Towards Secure and Trusted-by-Design Smart Contracts

Zaynah Dargaye, Florent Kirchner, Sara Tuccı-Piergiovanni, Önder Gürcan

Towards Secure and Trusted-by-Design Smart Contracts

Zaynah Dargaye, Önder Gürcan, Florent Kirchner, and Sara Tucci Piergiovanni

CEA, LIST, Point Courrier 174, Gif-sur-Yvette, F-91191 France
lea-zaynah.dargaye@cea.fr, onder.gurcan@cea.fr, florent.kirchner@cea.fr, sara.tucci@cea.fr

Abstract

Distributed immutable ledgers, or blockchains, allow the secure digitization of evidential transactions without relying on a trusted third-party. Evidential transactions involve the exchange of any form of physical evidence, such as money, birth certificate, visas, tickets, etc. Most of the time, evidential transactions occur in the context of complex procedures, called evidential protocols, among physical agents. The blockchain provides the mechanisms to transfer evidence, while smart contracts – programs executing within the blockchain in a decentralized and replicated fashion – allow encoding evidential protocols on top of a blockchain.

As a smart contract foregoes trusted third-parties and runs on several machines anonymously, it constitutes a highly critical program that has to be secure and trusted-by-design. While most of the current smart contract languages focus on easy programmability, they do not directly address the need of guaranteeing trust and accountability, which becomes a significant issue when evidential protocols are encoded as smart contracts.

1 Introduction

The rise of immutable distributed ledgers, or *blockchains*, extends the *"code is law"* vision to organizations and corporations through the Decentralized Autonomous Organization (*DAO*) concept. Indeed, blockchains allow the secure digitization of *evidential transactions* without relying on a trusted third-party [21] – where evidential transactions involve the exchange of any form of physical evidence, such as money, birth certificates, visas, tickets, etc. Most of the time, evidential transactions occur in the context of complex procedures encoded by specific programs, called *smart contracts*, executing within the blockchain in a decentralized and replicated fashion. A DAO, in particular, is an organization run by smart contracts encoding rules of governance. For those organizations, promises in terms of independence and savings are great advantages that smart contracts can enable, through the removal of both the trusted third-party and the need for repetitious (often manual) execution of contracts.

The smart contract concept thus plays a major role in a DAO. However, there is no official definition of smart contract: so far, no agreement has been established regarding this concept. Originally in 1994 in [21], Nick Szabo defined a smart contract as *a computerized transaction protocol that executes the terms of a contract*. The general objectives are to satisfy common contract conditions (such as payment terms, liens, confidentiality, and even enforcement) and minimize the need for trusted intermediaries. Related economic goals include lowering fraud loss, arbitration and enforcement costs, and other transaction costs. Nick Szabo believes that a contractual clause implemented in software and benefiting from cryptographical mechanisms would make the infringement of contracts very expensive – it is exactly this aspect that makes the contract "smart". Nowadays, a smart contract designates any piece of code running in the blockchain, losing the legal flavor but keeping its immutability properties: once in the blockchain the smart contract cannot be changed, becoming the "law" for the community using it.

Recently Josh Stark, Head of Operations & Legal at LedgerLabs, in the Ethereum community made an attempt to clarify these notions identifying two kinds of smart contracts (see [20]): *smart legal contracts* and *smart contract codes*. So far, most of the attention has focused on improving and designing smart contract code. Smart contract code languages, such as Solidity for Ethereum, lack both formal foundations and the expressiveness to program smart legal contracts in a secure way. To make things worse, a large number of vulnerabilities coming from the execution of the programs in a widely distributed network, i.e., the blockchain network, are fully exploitable today.

The lack of formalization and verification has already shown its impact: the famous attack on *The DAO* [9] was caused by a simple bug in the smart contract. The survey [3] shows that most attacks in Ethereum were caused by bugs or vulnerabilities of the execution platform (Ethereum Virtual Machine and blockchain network). Recently, the Tezos ledger has taken several steps to address these problems, with the use of OCaml for its implementation, and Michelson, a statically typed functional language for its smart contracts. However, the overall challenge of smart contract security remains unchanged. We advocate that to tackle this challenge there is the need (i) of a formal language specifically designed to capture the notion of trust (accountability, authentication, privacy) for the secure implementation of smart legal contracts in software, (ii) to capture and abstract the subtleties of the execution of smart contracts in a possible wide distributed environment and (iii) to provide full support for a compilation chain towards smart contract code language.

In this paper, we focus on the design of the smart legal contract language. We advocate that such a challenge could be addressed by extending so-called trust management frameworks to the blockchain. Trust management frameworks are formal frameworks allowing to formalize the specification of evidential protocols and to verify their correct implementation using a trusted-by-design approach. In particular, this paper presents a Coq implementation of a Cyberlogic framework that gathers the main features of smart legal contracts. Cyberlogic is a trust management framework, a first-order logic featuring an authority algebra for specifying protocols and their policies. Therefore, the Coq implementation enables to mechanically specify and verify Cyberlogic protocols, enabling to provide computable proofs. Moreover, as the Cyberlogic implementation is a shallow embedded one, a smart legal contract specified in Cyberlogic can be extracted to an executable language.

