Inter-Blockchain Protocols with the Isabelle Infrastructure Framework

Florian Kammüller
Middlesex University London, UK
Technische Universität Berlin, Germany
f.kammueller@mdx.ac.uk

Uwe Nestmann
Technische Universität Berlin, Germany
firstname.secondname@tu-berlin.de

Abstract

The main incentives of blockchain technology are distribution and distributed change, consistency, and consensus. Beyond just being a distributed ledger for digital currency, smart contracts add transaction protocols to blockchains to execute terms of a contract in a blockchain network. Inter-blockchain (IBC) protocols define and control exchanges between different blockchains.

The Isabelle Infrastructure framework has been designed to serve security and privacy for IoT architectures by formal specification and stepwise attack analysis and refinement[1]. A major case study of this framework is a distributed health care scenario for data consistency for GDPR compliance. This application led to the development of an abstract system specification of blockchains for IoT infrastructures.

In this paper, we first give a summary of the concept of IBC. We then introduce an instantiation of the Isabelle Infrastructure framework to model blockchains. Based on this we extend this model to instantiate different blockchains and formalize IBC protocols. We prove the concept by defining the generic property of global consistency and prove it in Isabelle.

2012 ACM Subject Classification Networks → Peer-to-peer protocols; Networks → Security protocols; Software and its engineering → Software verification and validation

Keywords and phrases Blockchain, smart contracts, interactive theorem proving, inter-blockchain protocols

Digital Object Identifier 10.4230/OASIcs.FMBC.2020.11

Funding *Florian Kammüller*: Research for this paper has been supported by CHIST-ERA grant 101112, SUCCESS.

1 Introduction

Inter-blockchain (IBC) protocols is a concept driven by industry. It serves to provide "reliable and secure communication between deterministic processes" [24] that run on independent blockchains or distributed ledgers. Practical application of IBC are for example the Cosmos Hub [5] "the first of thousands of interconnected blockchains" with the purpose of facilitating transfers between blockchains.

A formal specification of IBC within a Higher Order Logic theorem prover like Isabelle has the advantage that it provides a very rigorous model of the IBC concepts enabling mechanically verified properties. In principle, from such a formalization, executable code into many standard programming languages like Haskell or Scala can be generated. However, such code generation would always be understood to provide only reference implementations. Moreover, the major insights from specifying a practice oriented concept like IBC is that

[1] In this paper we do neither illustrate attack tree analysis nor security refinement.


© Florian Kammüller and Uwe Nestmann;
licensed under Creative Commons License CC-BY

the formal specification is mainly useful to provide a more abstract yet more precise model that carefully picks out the central concepts used within the application, here IBC. In doing this, the used methodology, here Isabelle, can provide as a framework existing work to immediately support the IBC specification. We rely heavily on the Isabelle Infrastructure framework [15] as an existing instantiation of Isabelle/HOL (which we will simply refer to as Isabelle within this paper). This framework offers a range of predefined concepts like Kripke structures and CTL, as well as state transition relations, actors, and policies that can be readily instantiated to the current application of IBC. Besides extracting a more abstract but precise specification of IBC, the resulting scientific advantage is to show that as a product of this process it becomes feasible to lay open crucial basic properties that result from the application domain (blockchain security). As the main result of this kind, we formally establish a global consistency property, define it formally on our IBC model and prove a consistency preservation theorem that shows the safety of our formal IBC semantics.

The contributions of this paper are

- summarizing the main features of IBC into a logical conceptual model,
- building a formal model of IBC in Isabelle as an instance of the Isabelle Infrastructure framework but extending it with sets of infrastructures,
- illustrating the feasibility of the formal model by expressing a global consistency property and formally proving it in Isabelle.

The last point seems to suggest that IBC can be seen as a "blockchain of blockchains".

1.1 Inter-blockchain protocols (IBC)

In this section, we summarize the main concepts of the IBC following the practice-oriented description [24]: we refer to the relevant section of the principal documentation[24], giving precise reference to section numbers. Figure 1 is a copy an overview architectural sketch provided by the main specification [24].

