DEDACS: Decentralized and dynamic access control for smart contracts in a policy-based manner

Kristof Jannes
imec-DistriNet, KU Leuven
Leuven, Belgium
kristof.jannes@kuleuven.be

Vincent Reniers
imec-DistriNet, KU Leuven
Leuven, Belgium
vincent.reniers@kuleuven.be

Wouter Lenaerts
KU Leuven
Leuven, Belgium
wouterlenaerts99@hotmail.com

Bert Lagaise
imec-DistriNet, KU Leuven
Leuven, Belgium
bert.lagaisse@kuleuven.be

Wouter Joosen
imec-DistriNet, KU Leuven
Leuven, Belgium
wouter.joosen@kuleuven.be

ABSTRACT

Distributed Ledger Technology (DLTs) or blockchains have been steadily emerging and providing innovation in the past decade for several use cases, ranging from financial networks, to notarization, or trustworthy execution via smart contracts. DLTs are enticing due to their properties of decentralization, non-repudiation, and auditability (transparency). These properties are of high potential to access control systems that can be implemented on-chain, and are executed without infringement and full transparency.

While it remains uncertain which use cases will truly turn out to be viable, many use cases such as financial transactions can benefit from integrating certain restrictions via access control on the blockchain. In addition, smart contracts may in the future present security risks that are currently yet unknown. As a solution, access control policies can provide flexibility in the execution flow when adopted by smart contracts.

In this paper, we present our DEDACS architecture which provides decentralized and dynamic access control for smart contracts in a policy-based manner. Our access control is expressive as it features policies, and dynamic as the environment or users can be changed, or alternative policies can be assigned to smart contracts. DEDACS ensures that our access control preserves the desired properties of decentralization and transparency, while aiming to keep the costs involved as minimal as possible. We have evaluated DEDACS in the context of a Uniswap token-exchange platform, in which we evaluated the costs related to (i) the introduced overhead at deployment time and (ii) the operational overhead cost. DEDACS introduces a relative overhead of on average 52% at deployment time, and an operational overhead between 52% and 80% depending on the chosen policy and its complexity.

KEYWORDS

Smart contracts, access control, blockchain

ACM Reference Format:
Kristof Jannes, Vincent Reniers, Wouter Lenaerts, Bert Lagaise, and Wouter Joosen. 2023. DEDACS: Decentralized and dynamic access control for smart contracts in a policy-based manner. In *The 38th ACM/SIGAPP Symposium on Applied Computing (SAC '23), March 27 – March 31, 2023, Tallinn, Estonia.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3555776.3577676

1 INTRODUCTION

Distributed Ledger Technology (DLTs) or blockchains have been strongly emerging in the last decade, and proven as potentially groundbreaking alternatives towards replacing centralized trust mechanisms. Potential use cases range from simple financial transactions using Bitcoin [11] or Ethereum [19]. The latter also allows for applications that make use of the decentralized network as a reliable trustworthy execution unit with the use of smart contracts. In all DLTs, data and the history of transactions that have been executed onto the blockchain are appended and remain immutable, and therefore provide a source of non-reputable information. DLTs can omit the requirement for trust in a centralized party, as the trust can be placed in the decentralized network and its consensus on the current state of the data ledger. In addition, the functions of smart contracts can be executed and trust can be placed in its next state and outcome as it is determined via consensus by the network. Many diverse use cases have emerged that can benefit from these properties, such as asset notarization (e.g. using NFTs), voting, financial systems, identity management, supply chain systems, auditable data sharing, or governance [13, 14, 16].

Similarly for classical software, security is of a major concern to software on the blockchain [18]. Developers wish to prevent the misuse of their code, and the development of smart contracts may hide risks for future security exploits. Due to the immutable nature of the blockchain, smart contracts once vulnerable may yet still remain on the blockchain. Such aspects complicates matter and hinders practical adoption in certain domains. In this paper, we therefore focus on solving such dillemma's by advancing the research into access control for smart contracts. Such access control could control which smart contracts are employed, or which rules are enforced, and consequentially which logic is applied for which actions. In the future, if a certain risk emerges, the access control

SAC '23, March 27 – March 31, 2023, Tallinn, Estonia

ACM ISBN 978-1-4503-9517-5/23/03…$15.00
https://doi.org/10.1145/3555776.3577676

system on the blockchain could determine, after consensus, to execute newly deployed logic with the patched vulnerability. Access control on the blockchain is no easy feat, and if a central entity has sway over the access control it could in fact compromise on the inherent desirable properties of DLTs, namely decentralization and removing the need to place trust in a centralized entity. Consequentially only after consensus on the network may the execution flow by policies change to preserve the properties of decentralization, while still enabling dynamic access control which is desired for both reasons of security and development flexibility.

The current state of practice in research has recognized the potential of DLTs for access control [4, 5, 7–9], as it brings benefits in terms of decentralization, auditability, transparency but also in its capabilities of executing logic via smart contracts, or simply bookkeeping policies on-chain in a tamper-proof manner. Current solutions in the domain of access control on the blockchain are shortcoming in terms of decentralized management, or their dynamicity and configurability. This paper researches how dynamic access control can be applied and enforced in smart contracts on the blockchain, featuring decentralized management thereof.