The paper is organised as follows: Section 2 presents the state of the art on the specification of evidential protocols detailing Cyberlogic and its implementation in Coq. Section 3 shows how to apply Cyberlogic to the design of trustworthy smart contracts. We corroborate our statement by specifying a smart contract in Cyberlogic and we present its implementation in a smart contract code. Finally, Section 4 presents our future work.

2 State-of-the-Art: Specifying Evidential Protocols

Evidential protocols by essence are based on trust between entities and particularly third-parties such as governments, banks or insurances. In an evidential protocol, those kind of entities exchange specific data, called evidences. A piece of evidence can be an official certificate, a license or a visa. It constitutes critical data, and is usually stated or claimed by a specific authority. Although accountability is endorsed by the authority which claims it, an evidence has to be formally specified as well as any other data in a protocol. Smart contracts have a lot in common with evidential protocols where a smart contract represents the interaction between the blockchain and the real world.

In this section, we detail the state-of-the-art of evidential protocols specification. First, we

present trust management frameworks which allow to specify trust and accountability. Second, we focus on the Cyberlogic framework which is an extended trust management framework allowing reasoning and verification. Finally, we detail our Coq implementation of Cyberlogic.

2.1 Trust Management

The underlying concepts of evidential protocols are the accountability of the evidence, its establishment and its usage. Therefore, the specification of evidential protocols requires being able to specify trust and accountability, which is the purpose of *trust management*.

PolicyMaker [7] is a trust management tool which associates to the management of the security (via public/private keys) a logic of predicates on keys. PolicyMaker is a language which features the ability to insert logic assertions. These assertions can express that a certain key permits (authorizes) a certain predicate. However, PolicyMaker does not allow formal reasoning for constructing a proof that validates the claim of an authority (ie., an abstraction of the entity that owns the key). The descendant of PolicyMaker is KeyNote [6], an other trust management framework is Fidelis [23]. This approach tackles the specification of policies in evidential transactions.

On the other hand, interesting contributions have been done about verification of trust in evidential protocols. In particular, a first alternative is to use the *a posteriori* validation of the authentication [2]. The proof accompanies the data and is verified at the reception by the authority. It has a cost and adds some work to the authority. [14] describes a delegation logic which considers the delegation of the authority. This logic permits expressing relation such as: "A claims P", "A delegates P to B with a delegation of k depth (a trusted chain of k authorities)", "A claims for B about P".

The Cyberlogic framework inherits the major features of trust management systems. In addition, Cyberlogic features native constructions to deal with distributed systems.

2.2 Cyberlogic

Cyberlogic [19] is designed for the transition from paper documents to electronic documents. In particular, Cyberlogic offers a framework for formalization and reasoning on attestation elements such as visas, certificates *etc.*... Cyberlogic is a logic and a distributed program in which protocols can play several exchanges of attestations. As Cyberlogic features a first order logic, it allows to reason upon the actions of a Cyberlogic protocol. Cyberlogic also features an authorities' algebra allowing it to reason upon trust and accountability.

In addition to the expression of certificates and visas, we believe that the authorities' algebra can also be useful in complex systems verification, and whenever trust into an entity takes the form of a formal verification : acceptance.

This could, for instance, be the case of a development using a black-box library. The library editor claims that the library has been formally verified. Then, the user can suppose that each function of this library obeys its specification. In that configuration, Cyberlogic provides feature to naturally express some hypotheses usually categorized as informal hypotheses, or working hypotheses. If $Spec$ is the specification of the function f of the library, then the user can use Cyberlogic to formalize its development. Hence, the specification of f becomes a formal formula: $E \triangleright Spec(f)$. If the editor has used Cyberlogic to formalize the library, then the library specification can be reused by the user. Finally, if f does not fill its specification and appears to bug in the development, the user can identify the editor as guilty and accountable.

An interpreter for Cyberlogic protocols and formulas has been developed at SRI International, using the PVS system [17]. Since 2013 at CEA List, thanks to two national projects

(SystemX MIC and FUI GeoTransMD) a Coq [15] implementation has been developed and used to verify authenticity properties of distributed protocols in communication systems.

In particular, the authors of [19] state that Cyberlogic is designed to enable evidential protocols implementation into agent-based frameworks where agents execute Cyberlogic protocols to carry out specific tasks that require the exchange of authorization and authentication information. Nowadays, this definition seems quite similar to blockchain protocols. In particular, we believe that Cyberlogic provides most of the properties required by a smart legal contract language.

2.3 Shallow Embedding Implementation of Cyberlogic in Coq

We developed a theorem prover for Cyberlogic as a shallow embedded implementation of Cyberlogic in the Coq theorem prover. The shallow embedding implementation allows to inherit all Gallina expressiveness and to use the standard libraries of Coq. Our implementation benefits the extraction mechanism as well. Moreover, it allows to focus only on the specific feature of Cyberlogic: the attestation. An attestation is a logic formula claimed by an authority. The other Cyberlogic constructors are those of the Coq language itself.

2.3.1 Authority and Attestations

Authorities are the owners of claims. A claim is a property of a data endorsed by an authority. The authority is accountable for the claimed property. An authority is the abstraction of a entity such as a person, a bank, a company or an organization, i.e. any actor of the system who can establish or claim a property. In Cyberlogic, an authority is uniquely identified and decidable comparable to other authorities.

