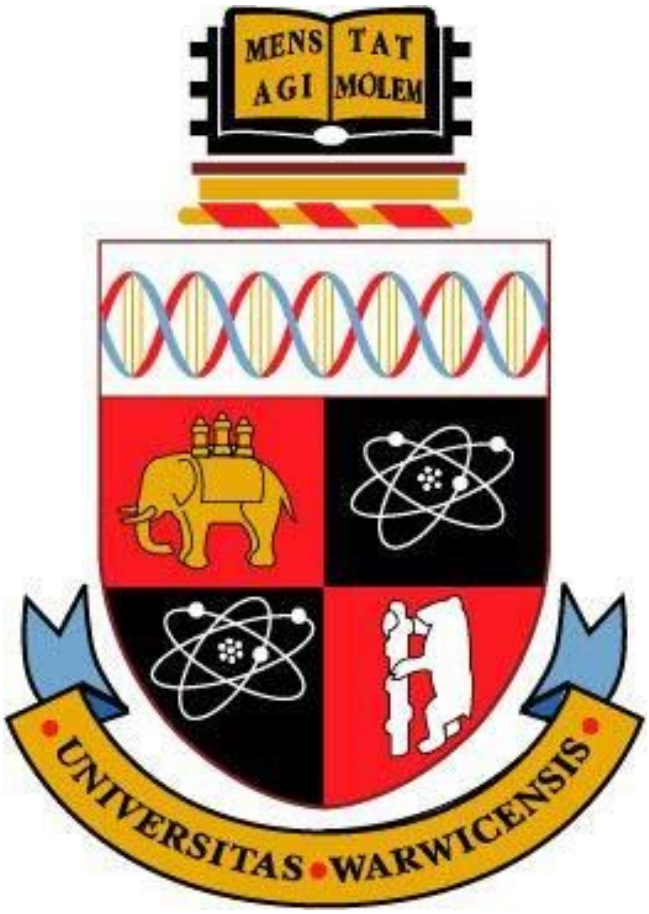
Private State Smart Contracts

Permissioned Information in Decentralised Networks



Final Report

by Artem Grigor

Third Year Computer Science Student
University of Warwick
2019-2022

Under the Supervision of **Dr. Charilaos Efthymiou**

Abstract

With the recent exponential increase in capabilities of decentralised networks, such as Ethereum, we have seen a range of applications appear on them. What holds back further growth of decentralised networks is the lack of privacy. Any Application's internal state, as well as all user interactions, are publicly visible. This made us ask ourselves how to enable privacy in decentralised settings. As a solution, we propose a system which allows applications running on decentralised networks to have secret, publicly inaccessible data. The solution is a trustless access control system for secret data and has been achieved using advanced cryptography (ZKP), hardware security computation modules (TEE) and consensus protocols. We have a working prototype, which can be used with decentralised applications hosted on Ethereum. In addition to the main result, we have proposed new techniques for managing time in State Machines in a hostile environment, approaches for combining TEE and ZKP and explored data revocation problem in TEEs. The main contribution of this project is to show how data access can be controlled in a decentralised system with no trusted participants. Future work might include reducing the security assumptions and making the system production-ready.

Keywords

Trusted Execution Environment, Blockchain, Data Privacy, Decentralised Networks, Zero Knowledge Proof, Access Control Management, Ethereum

Introduction

This section will take the reader through a brief history of decentralisation. We will follow this with and explanation of our motivation for conducting the project and then define the goals for this project.

Motivation

Road to Decentralisation

Centralised Computing

This section will give the reader a quick introduction to the origins of decentralised computing and how it has evolved. Then, we will talk about the first decentralised protocol, Bitcoin and its implications. Lastly, we will finish with the current state of the art technology and explain how it has enabled us to have Turing complete computations over the decentralised networks.

Since its foundation in the 1940s, the computing industry has primarily focused on exploring and building solutions in centralised settings. That is when a single party makes all sensitive computations, and you trust this party to work in accordance to set out rules and not abuse the system. An excellent example of such a popular system is Client- Server architecture which dominates the web. The client sends data to the server, trusting that it is not malicious and will perform correct computations, and then retrieves the result.

The reason why centralised computing was so popular is straightforward. Initially, only very few machines and groups of people could afford them. This meant that every connection between such machines was de-facto trusted. People knew who operated the machine was on the other end of the connection cable.

Decentralised Networks

This started to change in the 1980s [1] when computers became affordable. A great indicator of that is the first paper on decentralised networking by David Chaum [2], which later led to the infamous Tor [3] network now used for secure web browsing.

The body of decentralised computing research culminated in 2008 with the white paper by Satoshi Nakomoto, who proposed the first Peer-to-Peer Electronic Cash System (Bitcoin) [4]. This was the first time the world has heard about blockchain. It would be essential to note that the actual implementation of blockchain in our project does not concern us that much. Instead, we are interested in what blockchain technology enables users to do.

Before 2008, the world already had a solution for peer-to-peer communication in the face of network layer communication protocols, such as IP [5]. They did not depend on anyone's party to know the IP addresses of particular computers but rather worked by all participants sharing knowledge. This, with other cryptography techniques such as signing, was sufficient for building stateless decentralised networks. But a big problem persisted - how could the network participants have a shared state without centralised authority?

Decentralised Computational Networks

Bringing shared states to decentralised networks was precisely what Bitcoin did by leveraging Blockchain. Even though the original version only had minimal shared data of coin balances by all participants, Blockchain soon found its place in many other applications, such as Supply chain management [6], Elections [7] and much more.

The shared state in decentralised networks was extended to shared computation logic by Vitaly Buterin in 2014 when he unveiled Ethereum [8]. It was the first time a Turin Complete machine with finite computations could be run on top of the decentralised network [9]. This means participants could conduct arbitrary computation and arrive at a result that all of them would trust.

We will talk about the implications of having a Turin complete machine in the next section; However, now it is essential to understand that ever since the Ethereum release in 2015, the decentralised network could have, theoretically, replaced all centralised infrastructure we have had. We say 'theoretically' because there were limitations such as the amount of data stored, the number of state changes per second, and some other issues that the developers are trying to solve.

The Decentralised World

This section will show the reader the decentralised web we have today (web3). Then, we will briefly talk about how it works and mention its limitation as of today.

As mentioned in the previous section, Ethereum was the first decentralised network with Turing completeness. Now, there are many decentralised computation networks with similar functionality, yet Ethereum is still considered the standard of quality in the space. In Ethereum, anyone can deploy a Smart Contract [10] - a decentralised application. Then, anyone with the address of the application will be able to interact with it: read the state of the application or change (referred to as write) the state of the application. These smart contracts (decentralised applications) can be composed to make even more complex applications.

The fact that one can build almost any application and integrate it with other available applications has created a range of different services available on the blockchain. The most famous would be DeFi [11], which provides the users with decentralised financial applications such as exchanges, liquidity providers and loan providers. Overall, DeFi protocols currently hold more than 90bn USD as of May 2022 [12], which is could no longer be neglected.

The most significant novelty of all these services is that there are no trusted parties, so that applications can function without owners. The owner does not need to pay for server maintenance. Instead, the users pay for it while using applications. This allows services to function without commissions. And as all the applications are deployed to a shared database, anyone can see their source code and the current state of the application and all interactions that any other user did with such application.

This brings us to a crucial topic in decentralised computation networks - privacy. We will explore why it is vital in the next section.

Privacy with Decentralisation

So far, we have established a notion that the decentralised networks today can perform computation on a shared state (that is, de-

centralised computation networks) and that, currently, there is a range of financial and other applications working on top of the decentralised computation networks which use the computation resources they provide.

We will look into why privacy and the ability to store private data are significant for decentralised computation networks. We will bring up several examples of use cases where privacy is necessary and provide some analytical data on how much capital Web3 is missing out on because of lack of privacy.

Financial Privacy

Many people prefer to use decentralised financial services because they cannot be manipulated by any external authority and do not abide by the rules of any jurisdiction. Thus, the users can be exempt from the risk of loose cash due to a bank account being frozen or money being seized from it because of new regulations. One striking example of this is the situation in India in 1969 [13]

Yet, financial freedom comes with a price tag. All of the economic activity on the decentralised computation networks is completely visible to all other participants. Though acceptable for individual traders, this is a cause of significant concern for institutional investors, whose every move is tracked and measured. There can be no hidden orders [14] and dark pools [15]. And without it, any large crypto Sell order would immediately crash the market before any trade happened.

This prevents the big player from entering the game [16] and even further increases the range of services offered in decentralised computational networks.

Governance Privacy

Another excellent use case which arose in DCNs (decentralised computation networks) is decentralised governance, commonly referred to as DAOs [17] (decentralised autonomous organisations).

As we have mentioned before, the applications in DCNs usually have one owner and are controlled by a community of token holders that mimic shareholders in regular companies. However, as participants of the DCN do not need to give out their real identity, it is common for them only to be identified by a random string representing their network address. Thus, one can think that on a DCN, you have a new identity, not linked to your real one.

Since shareholders of the application on DCN are only known on a DCN, there were specific systems of governance developed on the DCN to help them with the decision process. Now, there are many such applications on the DCNs, such as Ethereum, that allow doing precisely that - start and run DAOs.