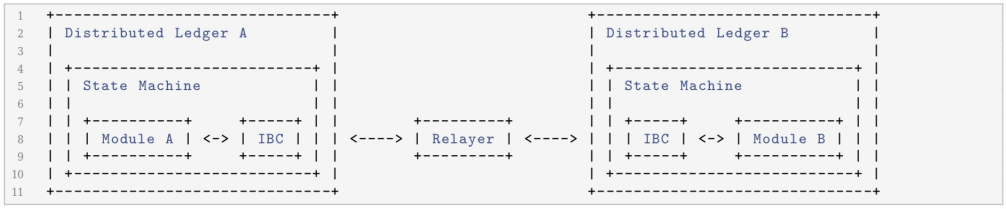

Figure 1 Architecture of IBC[24].

One of the main abstractions used in IBC comprising its architectural description is the *actor* [24, Section 1.1.1] which is the same as a *user*. Instances given to exemplify this are: a human end user, a module or smart contract running on a blockchain, or an off-chain *relayer* process. This relayer process represents the logical core of the IBC. It is a process that is outside any of the blockchains ("off-chain" [24]) that is responsible for "relaying" IBC data packets between blockchains. It can scan their states and submit data.

The notion of state machine is very central in IBC: the terms *machine*, *chain*, *blockchain*, or *ledger* are used interchangeably [24, Section 1.1.2] to denote a state machine that implements part or all of the IBC. In using the Isabelle Infrastructure framework – whose core part is the formal definition of a state machine semantics through a state transition relation – we follow this important architectural spirit.

Consensus is not explicitly defined but somewhat implicitly by the notion of consensus algorithm "the protocol used by the set of processes operating a distributed ledger to come to agreement on the same state" [24, 1.1.5] where "Consensus state" is defined next as information about the "state of a consensus algorithm" [24, 1.1.6]. We can safely understand consensus to mean the agreement of the actors on the next state with respect to the state transition relation.

1.2 Isabelle Infrastructure framework

The Isabelle Infrastructure is built in the interactive generic theorem prover Isabelle/HOL [19]. 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 [3] and framework [10].

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. The formal model of IBC in Isabelle uses the Isabelle

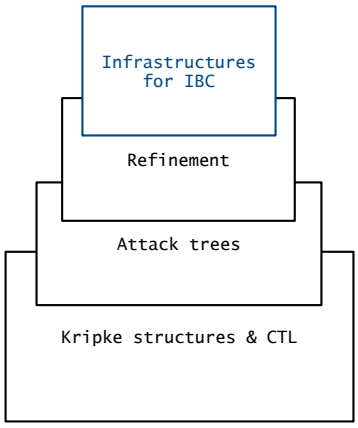

Figure 2 Generic Isabelle Infrastructure framework applied to Inter-blockchain protocols (IBC).

Infrastructure framework instantiating it by reusing its concept of *actors* for users, processes running on blockchains, or relayers running off-chain. Technically, an Isabelle theory file `IBC.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 IBC, express relevant and interesting properties and conduct interactive proofs (with the full support of the powerful and highly automated proof support of Isabelle). The IBC theory itself is an adaptation of the Infrastructure theory of the Isabelle Infrastructure framework and reuses (or slightly adapts) existing concepts. In the remainder of this paper, we introduce the model that we conceived for IBC. All Isabelle sources are available online [12].

2 IBC in Isabelle

2.1 Overview

In the following, we give a detailed description of the central parts of the formal Isabelle theory of IBC, pointing out and motivating special design decisions. In addition to the short general intro to the Isabelle Infrastructure framework of the previous section, we provide explanations of all used Isabelle specific specification concepts on the fly.