2.3.2 Different Kinds of Attestations

An authority can claim a property either directly or indirectly. If an authority claims something directly then it could have established it itself and is accountable for this claim. If an authority claims something indirectly, it means that another authority transmits her(him) this claim. In other words, the chain of trust can be traced to find the accountable authority.

In the formalization we name the qualification direct or indirect through the access concept.
`Inductive` **access** := | Direct : **access** | Indirect : **access**.

There are different kinds of claims (named `authority` in the Coq development). In addition to the combination of an authority and an access mode, time can be taken in account in some claims. As in modal logic, it is possible to claim a fact before, at or after a date. Then, a claim defines an owner, an access and a timing for a claim.

`Inductive` **authority** :=
| Key : *Authority* → **access** → **authority**
| KatT : *Authority* → **access** → time → **authority**
| Kbef : *Authority* → **access** → time → **authority**
| Kaft : *Authority* → **access** → time → **authority**.

2.3.3 Attestations

An attestation is a property which is claimed by a authority.
`Variable` *attestation* : **authority** → `Prop` → `Prop`.

For the sake of clarity, let's define some notations as similar as possible from those of the original paper [4].

$k \mid > \ f$	$k \ \triangleright \ f$	k attests directly f
$k \ * \mid > \ f$	$k \ :\triangleright \ f$	k attests indirectly f
$k \mid >= t \ f$	$k \ \triangleright_{=t} \ f$	k attests directly f at t
$k \ * \mid >= t \ f$	$k \ :\triangleright_{=t} \ f$	k attests indirectly f at t
$k \mid >< t \ f$	$k \ \triangleright_{<t} \ f$	k attests directly f before t
$k \ * \mid >< t \ f$	$k \ :\triangleright_{<t} \ f$	k attests indirectly f before t
$k \mid >> \ t \ f$	$k \ \triangleright_{>t} \ f$	k attests directly f after t

2.3.4 Delegation of Attestation

Cyberlogic authorities' algebra features a delegation system which appears in the indirect mode. An indirect claim models that this claim has been obtained by delegation from another authority. The accountable is the originate claimer. $D1(A, B, P)$ says that A claims P and A knows P from B, B is accountable for P. $D2(A, B, P)$ says that A has been directly delegated by some other authority (C) to claim P. But C is not accountable for P, B is. $D2$ is a special delegation chain of depth 3. A delegation is indexed by the length of the chain of authority from the original claimer to the authority which actually claims.

```
Definition Dinf (k k': Authority) (A:Prop) := (k' *|> A)-> (k *|> A).
Definition D1 (k k': Authority) (A:Prop):= (k'|>A) → (k *|> A).
Definition D2 (k k': Authority) (A:Prop):=
 (∀ (k0 : Authority), (k0 |> A) ∧ (k' |>((k0 |> A)->A))) → (k *|> A).
```

2.3.5 Time Reasoning in the Cyberlogic

Cyberlogic integrates features of modal logic in the authority algebra. A specific authority is accountable of time: Kt. Kt establishes the current time by the predicate `curr`, date in the past by the predicate `has_been` and in future by the predicate `in_futur`.

```
Variable hasbeen: time → Prop.
Definition is_time (t:time):Prop:= Kt *|> (hasbeen t).
Definition curr (t:time):Prop:= is_time t ∧ (∀ (t':time), (is_time t') → t'≤ t ).
Definition in_future (t:time):= Kt *|> ~(hasbeen t).
```

3 Cyberlogic for Specifying Smart Legal Contracts

Cyberlogic allows reasoning both on (i) actions, thanks to a first-order logic and (ii) on trust and accountability, thanks to the authority algebra. Being specifically designed for evidential protocols and benefiting from a implementation in Coq, we advocate that Cyberlogic is, indeed, a perfect candidate to be at the core of a smart legal contract language. To support our claim, in this section, we show Cyberlogic at work by specifying the Schengen visa management process as a Cyberlogic protocol and present the corresponding smart contract code in Solidity[1].

[1]The availability of a formal and proved specification opens the way for a formal compilation chain targeting the smart contract code; the compilation chain, however, is not in the scope of this paper (see section 4 for a detailed future work discussion.

3.1 Schengen Visa Management

The Schengen visa management which is described in accordance with the European Commission, is an evidential protocol where the visa represents an evidence delivered by a country of the Schengen area. In order to obtain that evidence, it also requires to gather other evidence (in accordance to the requirements of the visa management process). The visa itself is an evidence stating that all the requirements are satisfied.

Nowadays, the execution of visa management protocol is time consuming, with latency experienced for each required evidence, and mostly manual (photocopies, forms, ...). The digitalisation of the entire protocol through smart legal contracts is indeed a good opportunity to improve the protocol and make it more secure. In the remainder of this section we provide a characterisation of the visa management as a Cyberlogic protocol, highlighting the kind of reasoning that can be made. In particular, the Cyberlogic specification allows the verifier of the visa (e.g. the customs officer) to roll-up the entire chain of trust to find the authority accountable for the possible error.

3.2 The Schengen Visa as a Cyberlogic Protocol

We propose to implement the Schengen visa management protocol as a smart legal contract. The result would be to alleviate centralization and possible related bottlenecks by digitizing the process in a secure and decentralized way [2]. Let us note, indeed, that the intermediary here is the smart legal contract itself that will be later encoded as a smart contract code running in a blockchain. The smart contract code will, indeed, (i) encode all the rules pertaining to the management of the Schengen's visa, including rules on authorities that have the right to deliver it, and (ii) will be executed in a secured and decentralised fashion among the blockchain participants.

