





decode



D3.8



Decentralised models for data
and identity management:



Blockchain and ABC MVPs





Project no. 732546

DECODE

DEcentralised Citizens Owned Data Ecosystem

[D3.8] ["Decentralised models for data and identity management: Blockchain and ABC MVPs"]

Version Number: [V1.2]

Lead beneficiary: [RU]

Due Date: [dec 2018]

Author(s): Paulus Meessen (RU), Marloes Venema (RU), Alberto Sonnino (UCL), Shehar Bano (UCL)

Editors and reviewers: Alberto Sonnino (UCL), Sam Mulube (TH), Denis Jaromil Roio (DYNE)

Dissemination level:		
PU	Public	x
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Approved by: Francesca Bria (Chief Technology and Digital Innovation Officer, Barcelona City Hall)

Date: [31/12/2018]

This report is currently awaiting approval from the EC and cannot be not considered to be a final version.

Contents

1	Introduction	2
1.1	Document Outline	2
2	Decentralized Identity Management	3
2.1	Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers	3
2.1.1	Applications to Smart Contracts	3
2.1.2	System Overview	3
2.1.3	Implementation	4
2.1.4	Performance	4
2.1.5	Summary	4
3	Access Control	5
3.1	Enforcing Access Control with Attribute-Based Encryption	5
3.1.1	Key-policy or ciphertext-policy ABE	5
3.1.2	Collusion resistance and security	6
3.1.3	Decentralization	7
3.2	Revocation and other dynamic properties	7
3.2.1	ABE for DECODE	8
3.2.2	Conclusion	8
3.3	Credential Disclosures as Access Control Mechanism in Smart Contracts on Public Distributed Ledger Technologies	9
3.3.1	Introduction	9
3.3.2	A primer on Attribute Based Credentials ¹	10
3.3.3	A primer on Distributed Ledger Technologies	11
3.3.4	Using ABCs on DLTs as access control.	12
3.3.5	Practical Properties of DLTs to create a Nonce	12
3.3.6	Structures of Smart Contracts that preserve freshness	15
3.3.7	Analysis	19
3.3.8	Conclusion	21
3.4	Privacy-preserving petition	22
4	Conclusion	23

¹cf. [32]

1 Introduction

In DECODE, we propose an Attribute Based Cryptography access model in which some or all of the data, some or all of the rules, and even some or all of the credentials may be stored in a distributed manner on a ledger. Whenever data, rules and credentials meet at a DECODE node, we determine whether it may be released. This poses several challenges the DECODE project aims to address, providing answers that are informed by previous research like ABC4trust (FP7-ICT-2009-5, proj. nr. 257782) and Sentinels MobileIDM (IIPVV).

This task considers four questions about the use Attribute Based Cryptography in DECODE.

1. How to securely issue and use credentials which are stored in the distributed ledger?
2. How to exert control over such remotely or rather distributed stored credentials?
3. How to combine attribute-based encryption and attribute based credentials to enforce the secure storage of data on the distributed ledger while allowing the right entities to access data?
4. How to authorize transactions on the distributed ledger using anonymous credentials?

This task is about a decentralized model for Attribute Based Cryptography (ABC) and digital identity management and identifying functional technologies that can be used for the project.

In the first deliverable of the task; *initial Decentralized models for data and identity management: Blockchain and ABC* [33], we highlighted the relevant areas for research in the domains of attribute based cryptography, identity management and distributed ledgers. An overview of current state of the art technologies was provided and, when available, references to software implementations of these technologies as relevant to DECODE were included.

In this deliverable we also follow up on the Coconut credential system [53], which was initially proposed in [33]. We now provide an overview of the system and illustrate how it is being implemented in DECODE.

We also address the challenges set out in the previous deliverable regarding the use of attribute based cryptography as access control. We investigate Attribute Based Encryption as a means of access control of data in distributed systems like DECODE. Attribute Based Credentials are shown be usable a means of access control on the execution of smart contracts. We investigate the requirements of making ABCs a general building block access control of execution in distributed ledger technology.

Finally we show a contract for privacy-preserving petitions in DECODE.

1.1 Document Outline

Then sections 2 and 3 will address the specific challenges of Identity Management and Access Control respectively. Section 2 provides an overview of the Coconut Credential System developed for DECODE. Section 3 investigates two means of doing decentralized access control using Attribute Based Cryptography, and shows of a method for privacy-preserving petitions in DECODE.

2 Decentralized Identity Management

This section describes the Coconut credential system [53] as is used in DECODE.

We provide a general overview of Coconut authored by Shehar Bano. The text in this section was published earlier at Bentham's Gaze. Some adaptations to the formatting have been made in order to be part of this document.

2.1 Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers

Selective disclosure credentials allow the issuance of a credential to a user, and the subsequent unlinkable revelation (or 'showing') of some of the attributes it encodes to a verifier for the purposes of authentication, authorisation or to implement electronic cash. While a number of schemes have been proposed, these have limitations, particularly when it comes to issuing fully functional selective disclosure credentials without sacrificing desirable distributed trust assumptions. Some entrust a single issuer with the credential signature key, allowing a malicious issuer to forge any credential or electronic coin. Other schemes do not provide the necessary re-randomisation or blind issuing properties necessary to implement modern selective disclosure credentials. *No existing scheme provides all of threshold distributed issuance, private attributes, re-randomisation, and unlinkable multi-show selective disclosure.*

We address these challenges in our new work Coconut - a novel scheme that supports distributed threshold issuance, public and private attributes, re-randomization, and multiple unlinkable selective attribute revelations. Coconut allows a subset of decentralised mutually distrustful authorities to jointly issue credentials, on public or private attributes. These credentials cannot be forged by users, or any small subset of potentially corrupt authorities. Credentials can be re-randomised before selected attributes being shown to a verifier, protecting privacy even in the case all authorities and verifiers collude.

2.1.1 Applications to Smart Contracts

The lack of full-featured selective disclosure credentials impacts platforms that support 'smart contracts', such as Ethereum, Hyperledger and Chainspace. They all share the limitation that verifiable smart contracts may only perform operations recorded on a public blockchain. Moreover, the security models of these systems generally assume that integrity should hold in the presence of a threshold number of dishonest or faulty nodes (Byzantine fault tolerance). It is desirable for similar assumptions to hold for multiple credential issuers (threshold aggregability). Issuing credentials through smart contracts would be very useful. A smart contract could conditionally issue user credentials depending on the state of the blockchain, or attest some claim about a user operating through the contract - such as their identity, attributes, or even the balance of their wallet.

As Coconut is based on a threshold issuance signature scheme, that allows partial claims to be aggregated into a single credential, it allows collections of authorities in charge of maintaining a blockchain, or a side chain based on a federated peg, to jointly issue selective disclosure credentials.

2.1.2 System Overview

Coconut is a fully featured selective disclosure credential system, supporting threshold credential issuance of public and private attributes, re-randomisation of credentials to support multiple unlinkable revelations, and the ability to selectively disclose a subset of

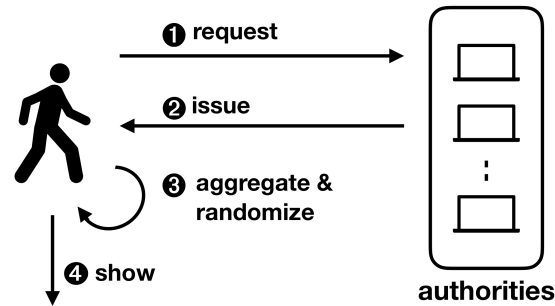


Figure 1: The Coconut architecture.

attributes. It is embedded into a smart contract library, that can be called from other contracts to issue credentials. The Coconut architecture is illustrated below. Any Coconut user may send a Coconut *request* command to a set of Coconut signing authorities; this command specifies a set of public or encrypted private attributes to be certified into the credential (1). Then, each authority answers with an *issue* command delivering a partial credentials (2). Any user can collect a threshold number of shares, aggregate them to form a consolidated credential, and re-randomise it (3). The use of the credential for authentication is however restricted to a user who knows the private attributes embedded in the credential—such as a private key. The user who owns the credentials can then execute the *show* protocol to selectively disclose attributes or statements about them (4). The showing protocol is publicly verifiable, and may be publicly recorded.