This paper introduces a decentralized dynamic attribute-based access control for smart contracts (DEDACS) in a policy-based manner. DEDACS starts from a similar approach and architecture as the solution proposed in [2], and makes several additional contributions over the state of the art:

- DEDACS provides highly extensible policies for smart contracts. The policies feature the same range of functionality that is possible for the actions that they control, as these policies are also implemented as smart contracts. The policy can examine much more than simply the roles of users, and take into account environmental attributes, or attributes passed to the calling function, and apply appropriate actions.
- The resources that are protected by DEDACS (i.e. smart contracts) have to implement a specific interface. This interface allows for standardization and re-use of the same policies for a multitude of resources. This avoids the potential cost involved with re-deploying policy contracts.
- Our solution is also portable accross multiple blockchain technologies, as the access control architecture are implemented as smart contracts in Solidity.
- Finally, DEDACS introduces a Governance Manager implementation, which decouples the management of DEDACS and its associated policy execution flow. This Governance Manager in essence decentralizes the management, which is important to avoid re-centralization and to therefore maintain the desired properties of DLTs.

2 MOTIVATING USE CASES FOR ACCESS CONTROL ON THE BLOCKCHAIN

The use cases of DLTs or blockchain technology range from cryptocurrencies, decentralized finance (DeFi) applications, to asset notarization, or any sort of use case that can outsource logic to a smart contract and eliminate centralized trust such as attestation of operations. Most cryptocurrencies, also referred to as tokens, do not enforce any or only a limited form of access control. Most users can send or receive tokens as they please, as long as a certain limit on the number of tokens the sender owns is not exceeded. At times however it is desired that a certain form of access control is applied on a token. For example, a game application which features a game token. It is possible that certain users in a higher level can use the token in different manners than otherwise possible. The blockchain can also be used to organize the sale of tickets for events. In order to prevent tickets from being sold at excessively high prices the organizer could enforce access control on these ticket artifacts. Such access control can limit the ways in which a token can be used.

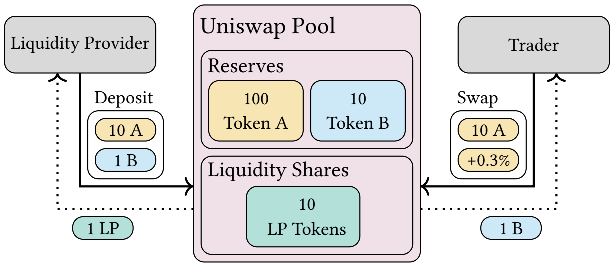

Figure 1: Overview of a Uniswap liquidity pool.

2.1 Primary use case: decentralized exchanges

An interesting use case of blockchain is that in the context of decentralized exchanges, which allow users to trade cryptocurrencies and other ERC-20 tokens. These platforms are of key interest to add additional access control on, as to control how and when users can perform certain actions (e.g. swap tokens), and we aim to investigate the techniques to do so while not compromising on the decentralized nature of these platforms. Decentralized exchanges, such as Uniswap [1], Curve, Balancer [10] are typically comprised of different sets of liquidity pools. Each liquidity pool contains an equal value of two different tokens, and allows users to exchange one kind of token to the other token in the pool. An exception to this is Bancor [6], which uses a system in which tokens are linked to each other in a network.

In general, decentralized exchanges use liquidity pools, which can be used by users in two manners: (i) either to trade one token for another, or (ii) to provide liquidity. This process is shown in Figure 1. In the first scenario, they pay a certain fee to trade one token (e.g. ETH) for another (e.g. USDC) against a certain market value which the decentralized exchange dynamically determines. In the second manner, the user provides his/her tokens as liquidity to a specific pool for a certain amount of time. This liquidity can then be used by other people to exchange tokens. In return for providing liquidity the user will receive a small percentage of each exchange which occurs via the pool for which they provide liquidity.

Governance. Most decentralized exchanges provide a system for governance, which is typically co-granted via a special governance token. The owners which posses such governance tokens can submit and vote for proposals for changes to the decentralized exchanges, such as regarding system parameters. The manner in which the tokens are distributed will determine the degree of decentralization of such systems. Uniswap [1] is the primary use case for our access control technique discussed in this paper. It has its own governance token UNI, and is comprised of a factory controlled by UNI token owners, where new liquidity pools can be created.

The case of decentralized exchanges present many interesting perspectives on how to integrate access control on the actions of users of (i) when being able to trade tokens, or (ii) capable of providing liquidity and the potential rewards. In the next subsection we further discuss the particularities of access control on the blockchain, and the importance of preserving properties such as decentralization.

2.2 Use case and access control requirements

*Dynamic access control (**R1**).* A first requirement given our use case is that of dynamic access control. At a certain point in time, for example, when certain security concerns come to light of existing smart contracts, it should be possible to deny access via the policies, or new policies. As a more practical example, it should also be possible to add more rules or new users to the policy-based system in the future as the development of the application progresses.

*Decentralized governance (**R2**).* The dynamic character of such an access control system also introduces a potential problem, namely the management of the access control. It is not desireable that rules or user rights can instantly change, or if users can assign additional rights for themselves. This brings out the requirement of decentralized governance. In a classical approach, such centralized management of the access rules would impede on the desired decentralized property of the blockchain. Therefore, the governance of adding and changing environment parameters, or user attributes and rules should be done via a consensus-based approach. This can be implemented for example via a collective of trusted developers, or via a community-based voting system, which decide to approve or reject proposals for change.

*Policies for expressiveness of access control (**R3**).* The access control rules should be expressive enough, and function on as much data as possible. This data can range from the current block, the enacting user, the attributes or roles, and or parameters passed by the calling function of the originating smart contract.