Let us now to explicit our smart legal contract. First of all, to satisfy the autonomy and the authority of the real states of the Schengen area, the smart legal contract includes a consulate (or prefecture) role as official authority having the rights to deliver the visa. At delivery time, this official authority delegates the visa to the requester, where: (i) the official authority is still the unique accountable for the visa and (ii) the requester is allowed to provide directly the required pieces of evidence.

Let us note that the consulate can be viewed as a trusted third party we are still relying on. On the other hand, we propose to formally delegate the entire procedure to the requester and to exploit blockchain immutability properties to correctly compute proofs on executed scenarios. This approach without fully realising the vision of a decentralised and autonomous organisation, represents a first step in this direction [3]. Anyway, a smart legal contract to manage the Schengen visa will allow active controls, transparency and detection of contradictory requirements. A controller will be able to "go behind" the visa itself and consult the requirement it represents. In case of two requirements are contradictory, the controller will be able to observe it.

The management of the Schengen visa as a smart legal contract in Cyberlogic is composed of 4 functions which are: its demand, its delivery, its control and its indictment. Let's informally define these 4 functions.

To demand a Schengen visa is to write evidential requirements of the Visa's protocol in the

[2] We are aware that the adoption of such a smart legal contract is not only a technical issue, since legal compliance with the proposed framework should also be evaluated. However, since our goal is to show how CyberLogic works, we are not taking into consideration legal issues.

[3] Our approach is still compliant with the actual organisation of States and their rules, a different fully decentralized organisation would be possible using self-certification and new governance rules.

blockchain, once and for all thanks to immutability. The accountability is preserved thanks to the Cyberlogic protocol, where each evidential requirement is a claim from the appropriate authority.
To deliver a Schengen visa is to write the visa in the blockchain, the visa is claimed by the consulate (or official state organization) and is delegated to the requester, thanks to the Cyberlogic delegation. The chain of trust from the visa is digitized thanks to the delegation mechanism of Cyberlogic.
To control a Schengen visa is a read in the blockchain, accessing to the evidential requirements by rolling-up the trust chain.
To suspect a Schengen visa is to extract the evidential requirements from the blockchain, analysing the evidential requirements and identifying the potential suspicious claims, i.e. computing accountability.

3.3 The Cyberlogic Protocol

We present the smart legal contract of the Schengen visa as a Cyberlogic protocol that exchanges transactions that carrying a visa, its demand or its control.

```
Inductive transaction:=
| Demand: Schengen_demand → transaction
| Deliver : visa → transaction
| Control : visa → transaction.
```

Queries are the suspicions that an officer can make. When an officer suspects a visa, he will make queries about it.

```
Definition query := visa → Prop.
Definition queries := list query.
```

A smart contract answers to a querry either that the visa is valid or that there is a list of suspicious claims about the visa.

```
Inductive answer:=| Valid : answer| Suspects : list Prop → answer.
```

Action designates the interaction API with the blockchain. It consists in three operations on transactions: reading the ledger, writing in the ledger and questioning the ledger about properties on a specific data. The API presented here is light but sufficient for highlighting the Cyberlogic properties.

```
Variable action:Type.
Variable write: Authority → transaction → action.
Variable read : Authority → transaction → action.
Variable verify : Authority → visa → queries → answer.
```

The functions of the Schengen visa smart contract in the Cyberlogic protocol becomes:

```
Definition demand (Requester: Authority)(C:country) (f:schengen_form) (pic:photo)
(pport:passport)(trvls:travel_itinerary)(ins_policy:travel_health)(accs:accommodations)
(suff:sufficient_means)(t:time):=
    write Requester (Demand (mkDemand f pic pport trvls ins_policy accs suff t)).
Definition deliver (cons:Authority)(v:visa):= write cons (Deliver v).
Definition control (officer:Authority)(v:visa):= read officer (Control v).
Definition suspect (officer:Authority)(v:visa) (ev:queries):= verify officer v ev.
```

3.4 The Protocol Policies

The policy of demanding a visa is a time-stamp verification. The requester is accountable of the demand.

```
Definition demanding (r:Authority) (t:time) (c:country) (d:Schengen_demand):=
  time_stamp d = t ∧ (r |>=t make (demand r c (form d) (picture d) (pass d) (travels d)
(insurance d) (lodgings d) (sufficient d) (time_stamp d))).
```

A consulate delivers a visa if the visa demand is valid. The appendix (see A) details the validity of a demand: the seven requirements are satisfied and claimed by the appropriate authority. For example, the validity of the passport of a citizen of a country C, is a claim of C, i.e. C is accountable for the validity of the passport.

```
Definition delivering_validation (cons req:Authority) (v:visa) (t:time) :=
∃ d, visa_of_demand d v ∧ demanding req (time_stamp d) (country_of cons) d ∧
  schengen_demand_validation req d ∧ (cons |>=t (make (deliver cons v))).
Definition delivering (cons req:Authority) (v:visa) (t:time) :=
(D1 req cons (cons |><t delivering_validation cons req v t) ).
```