The biggest problem with applications built to run DAOs is that the community has no privacy. Every community decision or a vote is immediately visible to the whole network and can be acted upon. There was once a case when a group of network participants decided to buy a copy of a Constitution; they gathered money of around 28ml and took part in an auction. Yet, everyone knew exactly how much they could bid. They were outbid by 1 dollar and lost the chance to buy the constitution [18]. Or should a group of members have decided to share a password, they would have needed to come up with another solution, as DCN does not offer private communication for a particular

subgroup.

Analytics

It can be estimated that if there were private data in the web3, it would have brought close to 5tr $ of investment value into it and enabled corporations to adopt it for their internal use [19]. Right now these corporations end up using custom DCNs to have their transactions private. However, they require maintenance and are not as secure as Ethereum.

Additionally, having a private shared state, such as a shared admin password to a forum for DAO members, would allow application developers more flexibility. Making applications more user friendly bring in even more capital and users to the web3, opening up a new expansion step.

The examples above and the analysis bring us to how important privacy is for the further development of the Web3 ecosystem. This is widely accepted by the industry and was mentioned by Vitaly Buterin in 2016 [20] when he pointed out that privacy is vital for the global adoption of decentralised networks, yet is extremely hard to achieve. This introduces us to the problem statement which we are trying to solve.

Problem Statement

As we have learned, decentralised networks have gradually evolved from handling peer communication to hosting numerous applications and services. These decentralised applications form the new internet, sometimes referred to as the Web3. Due to the nature of such Decentralised Computational Networks or DCNs, all data used by the hosted applications are public knowledge. Any decentralised network participant can see it. This makes it impossible to use the hosted applications privately. All information inputted will become publicly known. This poses a significant limitation for Web3 development.

The objective of this project was to address the issue of having no privacy on the DCNs. We have looked into how to integrate a notion of privacy into the current most popular DCNs, such as the most famous instance of Turing Complete Blockchain - Ethereum. Considering the scope of the issue being too wast and limited development and research time, we have decided to narrow the scope to **building a decentralised access control system integrated with DCNs**.

We will refer to the decentralised data access control system integrated with DCNs as simply Private State Smart Contracts (PSSC). This is because our system will allow smart contracts (applications in DCNs) to control access to the private data stored in our system.

Private State Smart Contracts have very versatile applications. One use case for PSSC is to allow one to have data that would be only accessible by a dynamic limited group of network participants. The group would be determined dynamically based on the current state of the DCN, with network participants losing and gaining access continuously based on their interactions in the network and changes to the shared state of the network. An example of this would be a stock market analytics subscription - you only get to see the issue if you have paid for it. Such service would only be possible with a centralised system as no decentralised solutions exist.

In the next section, we will formalise the project requirements for PSSC. The PSSC is a decentralised system used to manage private data access based on the state of the DCN.

Project Specification

This section clarifies the project aim by formalising what we mean by the Decentralised Access Control System integrated with the DCN, also referred to as Private State Smart Contracts. We also provide details on the project's scope, objectives, and requirements list.

Based on the problem statement in the section above, the project aims to construct a prototype of Private State Smart Contracts. It should be decentralised and allow private data access controlled by the state of a decentralised computing network.

We can further generalise the definition: **we want to construct a decentralised access control system, access rights for which are defined by an independent decentralised system. We need to make our system have no centralised components.**

Please note that the project's main deliverable is a protocol. The prototype we will build is only used to accompany the protocol and show that it works.

We believe that the project's result will advance the development of private decentralised systems on blockchains and other DCNs and help Web3 to grow even further.

Scope

We have set out to build a system that will store secret data and disclose it to any user who can prove the right to access the data. The right to access the data will be derived from the state of the DCN, that is, a state of a Smart Contract.

We will not explore how data can be deleted from such a system in the project. A case can be made that the data submitted to the system loses any ownership and should only abide by the contract rules. Additionally, creating such functionality has proved to open doors for many potential attack vectors, which would overcomplicate the project.

Additionally, this project will not consider the security of any used vendors' software solutions if the underlying technology is potentially secure. This is because the project is rather theoretical and is aimed to create a prototype with a functioning protocol rather than a production-ready solution. However, this will be included in the report's 'future work' portion.

Objectives

1. **Background research**

   The ambition: research which techniques can be used to create a decentralised access control system integrated with decentralised computation networks.

   The outcome: a set of tools and techniques suitable for developing the prototype and experience using such tools and techniques.

   The approach: talk to industry experts, examine the latest papers on the topic and, thus, accumulate the most common techniques. Additionally, test these techniques on a small scale to understand how feasible is it to apply them in real life.

2. **Prototype development**

   The ambition: develop a live system that fulfils all the PSSC requirements.
   The outcome: a working prototype which can store and disclose data. It should reveal the data only to someone who possesses the required access right based on the state of the real DCN, Ethereum.
   The approach: Follow an iterative design cycle, and use sprints to gradually extend functionality until the prototype is finished. Have a list of objectives for each sprint before starting the sprint.

3. **Prototype evaluation**

   The ambition: assess the solution security, scalability and performance.
   The outcome: a list of points stating the prototype's limitations, assumptions and performance. Additionally, an evaluation of all requirements for whether they have or have not been fulfilled.
   The approach: Document all limitations and issues encountered during development. Assess usability by trying real-life use case scenarios. Finally, manually check the outcome against a list of recorded requirements.

4. **Document insight**

   The ambition: record all learnings during the project work cycle and mention all future expansions if further work ensues.
   The outcome: the final report that reviews the work done, outlining anticipated insights and anticipated future work.
   The approach: after the completion of the project, evaluate it. How was project management done, what can be additionally developed, and how can the project be improved.

Side Objectives

1. Development experience in
   (a) Trusted Execution Environments [see Background section for details]
   (b) Zero-Knowledge Proofs [see Background section for more information]
2. Ethereum Smart Contracts
3. Theoretical knowledge in modern cryptography
4. Knowledge of novel Encrypted Computations Solutions
5. Knowledge of how cross-chain communication works
6. Learn about the internal Ethereum computer structure (EVM [21])

Prototype Requirements

The prototype produced as part of this project must follow the following requirements. We have additionally marked them using the MoSCoW prioritisation system [22] to follow project management best practices.

1. Data Submission should work correctly

Should Anyone can create new secret data

Could Any data could be used as secret data

Must Access rights should be specified during the creation of private data

Should Access rights should be flexible, with the option to link access to the state of any smart contract

Should Access management of secret data should be done independently of the submitter once the data is in the system

2. Data Access should work correctly

Must Data should not be disclosed to users with no access rights

Must Data should be disclosed to users with access rights

Should Users who no longer have an access right to the data should not be able to access the data

Should The access rights should be in sync with the DCN state, which is used for access right reference

Non-functional requirements

Should Access requests should be processed promptly - granted or rejected

Should The access control system must be decentralised with no single source of trust

Could There should be no special technical requirements for users' devices to request access to data

Could There should be no special technical requirements for users' devices to submit the data

Could The system must not require any special technical knowledge to use

Should The system must be protected against attacks by adversaries

Should The system must be capable of persistent continuous operation

Must Access right sync should happen promptly after every DCN state change

Should Access rights types should be easily adjustable, and new ones should be easily added.

Ethical Considerations

The project did not require any involvement from external participants, nor did it collect or process any private or personal data. Thus, there are no direct ethical issues to consider. However, it is essential to note that privacy is very controversial. It has been stated that privacy could potentially harm national security and, thus, be viewed as not ethical. [23]

As our project is working on developing new tools for preserving privacy, the possibility of it being abused must be taken into account.

1. The PSSC will enable us to store permissioned information on public systems. This might allow unlawful information to be stored and distributed without any means for the public authority to intervene due to its decentralised nature.

2. Private data submitted to the PSSC can potentially be abused and hacked. Thus an unintended leakage of personal data could be an implication of using our solution.

3. Finally, the system that we have built does not offer a way for the submitter to delete the data. If unintended data is submitted or the information should no longer be publicly accessed, it will not be possible to revoke the data unless most network participants agree. We will explore this more in the future work portion of the report.

Background

This part of the report will focus on what we have learned while conducting the industry survey. We will primarily focus on the tools and techniques we have used in the project. Thus we plan to accustom the reader to the lexicon we will use in the future parts of the report. Additionally, we will talk about other solutions already present and developed in the space, how they compare with the project, and their advantages and limitations.

Theoretical Privacy Techniques

This is the first section of our background survey. Here we will talk about the predominant theoretical techniques used in the industry to develop trustless private applications. We will mention both pure cryptographic methods as well as hardware-enabled solutions.

Trusted Execution Environments

A trusted execution environment or short for TEE, first introduced in 2009 [24], is a hardware privacy solution. It is a particular type of CPU such that any application executed in this CPU will run according to its specification. Even the owner cannot impact the execution of the application in any way except to terminate it by cutting the power. Moreover, trusted execution environments work in a fully encrypted mode, meaning that the owner will not even be able to know what execution steps are happening inside the TEE and which data it processes at the moment.

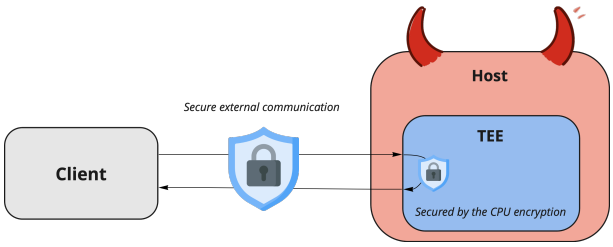


Figure 1: TEE Overview

To better understand the idea of TEE (Trusted Execution Environments), we can draw the following analogy. Let us imagine one has a computer and a friend who would like to use a computer. Usually, how it would work is the friend will use a computer in your presence, and you will be able to see everything that your friend does on the computer. This is convenient for you, as you are sure nothing malicious is executed on your machine. Yet your friend has no privacy; you can overlook any password they enter. A standard solution you and your friend agree to is that you should turn away when the other is doing anything private and Confidential. But it does require a great sense of friendship to do so. Moreover, you can always take the computer forcefully with all login credentials of your friend and take out all the cash from their account.

