
                 {(x,y). x ∈ agra G l ∧ y ∈ egra G l}))))
          (delta I)
   ⟹ I →_n I'
```

Another interesting rule for the state transition is the one for `move` whose structure resembles the previous one.

```
 move: G = graphI I ⟹ a @_G l ⟹ a ∈ actors_graph(graphI I) ⟹
    l ∈ nodes G ⟹ l' ∈ nodes G ⟹ enables I l' (Actor h) move ⟹
    I' = Infrastructure (move_graph_a a l l' (graphI I))(delta I)
 ⟹ I →_i I'
```

The semantics of this rule is embedded in the function `move_graph_a` that adapts the infrastructure state so that the moving actor `a` is now associated to the target location `l'` in the actor map `agra` and not any more at `l` and also the association of `efids` is updated accordingly.

```
move_graph_a n l l' g ≡
  Lgraph (gra g)
         (if n ∈ ((agra g) l) ∧  n ∉ ((agra g) l') then
           ((agra g)(l := (agra g l) - {n}))(l' := (insert n (agra g l')))
          else (agra g))
         (cgra g)(lgra g)
         (if n ∈ ((agra g) l) ∧  n ∉ ((agra g) l') then
               ((egra g)(l := (egra g l) - {efemid (cgra g n)}))
                         (l' := (insert (efemid (cgra g n))(egra g l')))
          else egra g)
         (kgra g)
```

Based on this state transition and the above defined `Corona_scenario`, we define the first Kripke structure.

```
corona_Kripke ≡ Kripke { I. Corona_scenario →* I } {Corona_scenario}
```

3.4 Attack analysis

For the analysis of attacks, we negate the security property that we want to achieve, usually the global policy.

Since we consider a predicate transformer semantics, we use sets of states to represent properties. The invalidated global policy is given by the set `scorona`.

```
scorona ≡ {x. ∃ n. ¬ global_policy x (Efid n)}
```

The attack we are interested in is to see whether for the scenario

```
Kripke_scenario ≡  Infrastructure ex_graph local_policies
```

from the initial state `Icorona` ≡{`corona_scenario`}, the critical state `scorona` can be reached, that is, is there a valid attack `(Icorona,scorona)`?

For the Kripke structure `corona_Kripke` we first derive a valid and-attack using the attack tree proof calculus.

$$\vdash [\mathcal{N}_{(\texttt{Icorona,Corona})}, \mathcal{N}_{(\texttt{Corona,Corona1})}, \mathcal{N}_{(\texttt{Corona1,Corona2})}, \mathcal{N}_{(\texttt{Corona2,Corona3})}, \mathcal{N}_{(\texttt{Corona3,scorona})}] \oplus_{\wedge}^{(\texttt{Icorona,scorona})}$$

The sets `Corona, Corona1, Corona2, Corona3` are the intermediate states where `Bob` moves to shop and `Eve` follows him collecting the Ephemeral Ids in each location. The collected information enables identifying Bob's Ephemeral Id.

The attack tree calculus [8] exhibits that an attack is possible.

```
corona_Kripke ⊢  EF scorona
```

We can simply apply the Correctness theorem `AT_EF` to immediately prove this CTL statement. This application of the meta-theorem of Correctness of attack trees saves us proving the CTL formula tediously by exploring the state space in Isabelle proofs. Alternatively, we could use the generated code for the function `is_attack_tree` in Scala (see [8]) to check that a refined attack of the above is valid.

4 Refinement

Intuitively, a refinement changes some aspect of the type of the state, for example, replaces a data type by a richer datatype or restricts the behaviour of the actors. The former is expressed directly by a mapping of datatypes, the latter is incorporated into the state transition relation of the Kripke structure that corresponds to the transformed model. In other words, we can encode a refinement within our framework as a relation on Kripke structures that is parameterized additionally by a function that maps the refined type to the abstract type. The direction "from refined to abstract" of this type mapping may seem curiously counter-intuitive. However, the actual refinement is given by the refinement that uses this function as an input. The refinement then refines an abstract to a more concrete system specification. The additional layer for the refinement can be formalized in Isabelle as a refinement relation $\sqsubseteq_{\mathcal{E}}$. The relation `mod_trans` is typed as a relation over triples – a function from a threefold Cartesian product to `bool`, the type containing true and false only. The type variables σ and σ' input to the type constructor `Kripke` represent the abstract state type and the concrete state type. Consequently, the middle element of the triples selected by the relation `mod_trans` is a function of type $\sigma' \Rightarrow \sigma$ mapping elements of the refined state to the abstract state. The expression in quotation marks after the type is again the infix syntax in Isabelle that allows the definition of mathematical notation instead of writing `mod_trans` in prefix manner. This nicer infix syntax is already used in the actual definition. Finally, the arrow $\Longrightarrow$ is the implication of Isabelle's meta-logic while $\longrightarrow$ is the one of the *object* logic HOL. They are logically equivalent but of different types: within a HOL formula P, for example, as below $\forall x. P \longrightarrow Q$, only the implication $\longrightarrow$ can be used.