The IBC is supposed to work for any type of blockchain, for example, Bitcoin or Ethereum, therefore the formal model abstracts from specific details of a specific blockchain. Similar to the IBC specification [24], the Isabelle formalization focuses on the central IBC concepts as depicted in Figure 1: ledgers, actors or modules, respectively, and the relayer process interacting via the IBC protocol with the modules within the distributed ledgers. In our formal model based on the Isabelle Infrastructure framework, we represent each blockchain as an infrastructure containing nodes on which the modules (actors) are running. Data items are assigned to actors. The ledgers of each infrastructure keep control over the data items. That is, a ledger is a unique assignment that controls the access to a data item and keeps a record of where the data item resides within this and other blockchains. The IBC enables just that: a unified view over a whole range of heterogeneous blockchains that exchange data consistently. Therefore, our formal model goes beyond the usual application of the Isabelle Infrastructure framework, e.g. [8], and considers sets of infrastructures (representing different blockchains).

2.2 Ledgers

Actors are a general concept provided by the Isabelle Infrastructure framework and can be used directly to represent the *actor* concept in IBC.

```
typedecl actor
type_synonym identity = string
consts Actor :: string ⇒ actor
```

Similar to the general Infrastructure framework, actors can perform actions. However, in this instantiation to IBC we redefine the actions representing the central activities of the relayer scanning each blockchain's state and submitting transactions (see Section 1.1).

```
datatype action = scan | submit
```

The Decentralized Label Model (DLM) [17] allows labeling data with owners and readers. We also adopt this definition of security labeled data as already formalised in [10]. Labeled data is given by the type `dlm × data` where `data` can be any data type.

```
type_synonym data = string
type_synonym dlm = identity × identity set
```

One major achievement of a blockchain is that it acts like a distributed ledger, that is, a global accounting book. A distributed ledger is a unique consistent transcript keeping track of protected data across a distributed system. In our application, the ledger must mainly keep track of where the data resides for any labeled data item. To express the system requirement that processing may not change the security and privacy labels of data, we introduce a type of security and privacy preserving functions.

```
typedef label_fun = {f :: dlm × data ⇒ dlm × data.
                       ∀ x. fst x = fst (f x)}
```

We formalize a ledger thus as a type of partial functions that maps a data item to a pair of the data's label and the set of locations where the data item is registered. Since all function in HOL are total, we use a standard Isabelle way of representing partial functions using the type constructor `option`. This type constructor lifts every type α to the type α `option` which consists of the unique constant `None` and the range of elements `Some x` for all `x`$\in \alpha$.

```
type_synonym ledger = data ⇒ (dlm × node set)option
```

Since the type `ledger` is a function type, it automatically constrains each data item `d` in its domain to have at most one range element `Some(l,N)`, that is, at most one valid data label `l` of type `dlm` and a list of current blockchain nodes `N` at which this data item is transcribed.

```
lemma ledger_def_prop: ∀ lg:: ledger. ∀ d:: data.
     lg d = None | (∃! l. (∃! L. lg d = Some(l, L)))
```

In an earlier application of the Isabelle Infrastructure framework to IoT security and privacy[15], we established a formal notion of blockchain. However, there we used a more explicit logical characterization in an Isabelle type definition which creates additional proof effort and makes formulas more complex. The current representation of the ledger type as a partial function type is more concise and implicitly carries the requested uniqueness properties. Note that the defining property of the ledger type is now proved from the used type constructors by the above lemma instead of being specified into the type as in the earlier formalization [15].

2.3 Infrastructures as blockchains

The datatype `sc_fun` formalizes any action that is sent or received between different blockchains and may have effects on the labeled data. Therefore the inputs to the send and receive messages are two identities of sender and receiver as well as the dlm label and the concerned data.

```
datatype sc_fun = Send identity × identity × dlm × data
                | Receive identity × identity × dlm × data
```

In addition to specifying the potential types of smart contracts, we need to provide a way of keeping track of the transactions that are executed within a blockchain. To this end, we define the following type of `transaction_record` which is a list of all executed smart contracts.

```
type_synonym transaction_record = sc_fun list
```