2.1.3 Implementation

We use Coconut to implement a generic smart contract library for Chainspace and one for Ethereum, performing public and private attribute issuing, aggregation, randomisation and selective disclosure. We evaluate their performance, and cost within those platforms. In addition, we design three applications using the Coconut contract library: a coin tumbler providing payment anonymity, a privacy preserving electronic petitions, and a proxy distribution system for a censorship resistance system. We implement and evaluate the first two former ones on the Chainspace platform, and provide a security and performance evaluation. We have released the Coconut white-paper, and the code is available as an open-source project on <https://github.com/asonnino/coconut> Github

2.1.4 Performance

Coconut uses short and computationally efficient credentials, and efficient revelation of selected attributes and verification protocols. Each partial credentials and the consolidated credential is composed of exactly two group elements. The size of the credential remains constant, and the attribute showing and verification are $\mathcal{O}(1)$ in terms of both cryptographic computations and communication of cryptographic material - irrespective of the number of attributes or authorities/issuers. Our evaluation of the Coconut primitives shows very promising results. Verification takes about 10 ms, while signing an attribute is 15 times faster. The latency is about 600 ms when the client aggregates partial credentials from 10 authorities distributed across the world.

2.1.5 Summary

Existing selective credential disclosure schemes do not provide the full set of desired properties needed to issue fully functional selective disclosure credentials without sacri-

ficing desirable distributed trust assumptions. To fill this gap, we presented Coconut which enables selective disclosure credentials - an important privacy enhancing technology - to be embedded into modern transparent computation platforms. The paper includes an overview of the Coconut system, and the cryptographic primitives underlying Coconut; an implementation and evaluation of Coconut as a smart contract library in Chainspace and Ethereum, a sharded and a permissionless blockchain respectively; and three diverse and important application to anonymous payments, petitions and censorship resistance.

We have released the Coconut white-paper, and the code is available as an open-source project on GitHub. ²

3 Access Control

This section provides three applications of Attribute Based Cryptography in DECODE. In the first subsection we have a comparative review of the state-of-the-art in Attribute Based Encryption, as it is relevant to the DECODE project. The second subsection investigates the general requirements to use Attribute Based Credentials as a means of access control on the execution of smart contracts in Distributed Ledger Technology. The final subsection provides the details of the smart contract used in the DECODE petition pilot.

3.1 Enforcing Access Control with Attribute-Based Encryption

Whereas attribute-based credentials (ABCs) may be used for authentication and proving possession of attributes, attribute-based encryption (ABE) [48] can be used as a means to enforce access control on the private data of a participant, i.e. the participant encrypts the private data under an access policy such that other participants in the system can only decrypt the data if they possess a set of attributes (and their associated keys) that satisfies the policy. This means that ABE provides us with a way to implement attribute-based access control on a cryptographic level. One of the features of ABE is that the setup can be decentralized: instead of allowing one authority to control all of the keys associated with attributes, multiple authorities generate the keys associated with their own unique set of attributes. The idea is that the user can authenticate towards the authority by showing his attribute-based credential, and therefore prove possession of the associated attribute, for which the authority then generates a key. However, how this is done is not entirely trivial, which means that further investigation is required in order to implement them in a joint fashion. In order to determine whether this is possible we first have to find a suitable scheme that provides functionality that DECODE requires.

3.1.1 Key-policy or ciphertext-policy ABE

In general, attribute-based encryption comes in two flavors: the access policy can be imposed on the keys and on the ciphertexts. In the former case, the access policies are specified and encoded in the secret keys by the key generation authority. The idea is that such a key can be received by a participant in the system when he possesses a set of attributes that satisfies the access policy. It is therefore also called key-policy attribute-based encryption (KP-ABE) [24]. In KP-ABE, the ciphertexts are associated with a set of attributes, which can be specified by the encryptor, and another participant can only decrypt these ciphertexts when he possesses a key associated with an access policy that

²The Coconut project is developed, and funded, in the context of the EU H2020 Decode project, the EPSRC Glass Houses project and the Alan Turing Institute.

is satisfied by the attributes associated with the ciphertext. In ciphertext-policy attribute-based encryption (CP-ABE) [8], the opposite is true. If we associate the ciphertexts with access policies, then the encryptor gets to specify the access policy and therefore who can decrypt the ciphertexts. Then, the secret keys are associated with attributes that the participant possesses, and a ciphertext can be decrypted whenever the associated access policy is satisfied by the set of attributes associated with the key. Because CP-ABE provides us with the functionality that we need to enforce access control, it is the more natural option for DECODE.

In ABE schemes, there are often restrictions on the access policies. These access policies are often represented as Boolean formulas, and ideally, we would like to allow for any Boolean formula to be integrated in the ciphertexts. However, not all schemes allow for this level of expressiveness. Some only allow for AND-statements [16], while others allow for all possible access policies to be integrated in the keys [37]. An important aspect that is related to this is the monotonicity of access policies, which means that the Boolean formula can either consist of negations or not. If we do not allow for negations, we say that the access policies are monotone [24]. If we do, then they are non-monotone [37]. While the non-monotonicity of access policies seem like a desirable property, we argue that it is not. Recall that we aim to use ABE in combination with ABC, i.e. participants prove their right to a certain secret key towards an authority by showing the associated credential. Whereas proving possession of an attribute is trivial in this case, proving the lack thereof is absolutely not. Because of this reason, we prefer monotone access structures, while still remaining as expressive as possible such that all access policies can be represented by monotone Boolean formulas.

Another property that is somewhat related to this is the set of attributes for which we generate secret keys. In ABE, this is called the universe of attributes [48]. While in the majority of the schemes [15, 16, 24, 30, 36, 37, 42, 48, 57, 60], this universe is fixed after the setup and the size of the universe is polynomially bound in the security parameter, there are also schemes that allow for more flexibility in the generation of system parameters [8, 24, 44, 48]. In particular, these constructions are practically unbounded in the number of attributes, and is therefore called a large universe construction, while its bounded counterpart is called a small universe construction. Because large universe constructions allow us to trivially add new attributes without having to run the entire setup again, and therefore provides a level of dynamicity that DECODE requires, we prefer large universe constructions over their small counterparts. Moreover, the fact that the system parameters are independent on the number of attributes also adds a level of scalability that small universe constructions do not enjoy.

3.1.2 Collusion resistance and security

In traditional public-key encryption schemes, semantic security is an important principle in ensuring that ciphertexts cannot be decrypted by unauthorized users. Whereas semantic security only provides security against attacks on the ciphertexts in the form of chosen-plaintext attacks, in attribute-based encryption, this type of security is not sufficient. In ABE, the users each have their own set of keys associated with the attributes they possess. While it might be the case that, individually, they cannot decrypt some ciphertext, it might be the case that a subset of the users can combine their keys in order to decrypt the ciphertext by colluding. Schemes that provide security against this type of attack are called collusion resistant.

3.1.3 Decentralization

While the most basic version of ABE requires a single trusted third party to generate the keys, it is also possible to distribute the trust across multiple parties, which is called multi-authority attribute-based encryption (MA-ABE) [13]. Whereas some schemes require either a central authority (such as [13] or communication between all of the authorities during the key generation [29], others are entirely decentralized after the global parameters are generated [30]. Because DECODE aims for the decentralization of trust, it is therefore also logical that access control is enforced in a decentralized setting, i.e. we should opt for a fully decentralized MA-ABE scheme.

In decentralized ABE, we allow for corruption of authorities to some extent. In most schemes, we either allow that almost all (except one or two) authorities may be corrupt without losing security guarantees. Other than actively attempting to decrypt ciphertexts, authorities might infringe upon other privacy rights of the participants. In MA-ABE, participants are identifiable with a global identifier GID , which is used to link keys to one participant. However, showing this GID to an authority opens up the possibility of a linkability attack: authorities can collude and link a set of attributes to the same GID . In order to prevent against this type of attack, there have been several proposals for anonymous key issuance and blind key generation algorithms [14, 25, 39].