```
refinement :: (σ Kripke × (σ' ⇒ σ) × σ' Kripke) ⇒ bool ("_ ⊑(_) _")
 K ⊑ℰ K' ≡ ∀ s' ∈ states K'. ∀ s ∈ init K'.
            s →*σ' s' ⟶ ℰ(s) ∈ init K ∧ ℰ(s) →*σ ℰ(s')
```

The definition of `K` $\sqsubseteq_{\mathcal{E}}$ `K'` states that for any state s' of the refined Kripke structure that can be reached by the state transition in zero or more steps from an initial state s of the refined Kripke structure, the mapping $\mathcal{E}$ from the refined to the abstract model's state must preserve this reachability, i.e., the image of s must also be an initial state and from there the image of s' under $\mathcal{E}$ must be reached with 0 or n steps.

4.1 Property Preserving System Refinement

A first direct consequence of this definition is the following lemma where the operator ` in $\mathcal{E}$`(init K') represents function image, that is the set, $\{\mathcal{E}(x). x \in$ `init K'`$\}$.

lemma `init_ref: K` $\sqsubseteq_{\mathcal{E}}$ `K'` $\Longrightarrow$ $\mathcal{E}$`(init K')` $\subseteq$ `init K`

A more prominent consequence of the definition of refinement is that of property preservation. Here, we show that refinement preserves the CTL property of $\mathsf{EF}s$ which means that a reachability property true in the refined model `K'` is already true in the abstract model. A state set s' represents a property in the predicate transformer view of properties as sets of states. The additional condition on initial states ensures that we cannot "forget" them.

```
theorem prop_pres:
   K ⊑_ℰ K'  ⟹ init K ⊆ ℰ`(init K') ⟹
   ∀ s' ∈ Pow(states K'). K' ⊢ EF s' ⟶ K ⊢ EF (ℰ`(s'))
```

It is remarkable, that our definition of refinement by Kripke structure refinement entails property preservation and makes it possible to prove this as a theorem in Isabelle once for all, i.e., as a meta-theorem. However, this is due to the fact that our generic definition of state transition allows to explicitly formalize such sophisticated concepts like reachability. For practical purposes, however, the proof obligation of showing that a specific refinement is in fact a refinement is rather complex justly because of the explicit use of the transitive closure of the state transition relation. In most cases, the refinement will be simpler. Therefore, we offer additional help by the following theorem that uses a stronger characterization of Kripke structure refinement and shows that our refinement follows from this.

```
theorem strong_mt:
ℰ`(init K') ⊆ init K ∧ s →_σ' s' ⟶ ℰ(s) →_σ ℰ(s')
⟹ K ⊑_ℰ K'
```

This simpler characterization is in fact a stronger one: we could have $s \rightarrow_{\sigma'} s'$ in the refined Kripke structure `K'` and $\neg(\mathcal{E}(s) \rightarrow_{\sigma} \mathcal{E}(s'))$ but neither s nor s' are reachable from initial states in `K'`. For cases, where we want to have the simpler one-step proviso but still need reachability we provide a slightly weaker version of `strong_mt`.