*Non-functional requirements (auditability, cost) (**R4**).* The major advantage of implementing an access control system on the blockchain is that it provides potential for auditability of on-going policy enforcement decisions. Any action taken and executed on-chain is logged in a non-reputable fashion, and can be audited by any public party that has access to the distributed data ledger. Another requirement is that related to cost, with the benefits of DLTs in decentralization and transparency, also comes a potential downside in financial or performance associated costs. Given the use case and intended solution, these parameters should be within bounds compared to a static access control setup, in which dynamic change is not possible. In order to keep these costs to a minimum, policies should be re-useable across multiple smart contracts. Finally, ideally the solution is re-useable or its architecture portable across different DLT technologies.

3 RELATED WORK

The properties of auditability and transparency have yielded much research in the area of access control on distributed data ledgers. We summarize key publications in this area, and we contrast the state-of-the-art to our solution DEDACS in terms of trade-offs or architectural differences.

Maesa et al. [4] present a blockchain-based access control system, with the policy-enforcement-point still off-chain. The policy administration point merely consults data stored on the blockchain to guide the Policy Enforcement Point (PEP). In a follow-up paper in 2018, the same authors moved the logic into smart contracts, referred to as a "smart policy", and involves attribute managers and attribute provider contracts [9]. The policies originate from parsing XACML into a smart contract, and are attributed-based (ABAC). According to our requirements, the expressiveness is met, but such policies may be potentially costly, and do not take into account the dynamicity of policies and governance. TRABAC [7] is an access control framework which uses NFTs to represent attributes, therefore it can provide fine-grained access control by limiting access to accounts that own a certain NFT attribute. However, this framework lacks dynamicity and policies are less expressive than our target. In 2019, Rouhani et al. present a publication on the state-of-the-art on blockchain-based access control systems [15]. All these previously discussed access control systems are systems (partially) implemented on the blockchain, to control access to a service outside the blockchain. DEDACS provides a solution to enforce access control on smart contracts, and this is an end-to-end process which takes place completely on-chain.

In 2019, Maesa et al. [5] present an on-chain attribute-based access control framework for protecting smart contracts. This is an extention of [9], an access control framework to protect other resources outside the blockchain. It inherits the same problems, namely that a central resource owner is used as Policy Administration Point (PAP), and that the resource must directly call the policy contract making the framework much less dynamic.

SMACS [8] is an access control system for smart contracts. However, the policies are not evaluated on-chain, but instead an external Token Service will evaluate a policy and returns a token to the client. The client can then use this token to call smart contracts on-chain. While this has benefits in terms of costs, as very little work has to be done on-chain, the trust moves to an off-chain entity (the Token Service), which defies the decentralization that a blockchain offers.

In [3], the authors propose a dynamic RBAC framework. They introduce a PermissionManager to decouple the policies from the business logic. We will adapt this notion of a PermissionManager in our solution. But we extend this with more expressive access control using ABAC and decentralized governance.

In [17], the authors present a formal model for RBAC and a code generator to generate Solidity smart contracts from the access control policies in the formal model. In this manner, developers can use formal verification tools to ensure that the smart contract correctly implements the envisioned policies. It tightly couples the policy with the smart contract that is to be protected by it. This makes the framework less dynamic, as it is impossible to change the policy at a later date.

OpenZeppelin [12] is a library for secure smart contract development for Solidity. It provides a contract for simple Role-Based Access Control. It lacks external and dynamic policies, decentralized governance and more extensive access control models such as ABAC.

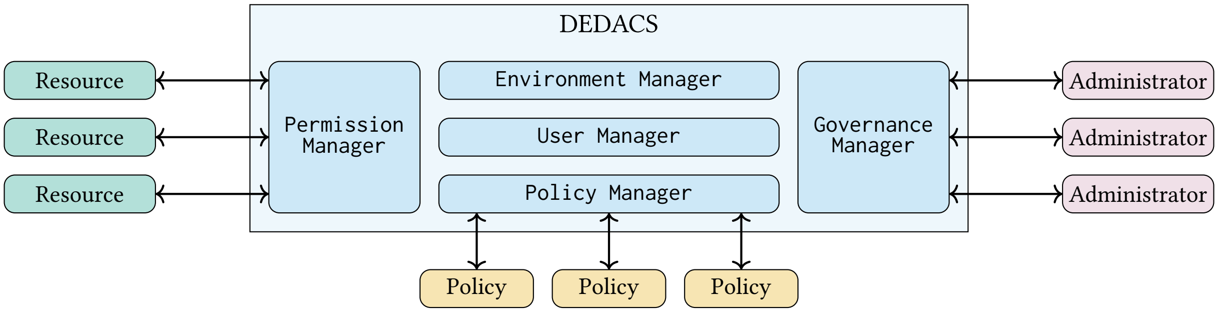

Figure 2: DEDACS: Architectural component overview.

4 ARCHITECTURE

Section 4.1 will first describe the architecture of DEDACS and an overview of all its components. Subsequently, Section 4.2 shows the workflow to enforce access control rules, and finally in Section 4.3 we detail the access control policies and their expressiveness.

4.1 Architectural overview and components

DEDACS can enforce attribute based access control on smart contracts, and is itself implemented via smart contracts using the Solidity programming language. In this manner DEDACS runs fully on-chain, which provides benefits in terms of (i) not requiring trust in a centralized or external entity, and (ii) provides transparency and auditability of access control operations. An overview of the solution architecture of DEDACS is presented in Figure 2. Our implementation is publicly available online[1]. We introduce five smart contracts that store relevant information, or play key roles in the access control decision process, namely: `User Manager`, `Environment Manager`, `Policy Manager`, `Permission Manager` and a `Governance Manager`.