The central component that builds the system state is an infrastructure. Since we use the Isabelle Infrastructure framework, we consider blockchains as infrastructures. The essential architecture of such an infrastructure is a simple graph of blockchain nodes on which the processes (actors) reside given as the first component `(node ×node)set` of the below datatype `igraph`. Besides this basic architecture, this infrastructure graph also stores the other components of the blockchain. The second input is a function that assigns a set of actor identities to each node in the graph representing the current location of the actors. The next input associates actors to a pair of string sets by a pair-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. An infrastructure graph also allows assigning a string to each location to represent some current state information of that location. Finally, the ledger is added as a separate component as well as the transaction record.

```
datatype igraph =
         Lgraph (node × node)set
                node ⇒ node set
                actor ⇒ (string set × string set)
                node ⇒ string
                ledger
                transaction_record
```

Corresponding projection functions for each of the components of an infrastructure graph are provided. They are omitted here for brevity but are available in the online version [12]); they are named `gra` for the actual set of pairs of locations, `agra` for the actor map, `cgra` for the credentials, and `lgra` for the state of a location `ledgra` for the ledger component in the graph and `trec` for the transaction record. Infrastructures contain an infrastructure graph 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] ⇒ apolicy set
```

There are projection functions `graphI` and `delta` when applied to an infrastructure return the graph and the policy, respectively.

Policies specify the expected behaviour of actors of an infrastructure. We define the behaviour of actors using a predicate `enables`: within infrastructure `I`, at location `l`, an actor `h` is enabled to perform an action `a` if there is a pair `(p,e)` in the local policy of `l` – `delta I l` projects to the local policy – such that action `a` is in 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
```

Compared to the applications of the Isabelle Infrastructure framework, e.g. [8], we do not make use of policies to model the constraints of our application. However different to previous applications, the IBC challenges the framework in other ways leading to slight extensions.

2.4 Relayer and set of blockchains

To model the relayer, we also use infrastructures: the relayer is a distinguished infrastructure. It could be thought of as another distributed application with various relayer processes to avoid bottlenecks but for simplicity, we assume that there is one specific actor `''relayer''` that resides on a specific node in the relayer infrastructure.

We express protocols as traces of execution steps of IBC transaction steps, that is, lists of smart contracts `sc_fun` (see previous section). Using traces of execution steps to represent protocols, follows the classical method of the inductive approach to security protocol verification originally devised by Paulson [22] and already successfully used for the Isabelle Infrastructure framework, for example, [13] and more recently [11, 9].

```
datatype ibc_protocol = Protocol sc_fun list set
```

The datatype `blockchainset` puts together the IBC protocol as a triple: as the first element it includes the IBC protocol, the second element is the list of infrastructures where each element is one blockchain involved in the IBC, and the third element is a single distinguished infrastructure, the relayer.

```
datatype blockchainset = Infs ibc_protocol
                              infrastructure list
                              infrastructure
```

To round off these new datatypes, we provide additional projection functions and constructors. For a given blockchain `Il`, the projection `trcs Il` returns the `sc_fun list set` representing the protocol, the projection `the_Il` returns the list of infrastructures of all involved blockchains, and `relayer Il` gives the distinguished infrastructure, the third element, which is the relayer infrastructure. To facilitate handling of data transactions, we define some update functions: the function application `upd_ld d lN I` updates a ledger at the data point `d` to now contain the pair `lN` of a dlm label and a set of nodes of residences of the data. Scaling this up to the level of infrastructures, the function application `upd_Il d lN Il` updates all blockchains in the infrastructure list of the blockchainset `Il` using the former ledger update `upd_ld`. A function `replace` allows to replace an infrastructure `I` in a blockchainset `Il`. See the online resources [12] for technical details and implementations of these definitions.