3.2 Revocation and other dynamic properties

Because the basic version of ABE does not satisfy any dynamic properties other than the level of dynamicity that large universe constructions provide, we might want to add a level of dynamicity in terms of revocation, because in DECODE, we might require that users are revoked. DECODE specifies that attributes are of a temporal nature, and attribute-based credentials may expire as well, which means that keys associated with one single attribute need to be revocable as well. Whereas in traditional public-key encryption schemes, we can simply blacklist keys, this is not an option for ABE, as all messages are encrypted under public keys for which holds that the private counterparts are held by more than one participant [38]. In literature, there are two generic solutions to this problem: direct or indirect revocation of users [7].

While the word ‘directly’ seems to imply that keys can be revoked in an immediate fashion, it actually specifies how revocation is implemented: either directly in the ciphertext or indirectly by re-encrypting the ciphertexts and renewing (a part of) the keys. In the former case, we do not need any communication between the key generation authorities and the users [9, 29, 52], while in the latter, we do [17, 40, 47, 60]. Because direct revocation does not provide us with a trivial way to revoke keys on an attribute level, it is less suitable for DECODE. Moreover, as directly revocable approaches require the list of participants to be public, as well as the revocation status, there might be some privacy issues related to this list. Another drawback is the scalability of such systems. The ciphertexts grow at least logarithmically in the number of users in the system, while its indirectly revocable counterparts keep their usual size. For these reasons, we prefer indirectly revocable ABE over its direct counterpart.

Other than revocable properties, we also need other dynamic properties. For instance, DECODE allows anyone to join the network and host a node. We would also like to implement this the decentralized ABE setting, i.e. any authority can join the scheme. In order to make this possible, we have to require that no inherently confidential parameters are generated in the global setup (which is the case in schemes such as [30]). This means that we cannot use any scheme that utilizes composite-order groups, but we cannot use any scheme that divides the generation of the master secret key across the authorities in some fashion (e.g. [13, 14]).

3.2.1 ABE for DECODE

In short, for DECODE we require an ABE scheme that is ciphertext-policy based, in which the policies are expressive but monotone, and the universe of attributes is large. Moreover, we also require that the key generation is fully decentralized and that the key issuance protocol is anonymous. Finally, the scheme has to provide some revocation functionality in order to make it truly compatible with the ABCs. If we consider the extensive survey in [55], we can conclude that such a scheme does not exist. However, there is one scheme that satisfies almost all properties: the scheme as proposed by Rouselakis and Waters in [45] satisfies almost all properties, but does not protect the privacy of the users towards the authorities in the form of unlinkability, and it does not provide any revocation functionality either. Because of the homomorphic properties of the scheme, we are confident that such functionality can be implemented by exploiting the generic design of revocation in [47].

Nevertheless, whereas this scheme satisfies almost all conditions that we formulated, there are also some issues that we have not taken into account yet, mostly related to security, but also the storage and computational costs. While the scheme is secure against chosen-plaintext attacks, we cannot use the generic methods as described in [12, 59] to obtain security against chosen-ciphertext attacks. We can use other generic methods that apply non-interactive zero-knowledge proofs in order to achieve the same security guarantees [31, 35, 46], but these generally lead to more expensive protocols. Moreover, the [45] scheme has a problem that many other (but certainly not all (e.g. [2])) ABE schemes have, i.e. the computational costs of the scheme depends heavily on the number of attributes and therefore becomes too expensive to run on smartphones and other IoT devices [56]. For this reason, we think that the resulting scheme is too inefficient in practical setups such as DECODE, which are supposed to be smartphone and IoT friendly.

Furthermore, we can imagine that some of the attributes that we want to use in DECODE have many possible values. For instance, the age of a user, or perhaps even the date of birth, which are examples of range attributes. Because in many ABE schemes the universe is small, we have to define a public parameter for each value of each attribute. While this is generally not a problem in large universe constructions (and this would be something we avoid altogether in choosing a large universe construction for DECODE), it does become an issue when a user attempts to encrypt a ciphertext under a policy that specifies an age range or range of dates spread across several years. If the ciphertexts grow in the number of these attributes, they potentially become extremely large, which is the case in the [45] scheme as well. In order to mitigate this problem, we can use generic conversion methods as proposed in [5] such that the ciphertexts still grow in the number of values, but only logarithmically, or we can solve this issue altogether by using a large-universe ABE scheme with constant-size ciphertexts [1, 6, 15, 20].

3.2.2 Conclusion

Because there are, to the best of our knowledge, no schemes that satisfy all of the properties that we require for DECODE, and because the best alternative scheme, namely the [45] scheme, has its own disadvantages that makes it less than ideal for DECODE, we believe that the perfect scheme does not exist yet. For this, there are several reasons:

- There are no efficient schemes that take access policies that denote ranges into account;
- There are no efficient *and* secure revocation techniques;

- There are some schemes that have constant-size ciphertexts and fast decryption algorithms, but they do not satisfy other desirable or even necessary properties.

While there are some schemes that have constant-size ciphertexts and fast decryption algorithms, there is only one that also supports large universes of attributes and monotone, expressive access policies, namely the scheme that is described in [1] and instantiated in [55]. However, this scheme is centralized and has large sets of secret keys.

Because of the lack of a truly suitable scheme, we conclude that more research has to be conducted in order to overcome some of the issues as described in this work.

3.3 Credential Disclosures as Access Control Mechanism in Smart Contracts on Public Distributed Ledger Technologies

3.3.1 Introduction

In this section we investigate the possibility of using Attribute Based Credentials [4, 10, 11] as a means of access control [49–51] in the execution of Smart Contracts [54] on Distributed Ledger Technology systems.

Initially this seems straightforward. We can use existing implementations of Attribute Based Credentials (e.g. [53],[27] or [28]), and translate the code for verification of credentials into code that runs as a Smart Contract on a DLT system.

Users will have a wallet-application with some Credentials issued in the existing system. For example, users may have a credential issued by their municipality with their name, address and date of birth as attributes. They can use their wallet to create a zero-knowledge proof: proving that they have such a credential, which contains the attribute that they live on ‘main street’, while keeping all the other attributes hidden.

This zero knowledge proof then may be used as an input for application that runs on a DLT. The application could for example be: a register where neighbors can announce a party or a barbecue, and everyone who is able to prove that they live in that street may be able to veto this event.

The main challenge arises from the inability to run interactive protocol on a Distributed Ledger system. The attribute based credential protocols [4] require a short, but interactive session between the user and the disclosing party to guarantee freshness of the proof. Freshness of disclosure proofs is required to make sure that the credentials shown to the smart contract are not replayed from a previous session. Since the DLTs we consider in this section are public systems, previous proofs will be available, and thus not trivially usable for an access policy.

In this section we propose three features present in most DLTs to be able to generate a `nonce` (‘number only used once’) that can be used as a proxy for the freshness.

- disclosure proofs must be bound to the change of global state (or equivalent proxy),
- disclosure proofs must be bound to a specific contract and a specific method,
- all inputs and parameters to that change are bound to the disclosure, and
- if the privacy of users is not affected (beyond the notion of (digital) identity that is inherent to the specific DLT system), disclosures should be bound to that identity.

We give example code of using these requirements in a Solidity [21] contract. This argues for the feasibility of actually using ABCs as an access control mechanism, even on (public) DLTs.

We do note that the practical differences between DLTs can still have a lot effect on the security and user experience. In particular to the availability of the service provided by smart contract.

Finally, since it is in line with the DECODE ambition [18] to allow all users to create these kind of ‘smart rules’, we argue that even our Solidity example is not simple enough, so provide some constraints and suggestions for improvement.

3.3.2 A primer on Attribute Based Credentials³

Attribute Based Credentials (ABCs) are a cryptographic alternative to traditional credentials like driver’s license and passports [4, 10, 11, 23, 53]. ABCs contain a set of attributes, typically encoded as numbers, that a users can selectively reveal to the verifier. Even when attributes are hidden, the verifier can still assess the validity of the credentials.

A typical ABC based scheme comprises of the following parties:

- **Issuer**

The issuer issues credentials to users. It ensures that the correct data are stored in the credential. A typical credential scheme has multiple issuers.