The `User Manager` and the `Environment Manager` are the Policy Information Point (PIP), and provide essential information that can be used by policies to make decisions. The `Policy Manager` is the Policy Administration Point (PAP), and enables administrators to register which policies are active for which resources. The Policy Enforcement Point (PEP) should be implemented in the resource, which itself is also a smart contract. In essence, DEDACS provides policies and access control that can be implemented into smart contracts. The PEP then forwards its calls to the Policy Decision Point (PDP), which are the `Policy` smart contracts that are assigned to the resource, as configured in the `Policy Manager`. We explain each managing smart contract in more detail.

User Manager. The `User Manager` is a smart contract that stores users and their attributes. Each user is uniquely identified by their public address on the blockchain. This user is not necessarily a person, but can also be a smart contract.

Environment Manager. The `Environment Manager` manages attributes such as time. Furthermore, it allows administrators to define custom attributes for their application.

Permission Manager. The `Permission Manager` forms the bridge between the applications using DEDACS and the DEDACS system itself. An operation of a resource that must be protected, should first call the `Permision Manager` to evaluate the policies, and only after a succesful evaluation, the operation itself should be executed. The `Permission Manager` itself will use the `Policy Manager` to direct the request to the correct policy contract.

Resources. Resources are the smart contracts that should be protected by DEDACS. They implement the business logic of the application. Every method that should be protected by DEDACS, should first call the `Permission Manager` to evaluate the policies. For this reason, they have to know the address of the `Permission Manager`. Note that they do not need the location of the other contracts of DEDACS or the policies itself. The `Permission Manager` will use the `Policy Manager` to evaluate the correct policy, and this policy can be changed dynamically without modifying the resource.

Policy Manager. The `Policy Manager` is responsible to map resources and operations to the correct policy contract. Administrators can register new policies based on the resource address and method name. The PEP can use the `Policy Manager` to verify whether an operation is allowed. It calls the `Policy Manager` with the resource address, the method name, the user address and the parameters of the operation. The `Policy Manager` then looks up the correct policy contract and calls it with the resource, the user address and the parameters. If no policy is registered for the given resource and operation, the `Policy Manager` will block the access.

Governance Manager. The `Governance Manager` is responsible for the governance of the system, and it controls all the other smart contracts contracts (i.e. managers) of DEDACS that feature states. It can change attributes of the user or the environment, and it can also change the policies. This contract is implemented as a multi-signature contract, which means that actions of the contract can only be executed after a majority has signed, and thus agreed, for a change. Such a multi-signature contract receives a list of administrators at creation time. The administrators can then all propose changes to the contract, and can vote on proposals from other administrators. However, the proposal will only be executed after a preconfigured majority has voted. The benefit of using such a multi-signature contract, is that it allows for a decentralized governance of the system, and thereby preserving the desired properties of DLTs in terms of decentralization. The `Governance Manager` is

[1] https://github.com/wouterlenaerts/DEDACS

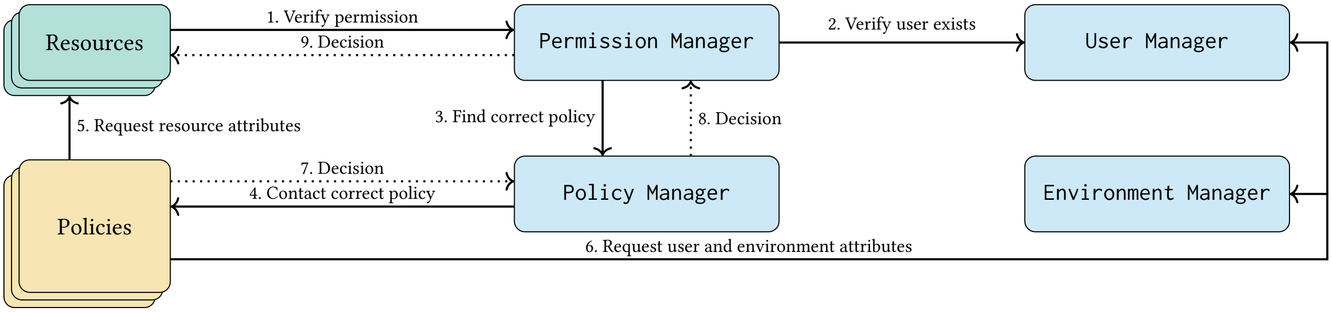

Figure 3: Overview of a workflow with three resources and two policies.

the owner of the `User Manager`, the `Environment Manager` and the `Policy Manager`. All methods that change the status of the access control system in these contracts can only be called by the owner. The `Permission Manager` does not have an owner, as it contains no state that has to be changed.

4.2 Access control and policy workflow

This section will describe the steps DEDACS takes to enforce access control using the components we previously discussed. An example workflow is shown in Figure 3. The resources are protected by policies. One policy can be used by many resources. When a user calls a method on a resource, that resource will first call the `Permission Manager` with the following arguments: the user its address, the method name, and the list of arguments passed by that method. The `Permission Manager` will verify if the user exists by contacting the `User Manager`. If the user does not exist, the `Permission Manager` will block the access. When the user exists, the `Permission Manager` will call the `Policy Manager` with the following arguments: the resource its address, the method name, the user its address, and the list of arguments.