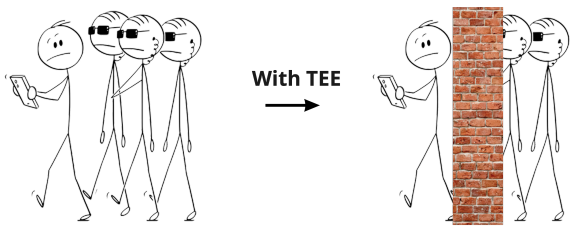


Figure 2: TEE Security improvement

TEE builds a brick wall between you and your friend using your computer. You will no longer be able to see what your friend is doing on your machine and influence or regain control of the laptop before your friend finishes and deletes all private credentials.

As we can see, TEE allows anyone to execute work on your machine without you being able to see what is being performed and influence the execution.

Now that the reader has this more general overview of the TEE, we will further explore how this is achieved, the drawbacks of TEE, and look into tools currently available for development.

Low-Level Implementation

The most mature and most used TEE solution is developed by Intel [25] and is called SGX (Software Guard Extensions). It has been integrated into their CPU for several personal and industry-grade servers. This section will discuss how SGX works, which transfers to general knowledge of how TEE works.

Intel CPUs which support SGX allow you to execute work on encrypted parts of memory. The encryption key to the memory is only known to the CPU itself as it is baked into it during manufacturing. This allows the CPU to create encrypted data that no one else but it can decrypt, process and reencrypt on the fly. The CPU can additionally check the integrity of the data using checksums [26][27]. All of this is sufficient to support secure computations that can not be manipulated or overseen by the owner of the CPU.

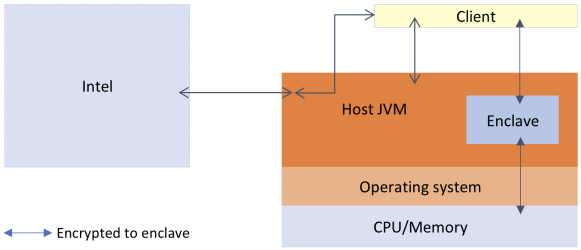


Figure 3: TEE Architecture in SGX

However, how does one know if the computations are performed in an authentic TEE rather than a regular vulnerable CPU? The SGX CPU can do it using its key and a concept of attestation. SGX CPU can generate proof that it is authentic. Then the owner of the SGX CPU sends the proof to Intel, who then will verify this proof. If the proof is correct, Intel will sign on it. If anyone sees a CPU that can demonstrate that it has authenticity proof signed by intel, they can trust the CPU is secure.

Let us also explain how the proof and verification work in more detail. First of all, when the CPU is manufactured, two keys are created - a public and private key, for example, following an RSA scheme. As follows from the name, the private key remains hidden in the CPU, while the public key is stored at Intel. The property of these keys is the following: the CPU can sign a message with the private key (PK), and the Intel can verify that a CPU PK has indeed signed the message by checking it using the Public Key (PBK) [28] that It has stored initially. If the verification succeeds, Intel then

signs a message using its PK, which the end-user can verify as the PBK of Intel is well known. It should be noted that the Private-Public Key Cryptography, which appeared in the 1970s, can be considered to be the first Zero-Knowledge Proof system. We will cover other ZKP systems in the next section.

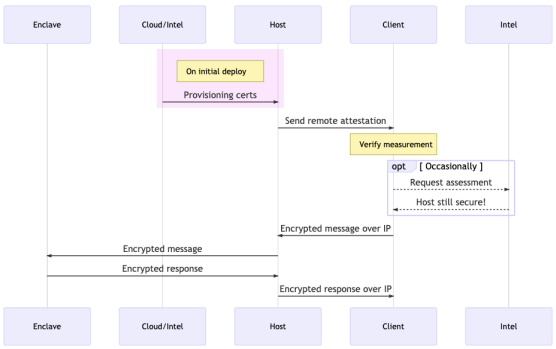


Figure 4: TEE verification sequence diagram

There are additional levels of security in the SGX system, such as updating and verifying the microcode running in the CPU[29]. However, we leave this up to the reader to explore as this is out of the project's scope.

Security Guarantees The TEE system gives us the following security guarantees:

1. The data that is processed is encrypted and inaccessible to external participants
2. The external participants can not influence the execution of the application inside the TEE
3. The TEE can prove to others that it does not have any known vulnerabilities and is up to date
4. The TEE communication is controlled by the owner, who can limit who can connect and interact with it
5. The attestation eliminates the possibility for someone to mimic the TEE with a regular processor

Security Concerns

It must be noted that the TEE is far from being protected from all possible attacks. However, it is secure against the computer owner the TEE applications run on. In this section, we have produced a list of several security issues of TEE. This list proved to be very useful during the development time as it allowed to control and prevent many attack vectors.

1. Malicious actors can still hack applications that run in the TEE the same way as the applications running on a trusted centralised system by providing malicious input.
2. The TEE can have unknown exploits which can compromise its security
3. The owner can learn undesired information by using side-channel attacks [30]
4. The TEE owner is in complete control of how TEE interacts with the external world. It can prevent messages from coming to the TEE and leaving it. We will address this issue in our implementation. Additionally, the host can reorder messages from different participants, thus manipulating the execution inside the TEE.
5. The owner can revert the state of the TEE to a previous state. This can be done by replacing the encrypted memory content with the earlier encrypted

state. TEE can be viewed simply as a State Machine, and the owner can move it to any previous state. [31] And one can quickly revert it

6. TEE has no trusted sense of time, meaning there is no inbuilt way to know how much time has elapsed in the TEE [31]

Development Tools

Here we will introduce tools used in the project with justification for selecting them.

As mentioned before, we have used SGX by Intel as a target TEE architecture. This choice has been made because the SGX has a wide range of development tools and easy access to processors with SGX support.

We have used the Conclave [32] development platform. Even though SGX has a C++ API available for developers, it requires much expertise and is not as developer-friendly. Additionally, due to it being too low level, there were no libraries which would support the project's ambitions. We have chosen Conclave for several reasons when choosing between available SDKs on top of SGX. First, it provides Java VM inside the TEE, with a vast developer community. Next, the Conclave allowed abstracting most of the SGX away from the application developers, taking only three lines of code to connect to the SGX. Last but not least, I contacted the core developers of Conclave, which proved to be very helpful when I had questions about the architecture and SDK functionality.

We have used the Microsoft Azure cloud platform for live deployment as our machine did not have an intel processor. Azure is currently the only cloud platform that supports SGX. Thus there was little choice as buying an SGX enabled computer would be too expensive.

Conclusion

The TEE is a very new technology. Most of the tools used were developed in the last five years, noticeable with a lack of documentation and standard practices. However, it has been relatively easy to start using and has not required much special knowledge. Most importantly, developers have built it for developers, which is why it was so comfortable to use.

Public Key Cryptography

We have mentioned the Public Key cryptography in the previous section when talking about Trusted Executing Environments. James Henry Ellis first proposed Public Key Cryptography in 1970. Its first implementation was done seven years later by the Rivest, Shamir and Adleman, known as RSA [33, Chapter 19: Public-Key Algorithms].

Let us now revisit what Public key cryptography is. Once again, everyone has two keys - public and private. The public is shared with all other participants and can be treated as the user's address. The private key is a key from the post box located at the address.

People can send messages to this address (public key), and they will only be decrypted by the true owner (private key owner), who can unlock the post box. There is another function for the public keys. The owner of the post box can authenticate that the message they have sent to the other partic-

ipant has indeed originated from their address. Then anyone could check that the message has been dispatched from the address they know. And if this is so, they will trust the sender.

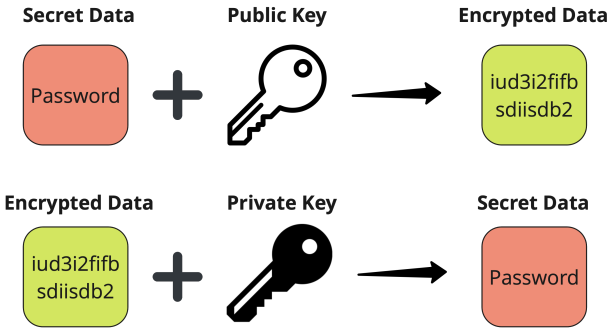


Figure 5: How public key encryption works

This relatively simple scheme can be used as a building block for other very complicated schemes. Here is an example of how it can distribute data between participants who have paid for data access [34].

We have used Public Key Cryptography in our project combined with other techniques. However, we have shown that it would not be sufficient to have just the Public Key encryption scheme to achieve the project objectives because the access to the data should be dynamic. Thus, it is impossible to specify who should be allowed to access the data in advance.