- **User**

The user holds a set of credentials, obtained from one or more issuers. She can disclose a (user defined) selection of attributes from any number of her credentials to a verifier to obtain a service.

- **Verifier**

The verifier, sometimes called relying party or service provider, checks that the credential is valid, the revealed attribute are as required and - when supported - if the credential is not revoked. Based on the outcome, it may provide a service to the user.

To this paper there is a particular interest in the disclosures.

From a user perspective, we assume that the disclosure proofs in the ABCs support selective disclosure, and have the unlinkability property. Selective disclosure means that the user can select a subset of attributes in a credential that are meant to be shown, while hiding the others. Unlinkability means that it is impossible to show that two disclosures originate from the same credentials based on the disclosure proofs.

Conventionally, disclosures happens in an interactive two step protocol between the service provider (the verifier) and the user.

0. The user starts the protocol by setting up a connection to the service provider.
1. The service provider tells the user which attributes, from which credentials are required and sends along a `nonce`.
2. The user creates a zero-knowledge proof that relies on the `nonce` for proper verification, proving ownership of the credentials and showing that the requested attributes are contained in the credentials used in the proof.

The service provider verifies the proof, and can subsequently provide the service to the user.

The service provider relies on the [41, p. 46] to prevent replay attacks. In a replay attack, a previously used or previously created proof is presented again, and if successful accepted by the verifier.

The security of accepting an ABC disclosure, therefore depends on the ability of the verifier to enforce the freshness of the `nonce`.

³cf. [32]

3.3.3 A primer on Distributed Ledger Technologies

A 'ledger' is traditionally an **append-only** (paper) ordered list of transactions. For example, such as bankers may have used for bookkeeping in the time before computers.

A *distributed* ledger is a distributed data structure, where a set of bookkeeping nodes collectively maintain a global state.

Distributed Ledger Technologies (DLT) are networks of computers that uses a (peer-to-peer) consensus mechanism maintain a distributed digital ledger of transactions.

It may depend on the implementation what kind of transactions the consensus is achieved on. For example, it can be about financial transactions [34] or correctness of computations [58].

Many of these distributed systems are open for participation and auditable by anyone. A typical public DLT system comprises of the following components:

- **Node**

The DLT system is built from nodes. All nodes are responsible for reaching (and maintaining) consensus. The nodes communicate to each other, about inputs received, the state of the network and the results from consensus.

- **Ledger**

The ledger is a append only data structure that is replicated across all the nodes in the system. Appends only occur through the consensus mechanism. A ledger of represents the *state* of a DLT. Sometimes this includes the input and output related to every successful interaction with the system. Sometimes it just contains a valid checksum to verify correctness of past events.

- **User**

Users interact with the network by sending it transactions. If the network supports user-identities, these usually are represented as pseudonymous public-private key pair. Transactions are signed using this key pair; binding them to a single user. Users often can also incentivise the network to prioritize the processing of their transactions by including monetary rewards to the processing.

To this paper there is a particular interest in the DLTs that allow Smart Contracts to be processed as transactions.

- **Smart Contract**

A user may include code in a transaction that must be stored in the network. In the future any other user, or node can find this code and execute it. A user may call an existing smart contract in the system with a certain input. Such a transaction will cause every node involved in the consensus to execute the code with that input. A transaction to a smart contract is only included in the ledger if consensus is achieved about the correctness of the execution. Sometimes it is relevant that the execution takes place in public on the network. At other times the smart contract only needs code to verify an input in order to accept a transaction.

For a call to a smart contract to be accepted on a DLT, all the nodes should be able to verify and audit transactions. That is: the input of a call will always result in the same outputs, given a global state.

In order to achieve this, all code for the verification of smart contracts is also included in the ledger. Often the verification code of a contract is the actual execution code [3].

Contracts can therefore only consider information already in the ledger, or inputs to the call for the verification. Dependencies on other sources of input will invalidate the integrity of the ledger in an audit, or cause immediate disagreement between the (bookkeeping) nodes about the verification.

3.3.4 Using ABCs on DLTs as access control.

Disclosures of Attribute Based Credentials can be used to make a decentralized access control decision on the execution of a service.

Initially there are seem very little limitations. The cryptographic code can be ported to run on many different DLTs as part of a smart contract. The user should be aware that the disclosure is not only public, but also irremovable from the ledger. However, this is precisely why the selective disclosure, and unlinkability properties are essential for this application. These guarantee that the user cannot be identified beyond the anonymity set of people that own credentials with these attributes - based on the disclosure proofs.

A smart contract however, cannot perfectly generate the functionality of a centralized server when it verifies the disclosure: A call to the DLT cannot be interactive, as it has to be replicated to all the nodes in the network. (and it is beyond impractical to have every user set up a session with every verifying or auditing node.)

This means that the generation of the `nonce` used for the freshness in the ABC protocol, now also has to be replicated across the nodes in the DLT. And this poses a challenge. We can no longer just use sequence numbers in live networked sessions, nor can we rely on a centralized source of randomness to make sure an attacker is unable to prepare a disclosure.

Conventionally, we still cannot trust the users that call the smart contracts to have any control over the `nonces` since these now can just look up the proofs from previous disclosures on the ledger.

The problem we aim to solve with the next sections, is to find an way to implement a version of a disclosure protocol that works for smart contracts, and still has the freshness property.

We must include these disclosures to the transaction calls to the DLT, and must make sure that it is recorded into the ledger in a manner that is verifiable and auditable.

This all must be practical for the users, and not impede on the availability of the contract.

Furthermore, we will make a suggestion for the syntax to use to present such a protocol to the writers of a smart contract. The writers should have an interest in the security that comes from proper freshness. The users of the smart contracts should also be able to understand which attributes from which credentials are requested and what the purpose of their use is [26].

3.3.5 Practical Properties of DLTs to create a Nonce

In this section we state four requirements to achieve freshness of disclosures on DLT using `nonces`.

In an initial attempt to solve our problem, we could try to move just the verifying code of the disclosure proof in the protocol to the DLT. Setting up the session and granting the service would still happen outside of the DLT system. The advantage of such a situation is that the relying party does not need to run the cryptographic code, and can rely on the high integrity results recorded in the ledger.

Instead of sending the `nonce` to the user, the relying party hard codes the `nonce` and other details for the disclosure proof into a smart contract and uploads this contract to some DLT system. The user receives an address to the location of this smart contract on that DLT. She can extract the `nonce` and other details from the smart contract on the ledger. This allows her to make a disclosure proof. The user then includes this disclosure proof in a call to that smart contract. The DLT evaluates the call to the smart contract, and if the proofs are valid, the transaction is recorded in the ledger. The relying party can now periodically check the ledger, and once it finds a valid proof to their smart contract has been recorded in the ledger, they will proceed and provide the service to the user.

The relying party must have a security model that allows this part of the interaction to be public! Issues where another user accidentally or maliciously creates a disclosure proof must be considered. Beyond that, there are no fundamental problems to move this part of the interaction to a DLT system.

The same mechanism could also be used in services that fully run on the DLT and have methods that are only ever called once. (Or in cases or where the outcome of such an access control decision is idempotent.)

One would use a flag in the global state to represent the access control decision, and append a method to the contract that only has the functionality to consume the proof and set that flag.

Envisioned use might be in a smart contract helps to organize some public event, but allows owners of specific attributes to *veto* the event. For example, the organization of neighborhood party, that can be vetoed by anyone living in the street where the event will take place.

The first challenge properly arises when we consider how to handle multiple disclosures to the same method.

There exists a special case of this, in the case where a single user will only call a particular method once. In that case, we demand that the disclosure proof is extended with another zero-knowledge proof, containing an encrypted hash of the secret key of the credential [19, 53]. This encrypted hash always needs to have the same value for a call this method. If the DLT permits to store a list of these hashes, we can reject multiple disclosures to the method from a single user, and by extension replay by others.

What we learn from this method of creating one time pseudonyms, is that any disclosure must necessarily be bound to the particular method and contract that is being called [19]. Otherwise one could re-use the disclosure proof for other contracts. A possible language pattern for doing disclosures on a DLT should therefore include an automatic reject of disclosure proofs that are not meant for that particular method in that specific contract.