The `Policy Manager` will then look up the correct policy contract based on the address of the resource and the name of the method. If no policy is registered, the `Policy Manager` will block the access. If a policy exists, the `Policy Manager` will call the policy contract with the following arguments: the address of the resource, the address of the user and the list of arguments. The policy can request the required attributes from the `User Manager`, the `Environment Manager` and the resource itself. Subsequently the policy makes a decision based on this extensive set of information whether to allow or block the operation in the originating smart contract (i.e. the resource which has to be protected).

The benefits of this access control occurring on the blockchain are that it is logged in a non-reputable fashion, and can be observed transparently. In addition, the access control is executed in a decentralized fashion without the possibility of a centralized entity interfering in this process.

4.3 Policies

Policies are smart contracts that have to implement the following interface, as shown in Listing 1. This method takes as input the address of the user, the address of the resource and the list of arguments. It outputs a boolean indicating wether the operaton is allowed or not. If the policy requires external attributes from the `User Manager` or the `Environment Manager`, it should receive the contract address of the `Governance Manager` as a constructor argument. The policy can request the location of the `User Manager` or the `Environment Manager` from the `Governance Manager`.

```
interface Policy {
        function isAllowed(
                address user,
                address resource,
                bytes [] memory arguments
        ) view external returns (bool);
}
```

Listing 1: DEDACS Policy interface

Policy size and composition. Since policies are smart contracts themselves, they can express any kind of access control policy. They are also not limited in size, or as long as a single smart contract is smaller than the block size. In theory, a policy can also be split into multiple smart contracts, of which the individual components could be re-used across other policies to save deployment costs. The `Policy Manager` will then call the first contract, which will call the second contract, which will call the third contract, etc.

Stateful policies. Smart contracts can store state internally, which makes it possible to implement stateful policies. For example, a policy can keep track of the number of operations that a user has performed. This policy can use this counter to block the user after a certain value.

Access delegation. Another example of a stateful policy is a policy that allows to delegate access to other users. If a user wants to perform an operation, the policy will verify whether the user has the required rights. If not, the policy can check if another user delegated his access to this user. If such a user exists, the policy can be re-evaluated with that second user. In such a case, and when the user does indeed have the required rights, the operation can be performed by the first user. This simplifies aspects in a certain regard as the `Policy Manager` does have to be updated via consensus, but rather individual policy contracts can introduce more flexible access delegation, which does however compromise via re-centralization of responsibilities when desired or allowed.

External information. Policies can call any other smart contract at the blockchain. This makes it possible to implement policies that use external data, for example by making calls to an oracle contract.

5 EVALUATION

This section will describe the evaluation of DEDACS. Section 5.1 starts with the research questions and objectives of the evaluation, next Section 5.2 details our experiment setup. Finally, Section 5.3 depicts the results involved with the costs when deploying DEDACS and different types of associated policies in terms of complexity.

5.1 Research questions and objectives

Deploying and executing smart contracts on the blockchain involves a transaction cost. The main goal of this evaluation is to determine if the associated cost makes DEDACS usable in practice or not. More specifically, our evaluation methodology and setup are designed to answer the following research questions:

RQ1 What is the *overhead cost of DEDACS at deployment time*, and how does it compare to an approach with hard-coded access control? More specifically, what is the deployment cost overhead of respectively the DEDACS architecture (**RQ1.A**) and different policies of varying complexity (**RQ1.B**)?

RQ2 What is *operational overhead cost of DEDACS* versus an approach with hard-coded access control, specifically the operational cost involved with DEDACS (**RQ2.A**) or during policy evaluation with policies of varying complexity (**RQ2.B**)?

5.2 Methodology

One of the more compelling use cases in distributed data ledger technologies are currently financial transactions and decentralized finance applications (DeFi). More specifically, we will in the context of our evaluation methodology use a decentralized token exchange platform, namely the previously discussed Uniswap [1] from Section 2.1. This use case is an ideal fit to implement access control on game tokens, and to therefore limit certain operations on a decentralized token exchange.

Experiment setup. In our Uniswap use case, one example policy rule is that only gamers who reached a certain level can swap an in-game token. To answer **RQ1**, and measure the overhead cost of our architecture at deployment time, we implemented two versions of this use case. The first setup for **RQ1.A** measures the deployment costs of a setup in which DEDACS manages the access control policy, and another setup in which the policies are hard-coded in the token contract itself. In addition, for **RQ1.B** we also measure the deployment cost of policies of varying complexity. We will use *hardhat-deploy* to deploy each combination locally on the EVM, and measure the gas cost for the deployment of all required smart contracts, as well as the average cost of a swap operation on Uniswap.

Policies. In order to answer the operational costs **RQ2**, we measure the operational cost of the DEDACS architecture **(RQ2.A)**, but also the operational impact of a policy's complexity on the involved cost **(RQ2.B)**. Both scenarios will be evaluated versus the hard-coded example without dynamic access control. To determine the cost of policies of varying complexities, both at deployment time **(RQ1.B)** and during operation **(RQ2.B)**, we performed the evaluation with 8 different policies:

P1 **Empty policy**: this policy has no constraints and will simply return true. In the hard-coded version, this is equal to a regular ERC-20 token without access control.

P2 **User policy**: this policy will only allow access if the user has a specific attribute.

P3 **Resource policy**: this policy will use an attribute from the resource to determine access, for example the creation date of the smart contract.