2.5 Consensus

The consensus algorithm may be different for each blockchain employed in the IBC. Therefore, we cannot make any assumptions at the general specification level of the IBC about it. Yet, we still want to use it in the description of the IBC protocol semantics. Therefore, we apply a trick: we declare `Consensus` to be a constant at the level of the specification of the IBC.

```
consts Consensus :: infrastructure ⇒ blockchainset ⇒ blockchainset
```

In Isabelle this means that `Consensus` is a function mapping an infrastructure and a system state of type `blockchain` to `blockchain` but there is no semantics attached to this constant. The constant is part of the theory `IBC.thy` and can be used in it like any other defined element but it has no meaning. However, a semantics can be later attached to it in an application of the IBC theory to specific blockchains. This could be done in the current context for example using a definition in a locale [14].

```
locale ConsensusExample =
fixes cons_algo :: infrastructure ⇒ blockchainset ⇒ infrastructure
defines cons_algo_def: cons_algo I Il = ...
fixes Consensus :: infrastructure ⇒ blockchainset ⇒ blockchainset
defines Consensus_def: Consensus I Il = replace (cons_algo I Il) I Il
```

The predicate `Consensus` redefines the semantics within the locale `ConsensusExample`. The first locale definition is omitted here for simplicity. We could imagine that it is a description of a consensus algorithm that can depend on all the state constituents, like actors, nodes, and policies of the blockchain `I` but also of the surrounding blockchainset including the relayer state and the current protocol state. The definition of the constant `Consensus` lifts the algorithm to the blockchain by using the replace function defined as part of the infrastructure for blockchainsets (see Section 2.4 or refer to the Isabelle code [12]).

2.6 IBC state transition semantics

The semantics of the IBC state machines is defined by a state transition relation over blockchain sets. That is, we define a syntactic infix notation `Il → Il'` to denote that blockchain sets `Il` and `Il'` are in this relation.

```
inductive state_transition_in ::
       [blockchainset, blockchainset] ⇒ bool "(_ → _)"
```

The rules of the inductive definition `state_transition_in` allow the definition of the intended behaviour of the relayer scanning an arbitrary blockchain (see Section 1.1). The relayer stores the results in its own transaction record. The following rule `scan` is the first of two inductive definition rules defining the transition relation →: if an infrastructure `I` is in the blockchainset `Il`, the actor (process, module) resides at node `n` in the graph `G` of `I`; `R` is the relayer and thus enabled to scan. The follow up state `Il'` of `Il` is given by extending any current protocol trace `l` using the specially defined function `insertp` by the transaction `Send(a,b,(a,as), d)`. Also the relayer's trace record `trec R` is extended by the same transaction.

```
 scan : inbc I Il ⟹ G = graphI I ⟹ a @_G n ⟹ n ∈ nodes G ⟹
        R = graphI (relayer Il) ⟹ r @_R n' ⟹ n' @_R nodes R ⟹
        relrole (relayer Il) (Actor r) ⟹
        enables I n (Actor r) scan ⟹
        ledgra G d = Some ((a, as), N) ⟹ r ∈ as ⟹
        R' = Infrastructure
                  (Lgraph (gra R)(agra R)(cgra R)(lgra R)
                          ((ledgra R)(d := Some((a, as),N)))
                          (trec R))
                  (delta (relayer Il)) ⟹
        l ∈ trcs Il ⟹ Consensus I Il = Il' ⟹
        Il' = insertp ((Send(a,b,(a,as), d)) # l) (replrel R' Il)
        ⟹ Il → Il'
```

Additionally, the relayer can submit data onto an arbitrary blockchain (see Section 1.1). The second rule `submit` of → defines its semantics: between the infrastructures `I` and `J` which are both in the blockchain set `Il` the relayer `R` can submit data `d` from an owner `a` to an owner `b` if the ledger component `ledgra R` of the relayer's infrastructure `R` is updated to the new owner in both blockchains. The update is achieved using the function update `:=` of Isabelle's function theory updating the point `d` to the new value `Some((b, bs), N)`. In the construction of the next state blockchainset `Il'` the specially defined update operators mentioned in Section 2.4 are used: `replrel` for updating the relayer and `bc_upd` for the infrastructure list representing the "client" blockchains. Note the latter realizes the consistent update in both involved infrastructures `I` and `J`.

```
submit : G = graphI I ⟹ inbc I Il ⟹ a @_G n ⟹ n ∈ nodes G ⟹
         ledgra G d = Some ((a, as), N) ⟹
         H = graphI J ⟹ inbc J Il ⟹ b @_H n' ⟹ n' ∈ nodes H ⟹
         ledgra H d = Some ((a, as), N) ⟹
         R = graphI (relayer Il) ⟹ r @_R n'' ⟹ n'' ∈ nodes R ⟹
         relrole (relayer Il) (Actor r) ⟹
         enables J n' (Actor r) submit ⟹
         r ∈ as ⟹
         R' = Infrastructure
                (Lgraph (gra R)(agra R)(cgra R)(lgra R)
                        ((ledgra R)(d := Some((b, bs),N)))
                        (trec R))
              (delta (relayer Il))  ⟹
         Il' = insertp (Receive(a,b,(a,as),d)# l)
                       (replrel R' (bc_upd d ((b,as), N) Il)) ⟹
```

```
        Consensus J Il = Il'
        ⟹ Il → Il'
```