Aside from this special case of single individual calls to a contract, we have to find a way of determining freshness when a smart contract may be called multiple times by a single user.

We need notion of freshness in DLTs, and this needs to be replicable among all the verifying nodes in the DLT. For example, by generation of a `nonce` - which may be non-trivial on certain DLTs [43].

The nodes then need to determine (mostly/in majority) identical values for this `nonce`. Once the transaction has been recorded in the network, anyone who audits the correctness of transactions, i.e. the change of the global state, should deterministically be able to reconstruct the same values.

At this point we do need to make an extra assumption on functionality of the DLT systems used. Many DLTs of course allow contracts to explicitly store (significant amounts of) information on the ledger. For privacy or performance reasons, some might actually not want have this feature [53]. To have a more universal notion of what a DLT can be,

we will assume that there is no ability to persistently store data(structures) on the DLT system.

This disables our option to start a contract with a given `nonce`, and update the value for every new disclosure by hashing it with an accepted disclosure proof, and store this nonce on the ledger. For performance this might even be a bad idea anyway. It creates a bottleneck for throughput, when users have to compete among each other to get the current value of the nonce.

Another way to introduce a notion of time and freshness, would be to install an oracle(-smart contract) somewhere on the DLT that can publish freshly-generated random numbers on a regular interval. This however introduces a central party in the system. Such central parties needs to be trusted by the writers of the smart contract, which goes counter to our goals [18] The availability of the central party also becomes a single point of failure for the methods that depend on disclosures.

DLTs that use a 'Blockchain' or hashchain as a data structure to represent the ledgers may however already something similar to such an oracle already built in. The hash-value of the last block in the chain often is a (pseudo) random number. It should only occur once, and needs to be out of the control of the users. Furthermore it represents an epoch of time, which can be agreed upon across the network of nodes.

To make this more general, if a transaction is processed and included in the ledger, it updates the global state of the DLT. The current global state should always be accessible to the nodes on the network, and as such to the users of the smart contracts. It therefore makes sense, to only accept disclosure proofs that are bound to the current representation of the global state. (This will only be useful if the epoch between changes of the global states is long enough to construct the call. But more on that later.)

Now we still have to account for the replay of proofs within an epoch. If the DLT does not support some system of user accounts, there is probably no mechanism to prevent replication with the epoch. The smart contracts therefore should be written in such a way that this can not cause ill effects.

Within the epoch have to make sure, that a call to a method will have the effects intended by the disclosing user. A contract that offers a toggle switch for some flag might functionally do precisely the opposite what the disclosing user had hoped for, if some third party manages to replays that call. While the possibility of a call failing might always exists, this now provides (limited) access to users who do *not* have the required credentials. If the user had bound the decision 'toggle to `on`' to their disclosure proof, no replay will grant agency with any unintended action.

Finally we have the option to bind disclosure proofs to the (pseudonymous) notions of identity present in the DLTs. If an account - for example to pay transaction fees - is bound to calls to the DLT: then no additional identifying information is leaked by also binding a disclosure to this an account. The writer of the smart contract can rely on the non-replayable nature of contract calls, to also prevent replay of disclosure proofs.

Summing up our requirements; in order to guarantee freshness for multiple disclosures to a smart contract, we have to make sure that:

- disclosure proofs are bound to the change of global state (or equivalent proxy),
- disclosure proofs are bound to a specific contract and a specific method,
- all arguments in the call to that method must also be bound to the disclosure proof, and
- if the privacy of users is not affected beyond the existing notion of (digital) identity in the DLT system, disclosure proofs should be bound to these identities.

3.3.6 Structures of Smart Contracts that preserve freshness

In this section we present two examples of smart contracts, written in Solidity [21]. The first contract is an idempotent disclosure method that sets a flag whenever it first encounters a valid disclosure proof based on a hard coded `nonce`. The second contract offers the ability to have access control through credentials for multiple disclosure proofs. It achieves this through implementation of the requirements stated in the previous section.

The code for the cryptographic functions to verify the disclosure proofs is omitted, since this is going to be specific for the ABC system used. In Ethereum [58] - the smart contract platform used for these examples, there are no native cryptographic building blocks to use in ABC systems. The cryptographical code therefore would have to be run as a smart contract, which means that implementations would be extremely slow and expensive. The same holds for the `scheme` of the attributes, if we wish to use their values as in the execution of the method. A credential scheme specifies how a translation is made between the attributes, and the mathematical objects used in the cryptography.

We use the `modifier` feature of functions in Solidity to create a syntax for the developers to specify which credentials are required [21]. A `modifier` is a function macro that allows code to be executed before and after an actual function is called. This allows credential wallets to parse the contracts, and prevents the methods to be executed if the disclosures are not valid. The `modifiers` in Solidity unfortunately do not allow us to enforce what arguments are passed through the function.

Idempotent disclosure The below code is a simple example that demonstrates the functionality and required overhead for a disclosure in the simple case.

The Data required to create the disclosure proof can be queried using the `Show_Nonce` and `Show_Public` methods. The `ABC_crypto_verify` function is omitted, but assumed to handle the cryptographic computations to verify the disclosure proofs for the ABC system.

The value of `_nonce` is hardcoded by the writer of the smart contract, but - depending on the security model - could also be some pseudo random value generated at the publication of the smart contract. For example the address of the contract on ledger. (Which is available as `address(this)`.)

Code

Listing 1: Solidity contract with single use access policy

```
pragma solidity ^0.4.25;

contract Idempotent_disclosure {

    bool private contract_is_activated = false;

    // hardcoded nonce.
    bytes32 private _nonce = 0
        xDEC0DED4C0FFEE4ABC4C0DE4ACEDFEED4DEADBEEF4ABBA4BA0BAB;
    // hardcoded public values for credential
    bytes[] private _public = [ /* ... */ ]

    function Show_Nonce()
    public
    view
    returns (bytes32) {
        return _nonce;
    }
}
```

```

function Show_Public()
public
view
returns (bytes[]) {
    return _public;
}

function Show_Is_Active()
public
view
returns (bool) {
    return contract_is_activated;
}

function ConsumeCredential (uint256 proof)
private
returns (bool)
{
    assert(ABC_crypto_verify(_public , proof , _nonce));
    return true;
}

function CredentialTrigger (uint256 proof)
public
returns (bool)
{
    if( contract_is_activated == false){
        assert (ConsumeCredential(proof));
        contract_is_activated = true;
        return true;
    }else{
        return true; // returns true if contract already has been
                    activated
    }

    return false; // should be unreachable. something went wrong
}

modifier Do_not_run_if_not_activated () {
    require(
        contract_is_activated,
        "Contract_has_not_been_activated,_please_provide_the_required
        _credential_proof_for_the_CredentialTrigger_method."
    );
    _;
}

// The rest of the methods in this contract can use the modifier to
// prevent execution if the contract has not been properly
// activated.
// Other contracts can use the 'Show_Is_Active' method to build
// functionality based on the activation of this contract.

function SomeExampleMethod()
Do_not_run_if_not_activated
public
returns () {
    /* some code */
}

```

```

    return;
  }
}

```

Multiple disclosures Contract The following contract has two new features: First, it offers the `disclosure` modifier, which moves verification outside of the methods used. We use a pattern here, where the modified method take the exactly same inputs as the modifier. This is to make sure all arguments of the method will be bound to the disclosure. Unfortunately there is no way of enforcing this in Solidity.

Similar constructs to Solidity's 'modifier', such as decorators in Python, do allow more rigid control over the arguments used on methods that use ABCs.

The example method we use in this contract - the `This_method_uses_ABC` method, works based on the fact that the parameters from the function call are in scope for the decorator, so can be passed on. The writer of the contract still does need to manually hard code the required credentials and/or attributes. (`REQUIREMENTS` in the code below)

Second, we provide a function for the generation of a `nonce`. We apply the ideas presented in the previous section.

- The `nonce` therefore will contain the address of the smart contract (`address(this)`), and the particular method (`msg.sig`) that is being called.
- The verification time is bound to the epoch in which the caller. All accepting nodes will find the hash of the head of the blockchain (`blockhash(block.number-1)`).
- The hash of the parameters of the function call are also explicitly bound to the value of `Nonce`.
- Since it is impossible in Ethereum to call a contract without an account, the account of the caller (`msg.sender`) has also been included in the `nonce`.