P4 **Environment policy**: this policy will use an attribute from the environment, for example the current time.

P5 **Attribute policy**: this policy will combine the different kind of attributes from P2, P3 and P4.

P6 **Argument policy**: this policy will use the arguments provided to the method call for which the policy needs to determine access.

P7 **Attribute and argument policy**: this policy combines P5 and P6.

P8 **Complex policy**: this policy will try to combine as much information as possible, it uses the arguments of the method, the role of the user, the security clearance of the user, the name of the user, the time, the address of the miner of the block, the security level of the environment, the security level of the resource, the owner of the resource and the date of creation of the resource.

Deployment with DEDACS. The full deployment and setup of the use case with DEDACS consists of 43 transactions on the blockchain. This includes the deployment of the required contracts of DEDACS (3 transactions), the deployment of the two game tokens (2 transactions), the deployment and provisioning of a Uniswap pool to swap these tokens (4 transactions), the creation of two users (3 transactions) and the configuration of the system (31 transactions). These steps are done eight times for the different policies.

Deployment with hard-coded approach. The deployment and setup of the hard-coded approach is much simpler, it consists only of 10 transactions. This includes the deployment of the two game-tokens (2 transactions), the deployment and provisioning of a Uniswap pool to swap these tokens (4 transactions), the creation of two users (3 transactions) and the configuration of the system (1 transaction). This deployment is also done eight times for the different policies. Instead of separate policies, the access control rules will be included hard-coded in the game tokens. The scenario with policy P1 is the baseline, with no access control present.

5.3 Evaluation results

We divide this section into respectively costs related at (i) deployment time of individually DEDACS, and the varying policies, and (ii) operational run-time cost of executing DEDACS, and individually the policies.

5.3.1 *Deployment cost (**RQ1**).*

The resulting gas costs of all scenarios are shown in Table 1. The first two rows show the cost for deployment (**RQ1**). To answer **RQ1.A**, we compare this deployment with the cost to deploy a similar system without DEDACS and its dynamicity. The cost is on average 52% higher when using

	Empty Policy **(P1)**	User Policy **(P2)**	Resource Policy **(P3)**	Environment Policy **(P4)**	Attribute Policy **(P5)**	Argument Policy **(P6)**	Attr. and arg. policy **(P7)**	Complex Policy **(P8)**
Deployment costs (RQ1)								
DEDACS	22,133,238	22,240,146	22,242,873	22,175,680	22,354,973	22,293,299	22,516,725	22,820,592
Hard-coded	14,640,498	14,695,566	14,690,355	14,745,642	14,766,189	14,727,158	14,815,792	14,897,580
Operational costs (RQ2)								
DEDACS	180,964	189,502	186,742	186,212	200,992	186,958	206,978	237,050
Hard-coded	118,074	118,202	118,038	122,398	122,366	118,406	122,794	131,466
Administration (governance)								
DEDACS	354,549	354,549	354,549	354,537	354,537	354,549	354,537	354,549
Hard-coded	-	-	-	-	-	-	-	-

Table 1: Transactional cost of using DEDACS for Uniswap expressed in units of gas.

DEDACS. The deployment cost of the use case with DEDACS is similar for all different policies (**RQ1.B**). The minor increase in cost for more complex policies is only due to the increase in size of the policy contract. The cost to deploy the DEDACS contracts stays the same. Since the deployment cost of DEDACS and the Uniswap pool is quite large, the effect of the policies is small (**RQ1.B**).

*5.3.2 Operational cost (**RQ2**).* Once the system is deployed, users can swap tokens using the Uniswap pool, and we can measure the operational cost involved with DEDACS (**RQ2.A**), and the respective policies and their complexity (**RQ2.B**). However, whether such a swap is allowed for a user depends on the different policies. The third and fourth rows of Table 1 show the cost for such a swap operation, including evaluating the policy. With this experiment, we are able to answer **RQ2.A** and **RQ2.B**. These transactions are much cheaper than the deployment. The overhead of using DEDACS is even greater, this ranges from 52% to 80% depending on the policy. The complexity of the policy also greatly influences the total cost, with P8 begin 31% more expensive then the empty policy P1.

A third kind of operation is changing the access control policy. In DEDACS, this is possible via the `Governance Manager`. Administrators can propose changes, vote for them, and confirm them if enough administrators agree. The cost to change one user attribute is shown in the fifth row of Table 1. This is the sum of the three transactions to submit, confirm and execute a change if their is only one administrator. If there are multiple administrators, this cost will increase with 74,652 gas per extra administrator that confirms a proposal. There is no equivalent evaluation for the hard-coded case, as the access control policy is hard-coded in this case, and no chages can be done. Only alternative in this case would be to redeploy all contracts again.

6 DISCUSSION

Section 6.1 discusses the architecture of DEDACS and possible design considerations in terms of: decentralized governance, or tailoring it to other applications (e.g. which do not require users, or alternative attributes). Next, Section 6.2 discusses an analysis of the realistic cost when deploying DEDACS on a variety of blockchain platforms.

6.1 Architectural design decisions

This section discusses some properties and design decisions of DEDACS, with its advantages and disadvantages.

6.1.1 Decentralized governance mechanisms. The `Governance Manager` decouples the administration of DEDACS from the rest of the system. If a developers wants a different form of governance, he can adapt the `Governance Manager` to his needs, without changing the rest of the system. It is for example possible to adapt a centralized governance model, where a single administrator can change the policies. Or it can be adapted to a decentralized governance model used in so-called Decentralized Autonomous Organizations (DAOs), where tokens holders can vote on proposals.