Of course only specific persons can control a visa, this officer has to be one of the Schengen area. If an officer has some suspicions regarding specific properties of the visa, two cases can happen. First case, this was a false alert and the officer claims a proof that all queries are satisfied. Second case, this was a real alert and the officer claims a list of suspicious claims that have to be checked.

```
Definition controlling (officer:Authority) (v:visa) (t:time):=
  schengen_officer officer ∧ curr t ∧ (officer |>=t (make (control officer v))).
Definition false_alert (officer:Authority) (v:visa)(ev:queries) (t:time):=
 schengen_officer officer ∧ (officer |>=t (make_answer (suspect officer v ev))) ∧
 suspect officer v ev= Valid.
Definition raise_alert (officer:Authority)(v:visa)(ev:queries) (evidences: list Prop) (t:time):=
 schengen_officer officer ∧ (officer |>=t (make_answer (suspect officer v ev))) ∧
 suspect officer v ev= Suspects evidences.
```

3.5 The Cyberlogic Protocols at Work

In this section, we present a scenario specified in the Cyberlogic implementation to highlight the accountability computation, its role in conflict detection and the fact that it allows to raise an alert. Today, this facility can only be applied at design time while playing scenarios to improve Cyberlogic protocol. We plan to transpose this facility at runtime via a monitoring smart contract or service. To do this, the Cyberlogic framework has to be extended to consider transactions as first class citizens and to express accountability on transaction instead than only on properties. The scenario is as the following:

Jon Snow requests a Schengen visa to the French consulate for 3 months. He plans to stay in Paris from the 1st June 2018 to 31st August 2018 in the Icy Wall.

```
Definition JSaccs := mkAcc IcyWall FirstJune2018 ThirtyFirstAugust2018.
Definition JSaccs := JSacc::nil.
```

His flight tickets are:

Drogo airline	from Winterfell (Essos) 1st June 2018	to Paris (France) departure: 3am	on the flight 3 arrival: 3:30am
Drogo airline	from Paris (France) 31st August 2018	to Winterfell (Essos) departure: 4pm	on flight 10 arrival 4:30pm

```
Definition JSoutward :=
 (mkFlight Drogo 3 JonSnow (Winterfell,FirstJune2018+Wdep_t)
          (France,FirstJune2018+Farr_t) W_IATA F_IATA 100).
Definition JSreturn :=
 (mkFlight Drogo 10 JonSnow (France,ThirtyFirstAugust2018+Fdep_t)
          (Winterfell, ThirtyFirstAugust2018+Warr_t) F_IATA W_IATA 100).
Definition JStravels := JSoutward::JSreturn::nil.
```

Thanks to his new job as King of The North, he provides his employment contract as a mean of sufficient.

```
Definition was_KingOfTheNorth:= KingOfTheNorth JonSnow demand_t.
Definition JSsuff:= Employment Cwinterfell was_KingOfTheNorth.
```

He also provides its passport number (delivered by Winterfell) and a reference of its travel health insurance delivered by Three-eyed crow &cie.). We have axiomatized the no significant part of the requirements.

```
Definition JSdemand:= (mkDemand JSform JSpic JSpassport JStravels JSinsurance JSaccs JSsuff demand_t).
```

He obtains his visa, `JSvisa`. The 1st July 2018, a police officer controls his visa. The officer recognizes him and knows, thanks to the Winterfell Times that Lady Sansa Stark. It reveales that she took his job since 4 months. So he suspects his visas and asks for evidence of sufficient means.

```
Definition suspicious_clue := WinterfellTime |> (KingOfTheNorth SansaStark (demand_t - 5)).
Definition JSsuff_query (v:visa):=
 ∀ d, visa_of_demand d v → (sufficient d) = JSsuff → KingOfTheNorth JonSnow demand_t.
```

The protocol returns the claim of the employment contract, which is a claim of the Winterfell kingdom. At this stage, the smart contract waits for physical world intervention and raises an alert.

```
Axiom suspicious_sufficient_means:
demanding JonSnow demand_t France JSdemand → delivering CFrance JonSnow JSvisa deliver_t
→
raise_alert JaimeL JSvisa (JSsuff_query::nil) ((Cwinterfell |> was_KingOfTheNorth)::nil) suspicious_t.
```

While the expressiveness of Cyberlogic allows to specify and reason on smart legal contracts and execution scenarios, a lot has to be done to provide mechanisms to monitor accountability at runtime in a blockchain-based execution environment [4]. Moreover, the authorities algebra allows to claim properties while in a smart legal contract language, this is the object/data of a transaction that has to be claimed. The chosen example is an evidential protocol for which Cyberlogic is particularly suitable. It also highlights the trust and accountability issues that DAO has to manage. However, to cover the whole kind of DAO the Cyberlogic has to be extended to handle contractual relationship (see section 4). The scenario we had unfolded highlight the detection of conflict and active control. To treat more complex scenarios, with several demands, the write, read and query operations have semantics concording with the distributed and immutable characteristics of the ledger. Therefore, those operations have to be first-class citizen of the smart legal contract language. In the next section the blockchain-based smart contract, i.e. the code counterpart of the Cyberlogic protocol is presented.

[4] In section 4 this issue is discussed as work-in-progress.