Zero-Knowledge Proofs

Zero-knowledge proof (ZKP) is a powerful cryptographic method that allows one party to prove to the other party some statement without conveying some secret knowledge required to verify the statement [Applied Cryptography by Bruce Schneier, Chapter 5: Advanced Protocols]. We have already seen an example of a ZKP protocol in the face of public-key cryptography. When signing a message, a user proves he knows the private key without revealing it, and the receiver of the message can check (verify) the signature using the public key and not knowing the private key.

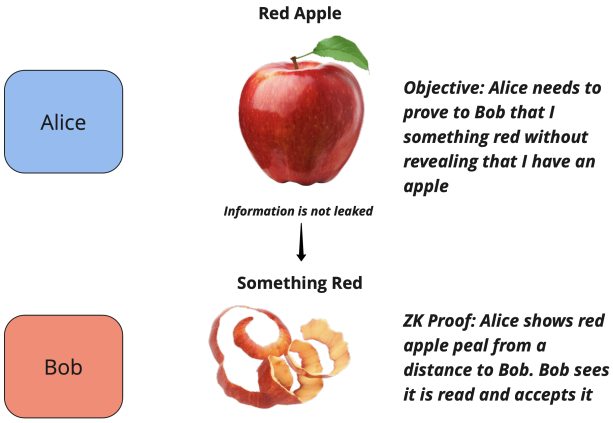


Figure 6: Example of Zero Knowledge Proof

Another famous example of the ZKP protocol that will explain how powerful this paradigm is is 3-Coloring. In it, a prover can prove that he knows the solution to the graph colouring without revealing it [35]. As we know, 3-Coloring is NP-complete [36, Chapter 36: NP Completeness]; thus, any other P and NP problem can be reduced to it. This implies that we can prove that we have done any NP computation satisfying some constraint to the verifier without disclosing the output. However, as expected, this approach has enormous overheads and would not be used in real life.

However, we now have new techniques to efficiently create proof without extensive interactions between the prover and verifier. Currently, ZKP techniques all follow one of two main branches. They are either based on the zk-SNARK or zk-STARK approach.

zk-SNARKs

zk-SNARK or zero-knowledge succinct

non-interactive argument of knowledge is considered an older and much more established approach to constructing zero-knowledge proofs. To give perception for the reader of how novel the space is, the zk-SNARK was introduced only ten years ago by Nir Bitansky et al [37].

Now we will give a very brief description of how zk-SNARKs work.

The most crucial part is a polynomial commitment scheme [38]. The prover discloses only a gist of a polynomial to the world without revealing the whole polynomial. Then, the proper can evaluate the polynomial at any specific point and send this value with some additional information (witnesses) to the verifier. The verifier would use the information and the commitment to verify that the prover has provided the correct polynomial evaluation. An example of how much commitment can be created is the KATE commitment scheme, also known as KZG [39]. It should be mentioned that this approach requires knowledge of Elliptic Curves and their pairings [40, Chapter 9 - Pairings], which is itself a very advanced topic.

Having the commitment scheme, it is relatively easy to construct zk-SNARKs. We could show that an arithmetic circuit with a set of public and private inputs and outputs (private inputs are not hidden from the verifier) can be converted to a set of polynomial constraints, proven using the polynomial commitment scheme. We have simplified many steps, yet, this hopefully gives an overview of how zk-SNARKs work.

The most popular zk-SNARK methods for constructing from arithmetic circuits a list of polynomial commitments [41] include Groth16 [42], Halo [43] as well as PLONK (Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge) [44].

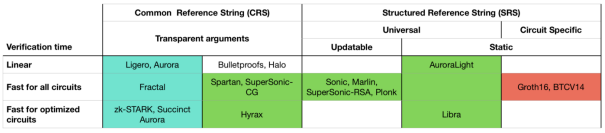


Figure 7: Comparison of ZKP Protocols

The biggest downside for all zk-SNARKs is that they require a trusted setup - there is an original secret needed to set up the environment for creating proofs. If the secret is not discarded, the verifier can potentially use it to extract secret information from the proofs that it verifies.

zk-STARKs

The most significant advantages of zk-STARKs over zk-SNARKs are that they do not require an above-mentioned trusted setup and are additionally post quantum secure. Meaning that once we achieve quantum supremacy [45], the prover still will not be able to fool the verifier (soundness property). In contrast, SNARKs are not suitable for this due to the limitations of pairings in elliptic key cryptography [46].

The structural difference between zk-SNARKs and zk-STARKs is how the polynomial commitments work. Zk-STARKs utilise fast Fourier transform and hashing [47] instead of elliptic curve cryptography.

The lack of efficient implementations and developer tools is the most significant current limiting factor for developing zk-STARKs. The original paper introducing zk-STARKs was published in 2018, and the most efficient zk-STARKs proof sizes are still much larger

than those of PLONK or HALO2. However, zk-STARKs have potential to become a dominant technology in the long term. Conclusion zk-STARKs and zk-SNARKs are effective cryptographic techniques which allow proving statements without revealing them. They offer additional security compared to TEE as their security is mathematically proven. However, the technology requires exceptionally advanced knowledge of cryptography, and it does not provide a universal computation solution. Instead, one must convert problems to the ZKP settings, which is impossible for some problems.

Multi-party computation

Short for MPC, this is a more general version of the zero-knowledge proof system. Instead of a prover and verifier, we have a set of participants whom all have a secret and would like to compute a function on their collective data without revealing it. The first specific case applications were produced in the 1970s, followed by a universal solution first offered by Andrew C. Yao in 1986 [48] for 2 parties and Goldreich Micali Widgerson for more than 2 parties case. Since then, the development in the space has focused on how to make the protocol more efficient.

The most significant disadvantage of current protocols is that they require many interactions between parties, which means all participants need to be online. This in itself is a stringent requirement. Additionally, there is no tooling to convert common computational tasks efficiently to the MPC paradigm, which made us avoid it during the project.

Fully Homomorphic Encryption

This cryptography technique works similarly to the TEE. However, instead of trusting the hardware, you need to trust the mathematics. Fully Homomorphic Encryption defines how to create functions that accept encrypted data, perform some computation and output encrypted data [49]. The essential part is that the computation happens over the encrypted data. Thus the person conducting the computation does not know what is being computed and the output. Only the person who provided the input data would be able to decrypt it.

	Trusted Execution Environments	Homomorphic Encryption
Type of privacy	• Securely outsourcing to a server, or cloud, computations on sensitive data	• Securely outsourcing specific operations on sensitive data • Safely providing access to sensitive data
Privacy risk addressed	• Revealing sensitive attributes present in a dataset	• Revealing sensitive attributes present in a dataset
Data protected	• In storage • During computing	• In storage • During computing
Benefits	• Commercial solutions widely available • Zero loss of information	• Can allow zero loss of information • FHE* can support the computation of any operation
Current limitations	• Many side-channel attacks possible	• FHE currently inefficient, but SHE* and PHE* are usable • Highly computationally intensive; bandwidth and latency issue • Running time • PHE and SHE support the computation of limited functions • Standardisation in progress

	Secure Multi-Party Computation	Differential Privacy
Type of privacy	• Enabling joint analysis on sensitive data held by several organisations	• Organisation releasing statistics or derived information – generally an organisation that holds a large amount of data
Privacy risk addressed	• Revealing sensitive attributes present in a dataset	• Dataset or output disclosing sensitive information about an entity included in the dataset
Data protected	• During computing	• At point of dataset or result disclosure
Benefits	• No need for a trusted third party - sensitive information is not revealed to anyone • The parties obtain only the resulting analysis or model	• Formal mathematical proof / privacy guarantee • The user can set the level of protection desired, in particular by reasoning about the number of times the data might be queried
Current limitations	• Highly compute and communication intensive	• Noise and loss of information, unless datasets are large enough • Setting the level of protection requires expertise

Figure 8: Comparison of Cryptography computation protocol

It must be noted that FHE is currently very inefficient. It is 1000 times slower than the TEE [50]. The first solutions to building Turing Complete Fully Homomorphic Encryption schemes are dated 2009 [51]. Since then, it has gradually improved its efficiency. Additionally, it does not have a state, and building stateless applications is an additional difficulty in an already very ambitious project. Thus we will only use it as a reference and not in actual project applications.

Summary

Overall, we have evaluated five different privacy preservation solutions. Yet as the

reader has noticed, we have primarily focused on the Trusted Execution Environment (TEE) and Zero-Knowledge Proofs (ZKP). Its reasons include how well developed the technology is, its efficiency, and its application in the project.

Distributed Privacy Applications

In this section, we will focus on projects that have been developed for the Decentralized Networks and focus on user privacy. For each such project, we evaluate the use case and how it is compared to the scope of our project. Additionally, we check if there are any limitations to their technology. We classify the projects according to the theoretical solution on which the project was based.

ZKP Applications

Here we will run through several most notable privacy solutions in decentralised computational networks, which utilise ZKP.

Zcash

Overview This is a blockchain protocol which uses as part of its consensus mechanism Zero-Knowledge proof. The proof confirms that someone has created a valid transaction without revealing the transaction details. The sender can hide their identity or the amount of money transferred. This makes it impossible to track how money flows in the system and identify users using side information, the way the FBI did it in Bitcoin [52].

Limitations Being one of the first practical applications of ZKP, Zcash's most significant limitation is that the platform is not versatile. We can not call Zcash a DCN as, besides moving funds, it currently can not be used in another way.

Project Relationship Concerning our project, Zcash is an example of how ZKP can be used in complex systems to achieve consensus. However, it does not offer an answer on creating an access control mechanism as ZK is used only to verify if the action is correct rather than grant access.