Code

Listing 2: `nonces` in Solidity for multiple Disclosures

```

pragma solidity ^0.4.25;
pragma experimental ABIEncoderV2;

contract ExampleDisclosure {

    modifier disclosure (string[] parameters, uint256[] credentials,
        uint256 proof, bytes32 nonce){
        assert(true);
        assert(ConsumeCredential(parameters,proof,credentials));
        _;
    }

    function Nonce(bytes32 lasthash, address caller, address me, bytes8
        method, string[] parameters)
    public
    pure
    returns (bytes32) {

        bytes memory constructed_nonce = (abi.encode(
                                                    abi.encode(lasthash),
                                                    abi.encode(caller),
                                                    abi.encode(me),
                                                    abi.encode(method)

```

```

    ));

    return keccak256(constructed_nonce) ^ keccak256(abi.encode(
        parameters));
}

function ConsumeCredential (
    string[] parameters,
    uint256 proof,
    uint256[] credentials
)
public
view
returns (bool)
{
    bytes32 nonce = Nonce(blockhash(block.number-1), msg.sender
        , address(this), msg.sig, parameters);
    assert(VerifyProof(credentials, proof, nonce));
    return true;
}

function VerifyProof(uint256[] credentials, uint256 proof, bytes32
nonce)
public
pure
returns (bool) {
    return true;
}

uint256[] a;

function This_method_uses_ABC (string[] parameters, uint256
proof, bytes32 nonce)
public
disclosure(parameters, REQUIREMENTS , proof, nonce)
{
    // does what you actually want
    // only runs if the disclosure modifier completes
    // has parameters available as strings
    return ;
}
}
}

```

A user can find out what the last block number is by calling another contract. The rest of the parameters for the `nonce` generation are available upon the discovery of the smart contract. This allows us to generate the `nonce` on the client side, while the nodes in the network will also come up with the same value during the mining of the current block.

Listing 3: Solidity contract to find the most recent blockhash value

```

contract LastHash {
    function show()
    public
    view
    returns (bytes32) {
        return blockhash(block.number-1);
    }
}

```

}

3.3.7 Analysis

In the previous section we have seen two examples of how the protocol of ABC could be adapted for use with the Ethereum blockchain. We show that the ideas presented in the previous section are feasible in implementations.

In this section we address some of the particular problems that might arise when coding a similar smart contract for a different DLT. First we look at some practical differences that arise from the speed at which the global state is updated within different DLTs. Finally we address some practical concerns about the syntax to set attribute based credentials as access control for methods in smart contracts. The goals in DECODE specify that such smart rules should be usable by all users in the network.

Update Cycles The first issue relates to binding to ‘the current head of the blockchain’. All DLTs have a data structure that represents the ledger. This ledger is then distributed over the nodes as a shared ‘global state’.

The rate at which this global state is updated can vary a lot. Some DLTs update every several minutes (bitcoin [34]), others a new block every couple of seconds (Ethereum [58]), while some can process transactions so quickly the state-objects might update multiple times per second (Chainspace [53]). This rate also affects the time it takes to make sure that the transaction actually has been processed and recorded. (‘confirmation time’)

The DLTs mentioned above also differ in what triggers the update of the state. In most networks, like in Ethereum, the global state is updated continuous, and on a mostly regular time interval. The objects that represent state in Chainspace however, are only updated when they are consumed in a transaction.

These variations are important for the assessment whether our proposed method is usable for a specific smart contract application on a specific DLT. The effects will be in the user experience, but are also relevant for the security.

The faster the update cycle, the less opportunity an attacker has to benefit from replay or other manipulation of the call to the contract. So the longer the epoch of the update cycle is, the more one has to rely on the other properties, such as the binding of arguments or a strong notion of accounts, to make sure disclosure proofs are fresh within the epoch.

The faster the update cycle, the higher the likelihood will be that the call cannot be completed within that cycle. This in particular becomes an issue in the DECODE application, where a lot of the cryptographic operations are computed on embedded internet-of-things devices or smartphones, which may take multiple seconds or even minutes to prepare a single transaction. It is going to be a very inconvenient user experience if your calls to the DLT are rejected too often, or require more powerful and expensive hardware. In fact this affects the availability of your application on the DLT.

If the update of the global state happens on a regular basis, one could adapt the method to compute a range of nonces for the last n blocks, one of which must be used in the call. The value of n should remain as low as possible.

This problem might be more difficult to mitigate in the case where the local state of a contract is represented by an object in the ledger which is only ever updated when the related method is actually called. This has the advantage that slower devices may use more time to prepare the call to the contract. It does, however force all the users of the contract to compete in order to obtain the new state of the object and submit their call before some other user makes the state change.

This problem fortunately only arises when the contract gets popular. It then imposes an additional constraint on the writer of the contract. They must consider what the implications may be of limited availability. It can also negatively impact the fairness among the users in the network.

Composing Smart Contracts with ABCs The second consideration relates to the way writers of smart contracts can utilize the features of ABCs in their methods. This subsection presumes that the main functionality is access control of the execution. The implementation and syntax of this should not require the writer to write or comprehend any cryptographical code. This is because the goal in DECODE is to put users in control of their data, and as such ‘any’ user should be able to create such an access policy [18].

The writer of smart contracts must have some simple functionality available to add to a method that prevents execution without a valid disclosure. In our case this was a modifier in Solidity. While the modifier in our example successfully hides the `nonce`-creation and the other cryptographic functions, we still requires a lot of code to be specified and filled in by the writers of the smart contract. For example, the parameters of the call to the method still have to be passed into the modifier explicitly. It also remains easy to write functions that have logic and inputs that are not bound to the `nonce` and thus disclosure proof.

Our example code should therefore not be used as a template for the syntax and semantics of creating such an access control policy. It fundamentally is not simple enough.

A better example can be set using Python’s decorators:

Listing 4: Decorators in Python

```
import DLT # some DLT specific library to get state.
import ABC # ABC crypto functions.

def requires(req_attributes):
    def decorator(method):
        def inner(proofs, *_):

            # before method call:
            if not ABC.verify(req_attributes, proofs, _):
                raise Exception("Unable_to_verify_proof._Call_aborted.")
            else:
                plaintext_attributes = ABC.get_Attributes_from_proof(proofs,
                    DLT.state, req_attributes, _ )

            # method call:
            return method(plaintext_attributes, *_ )

            #after method call
            DLT.ABCcleanup(proofs)
        return inner

    return decorator

@requires({{issuer : 'brp', scheme : 'IRMA', type : 'age'}})
def age_check(attributes):
    if int(attributes.brp.age) > 18:
        return True
    return False
```

The `requires` decorator in this case, not only allows us to use credentials as an access policy, but also explicitly set controls on the arguments passed into the method. In this case the disclosed attributes, on successful verification, even become available in the (decorated) method as a dictionary of values.

When a function with this decorator is actually called, the method expects the disclosure proofs as arguments. (`inner(proofs, *_)`) The developer can now write the method with the only constraint that the dictionary of attributes is expected to be the first argument of the function.

Specifying the required credentials now also happens at a more readable and convenient place. The `requires` decorator only takes one argument; and that will be an access control policy of the required credentials (and/or attributes) to run the method. The @-notation of prepending a decorator before a method makes it fairly clear what is going on in the code. This makes for convenient human- and machine readability of these requirements from the smart contract source.

In a language that has purposefully been designed to support privacy friendly features in smart contracts, these safeguards can be implemented at an even lower level. In DECODE, Zenroom⁴ will likely contain this functionality and might use representations inspired by [22] for ease of use to programmers and users of the contracts alike.

3.3.8 Conclusion

In this subsection we have investigated the possibility of using Attribute Based Credential as a means of access control in the execution of Smart Contracts on Distributed Ledger Technology systems.

The main challenge arises from the inability to have a interactive protocol with the disclosing party, in order to guarantee the freshness of the disclosure proof. If these proofs are accepted without the freshness property, they can be replayed. Since the DLTs we consider are public systems by design, previous proofs will always be available to attackers. This would make ABCs unusable as a means to specify and access policy.