The real advantage of the Isabelle Infrastructure framework comes into play when using the possibility of instantiation of axiomatic type classes provided by Isabelle. Since state transitions have been defined by an axiomatic type class in the framework within the theory for Kripke structures and CTL, we can now instantiate blockchainsets as state and thereby inherit the entire logic, constructors and theorems.

```
instantiation blockchainset :: state
```

3 Global consistency

To illustrate the use of the abstract formal model of IBC presented in this paper, we show that we can exhibit an important property: global consistency. That is, if the IBC scans and submits between blockchains it must not introduce inconsistencies.

Expressing this property alone represents a proof of concept since it shows that our IBC model is detailed enough to capture explicitly the notion of consistent data representation across different blockchains. *Proving* the property is a non-trivial contribution (see proof scripts [12]) that helped exhibiting a range of useful auxiliary definitions and lemmas as we will highlight in this section when discussing the global consistency theorem. The proofs were greatly helped by the recent advances in proof automation in Isabelle using sledgehammer [21]. The fact that the property is provable shows that the model and in particular its semantics conform to the intuition described in [24]. The formalization and proof also highlight the pros and cons of our model as discussed in the Conclusions in Section 4.

We first define global consistency as the property that the individual ledgers in each blockchain in an IBC blockchainset agree on the data, that is, they all hold consistent information about the access control of the data (the first part of type `dlm` of the `ledgra` output (see Section 2.2)) and where the data resides: the set of nodes that are the second component of the `ledgra` output.

```
Global_consistency Il = (∀ I I'. inbc I Il → inbc I' Il →
       (∀ d. (ledgra (graphI I') d) = (ledgra (graphI I) d)))
```

The theorem shows that if global consistency holds, then a step of the state transition does preserve it.

```
theorem consistency_preservation:
global_consistency Il ⟹ (Il → Il') ⟹ global_consistency Il'
```

Preservation of global consistency guarantees that any transaction happening within IBC preserves one consistent view over all data, their access control, and residence. If initially data is not visible on all blockchains, not all ledgers are equal. However, if eventually data has traveled across, all ledgers become the same: the blockchainset becomes like one blockchain: a "blockchain of blockchains".

4 Conclusions, related work, and outlook

In this paper, we have provided an abstract formal model of the Inter-blockchain protocol (IBC) [24] as an instantiation of the Isabelle Infrastructure framework. We have detailed the formal presentation in Isabelle and the extensions to the Isabelle Infrastructure framework,

most notably by defining sets of (heterogeneous) blockchains including protocols and a distinguished relayer. The abstraction we conceived for this model has been first validated by a proof of concept by sketching how the abstract notion of Consensus can be instantiated by a locale (Section 2.5). Furthermore, we have defined a global consistency property over blockchainsets proving that our abstraction yields the desired expressivity (Section 3). We have proved a preservation theorem for global consistency in Isabelle. Summarizing, our model allows to prove meta-theoretical results but is not too abstract to allow instantiation onto concrete blockchains and their Consensus algorithms. As a more general thought, the dealings with global consistency seem to suggest that IBC creates a blockchain of blockchains.