3.6 A solidity smart contract code

We implemented the Schengen Visa smart contract given in Section 3.2 in the Solidity language (Algorithm 1). Solidity is a contract-oriented, high-level language whose syntax is similar to that of JavaScript and it is designed to target the Ethereum blockchain framework [10]. It is statically-typed, supports inheritance, libraries and user-defined abstract data types. Solidity contracts bundle data with the functions operating on that data and have mechanisms for restricting direct access to some of their components.

The Schengen Visa smart contract functions are implemented as demand(), deliver(), control() and suspect() functions respectively in Algorithm 1. It is assumed that the identity (address) of the consulate and the officer are already known by the contract, and they are used by the contract to restrict access to its functions. External calls to the contract functions bring a msg.sender parameter that returns the address of the caller. The contract can then use these information to verify the identities when needed using the modifier construct of Solidity together with the msg.sender parameter. Modifiers are used to amend the semantics of functions in a declarative way. The modifier onlyConsulate() is used for verifying that msg.sender is a valid consulate (Algorithm 1 line 6) and the modifier onlyOfficer() used for verifying that msg.sender is an officer of schengen aera (Algorithm 1 line 7). This way, it is assured that the deliver() function can only be called by the consulate (Algorithm 1 line 18) and the control() and suspect() function can only be called by the officer (Algorithm 1 line 30 and 34).

4 On-going and Future Research Perspectives

Our aim is to provide a framework to design and implement smart contracts in a secure and trusted manner. Our point of view is that a smart legal contract has to be compiled in a smart contract code. We advocate that an extension of Cyberlogic is a good candidate to specify the smart legal contract. As transparency and trust are at core of our vision, we plan to equip our framework with formal analyzers as well as a formal verification of the compiler. In this section, we discuss the different current works in progress and futures research perspectives that we intend to explore.

4.1 Specification of Legal Artefacts

Smart contracts aim at digitized parts of legal contracts between entities. Specifying law is a recurrent holy Grail in formal methods [18]. However, deontic logic [16] with some restriction to avoid paradoxes have been embedded in formal frameworks. In particular, the Contract Language [11] is an action-based language featuring concepts for permission, obligation and prohibition as first-class citizen. The tool CLAN allows to analyze specification in CL and detect conflicts. On the other hand, CL does not allow to reason about trust and accountability as Cyberlogic does. We aim at extending Cyberlogic with CL-like mechanisms to provide more specific expressiveness dedicated to smart legal contracts. The open project Legalese also aims at providing such a language based on *Contract Language*.

4.2 Targeting Secure Smart Contract Code

Cyberlogic protocols in our Coq implementation are Gallinea programs. Therefore, they can be extracted to Ocaml code and then be compiled into a smart contract code language. In particular, we target Ocaml based languages as they might provide a formal base such as Michelson for the smart contract language of Tezos [13]. Another possibility is to target a

Algorithm 1 The solidity implementation of the Schengen Visa smart contract. It is assumed that the identity (address) of the consulate and the officer are already known by the contract. Due to the space limitation the abstract data types Demand and Visa are not shown.

```
1: contract SchengenVisa {
2:
3: address public consulate;
4: address public officer;
5:
6: modifier onlyConsulate() { require(msg.sender == consulate); _; }
7: modifier onlyOfficer() { require(msg.sender == officer); _; }
8:
9: mapping (address => Demand) demands;
10: mapping (uint => Visa) visas;
11: ...
12:
13: function demand(uint sch_form_id, uint photo_id, string passport_id, uint[] travel_ids,
    uint travel_health, uint[] accommodation_ids, uint sufficient_means, uint time_stamp)
    public {
14:     address visaDemander = msg.sender;
15:     demands[visaDemander] = Demand(sch_form_id, photo_id, passport_id, travel_ids,
16:         travel_health, accommodation_ids, sufficient_means, time_stamp); }
17:
18: function deliver(Demand demand, string country, string duration) public returns (uint
    visaId) onlyConsulate {
19:     if (isValid(demand)) {
20:         visas[++visaId] = Visa(visaId, consulate, country, duration);
21:         return visa.id;
22:     } else return -1; }
23:
24: function isValid(Demand demand) returns (bool valid) private {
25:     valid = validateTravels(demand.travel_ids, demand.accommodation_ids);
26:     valid &= (demand.sufficient_means >= 5000);
27:     return valid;
28: }
29:
30: function control(uint visaId) public returns (Visa visa) onlyOfficer {
31:     var visa = visas[visaId];
32:     return visa; }
33:
34: function suspect(uint visaId, string reqField) public returns (var field) onlyOfficer {
35:     var visa = visas[visaId];
36:     var field = visa[reqField];
37:     return field; }
38:
39: } // contract SchengenVisa
```

subset of Solidity [10] of the Ethereum Virtual Machine EVM [22]. The execution infrastructure should also include a monitoring mechanism for trace/scenarios analysis which is linked to the Cyberlogic formalisation and allowing reasoning.

4.3 Specification of the Blockchain at Low-Level

The inherent non-determinism caused by the execution of the programs in a widely distributed environment subject to Byzantine failures and network partitions exposes the programs to several vulnerabilities, which are often misunderstood by programmers. The survey [3] shows, indeed, that most attacks in Ethereum were caused by vulnerabilities of the execution platform (Ethereum Virtual Machine and blockchain protocols).