Dark Finance

Overview Primarily based on the Zcash ZKP technology, Dark Finance [53] is built to bring privacy to decentralised financial applications. It can be viewed as an advanced version of Zcash, developed on top of the Ethereum blockchain. Applications built on Dark Finance include anonymous money transfers, like ZCash, private exchanges and auctions.

Limitations The most significant limitation of Dark Finance is that the project is less than a year old, with the SDK for building private decentralised applications not being publicly available yet. Moreover, the development language is much different from ordinary development languages. This is why it takes a long time to get acquainted with it.

Project Relationship Overall, it can not be said that Dark Finance on its own is sufficient to build the Access Control system we

have outlined in the introduction. As with ZCash, it can not be used to encrypt and decrypt data as ZKP does not support this. However, Dark Finance is an excellent example of how ZKP can be applied and could be used for some parts of the project, such as credential verification.

Panther

Overview The Panther can be seen as being very similar to Dark Finance. Its website says that they are a Private scalable blockchain protocol for institutions joining DeFi [54]. The main difference is that they want to build a Turing Complete Decentralised Network that will work purely on ZKPs. This will allow private firms to safely do business on blockchain and solve some of the issues we have outlined in the introduction section.

Limitations The most significant limitation of the project is the same as for Dark Finance. The protocol does not solve the issue of private data access we have originally online in the project. It only focuses on hiding the meaning of the actions and not on how to have private information shared across the system. It also should be noted that due to the inherited limitation of the ZKP DCNs, this approach will not scale [55].

Project Relationship As mentioned above, the Panter does not solve the issue which our project has set out to solve. However, it can be potentially used as a part of a more complex system. The Panther will be used for access right evaluation, and another complex system will read from it and enforce these restrictions. Yet the protocol does not answer how to implement these restrictions.

TEE Applications

This section will mention the known applications of TEE in the distributed network settings and evaluate their limitations and how they match our project's ambitions.

Secret Network

Overview Similar to Panther, Secret Network [56] attempts to build a DCN with total privacy. They currently only focus on the transaction part. However, they have plans to launch DeFi or, better say, PriFi very soon. The system is very versatile and already offers private states smart contracts. There is additionally a language which the team provides built as an extension of Rust.

Limitations The internal TEE system is very inefficient for working on Secret Network. They do not utilise the total capacity of the TEE and are limited by the initial design choices. Moreover, as Secret Network is currently not compatible with Ethereum, it has minimal adoption.

Project Relationship We have tried writing the Secret Network contracts and have faced a bottleneck in how the transactions function. However, we must say that theoretically, the Secret Network could be our use case. The only limitation was that Secret Network is an isolated DCN with no built-in integration with the major DCNs such as Ethereum. This would make us build a bridge between these DCNs to pull data between them, which is a very complicated

and error-prone task due to the number of security issues [57]. Moreover, because Secret Network has a limited API compared to the available TEE API, we have anticipated that the development using the Secret Network toolstack is not guaranteed to be successful and terminate in time.

Obscuro

Overview This project is identical to the Secret Network but for one thing. It is connected to the Ethereum ecosystem with a pre-provided bridge, making it more secure. Additionally, it has no internal privacy guarantees, rather instead focusing on only temporary privacy that will be disclosed after a few minutes or seconds.

Limitations The two significant limitations are the lack of internal privacy and the fact that there is no inbuilt way to pull data from Ethereum. Obscuro [58] was launched after our project began; it is still reasonably young, with plans to launch the alpha version earlier this summer.

Project Relationship We have worked closely with the Obscuro team, having met several times. However, the Pbscuro project is still very young and can not be used as part of our project.

Chainlink

Overview Chainlink is a network of Oracles that provide services for bridging several DCNs together. One can see a DCN as an island and oracles as ships that send goods between the islands and exchange information. Chainlink uses the TEE to allow anyone to build a custom oracle, which will gather data from system A, process it in a specific manner and then move it to system B. The systems could be DCNs.

Limitations The major limitation of Chainlink is that it only offers oracle service, which is only part of the puzzle of this project.

Project Relationship Chainlink can be a starting point for constructing oracle systems and getting information from Ethereum to other chains.

Summary

We have evaluated several prominent ZKP and TEE focused applications. We have gathered the following insight from them.

The ZKP applications can be used to solve a part of the problem - how to evaluate if someone has or has no right to access data without leaking private information. However, they do not have an explanation on how to enforce these constraints. Thus, we find that they do not have a solution for a problem we have set out to solve in this project.

However, it must be noted that ZKP protocols have an exciting quality - they prevent information leakage. Thus our system can be enhanced by using them to achieve total privacy for the users.

Concerning TEE, we have seen them as a lot more promising, allowing us to achieve almost what the project ambition is. Yet the fact that TEE applications are not mature enough and have limited functionality compared to what our project requires, we could

not have just used these applications to combine them into one system.

Nevertheless, we see that TEE is more suitable for the most crucial part of the project - data access control. And it can be further improved with oracles and ZKP to abstract who accesses the data and prevent information leakage.

Implementation

So far, we have provided the reader with the motivation for conducting the project, project objectives and background information about privacy preservation techniques available. We have also justified why we have chosen particular privacy preservation techniques to complete the project.

We will now explain how the protocol we have developed works, how the project has been structured, what we have developed, and walk through challenges we have faced along the way. This is the technical portion of the report, where the reader will see examples of code snippets and technical terms introduced in the background section.

Solution Description

This part of the report will describe how the application works. We build on this knowledge in the following parts of the report. This is an essential part, as we introduce many concepts necessary to understand the other parts of the report. Please note that we will not go into implementation details in this section but rather focus on the design decisions.

The system we are building aims to **create a decentralised access control system integrated with DCNs**. The exact requirements and clarifications on each of these terms have been outlined in the Introduction part of this report.

As discussed in the background portion of the report, we have decided to build the application using a TEE to preserve privacy. We will refer to the application we are developing as an Enclave.

System Workflow

1. The application accepts secret data submissions from different users with conditions on whom disclose the secret data to
2. The data submitters share the data-id with the data requestors, this happens out of the scope of the project
3. Anyone can request the data from the application using the unique data-id. The user will also need to prove that he can access the data.
4. To verify the proof, the TEE will request the oracle network to confirm if the condition is correct. Oracles are a group of independent nodes connected to the TEE that perform a specific set of actions if asked to. See Chainlink for details on Oracles [59].
5. If most Oracles have successfully verified the proof, the TEE accepts it, and the requestor receives the secret data.

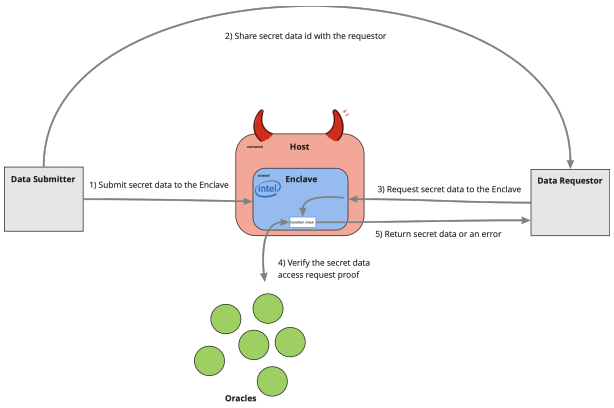


Figure 9: Workflow Diagram with all interactions labeled

System Participants

1. **Data Submitter** - submits the secret data to our application, Enclave, with conditions that need to be fulfilled for anyone to access this data.

2. **Enclave** - the application that runs inside the TEE. It stores secret data and decides whether to disclose it to data requestors by verifying if the data submitter conditions have been fulfilled. If the verification is not deterministic, Enclave employs the Oracles to provide the answer.

3. **Enclave Host** - a party that maintains the hardware on which the Enclave runs. We consider this party to be untrusted.

4. **Data Requestor** - anyone who wants to access the data stored in the Enclave. With the requested data-id, this participant provides proof that he can access the data.

5. **Oracle** - nodes connected to the Enclave and any other DCN, which help verify if the proof provided with the data request is valid. After each Oracle comes to an answer, Enclave aggregates their results to decide whether to provide data or not.

Important Concepts

1. **Secret Data** - data submitted by the data provider which should only be disclosed to parties satisfying a set of requirements.

2. **Disclosure Requirements** - the conditions to evaluate if someone is allowed to access the secret data stored in the Enclave. The requirements can be either deterministic or non-deterministic. Non-deterministic requirements need external knowledge not initially available in the TEE to assess if the requirements are fulfilled. Note that our system supports non-deterministic requirements, making it unique compared to other systems we reviewed in the report's background portion.

3. **Proof of Access Right** - what the data requestor provides to the Enclave. It proves that the requestor is allowed to access the data and fulfils the Disclosure requirements. It must be noted that this is a potential source of information leakage as this information or part of the proof can be forwarded to the oracles to verify it. And oracles can, thus, gain knowledge about who is using the system.

4. **Smart Contracts** - we have previously mentioned smart contracts as an application running on the DCN. How-

ever, in the report's scope, we will view them purely as a source of accessible DCN's state information. The data provider can specify as disclosure condition the fact that a particular Smart Contract on the DCN evaluates for a specific value, which oracles assist in checking.