```
theorem strong_mt':
ℰ`(init K') ⊆ init K ∧ (∃ s0 ∈ init K'. s0  →* s)
 ∧ s →_σ' s' ⟶ ℰ(s) →_σ ℰ(s') ⟹ K ⊑_ℰ K'
```

This idea of property preservation coincides with the classical idea of trace refinement as it is given in process algebras like CSP. In this view, the properties of a system are given by the set of its traces. Now, a refinement of the system is given by another system that has a subset of the traces of the former one. Although the principal idea is similar, we greatly extend it since our notion additionally incorporates refinement. Since we include a state map $\sigma' \Rightarrow \sigma$ in our refinement map, we additionally allow structural refinement: the state map generalizes the basic idea of trace refinement by traces corresponding to each other but allows additionally an exchange of data types. As we see in the application to the case study, the refinement steps may sometimes just specialize the traces: in this case the state map $\sigma' \Rightarrow \sigma$ is just identity.

In addition, we also have a simple implicit version of *action refinement*. In an action refinement, traces may be refined by combining consecutive system events into atomic events thereby reducing traces. We can observe this kind of refinement in the second refinement step of the Corona-virus warning app considered next.

4.2 Refining the Specification

Clearly, fixed Ephemeral Ids are not really ephemeral. The model presented in Section 3 has deliberately been designed abstractly to allow focusing on the basic system architecture and finding an initial deanonymization attack. We now introduce "proper" Ephemeral Ids and show how the system datatype can be refined to a system that uses those instead of the fixed ones.

For the DP-3T Ephemeral Ids **[TODO:[]**todoC][2], for each day t a seed SK_t is used to generate a list of length $n = 24 * 60/L$, where L is the duration for which the Ephemeral Ids are posted by the smart phone

```
EphId1 || ... || EphIdn = PRG(PRF(SKt,‘‘broadcast key’’))
```

"where PRF is a pseudo-random function (e.g., HMAC-SHA256), "broadcast key" is a fixed, public string, and PRG is a pseudorandom generator (e.g. AES in counter mode)" **[TODO:[]**todoC][3].

From a cryptographic point of view, the crucial properties of the Ephemeral Ids are that they are purely random, therefore, they cannot be guessed, but at the same time if – after the actual encounter between sender and receiver – the seed SK_t is published, it is feasible to relate any of the `EphIDi` to SK_t for all `i` $\in \{1, \ldots, n\}$. For a formalisation of this crucial cryptographic property in Isabelle it suffices to define a new type of list of Ephemeral Ids `efidlist` containing the root SK_t (the first `efid`), a current `efid` indicated by a list pointer of type `nat`, and the actual list of `efid`s.

```
datatype efidlist = Efids "efid" "nat" "efid list"
```

We define functions for this datatype: `efidsroot` returning the first of the three consituents in an `efidlist` (the root SK_t); `efids_index` giving the second component, the index of the current `efid`; `efids_inc_ind` applied to an `efidlist` increments the index; `efids_cur` returning the current `efid` from the list and `efids_list` for the entire list (the third component).

The first step of refinement replaces the simple `efid` in the infrastructure type by the new type `efidlist`. Note, that in the new datatype `igraph` this change affects only the thrird component, the credentials. The last two componenets, the set of currently present efids, and the knowledge set, still operate on the simple type `efid`.

```
datatype igraph =
        Lgraph (location × location)set
                location ⇒ identity set
                identity ⇒ (string set × string set × efidlist)
                location ⇒ string × (dlm × data) set
                location ⇒ efid set
                actor ⇒ location ⇒ (identity × efid) set
```

[2] **[TODO:[]**todoC]**BIB-REF**: DP-3T Whitepaper
[3] **[TODO:[]**todoC]**BIB-REF**: DP-3T Whitepaper

The refined state transition relation implements the possibility of changing the Ephemeral Ids by the rule for the action put.

```
 put: G = graphI I ⟹  a @_G l ⟹ enables I l (Actor a) put ⟹
      I' = Infrastructure (put_graph_efid A l (graphI I))(delta I)
 ⟹ I →_i I'
```

The change to the infrastructure state is implemented in the function `put_graph_efid` that increases the current index in the `efidlist` in the credential component `cgra g n` for the "putting" actor identity `n` and inserts the current `efid` from that credential component into the `egra` component, the set of currently "present" Ephemeral Ids at the location `l`.