6.1.2 Extensions to system attributes. The `User Manager` and the `Environment Manager` can be extended to keep track of extra attributes. However, attributes have to be defined before the deployment. Once deployed, no new attributes can be added anymore since the contract is immutable. If new attributes are required, one has to redeploy a new `User Manager` that contains these attributes and update the `Governance Manager` to point to this new contract. Future policy calls will then use the new `User Manager` and can use the added attributes.

6.1.3 Non-user application. The current prototype assumes that all users are known to the system in the `User Manager`. However, not all applications require that a user is registered upfront. DEDACS can be easily adapted to facilitate this by skipping the check in the `Permission Manager` whether the user exists.

6.1.4 Solidity programming language drawbacks. Since policies are smart contracts implemented in Solidity on the Ethereum Virtual Machine (EVM), they inherit all drawbacks. One example is that

	Deployment **(RQ1)**		Operation **(RQ2)**	
	DEDACS	Baseline	DEDACS	Baseline
Ethereum	$ 2770.3	$ 1814.9	$ 24.4	$ 14.6
BSC	$ 50.9	$ 33.3	$ 0.45	$ 0.27
Avalanche	$ 54.7	$ 35.9	$ 0.48	$ 0.29
Polygon	$ 3.54	$ 2.32	$ 0.03	$ 0.02

Table 2: Transaction cost in dollar for different EVM compatible blockchains. Gas price and token price are the averages for the first three quarters of 2022.

Solidy will only give back the hashes of the most recent blocks. Requesting attributes from older blocks is therefore not possible.

6.2 Realistic cost of DEDACS and policies across blockchain technologies.

The previous section showed the results of the evaluation in gas. However, gas is an abstract metric and not directly important for the end-user. Table 2 shows the true cost in dollars when executing these operations on the most popular EVM compatible blockchains. These dollar prices are derived from the gas cost in Table 1, multiplied by the gas price and the token price. Since prices of blockchain tokens are quite volatile, we used the average price in the first three quarters of 2022. For the gas price we used also the average in that same period. The results for DEDACS use the average cost of all policies (P1 - P8). The baseline are the costs for the hard-coded approach with policy P1, this is the empty policy, meaning that this presents the case where there is no access control logic present. This is equal to the cost users pay today when using Uniswap with a regular ERC-20 token.

DEDACS makes deployment much more expensive in absolute terms, and relatively involves an increase of 52.6%. In absolute terms this may vary quite significantly depending on the chosen blockchain technology. For example, in the case of Polygon this increases the cost from $2.32 to $3.54. On Ethereum for example, it adds an extra $950 to the transaction cost over the already quite significant baseline cost of $1814. However, in return, applications get a dynamic access control system, where policies and attributes can be changed afterwards. This dynamicity can even save costs, as updating the system is much cheaper. For the hard-coded approach, making such an update would require to deploy the contracts again.

The operational costs of DEDACS is also more expensive. For example, in the case of Avalanche the operational cost increases from $0.29 to $0.48. When the Ethereum blockchain is used, this operational overhead adds $10 to each transaction. The chosen blockchain technology and resulting cost will determine whether DEDACS can bring additional value or not. Depending on the application, and how many transactions users do on average, this overhead will be significant or not compared to the overall value the user puts into the system. Layer-2 solutions on top of Ethereum, which reduce the transaction costs, will make DEDACS more feasible. On other chains, DEDACS only adds a few cents, making it usable for many applications.

7 CONCLUSIONS

Distributed Ledger Technologies (DLTs) have been steadily emerging in the recent decade, and present beneficial properties in terms of decentralization (i.e. removing the need for trust in centralized entities), as well as transparency and auditable as everything is shared and determined by consensus on the network. These features and benefits have led to a wide number of use cases, such as financial transactions (e.g. cryptocurrencies), asset notarization (e.g. NFTs), supply chain tracking, and decentralized finance (DeFI) applications to name a few. While is not yet sure which use cases will remain viable in the future, such systems can highly benefit of access control on their operations, as future security vulnerabilities may arise it can be convenient to provide a form of dynamic access control to restrict or alter operations. Such access control mechanisms should be implemented on-chain, and their governance should also be decentralized to preserve the desired properties of DLTs.

In this paper we present our DEDACS architecture for decentralized and dynamic access control for smart contracts in a policy-based manner. The policies are highly expressive as these are also implemented as smart contracts, and function on attributes, user roles, and method parameters passed by the protected smart contract (i.e. the resource). The dynamicity of the system comes from the potential to control access based on such parameters and information, but also the possibility to govern DEDACS and change environment attributes, user roles, or assign new policies to new or existing smart contracts. The DEDACS architecture is importantly governed not via a centralized entity, as this would comprise the beneficial properties of DLTs in terms of decentralization. Instead, a multi-signature contract ensures that a certain number of administrators have to reach consensus, which means in practice a majority vote on proposals for change. We have proposed an end-to-end on-chain solution to enable access control for smart contracts in a dynamic manner. DEDACS is implemented in Solidity, and therefore portable across various blockchain technologies that are compatible with the Ethereum VM (EVM).