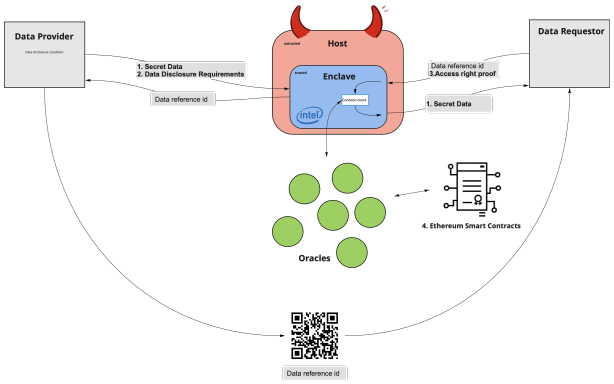


Figure 10: Workflow Diagram with all concepts labeled

System Components

The deliverable components mimic the participant set, with each participant having their own application to run. Here we will describe each deliverable and the important subparts they consist of.

1. Enclave application
   (a) Encrypted Database - stores secret data submissions.
   (b) Messaging System - receives and sends messages to and from different participants.
   (c) Access Evaluation System - evaluate if the requestor has provided valid proof to grant them access to the secret data.
   (d) Oracle Management System - controls oracles, removes those who have timed out and much more.
   (e) Time System - estimates time occurred in the Enclave
2. The application used to submit data to the Enclave
3. The application used to request data from the Enclave
4. Oracle application
   (a) DCN Query Service - requests the state of the DCN from the known RPCs
   (b) Enclave Service - connects to the Enclave and receives tasks to verify the proofs

Development Tools

List of the development tools as well as a justification for choosing them.

- *JVM* - Java Virtual Machine, execution engine for our applications. Chosen due to our experience with it and a lot of mature developer documentation available.
- *Conclave SDK* - an SDK to develop in TEE, used with the Intel SGX. Chosen due to how easy it is to create applications with it and that it targets JVM, with which we have much experience.
- *Java 11* - language to write JVM programs. The particular version was selected due to the Conclave SDK compatibility restrictions.

- *Kotlin* - a language which compiles to the Java 11. Chosen as it provides a lot of useful abstractions and has qualities of functional programming. Very useful in security-focused development [60].
- *Gradle* - package manager for the JVM. Allows us to automatically add dependencies and libraries, structure the project, and automate compilation. Selected due to it being the industry standard.
- *Git* - software version control manager. Chosen as is the most standard solution for this task.
- *Docker* - a container manager which allows to run of applications in specific environments. Selected for it is the industry standard.
- *Web3j* - cryptography and crypto library, allowing JVM applications to interact with the Ethereum blockchain and generate keys, check signatures, etc. Selected as the only maintained library for Java.
- *Ethereum* - a public DCN. It is selected because it is the biggest DCN right now, with the computation architecture standard EVM being mimicked and supported by other DCNs.
- *PicoCLI* - a library used to build a quick and durable Command-line interface. Picked due to the project's focus on the underlying technology and not on how pretty the user-focused applications look. The objective of the project was to develop a prototype.
- *Kotlinx.Serialization* [61]- serialisation library. Selected due to very tight integration with Kotlin, a language in which most development has been done.
- *Amazon Azure* - cloud computing platform. Selected due to vast popularity in the industry and is the most prominent player supporting Intel SGX.
- *Log4j* - a popular Java logging library. Even though a known vulnerability was discovered in the log4j [62], we still considered using it since the issue has been resolved and much documentation has been developed for log4j.

Development Methodology

This section will talk about the methodology we have used to develop the prototype. We will talk about how we have managed the whole project later in the Evaluation section of this report.

Before we start, we will justify dedicating a whole report section to this. It might be surprising to the reader, but about 1/3 of all projects fail, based on the statistics gathered by Wrike [63]. Having no plan on how to manage the project has been linked to this low success rate [64]. Thus, it is vital to understand the project settings and select the proper methodology to manage it.

Project Settings

In this section, we will talk about the project setup, which will later influence the choice of our development methodology.

The first key factor in selecting a methodology was that only one person would be de-

veloping the project. This meant no communication overhead and potential loss of sync between different project members. Moreover, as we could not have expanded the team, it would be tough to speed up the development time if necessary. Instead, the opposite created a risk of a single point of failure in the face of only one developer.

Another significant factor was that even before starting the project's development, we had a list of concrete requirements of what we wanted to achieve and a fixed deadline for when it should be done. Moreover, we knew that there would be only one person working on the project, which meant we would not be able to expand the team.

The last factor in selecting the methodology was that we were not well acquainted with several development tools. Even with a list of requirements, it was impossible to prepare a detailed prototype plan in advance as many aspects were not known how to implement. This meant that the project had a considerable underlying risk and would require quick adjustments if something went wrong.

Selecting Methodology

The development is usually classified on the scale between Agile and Waterfall methodologies. Agile is generally considered very flexible, while Waterfall is relatively fixed and has a hard time adjusting to new environments.

Considering that going in either of two extremes would not have been helpful for us - Waterfall can not adjust to high risks present in the project, Agile does not embrace the fact that we have fixed requirements and deadlines - we have chosen an approach in the middle.

For that reason, we have selected RAD (Rapid Application Development) [65]. It would allow us to quickly get the prototype out by performing Develop Evaluate Refine loop. Additionally, the RAD methodology provided enough flexibility if the technology would have been incompatible or if we had faced any other issues during the development. Moreover, as the prototype would be out soon, we were confident we would get some tangible output by the development deadline.

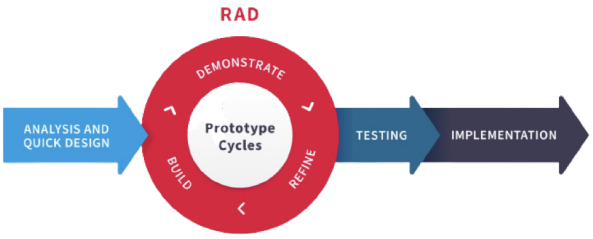


Figure 11: RAD Development Cycle [https://getbreakout.com/no-code/rapid-application-development/]

Because we have selected the RAD, we have structured our work into springs; each sprint aims to improve the previous prototype and add new functionality. Moreover, each sprint would have produced a working application. Thus, even if we would not manage to finish the last sprint, we would still have something to show. This flexibility proved to be very useful for us afterwards.

This concludes our introduction to the prototype. Now, we will explain how the development went and outline interesting problems we came across while working with it.

Sprints

Overall, we have finished three sprints and have started the 4th one. Here we will explain to the reader what we have achieved in each sprint, how the solution has evolved and which challenges we have faced. This is where we will talk about the technical details of our implementation.

The specific form of talking about the implementation sections of sprints has been selected because it offers a gradual introduction for the reader to the final prototype. We have developed more and more advanced components every next sprint. Jumping into them straight away would be disorienting for the reader. Additionally, we have hoped it would make it easier for the reader to understand our design choices and motivation.

SPRINT 1

This was the initial sprint. Its objective was to develop the initial prototype which could handle Access Control Management. We have simplified the access requirements for the first version and have not considered integration with DCNs.

Prototype 1 Objective

1. Create a project base
2. Develop an Encalve with functionality to store secret data as well as a solution to evaluating access requests which do not require oracles
3. Develop a Data Submitter CLI to support specifying basic constraints
4. Develop a Data Requestor CLI to support requesting secret data by providing proof of right for data access

Project Base

This task involves initiating the project by using Gradle to configure project modules to easier manage code. Not being a difficult task, it was still very time-consuming. The structure of the project modules is available in the diagram 12.

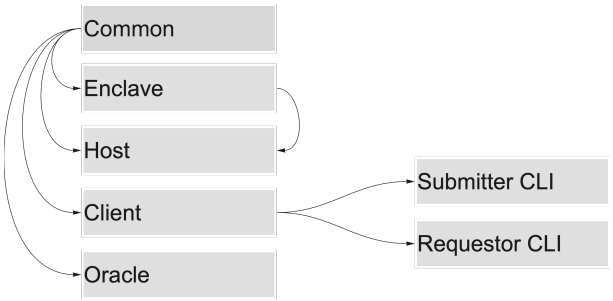


Figure 12: Diagram of project modules and their dependency

The most significant difficulty in this step was correctly configuring the Conclave SDK we have used to develop the TEE application - Enclave. Between the background research stage and implementation stage, the Conclave team has released a new version with a plugin which should have taken care of several development tasks, such as creating a server for the Enclave Host and handling communication between the Enclave and other parties. It took an effort to set it up correctly with the Gradle project management.

Conclave Host Plugin The host provides messages to the Enclave and is responsible for handling communication between the Encalve and external participants. As one can

imagine, building the host server and defining communication protocol required considerable effort and knowledge of how Client-Server communication works. Previously, we would have to create a Spring Boot [66] server and define Rest API [67] for external nodes to communicate.

The new SDK version has simplified this to a single line of code in a Gradle configuration. Yet it had other implications, meaning we had to learn how to do provided message communication, which some of our previously developed software has not been compatible with.

```
// Use the host web server for receiving and sending mail to the clients.
// This means the client needs to use the conclave-web-client library.
runtimeOnly "com.r3.conclave:conclave-web-host:$conclaveVersion"
```

Figure 13: Code snippet of creating host server with the SDK [Gradle]

Additionally, the plugin did not allow one to configure the host server. We could not specify any actions for the host to perform periodically, which has impacted how we have managed time. We will come back to this in the 3rd Sprint. **Enclave** Enclave

can be seen as a state machine. It has some internal state and only moves from one state to the other due to the external input, such as a message from an external participant.