```
put_graph_efid n l g  ≡
  Lgraph (gra g)(agra g)
         ((cgra g)(n := (credentials (cgra g n), roles (cgra g n),
                          efids_inc_ind(efemid (cgra g n)))))
         (lgra g)
         ((egra g)(l := insert (efids_cur(efemid (cgra g n)))(egra g l)))
         (kgra g)
```

We can now apply the refinement by defining a datatype map from the refined infrastructure type `InfrastructureOne.infrastructure` to the former one `Infrastructure.infrastructure`.

```
definition refmap :: InfrastructureOne.infrastructure ⇒
                           Infrastructure.infrastructure
where refmap I =
  Infrastructure.Infrastructure
    (Infrastructure.Lgraph
       (InfrastructureOne.gra (graphI I))
       (InfrastructureOne.agra (graphI I))
        (λ h. repl_efr
           (InfrastructureOne.cgra (graphI I)) h)
           (InfrastructureOne.lgra (graphI I))
           (InfrastructureOne.egra (graphI I))
           (λ a. λ l.
  (λ (x,y).(x, efids_root(efemid(InfrastructureOne.cgra (graphI I) x))))
                      `(InfrastructureOne.kgra (graphI I)) a l))
```

This is then plugged into the parameter $\mathcal{E}$ of the refinement operator allowing to prove `corona_Kripke` $\sqsubseteq_{\texttt{refmap}}$ `corona_KripkeO` where the latter is the refined Kripke structure.

Surprisingly, we can still prove `corona_KripkeO` ⊢`EF scoronaO` by using the same attack tree as in the abstract model: if Bob moves from pub to shop, he is vulnerable to being identifiable as long as he does not change the current `efid`. So, if Eve moves to the shop as well and performs a get before Bob does a put, then Eve's knowledge sets permits identifying Bob's current Ephemeral Id as his.

Second refinement step The persistent attack can be abbreviated informally by the action sequence `[get,move,move,get]` performed by actors Eve, Bob, Eve, and Eve again, respectively. How can a second refinement step avoid that Eve does the last `get` by imposing that after the first `move` of Bob a `put` action must happen before Eve can do another `get`? A very simple remedy to exclude this attack is to bind a `put` action after every `move`. We can implement that change by a minimal update to the function `move_graph_a` (see Section 3) by adding an increment (highlighted in the code snippet) of the currently used Ephemeral Id before updating the `egre` component of the target location.