4.1 Related Work

Relevant examples for the investigation of formal support for blockchains and smart contracts can be found in abundance in the proceedings of the first FMBC workshop [2]. We only discuss the few most closely related ones from there since others are either focusing on specific blockchains (unlike the generic IBC we consider) or are differing in the formal approach (not using theorem provers and thus not addressing the same level of expressivity and assurance).

A range of works formalizes smart contracts typical for the Ethereum virtual machine. For example, using the K framework [23], the Lem language [7], and F* [6]. We focus here on the work that has been performed in the K-framework [23]. The K-framework is a semantics framework enabling to produce executable operational semantics for programming languages. K also provides tools like parsers, interpreters, model-checkers and program verifiers. It has been applied to provide a verification environment for the Ethereum Virtual Machine EVM [20] which is useful for verifying programme modules within Ethereum's smart contract systems, for example, Ethereum's Name Service (ENS) [25].

In comparison to those dedicated verification environments for specific blockchains, like Ethereum, our formal model strongly abstracts from technical detail. This abstraction is necessary to accommodate a global view that allows to reason about the communication between a heterogeneous set of blockchains.

A few works use model checkers and SMT solvers, for example [4]. Deductive verification platforms like Why3 [11,13] have been also used for smart contracts. Interactive proof assistants (e.g. Isabelle/HOL or Coq) have been used before for modeling and proving properties about Ethereum and Tezos smart contracts [1].

Very related is the work by Nielsen and Spitters on Smart Contract Interactions in Coq [18]. The authors construct a model of smart contracts that allows for inter-contract communication generalizing over depth-first execution blockchains like Ethereum and breadth-first execution blockchains like Tezos. They use Coq's functional language Galina to express smart contracts. Besides the obvious difference of being a Coq development rather than an Isabelle development, we address the high level protocol language IBC instead of focusing on generalized smart contracts.

Maybe even more closely related is the work on the specification of the dedicated security framework Cap9 in Isabelle [16]. Compared to us it focuses again on the expression of smart contracts and does not have the inter-blockchain aspect like our IBC.

4.2 Outlook

The global consistency preservation theorem proves the concept of the IBC specification and also shows that the formalization in itself is a useful experiment: extracting a closed abstract model of the IBC from the technical specification [24] has immediately produced

the consistency question. The abstraction allowed to define semantics in which a strong global consistency theorem could be proved within Isabelle in reasonably short time. It should be understood that these are first steps that mainly serve to prove the concept of using the Isabelle Infrastructure framework for advancing the IBC. A clear next step is to elaborate the sketched application example of Section 2.5 of a concrete blockchain and its consensus algorithm. A much more challenging next step is to refine the model by elaborating a more concrete IBC protocol example by instantiation of the `ibc_prot` component of the `blockchainset` type. This would be a fruitful future avenue for applied research in collaboration with the designers of IBC.

The notions of attack trees and security refinement have not been applied in this application of the Isabelle Infrastructure framework but can be seen in other applications, for example to auction protocols [13], GDPR [8], or IoT security [10]. Nevertheless, the current application has brought about much improvement on the formalization of the ledger datatype as well as instantiating the generic state of the framework to sets of infrastructures and defining their state transition.

The Isabelle Infrastructure framework subsumes the earlier Isabelle Insider framework, for example [13]. Thus there is the possibility to reason about malicious agents that are in the group of trusted participants. This could be used to reason about participants that do not comply to the IBC protocol and in terms of Consensus it would enable reasoning on Byzantine fault tolerance. Using attack tree analysis and security refinement in a security engineering cycle [15] could then be used to develop secure IBC solutions.

References

1 S. Amani, M. Bégel, M. Bortin, and M. Staples. Towards verifying ethereum smart contract bytecode in isabelle/hol. In *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs*, pages 66–77. ACM, 2018.

2 N. Catano, D. Marmsoler, and B. Bernardo, editors. *Pre-proceedings of the First Workshop on Formal Methods for Blockchains, FMBC*, 2019. Selected papers to appear in Springer LNCS. URL: `https://sites.google.com/view/fmbc`.