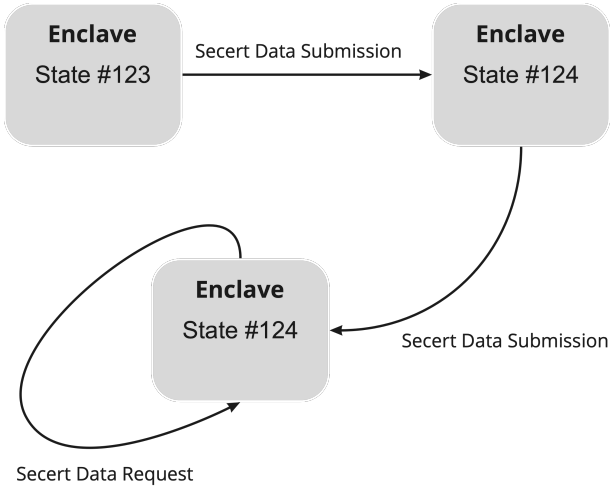


Figure 14: Diagram of an Enclave as a state machine

When developing the Enclave, we first focused on how the message communication would work and then on the database and the access control management parts. We will talk about each of them separately.

Message Handling To ensure that the enclave host can not intervene in messages and manipulate or read, nodes that communicate to the Enclave encrypt them with the SGX public key that has Enclave running in it. That way, only the SGX instance will be able to decrypt the messages, preventing the host from eavesdropping on the messages. This implies that messages are packed to bytes before being sent - serialised and then need to be deserialised inside the enclave from bytes to JVM classes.

For the first iteration of the prototype, we had two types of messages the Encalve could receive - submission of secret data and a request for secret data. Both messages would be first serialised to bytes before being sent to the Enclave and then deserialised inside the enclave to learn what type of message they were. To simplify the development and improve code readability, we have used the

Object-Oriented feature accessible in Kotlin - Polymorphism. All our messages have been derived from a single *Message* class.

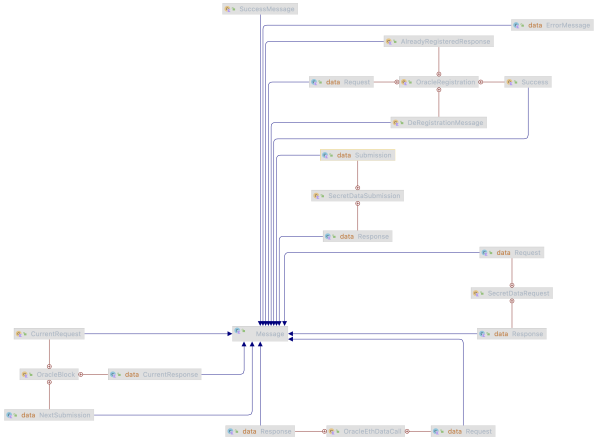


Figure 15: Message class diagram of inheritance

Instead of verifying if an array of bytes can be deserialised correctly to a specific message type, deserialised the bytes as a general message, this has allowed us to check if it is a particular type using the is construction. Then if we saw that the message corresponded to a specific kind, we would invoke a dedicated function for processing this particular message type. This provided a clear structure, where each message corresponded to a particular action inside the enclave.

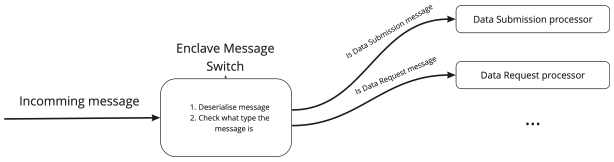


Figure 16: A diagram showing how messages are processed

Database As all of the information stored inside the Enclave is encrypted by default, we did not need to worry about secret data encryption. Instead, we primarily focused on how efficient would the data structure be for the insertion and lookup of data. As mentioned in the introduction, Deletion was out of the project's scope.

As an underpinning data structure of the database, we have selected a HashMap, notorious for its efficiency in data insertion and lookup [68]. As a lookup key, we have used a unique universally distributed identification number (UUID), which we have created for each secret data submitted to the system. Due to the ID being truly random, we have minimised the chance of collisions when populating the HashMap, thus further improving the performance [69]. This tactic additionally further anonymised the system, making any possibility of gaining knowledge about secret data from the data-id impossible.

Even though we have not done this in our project, we have noticed that to optimise the storage use inside the enclave, we could store only private keys to decrypt the data. Then, when the data requestor has proved that he has access to the data, he would have received a symmetric encryption key [cite the crypto book above], which can decrypt the secret data stored in some public location such as IPFS. The security guarantees would be the same, reducing the enclave's network bandwidth and internal size.

Access Control We have used the public key constraint as a requirement for data access. The Data Submitter would provide a list of public keys with the secret data. Any user who owns a public key can prove that he knows the corresponding private key and would be granted access to the data.

The proof of owning a public key consisted of a secret data-id as well as the signature over it. As it is possible to extract the public key from the signature [70], the En-

clave would evaluate the public key used to sign the message and then check if the key is present in the list of keys allowed to access the data. The enclave will return the secret data to the requestor if that holds. Otherwise, an error would be returned.

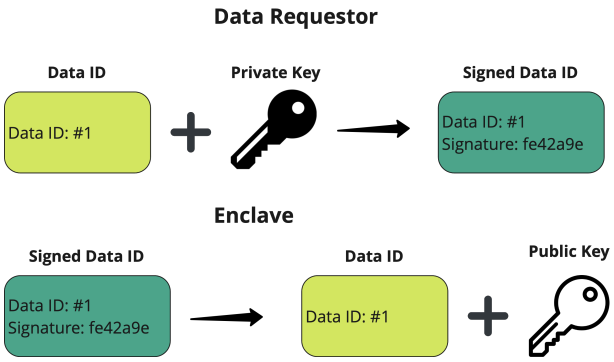


Figure 17: Diagram showing how proving and verification works

To sign and verify signatures, we have used the Web3j library. As the signature also needs to be serialised, which is not done by default in the library, we have built an extension to it to handle this use case.

One crucial problem usually presented in these public key ownership verification schemes is the possibility of a replay attack. If anybody sees valid proof, they can send it again and replay the valid message sent by the requestor. This way, they will impersonate the requestor and gain access to the secret data [71]. There are multiple solutions. For example, in Bitcoin, there is an increasing nonce that needs to be included in the request. If a request with the same nonce has already been seen, it will be considered invalid.

However, in our system, the Conclave SDK handles the replay protection, which means we do not need to include nonce for each request the requestor sends to the enclave.

Error Handling Due to the nature of the application, we had to be very careful with which internal errors to disclose or not to disclose to the external participants, such as data requestors. For example, we had decided to show the same error if the condition was partially fulfiled as when the condition was not fulfiled at all. This way, information extraction was made to be a lot harder.

Submitter CLI The CLI used by the data submitter has been built to be very simple. The user would input secret data and a list of public keys allowed to access the data. The application would then pack this into a request sent to the enclave. Upon receiving the message and successfully processing it, the Enclave would return the unique identification number of the data. This identifier is then supposed to be used by the requestor to tell the enclave which data they want to access.

Built with the PicoCLI, you can find all implementation details in the source file inside the client module.

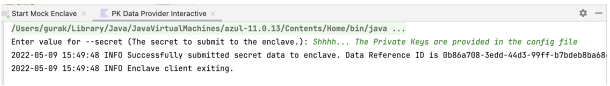


Figure 18: Submitter CLI output sample

Requestor CLI The CLI allowed the user to provide the secret data-id and their private key to the system. Then the application would create a signature over the secret data-id and send it to the Enclave. It would wait for the Enclave to evaluate the signature and then either output the secret data or show an error that has occurred.

Built with the PicoCLI, you can find all

implementation details in the source file inside the client module.

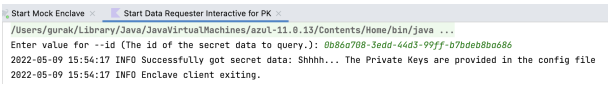


Figure 19: Requestor CLI output sample

SPRINT 2

This sprint has expanded on the initial prototype developed during the 1st sprint. The main objective for this sprint was to create access control based on the state of a DCN. We only considered we had one Oracle node for this sprint. We also assumed the oracle node was trusted. We have gone away from this assumption in the 3rd sprint.

Prototype 2 Objective

1. Conceptualise how access control will work with the DCN
2. Create an Oracle application that would accept a verification request from the Enclave and verify it on the DCN
3. Extend the Enclave to work with Oracles. It should delegate them verification, read their answer and decide if secret data can be shared with the requestor
4. Extend the Submitter CLI to support specifying the new type of constraints

Access Control based on the DCN state

As the reader might remember, the objective of this project has been to develop a data access control system integrated with a DCN. Here is actually when we come to implementing such a system.

As mentioned in the Development tool list, we have selected Ethereum as our target DCN. Ethereum has decentralised applications running in its Ethereum Virtual Machine. We will use their state as a reference point for the constraints imposed by the data submitter.

The data submitter will be able to specify which application (smart contract) they want to use as a reference point and the name of the specific getter (view) function that will return this state to an observer. In addition to this, the data submitter will be able to specify the value that the smart contract should return.

Suppose the data requestor provides correct inputs to the view function of the smart contract, which makes it return the desired value. In that case, we will consider that the requestor has passed the verification and is allowed to see the secret data.