Our solution has been evaluated in the compelling use case of decentralized token exchanges, more specifically we integrated DEDACS into the Uniswap token exchange case. Such decentralized finance (DeFi) applications may benefit greatly from implementing a degree of restrictions or access control on transactions. Our evaluation investigated both the overhead in terms of (i) cost at deployment time, as well as (ii) operational costs. The overhead of at deployment is on average 52% higher when using DEDACS. In terms of operational costs, the overhead in transactions ranges from 52% to 80% depending on the policy its complexity. Eventually, the end-user will have to decide whether the downsides in increased cost is acceptable at the benefits of dynamic and expressive policy-based access control. It is also possible to re-use certain policies for multiple smart contracts. In future work, it would be interesting to analyze how this cost can be reduced further, and which alternative systems are possible for the decentralized governance of the DEDACS architecture.

REFERENCES

[1] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. Uniswap v3 Core. https://uniswap.org/whitepaper-v3.pdf.

[2] Arnab Chatterjee, Yash Pitroda, and Manojkumar Parmar. 2020. Dynamic Role-Based Access Control for Decentralized Applications. In *Blockchain – ICBC 2020*. Lecture Notes in Computer Science, Vol. 12404. Springer International Publishing, Cham, 185–197. https://doi.org/10.1007/978-3-030-59638-5_13

[3] Arnab Chatterjee, Yash Pitroda, and Manojkumar Parmar. 2020. Dynamic Role-Based Access Control for Decentralized Applications. In *Blockchain – ICBC 2020*. Springer International Publishing, Cham, 185–197. https://doi.org/10.1007/978-3-030-59638-5_13

[4] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. 2017. Blockchain Based Access Control. In *Distributed Applications and Interoperable Systems*. Springer International Publishing, Cham, 206–220. https://doi.org/10.1007/978-3-319-59665-5_15

[5] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. 2019. A blockchain based approach for the definition of auditable Access Control systems. *Computers & Security* 84 (2019), 93–119. https://doi.org/10.1016/j.cose.2019.03.016

[6] Eyal Hertzog, Guy Benartzi, Galia Benartzi, and Omri Ross. 2018. Bancor Protocol Continuous Liquidity for Cryptographic Tokens through their Smart Contracts. https://storage.googleapis.com/website-bancor/2018/04/01ba8253-bancor_protocol_whitepaper_en.pdf.

[7] Aisyah Ismail, Qian Wu, Mark Toohey, Young Choon Lee, Zhongli Dong, and Albert Y. Zomaya. 2021. TRABAC: A Tokenized Role-Attribute Based Access Control using Smart Contract for Supply Chain Applications. In *2021 IEEE International Conference on Blockchain (Blockchain)*. 584–589. https://doi.org/10.1109/Blockchain53845.2021.00088

[8] Bowen Liu, Siwei Sun, and Pawel Szalachowski. 2020. Smacs: smart contract access control service. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 221–232. https://doi.org/10.1109/DSN48063.2020.00039

[9] Damiano Di Francesco Maesa, Paolo Mori, and Laura Ricci. 2018. Blockchain based access control services. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE, 1379–1386. https://doi.org/10.1109/Cybermatics_2018.2018.00237

[10] Fernando Martinelli and Nikolai Mushegian. 2019. A non-custodial portfolio manager, liquidity provider, and price sensor. https://balancer.fi/whitepaper.pdf.

[11] Satoshi Nakamoto. 2009. Bitcoin: A peer-to-peer electronic cash system. http://www.bitcoin.org/bitcoin.pdf

[12] OpenZeppelin. 2022. OpenZeppelin Access Control. https://docs.openzeppelin.com/contracts/4.x/access-control

[13] Danda B Rawat, Vijay Chaudhary, and Ronald Doku. 2019. Blockchain: Emerging applications and use cases. *arXiv preprint arXiv:1904.12247* (2019).

[14] Vincent Reniers, Yuan Gao, Ren Zhang, Paolo Viviani, Akash Madhusudan, Bert Lagaisse, Svetla Nikova, Dimitri Van Landuyt, Riccardo Lombardi, Bart Preneel, and Wouter Joosen. 2020. Authenticated and Auditable Data Sharing via Smart Contract. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing (SAC ’20)*. Association for Computing Machinery, New York, NY, USA, 324–331. https://doi.org/10.1145/3341105.3373957

[15] Sara Rouhani and Ralph Deters. 2019. Blockchain Based Access Control Systems: State of the Art and Challenges. In *IEEE/WIC/ACM International Conference on Web Intelligence (WI ’19)*. Association for Computing Machinery, New York, NY, USA, 423–428. https://doi.org/10.1145/3350546.3352561

[16] Bela Shrimali and Hiren B. Patel. 2022. Blockchain state-of-the-art: architecture, use cases, consensus, challenges and opportunities. *Journal of King Saud University - Computer and Information Sciences* 34, 9 (2022), 6793–6807. https://doi.org/10.1016/j.jksuci.2021.08.005

[17] Jan-Philipp Töberg, Jonas Schiffl, Frederik Reiche, Bernhard Beckert, Robert Heinrich, and Ralf Reussner. 2022. Modeling and Enforcing Access Control Policies for Smart Contracts. In *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*. 38–47. https://doi.org/10.1109/DAPPS55202.2022.00013

[18] Zeli Wang, Hai Jin, Weiqi Dai, Kim-Kwang Raymond Choo, and Deqing Zou. 2021. Ethereum smart contract security research: survey and future research opportunities. *Frontiers of Computer Science* 15, 2 (2021), 1–18. https://doi.org/10.1007/s11704-020-9284-9

[19] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. https://files.gitter.im/ethereum/yellowpaper/VIyt/Paper.pdf