3 CHIST-ERA. Success: Secure accessibility for the internet of things, 2016. http://www.chistera.eu/projects/success.

4 Sylvain Conchon, Alexandrina Korneva1, and Fatiha Zaidi. Verifying smart contracts with cubicle, 2019. Selected papers to appear in Springer LNCS. URL: `https://sites.google.com/view/fmbc`.

5 Cosmos. Cosmos hub, 2020. accessed 23.1.2020. URL: `https://hub.cosmos.network/master/hub-overview/overview.html`.

6 I. Grishchenko, M. Maffei, and C. Schneidewind. A semantic framework for the security analysis of ethereum smart contracts. In L. Bauer and R. Ksters, editors, *Principles of Security and Trust*, Lecture Notes in Computer Science, pages 243–269. Springer, 2017.

7 Y. Hirai. Defining the ethereum virtual machine for interactive theorem provers. In M. Brenner, K. Rohloff, J. Bonneau, A. Miller, P. Y. Ryan, V. Teague, A. Braccialiand M. Sala, F. Pintore, and M. Jakobsson, editors, *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 520–535. Springer, 2017.

8 F. Kammüller. Formal modeling and analysis of data protection for gdpr compliance of iot healthcare systems. In *IEEE Systems, Man and Cybernetics, SMC2018*. IEEE, 2018.

9 F. Kammüller. Attack trees in isabelle extended with probabilities for quantum cryptography. *Computer & Security*, 87, 2019. URL: `//doi.org/10.1016/j.cose.2019.101572`.

10 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.

11 F. Kammüller. Qkd in isabelle – bayesian calculation. *arXiv*, cs.CR, 2019. URL: https://arxiv.org/abs/1905.00325.

12 F. Kammüller. Isabelle infrastructure framework for ibc, 2020. Isabelle sources for IBC formalisation. URL: https://github.com/flokam/IsabelleSC.

13 F. Kammüller, M. Kerber, and C.W. Probst. Towards formal analysis of insider threats for auctions. In *8th ACM CCS International Workshop on Managing Insider Security Threats, MIST'16*. ACM, 2016.

14 F. Kammüller, M. Wenzel, and L. C. Paulson. Locales – a sectioning concept for Isabelle. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Thery, editors, *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99*, volume 1690 of *LNCS*. Springer, 1999.

15 Florian Kammüller. A formal development cycle for security engineering in isabelle, 2020. arXiv:2001.08983.

16 Mikhail Mandrykin1, Jake O'Shannessy, Jacob Payne, and Ilya Shchepetkov. Formal specification of a security framework for smart contracts, 2019. Selected papers to appear in Springer LNCS. URL: https://sites.google.com/view/fmbc.

17 A. C. Myers and B. Liskov. Complete, safe information flow with decentralized labels. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, 1999.

18 J. Botsch Nielsen and B. Spitters. Smart contract interactions in coq, 2019. Selected papers to appear in Springer LNCS. URL: https://sites.google.com/view/fmbc.

19 T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.

20 Daejun Park, Yi Zhang, Manasvi Saxena, Philip Daian, and Grigore Roundefinedu. A formal verification tool for ethereum vm bytecode. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 912–915, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3236024.3264591.

21 Lawrence Paulson and Jasmin Blanchette. Three years of experience with sledgehammer, a practical link between automatic and interactive theorem provers. *Proceedings of the 8th International Workshop on the Implementation of Logics*, February 2015. doi:10.29007/tnfd.

22 Lawrence C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6(1-2):85–128, 1998. URL: http://iospress.metapress.com/content/5wlu8p2am1du051d/.

23 Grigore Rosu. Specifying languages and verifying programs with k. *2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 28–31, 2013.

24 The IBC Specification Team. The interblockchain communication protocol, 2020. 4th May 2020 — 1.0.0-rc5. URL: https://github.com/cosmos/ics/blob/master/spec.pdf.

25 Duy Minh Vo. *Verification of Smart Contracts using the K-framework*. PhD thesis, Technische Universität Berlin, 2018.