```
move_graph_a n l l' g ≡ ...
  (l' := insert (efids_cur( efids_inc_ind(efemid (cgra g n))))(egra g l))...
```

This is an action refinement because the move action is changed. It is a refinement, since any trace of the refined model can still be mapped to a trace in the more abstract model just omitting a few steps (the refinement relation is defined using the reflexive transitive closure $\rightarrow^*$.

5 Conclusions

5.1 Protection goals of attacks

5.2 Summary and discussion of relevance of the approach

We can establish in our formal framework an attack that even a system using changing Ephemeral ids can be broken if the attacker physically follows a victim. This is a basic attack on anonymity: a user's connection between his Ephemeral Ids and personal details (Iphone MAC or name) is revealed to the attacker. The protection goal of privacy is thereby destroyed.

When establishing the attack we start from a simplistic scenario that does not use Ephemeral Ids but fixed ids. In this (over)simplified model the attack is established. We then define a formal Refinement calculus for the Isabelle Infrastructure framework to refine the attack to a system with Ephemeral Ids that change in fixed time intervals obfuscating the relationship between user and his pseudonym[4].

Now, the refinement shows that although the Ephemeral Ids change regularly the same attack that has been identified on the very abstract level (fixed ids) persists. The refinement allows refining the datatype (Id $\mapsto$ EphId) but also delivers the usual trace refinement (behaviours of the refined systems are a subset of the traces of the abstract system). This persistence of the attack precisely shows which part of the system behaviour is responsible for the attack. In a second refinement step using action refinement based on this insight from the (repeated) attack, we can exclude the dangerous attack trace[5].

[4] We identify the smartphone and the user which might be also recognized by his appearance (face)

[5] Implication on the actual protection goals that are endangered by the attack on privacy are discussed in the larger context of the Corona-warning app (see previous Section)

Clearly, the abstract attack we establish is obvious on an informal level but the persistence of the attack on the refined system is less obvious. The remedy by a second refinement step is an evident restriction of the system behaviour which gives a clear specification of a system secured against this attack. The use of formal methods therefore lies not in the discovery of an obvious attack on a simplified system but in showing how a formal specification including security refinement can lead to a stepwise improvement that is accompanied by formal proof in the Isabelle Infrastructure framework. The solution to exclude the attack in the second refinement step binds the action move together with a put action. This shows that besides datatype refinement and trace refinement our Refinement calculus also entails action refinement. This action refinement is implemented implicitly by changing the effects of the actions in the semantic state transition relation. In future work, we could think about making the action refinement more explicit by considering a relationship between semantic rules or by refining the refinement notion to a more explicit layer of protocol steps – similar to what has been done in previous applications of the Isabelle Infrastructure framework for example to Auction protocols [13] or the Quantum Key Distribution [10].

The security refinement in general might seem pointless, as in the first step of refinement the attack persists and even though the second refinement gets rid of this specific attack, it doesn't exclude the reachability of the attack goal altogether (if the attacker Eve gets Bob on his own in a location she can map all used EphIDs to him). However, the refinement makes the system relatively more secure in that for a larger number of traces the abstract attack does not work anymore[6]. It is important to emphasize that security refinement is a cyclic process that improves security but does not usually terminate (like a loop) with a fixed point of 100% secure system. In a refined model new detail may give rise to new attack possibilities. These additional attacks can be identified using the Attack Tree calculus and trigger further refinement steps.

References

1. D. Bundesregierung. Die Corona-Warn-App: Unterstützt uns im Kampf gegen Corona, 2020. German government announcement and support of Coronavirus warning app.
2. CHIST-ERA. Success: Secure accessibility for the internet of things, 2016. http://www.chistera.eu/projects/success.
3. Corona-Warn-App, 2020. https://github.com/corona-warn-app/cwa-documentation/blob/master/solution_architecture.md.
4. Corona-Warn-App. Corona-warn-app solution architecture, 2020. https://github.com/corona-warn-app/cwa-documentation/blob/master/solution_architecture.md.
5. DP-3T. Decentralized Privacy-Preserving Proximity Tracing, 2020. https://github.com/DP-3T.

[6] Adding probabilities as in [10] enables quantifying this.

6. DP-3T. Decentralized privacy-preserving proximity tracing - White Paper, 2020. https://github.com/DP-3T/documents/blob/master/DP3T White Paper.pdf.
7. DP-3T. README: Apple / Google Exposure Notification, 2020. https://github.com/DP-3T/documents.
8. F. Kammüller. Attack trees in isabelle. In *20th International Conference on Information and Communications Security, ICICS2018*, volume 11149 of *LNCS*. Springer, 2018.
9. F. Kammüller. Isabelle infrastructure framework with iot healthcare s&p application, 2018. Available at `https://github.com/flokam/IsabelleAT`.
10. F. Kammüller. Attack trees in isabelle extended with probabilities for quantum cryptography. *Computer & Security*, 87, 2019.
11. F. Kammüller. Combining secure system design with risk assessment for iot healthcare systems. In *Workshop on Security, Privacy, and Trust in the IoT, SPTIoT'19, colocated with IEEE PerCom*. IEEE, 2019.
12. F. Kammüller. Isabelle infrastructure framework for ibc, 2020. Isabelle sources for IBC formalisation.
13. F. Kammüller, M. Kerber, and C. Probst. Insider threats for auctions: Formal modeling, proof, and certified code. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA)*, 8(1):44–78, 2017. Special Issue on Insider Threat Solutions - Moving from Concept to Reality.
14. F. Kammüller and U. Nestmann. Inter-blockchain protocols with the isabelle infrastructure framework. In *Formal Methods for Blockchain, 2nd Int. Workshop, colocated with CAV'20*, Open Access series in Informatics. Dagstuhl publishing, 2020. To appear.
15. F. Kammüller. A formal development cycle for security engineering in isabelle, 2020.
16. F. Kammüller and M. Kerber. Applying the isabelle insider framework to airplane security, 2020.
17. T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.
18. PEPP-PT. Pan-European Privacy-Preserving Proximity Tracing, 2020. https://github.com/PEPP-PT.
19. ROBERT. ROBust and privacy-presERving proximity Tracing protocol, 2020. https://github.com/ROBERT-proximity-tracing.