We propose three features present in most DLTs to be able to generate a `nonce` value that can be used as a proxy for the freshness.

- disclosure proofs are bound to the change of global state (or equivalent proxy),
- disclosure proofs are bound to a specific contract and a specific method,
- all arguments in the call to that method must also be bound to the disclosure proof, and
- if the privacy of users is not affected beyond the existing notion of (digital) identity in the DLT system, disclosure proofs should be bound to these identities.

We provide example code of implementing these requirements in a Solidity contract. This argues for the feasibility of actually using ABCs as an access control mechanism on (public) DLTs.

We argue that the practical differences between DLTs can still have a lot of effects on the security and user experience. In particular for availability of the service or application offered by the smart contract.

Furthermore, since this is in line with the DECODE ambition to allow all users to create these kind of smart rules, we argue that even our Solidity example is not simple enough. We take inspiration from Python decorators to suggest a way forward in allowing users a syntax to specify access policies. These policies should be easily readable for humans and machines.

⁴<https://zenroom.dyne.org/>

3.4 Privacy-preserving petition

We consider the scenario where several authorities managing the country C wish to issue some long-term credentials to its citizens to enable any third party to organize a privacy-preserving petition. All citizens of C are allowed to participate, but should remain anonymous and unlinkable across petitions. This application extends the work of Diaz *et al.* [19] which does not consider threshold issuance of credentials.

Our petition system is based on the library contract and a simple smart contract called “petition”. There are three types of parties: a set of signing authorities representing C , a petition initiator, and the citizens of C . The signing authorities create an instance of the smart contract as described in [53]. As shown in Figure 2, the citizen provides a *proof of identity* to the authorities (❶). The authorities check the citizen’s identity, and issue a blind and long-term signature on her private key k . This signature, which the citizen needs to obtain only once, acts as her long term *credential* to sign any petition (❷).

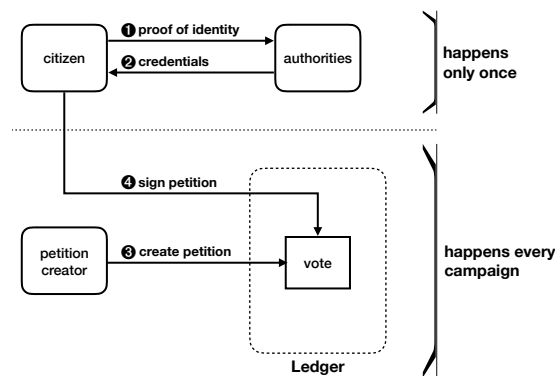


Figure 2: The petition application.

Any third party can *create a petition* by creating a new instance of the petition contract and become the “owner” of the petition. The petition instance specifies an identifier $g_s \in \mathbb{G}_1$ unique to the petition where its representation is unlinkable to the other points of the scheme⁵, as well as the verification key of the authorities issuing the credentials and any application specific parameters (e.g., the options and current votes) (❸). In order to *sign* a petition, the citizens compute a value $\zeta = g_s^k$. They then adapt the zero-knowledge proof of the algorithm of [53] to show that ζ is built from the same attribute k in the credential; the petition contract checks the proofs and the credentials, and checks that the signature is fresh by verifying that ζ is not part of a spent list. If all the checks pass, it adds the citizens’ signatures to a list of records and adds ζ to the spent list to prevent a citizen from signing the same petition multiple times (prevent double spending) (❹). Also, the zero-knowledge proof ensures that ζ has been built from a signed private key k ; this means that the users correctly executed the callback to prove that they are citizens of C .

Security consideration. Its blindness property prevents the authorities from learning the citizen’s secret key, and misusing it to sign petitions on behalf of the citizen. Another benefit is that it lets citizens sign petitions anonymously; citizens only have to go through the issuance phase once, and can then re-use credentials multiple times while staying anonymous and unlinkable across petitions. It allows for distributed credentials issuance, removing a central authority and preventing a single entity from creating arbitrary credentials to sign petitions multiple times.

⁵This identifier can be generated through a hash function $\mathbb{F}_p \rightarrow \mathbb{G}_1 : \tilde{H}(s) = g_s \mid s \in \mathbb{F}_p$.

4 Conclusion

In the first deliverable of the task; *initial Decentralized models for data and identity management: Blockchain and ABC* [33], we highlighted the relevant areas for research in the domains of attribute based cryptography, identity management and distributed ledgers

In this document we have followed up on the Coconut credential system [53], which was initially proposed in [33]. We provide an overview of the system and illustrate how it is to be implemented in DECODE.

We also address the challenges set out in the previous deliverable regarding the use of attribute based cryptography as access control.

We investigated Attribute Based Encryption as a means of access control of data in distributed systems like DECODE. Because there are, to the best of our knowledge, no schemes that satisfy all of the properties that we require for DECODE, and because the best alternative scheme, namely the [45] scheme, has its own disadvantages that makes it less than ideal for DECODE, we believe that the perfect scheme does not exist yet.

Attribute Based Credentials are shown be usable a means of access control on the execution of smart contracts. We investigate the requirements of making ABCs a general building block access control of execution in distributed ledger technology. This seems universally achievable for most DLT technologies, yet the large variety among these systems still requires caution and awareness.

Finally we have a method for privacy-preserving petitions in DECODE. The application we detail, builds upon the work of [19], and combines this with the Coconut credential scheme.

References

- [1] AGRAWAL, S., AND CHASE, M. A study of pair encodings: Predicate encryption in prime order groups. In *Theory of Cryptography Conference* (2016), Springer, pp. 259–288.
- [2] AGRAWAL, S., AND CHASE, M. Fame: Fast attribute-based message encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (2017), ACM, pp. 665–682.
- [3] AL-BASSAM, M., SONNINO, A., BANO, S., HRYCYSZYN, D., AND DANEZIS, G. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778* (2017).
- [4] ALPAR, G. *Attribute-based identity management: bridging the cryptographic design of ABCs with the real world*. Radboud University Nijmegen, 2015.
- [5] ATTRAPADUNG, N., HANAOKA, G., OGAWA, K., OHTAKE, G., WATANABE, H., AND YAMADA, S. Attribute-based encryption for range attributes. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 101, 9 (2018), 1440–1455.
- [6] ATTRAPADUNG, N., HERRANZ, J., LAGUILLAUMIE, F., LIBERT, B., PANAFIEU, E. D., AND RÀFOLS, C. Attribute-based encryption schemes with constant-size ciphertexts. *Theoretical Computer Science* 422 (2012), 15–38.
- [7] ATTRAPADUNG, N., AND IMAI, H. Attribute-based encryption supporting direct/indirect revocation modes. In *International Conference on Cryptography and Coding* (2009), Springer, pp. 278–300.
- [8] BETHENCOURT, J., SAHAI, A., AND WATERS, B. Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)* (2007), pp. 321–334.
- [9] BOLDYREVA, A., GOYAL, V., AND KUMAR, V. Identity-based encryption with efficient revocation. In *Proceedings of the 15th ACM conference on Computer and communications security* (2008), ACM, pp. 417–426.
- [10] BRANDS, S., AND PAQUIN, C. U-prove cryptographic specification v1. *Microsoft Corporation* (2010).
- [11] CAMENISCH, J., KRENN, S., LEHMANN, A., MIKKELSEN, G. L., NEVEN, G., AND PEDERSEN, M. Ø. Scientific comparison of abc protocols. *ABC4Trust* (2014).
- [12] CANETTI, R., HALEVI, S., AND KATZ, J. Chosen-ciphertext security from identity-based encryption. In *International Conference on the Theory and Applications of Cryptographic Techniques* (2004), Springer, pp. 207–222.
- [13] CHASE, M. Multi-authority attribute-based encryption. In *Theory of Cryptography Conference* (2007), Springer, pp. 515–534.
- [14] CHASE, M., AND CHOW, S. S. M. Improving privacy and security in multi-authority attribute-based encryption. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (2009), CCS '09, ACM, pp. 121–130.