Available formal analyses of security of the blockchain [12] have several limitations, assuming a perfect message diffusion mechanism, instantaneous communication and a fixed number of participants. These assumptions are far from being realistic so security thresholds as the famous "majority assumption" for Bitcoin (the system is secure if the majority the hashing power is in the hands of honest nodes) falls short in more realistic settings. A formal definition of blockchain (first attempts in [1, 8]) is indeed needed to (i) allow the secure design of blockchain protocols and (ii) to gain trust in the smart contract execution by defining formal semantics to specify the properties of the execution context.

5 Conclusion

We advocate that since the final goal of smart contracts is to obviate the use of trusted third-parties, a smart contract specification must allow reasoning about trust and accountability. By considering smart contracts as evidential protocols we take advantage of Cyberlogic to specify and verify them. In this paper we have shown the use of Cyberlogic through an illustrative example and we have described the overall approach along with the elements needed to provide a trustworthy framework to design and implement secure and trusted-by-design smart contracts. Finally, we are currently exploring extensions of Cyberlogic with deontic supports and dedicated features to interact with the underling immutable distributed ledger.

References

[1] E. Anceaume, R. Ludinard, M. Potop-Butucaru, and F. Tronel. Bitcoin a distributed shared register. In *Stabilization, Safety, and Security of Distributed Systems - 19th International Symposium, SSS 2017, Boston, MA, USA, November 5-8, 2017, Proceedings*, pages 456–468, 2017.

[2] A. W. Appel and E. W. Felten. Proof-carrying authentication. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, CCS '99, pages 52–62, New York, NY, USA, 1999. ACM.

[3] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts sok. In *Conference on Principles of Security and Trust - Volume 10204*, New York, USA, 2017.

[4] V. Bernat, H. Ruess, and N. Shankar. First-order cyberlogic. Technical report, SRI International, 2005.

[5] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin. Formal verification of smart contracts. In *Workshop on Programming Languages and Analysis for Security*, PLAS '16, New York, USA, 2016.

[6] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures (position paper). In *Security Protocols, 6th International Workshop, Cambridge, UK, April 15-17, 1998, Proceedings*, pages 59–63, 1998.

[7] M. Blaze, J. Feigenbaum, and M. Strauss. Compliance checking in the policymaker trust management system. pages 254–274. Springer, 1998.

[8] T. Crain, V. Gramoli, M. Larrea, and M. Raynal. (leader/randomization/signature)-free byzantine consensus for consortium blockchains. *CoRR*, abs/1702.03068, 2017.

[9] P. Daian. Analysis of the dao exploit. http://hackingdistributed.com/2016/06/18/analysis-of-the-dao-exploit/.

[10] C. Dannen. *Introducing Ethereum and Solidity: Foundations of Cryptocurrency and Blockchain Programming for Beginners*. Apress, Berkely, CA, USA, 1st edition, 2017.

[11] S. Fenech, G. J. Pace, and G. Schneider. *CLAN: A Tool for Contract Analysis and Conflict Discovery*, pages 90–96. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[12] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310, 2015.

[13] L. Goodman. A self-amending crypto-ledger. tezos white paper. 2014.

[14] N. Li, B. N. Grosof, and J. Feigenbaum. A practically implementable and tractable Delegation Logic. In *Proceedings of the 2000 IEEE Symposium on Security and Privacy*, pages 27–42. IEEE Computer Society Press, May 2000.

[15] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Ver. 8.0.

[16] P. McNamara. Deontic logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2014 edition, 2014.

[17] S. Owre, J. M. Rushby, and N. Shankar. Pvs: A prototype verification system. In *Conference on Automated Deduction: Automated Deduction*, CADE-11, pages 748–752, London, UK, UK, 1992. Springer-Verlag.

[18] H. Prakken and G. Sartor. *The Role of Logic in Computational Models of Legal Argument: A Critical Survey*, pages 342–381. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[19] Rueß and N. Shankar. Introducing cyberlogic. In B. Martin, editor, *HCSS'03—High Confidence Software and Systems Conference*, Baltimore, MD, 1-3 April 2003.

[20] J. Stark. Making sense of blockchain smart contracts. https://www.coindesk.com/making-sense-smart-contracts/.

[21] D. Tapscott and A. Tapscott. *The blockchain revolution:how the technology behind bitcoin is changing Money,Business and the World*, pages 72,88,101,127. TNew York, New York : Portfolio / Penguin, 2016. ISBN-13: 978-1101980132.

[22] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. http://bitcoinaffiliatelist.com/wp-content/uploads/ethereum.pdf, 2014. Accessed: 2016-08-22.

[23] W. Yao. Trust management for widely distributed systems. Technical Report UCAM-CL-TR-608, University of Cambridge, Computer Laboratory, Nov. 2004.

A Specification of the Verification of Schengen Visa Requirements

A smart contract for Schengen visa requests a generic specification of visa. A visa is delivered by a specific authority, has a duration or an expiry date, is an evidence that allow someone to circulate in a country.