- [15] CHEN, C., ZHANG, Z., AND FENG, D. Efficient ciphertext-policy attribute-based encryption with constant-size ciphertext and constant computation-cost. In *International Conference on Provable Security* (2011), Springer, pp. 84–101.
- [16] CHEUNG, L., AND NEWPORT, C. Provably secure ciphertext policy abe. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 456–465.
- [17] CHOW, S. S. M. A framework of multi-authority attribute-based encryption with outsourcing and revocation. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies* (2016), ACM, pp. 215–226.
- [18] DANEZIS, G., BANO, S., BASSAM, M. A., AND SONNINO, A. D1.4 de-code first version of decode architecture. <https://www.decodeproject.eu/publications/decode-architecture-first-version>, 2017.
- [19] DIAZ, C., KOSTA, E., DEKEYSER, H., KOHLWEISS, M., AND NIGUSSE, G. Privacy preserving electronic petitions. *Identity in the Information Society* 1, 1 (2008), 203–219.
- [20] EMURA, K., MIYAJI, A., NOMURA, A., OMOTE, K., AND SOSHI, M. A ciphertext-policy attribute-based encryption scheme with constant ciphertext length. In *International Conference on Information Security Practice and Experience* (2009), Springer, pp. 13–23.
- [21] ETHEREUM. Solidity — solidity 0.5.0 documentation. <https://solidity.readthedocs.io/en/v0.5.0/>, 2018.
- [22] FEDERICO BONELLI, T. v. D. Design & implementation interface for smart rules. <https://www.decodeproject.eu/publications/design-and-implementation-interface-smart-rules>, 2017.
- [23] GARMAN, C., GREEN, M., AND MIERS, I. Decentralized anonymous credentials. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014* (2014).
- [24] GOYAL, V., PANDEY, O., SAHAI, A., AND WATERS, B. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (2006), CCS '06, ACM, pp. 89–98.
- [25] HAN, J., SUSILO, W., MU, Y., ZHOU, J., AND AU, M. H. Improving privacy and security in decentralized ciphertext-policy attribute-based encryption. *IEEE Transactions on Information Forensics and Security* 10, 3 (2015), 665–678.
- [26] HOEPFMAN, J.-H., BANO, S., BASSI, E., CIURCINA, M., FREIRE, A., AND HAJIAN, S. D1.2 privacy design strategies for the decode architecture, 2017.
- [27] IBM RESEARCH. Identity mixer. https://www.zurich.ibm.com/identity_mixer/.
- [28] JAN CAMENISCH, A. L. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding* (2001), pp. 93–118.

- [29] JUNG, T., LI, X. Y., WAN, Z., AND WAN, M. Privacy preserving cloud data access with multi-authorities. In *INFOCOM, 2013 Proceedings IEEE* (2013), IEEE, pp. 2625–2633.
- [30] LEWKO, A., AND WATERS, B. Decentralizing attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2011), Springer, pp. 568–588.
- [31] LINDELL, Y. A simpler construction of cca2-secure public-key encryption under general assumptions. In *International Conference on the Theory and Applications of Cryptographic Techniques* (2003), Springer, pp. 241–254.
- [32] LUEKS, W., ALPÁR, G., HOEPMAN, J.-H., AND VULLERS, P. Fast revocation of attribute-based credentials for both users and verifiers. *Computers & Security* 67, Supplement C (2017), 308 – 323.
- [33] MEESSEN, P., AND SONNINO, A. Initial decentralised models for data and identity management: Blockchain and abc mvps. <https://www.decodeproject.eu/publications/initial-decentralised-models-data-and-identity-management-blockchain-and-abc-mvps/> 2017.
- [34] NAKAMOTO, S. Bitcoin: A peer-to-peer electronic cash system, 2008.
- [35] NAOR, M., AND YUNG, M. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing* (1990), ACM, pp. 427–437.
- [36] NISHIDE, T., YONEYAMA, K., AND OHTA, K. Attribute-based encryption with partially hidden encryptor-specified access structures. In *International Conference on Applied Cryptography and Network Security* (2008), Springer, pp. 111–129.
- [37] OSTROVSKY, R., SAHAI, A., AND WATERS, B. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 195–203.
- [38] PIRRETTI, M., TRAYNOR, P., MCDANIEL, P., AND WATERS, B. Secure attribute-based systems. *Journal of Computer Security* 18, 5 (2010), 799–837.
- [39] QIAN, H., LI, J., AND ZHANG, Y. Privacy-preserving decentralized ciphertext-policy attribute-based encryption with fully hidden access structure. In *International Conference on Information and Communications Security* (2013), Springer, pp. 363–372.
- [40] QIAN, H., LI, J., ZHANG, Y., AND HAN, J. Privacy-preserving personal health record using multi-authority attribute-based encryption with revocation. *International Journal of Information Security* 14, 6 (2015), 487–497.
- [41] RANNENBERG, K., CAMENISCH, J., AND SABOURI, A. Attribute-based credentials for trust. *Identity in the Information Society*, Springer (2015).
- [42] RAO, Y. S., AND DUTTA, R. Decentralized ciphertext-policy attribute-based encryption scheme with fast decryption. In *IFIP International Conference on Communications and Multimedia Security* (2013), Springer, pp. 66–81.
- [43] REUTOV, A. Predicting random numbers in ethereum smart contracts. <https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620/> 2017-01-31.

- [44] ROUSELAKIS, Y., AND WATERS, B. Practical constructions and new proof methods for large universe attribute-based encryption. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & communications Security* (2013), ACM, pp. 463–474.
- [45] ROUSELAKIS, Y., AND WATERS, B. Efficient statically-secure large-universe multi-authority attribute-based encryption. In *International Conference on Financial Cryptography and Data Security* (2015), Springer, pp. 315–332.
- [46] SAHAI, A. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Foundations of Computer Science, 1999. 40th Annual Symposium on* (1999), IEEE, pp. 543–553.
- [47] SAHAI, A., SEYALIOGLU, H., AND WATERS, B. Dynamic credentials and ciphertext delegation for attribute-based encryption. In *Advances in Cryptology—CRYPTO 2012*. Springer, 2012, pp. 199–217.
- [48] SAHAI, A., AND WATERS, B. Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2005), Springer, pp. 457–473.
- [49] SAMARATI, P., AND DE VIMERCATI, S. C. *Access Control: Policies, Models, and Mechanisms*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, pp. 137–196.
- [50] SANDHU, R. S. Lattice-based access control models. *Computer* 26, 11 (1993), 9–19.
- [51] SANDHU, R. S., COYNE, E. J., FEINSTEIN, H. L., AND YOUMAN, C. E. Role-based access control models. *Computer* 29, 2 (Feb. 1996), 38–47.
- [52] SHI, Y., ZHENG, Q., LIU, J., AND HAN, Z. Directly revocable key-policy attribute-based encryption with verifiable ciphertext delegation. *Information Sciences* 295 (2015), 221–231.
- [53] SONNINO, A., AL-BASSAM, M., BANO, S., AND DANEZIS, G. Coconut: Threshold issuance selective disclosure credentials with applications to distributed ledgers. *CoRR abs/1802.07344* (2018).
- [54] SZABO, N. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials* 6 (1997).
- [55] VENEMA, M. Decentralized attribute-based encryption for decode. Master's thesis, Radboud University Nijmegen, 2018.
- [56] WANG, X., ZHANG, J., SCHOOLER, E. M., AND ION, M. Performance evaluation of attribute-based encryption: Toward data privacy in the iot. In *2014 IEEE International Conference on Communications (ICC)* (2014), IEEE, pp. 725–730.
- [57] WATERS, B. Ciphertext-policy attribute-based encryption - an expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography* (2011), Springer, pp. 53–70.
- [58] WOOD, G. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151 (2014).
- [59] YAMADA, S., ATTRAPADUNG, N., HANAOKA, G., AND KUNIHIRO, N. Generic constructions for chosen-ciphertext secure attribute based encryption. In *International Workshop on Public Key Cryptography* (2011), Springer, pp. 71–89.

- [60] YU, S., WANG, C., REN, K., AND LOU, W. Attribute-based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (2010), ACM, pp. 261–270.