`Variable` *visa*: `Type`.

```
Variable visa_delivered_by : visa → cyber.Authority.
Variable visa_duration: visa → cyber.time.
Variable visa_kind : visa → Prop.
Variable visa_country: visa → country.
```

1-The Visa Application form must be fully completed and signed in the corresponding blanks.

```
Variable schengen_form:Type.
Variable schengen_from:schengen_form → time.
Variable schengen_to: schengen_form → time.
Variable schengen_requester:schengen_form → Authority.
Variable schengen_country : schengen_form → country.
Variable schengen_form_requirement: schengen_form → Prop.
```

2- A passport is delivered by an authority, has a expiry date and contains a list of visas.

```
  Variable photo: Type.
Variable passport_photo: photo → Prop.
Variable passport:Type.
Variable passport_delivered_by: passport → Authority.
Variable visas_of_passport : list visa.
Variable passport_expericy_date: passport → time.
```

Passport_of is a mapping that associates a passport to its owner.

```
Variable passport_of: Authority → passport.
```

3- The passport as well as all the copies of your previous visas, valid for at least 3 months prior to your departure is required. The passport must have at least two blank pages.

```
Definition valid_passport_at (p:passport) (departure_time: time):=
 (passport_expericy_date p) ≤ (departure_time - (months 3)).

Variable valid_passport: passport → time → Prop.
Definition valid_at_time (p:passport)(departure_time: time):=
 (valid_passport p departure_time) → (valid_passport_at p departure_time).
```

4- Round trip reservation or itinerary with dates and flight numbers specifying entry and exit from the Schengen area. You can use the visa consultation services like this one. These guys can handle most of your visa requirements such as flight itineraries, hotel reservations along with free consultation over email.

```
  Record flight := mkFlight
{ fl_airline: Authority; fl_id: nat; fl_for: Authority; fl_departure: country × time;
  fl_arrival: country × time; fl_dep_aipport: IATA; fl_arr_airport:IATA; fl_price: nat; }.

Definition trav_itinerary := list flight.
```

trav_valid holds is a flight that is valid recording the requirement of the Schengen visa. That would be a property claimed by an authority.

```
Variable trav_valid: flight → Prop.
Definition travs_valid (l:trav_itinerary) := ∀ fl, List.In fl l → ((fl_airline fl) |> trav_valid fl).
```

travs_consistency is a verified property that is not a claim.

```
Fixpoint travs_consistency (tvls : trav_itinerary) (tfrom tto:time):Prop:=match tvls with
| a::m ⇒ (snd (fl_departure a)) < (snd (fl_arrival a)) ∧
        (match m with
         | nil ⇒ (snd (fl_departure a)) = tfrom ∧ (snd (fl_arrival a)) = tto
         | m ⇒ (snd (fl_departure a))=tfrom ∧ (travs_consistency m (snd(fl_arrival a)) tto)
         end)
```

```
| nil ⇒ True end.
```

5- The travel health insurance policy is to be secured, covering any medical emergency with hospital care and travel back to ones native country due to medical motives. This health insurance policy has to cover expenses up to 30,000 euros, the sum depending on the residing days, and also it has to be valid in all Schengen countries. The health insurance policy must be purchased before picking up the visa and if your visa is refused you can cancel it!

```
Variable trav_health :Type.
Variable trav_health_of: trav_health → Authority.
Variable trav_health_emitter: trav_health → Authority.
Variable trav_health_valid : trav_health → Prop.
```

6-Proof of accommodation for the whole duration of the intended stay in the Schengen area.

```
Record accd:= mkAcc{ shelter_at: Authority; from : time; to:time;}.
Definition accds:= list accd.
Variable accd_valid: accd → Prop.
Definition accds_valid (acc:accds):= ∀ ac, List.In ac acc → ((shelter_at ac) |> accd_valid ac).

Fixpoint accds_consistency (accs:accds) (tfrom tto:time):Prop:=match accs with
| a::m ⇒ (from a) < (to a) ∧
        (match m with
         | nil ⇒ (from a) = tfrom ∧ (to a) = tto
         | m ⇒ (from a)=tfrom ∧ (accds_consistency m (to a) tto)
         end)
| nil ⇒ True end.
```

7- Proof of sufficient means of subsistence during the intended stay in the Schengen area. Varies from country to country. To complete Supporting document to attest sponsor's readiness to cover your expenses during your stay Proof of prepaid accommodation Document about accommodation in private Proof of prepaid transport.

```
Inductive sufficient_means:=
| Bank_statement : (Authority → Prop) → sufficient_means
| Credit_card : (Authority → nat → Prop)-> sufficient_means
| Cash : (nat → Prop) → sufficient_means
| Employment : (Authority → Prop) → sufficient_means.

Variable means_of_sufficiency : country → sufficient_means → Prop.

Record Schengen_demand := mkDemand
{ form: schengen_form; picture: photo;pass: passport; travs: trav_itinerary; insurance: trav_health;
  lodgings: accds;sufficient: sufficient_means; time_stamp:time; }.

Definition schengen_demand_valid (requester:Authority)(d_form:Schengen_demand):=
 let demand := form (d_form) in let C := schengen_country demand in
 let Consul:= consulat_of C in let tfrom := schengen_from demand in
  let tto:= schengen_to demand in
 (Consul |> (schengen_form_requirement demand)) ∧
 (requester |> passport_photo (picture d_form)) ∧
 (travs_valid (travs d_form) ∧ travs_consistency (travs d_form) tfrom tto)/\
 ((trav_health_emitter (insurance (d_form))) |> ( trav_health_valid (insurance (d_form)))) ∧
 ((accds_valid (lodgings d_form))/\ (accds_consistency (lodgings d_form) tfrom tto)) ∧
 ((means_of_sufficiency C) (sufficient d_form)).
```