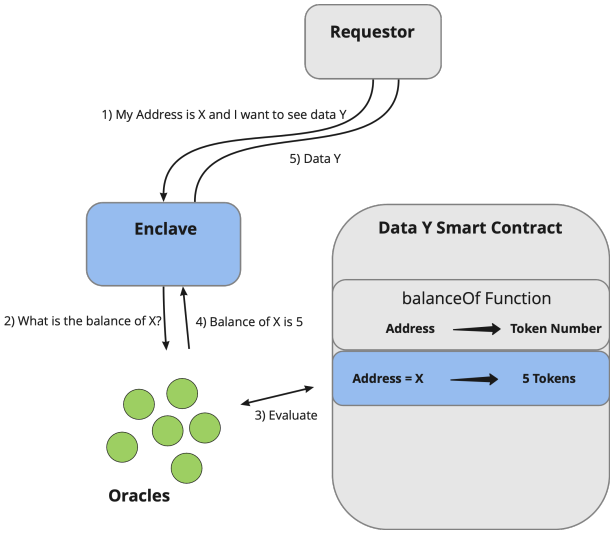


Figure 20: Diagram of how contract requests work

To better understand how this can be used in real life, let us provide the reader with an example. We have a token on Ethereum,

representing being part of a particular group of people. The data submitter wants to make some secret data accessible only to this specific group of people. Thus he would like to verify that the person requesting data access has the token. To do so, he will invoke a function balanceOf on the token smart contract with the address of the requestor. If the balance is not 0, the requestor indeed should have access. We will need to invoke a view function on the token contract with a particular input - requestor address to have this functioning, which we have specified above.

Smart Contract Parametrisation View functions in Ethereum Smart Contracts are such that they do not manipulate the state of the DCN but rather only extract information from it that they return as a result. There is a subset of view functions called pure functions.

View functions can require input parameters, which they will use to evaluate the output. For example, a view function in the example above needed the address to assess the balance. It did it by using the address as a key of a hashmap, where the value is a balance [72] of the provided address.

For this sprint, we have considered a straightforward system. In it, the data submitter needs to provide all the parameters for the view function input parameters besides the address of the requestor. The requestor, on his side, when making a request, provides their address which is a truncated public key. This design choice has been made because the data requestor needs to provide input parameters, a password, or anything else private to them. Oracles could potentially gain private information. We have only allowed addresses as they could be verified in Zero-Knowledge. It is worth pointing out that we planned later to add other input parameters, which we will talk about in **Sprint 4**.

	Data Submitter	Oracle	Data Requestor
Data Provided	Secret Data	None	Public Address
Who can access	Permitted data requestor		Oracles
Information leakage	Allowed		Undesired, should be minimized

Figure 21: Table depicting personal and private data present in the system

It should be noted. Anyone can create a request with an input address that they do not own but belongs to a well known token holder. Such checks are not done in Ethereum Smart Contracts as they are usually done by the EVM [73]. We still need to do them as we do not use the EVM. We retain the system developed in the first sprint, where the requestor still needs to sign a message that the Enclave then validates. **Oracle**

Oracle was required to bridge the gap between the TEE and the DCN. Both systems are considered trusted. However, it is hard to get information between them. Oracles are a ubiquitous approach to do so. If we have enough independent oracle nodes, as long as half of the oracles do not collude, the information transfer between the two systems is secure.

As mentioned before, the objective of this sprint has been to create a single oracle node that could transfer information from DCN - Ethereum to our Enclave application.

Evaluation of View Functions We have developed the Main Oracle class with a range of subsystems. This allowed it to handle multiple concurrent requests following the mi-

croservice architecture pattern [74]. Now we will talk about the *EthOracleService*. We have used the Web3j library to interact with the Ethereum blockchain. It allowed us to establish a connection with an Ethereum RPC, which we could query about the state of the Ethereum.

To do such queries, we received the following information from the Enclave, shown in 22.

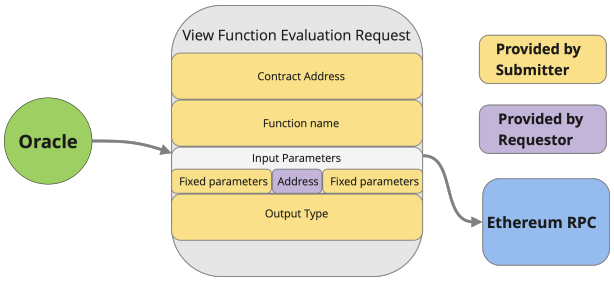


Figure 22: The diagram of what Oracle recieves from the Subitter and the Requestor

We then created a function call request following Ethereum JSON-RPC API[75], waiting for the result of the execution of the function was successful and forwarded it back to the Enclave. **Enclave**

Oracle Registration As Enclave would need to verify the data requests with Oracles, it required to have a registry of current active Oracles to forward them a request. That is why we have added this step, where each new Oracle needs to first sign up with the Encalve to receive the requests from the Enclave later. The signup process consisted of the Oracle sending a Hello message and the Encalve replying with a confirmation that the registration of the oracle has been completed successfully.

```kotlin
object OracleRegistration {
    /**
     * Message sent from the Oracle to the Host to register as an Oracle.
     */
    @Serializable
    data class Request(val key: String) : Message()

    /**
     * Response after a successful Oracle registration.
     */
    @Serializable
    object Success: Message()
```

Figure 23: Oracle registration message class code snippet [Kotlin]

Access Management Improvement In our previous iteration, the data access verification has been synchronous in the enclave. We did not wait for any external party or system to provide additional data to evaluate the request. Now, we had to change this to an asynchronous operation. As the reader might remember, our Enclave is a state machine which does not support asynchronous functions. This meant we had to develop a system to overcome this limitation.

The way we approached this was to create a database of all pending verification requests which required information from the Oracle. Each such request would be assigned a unique ID and stored in the HashMap. Then, when an Oracle came back with the result, we would remove the request from the pending queue and evaluate it with all available information.

```kotlin
/** Map between the Topic of the request sent to Oracle to the UUID of the */
private val pendingDataRequests : MutableMap<UUID, PendingDataRequest> = mutableMapOf()
```

Figure 24: Code snippet of a pending data request structure [Kotlin]

If the verification passed and the constraints provided by the Data Submitter were fulfilled, we would have sent the Data Requestor the data, otherwise an error message.

We have additionally allowed the condition provided by the Data Submitter to

be of different types - either Public-Key or Ethereum Smart Contract State constraint. The way we have written our code made it very simple for the additional constraint types to be introduced by adding a specific function to check them.

Ethereum type serialisation It must be told to the reader that Ethereum has a coding language called Solidity[76], which is used to write smart contracts and compiles to byte code which is executed in the EVM. Solidity is very similar to C, with the same syntax and an option to write assembly. It has very rudimental classes and interfaces, but it can hardly be considered an Object-Oriented language.

One unordinary thing about Solidity is the number of types. Overall, there are over 30 types, most of which are Integers of different byte lengths. There are int8, int16, ... int256 and unsigned versions of them. Providing the programmers with additional versatility has posed a big challenge for us.

As one might remember, when defining constraints, the Data Submitter needs to specify the contract and the function to execute and the expected data. These values are expressed in the Web3j type system, which has a particular class implemented for each Solidity type. And as the Data Submitter first needs to serialise their submission, we would need to serialise all of these classes, which the Web3j library does not handle.

This has been only part of the challenge. This is because besides having to serialise, we also had to deserialise. And we would have to deserialise to generic types first, then move to more concrete types. This would require us to register all the serialisers under the polymorphic. It took some time and effort to finally design such serialisers and a serialisation plugin that the whole system would work. Our most significant achievement is that we only had three serialises instead of 30+ as we have successfully used the polymorphism between all the integer types.

However, this was only part of the story. Additionally, there has been a Reference class in the Web3j. Its instances represented the Solidity type. For some reason, the developers have not made the solidity types as objects or enumerate but as a class which accepts the Web3j implementation of solidity type as a type parameter.

We had to find a way to serialise this as well as the submitter had to provide the signature of the function output, which consisted of the Reference instances. We could not have just serialised this as the java type erosion [77] lost the information for what the Reference held. We ended up serialising this separately by representing the referenced data type as a string and reverting it from a String using a class lookup.

This is an insecure implementation as an adversary can substitute the class name with something malicious and cause remote code execution inside the Oracle [78]. However, as this was a prototype, we accepted the risk.

Data Submitter CLI

he data submitter CLI had to be changed to support new type constraints. We have created a new CLI for the Ethereum State constraint so that we could create either Ethereum State or Public Key constraint.

With the serialisation problem solved, it was straightforward to modify. We have hardcoded the constraint to keep the appli-

cation simple and quick to test. It would require the requestor to have a specific ERC20 token on Ethereum to disclose the information. This followed the example we gave when describing how the Access Control would work with the DCN.

Data Requestor CLI

As in the current version of the prototype, we only require the address of the Data Requestor, we were able to manually calculate it in the Encalve from the signature, thus not needing to change the Data Requestor CLI.

SPRINT 3

This sprint was focused on fixing some simplifications we have made to have a more resilient system. The biggest challenge we have faced was adding time notion to the Enclave, yet we will cover this in more detail inside this section.

Prototype 3 Objective

1. Extend the supported versatility of the Access Control System Ethereum State constraints
2. Support multiple oracles and have a consensus mechanism for them

Extending the constraint versatility

By now, we have designed the following constraint system. The Data Submitter could specify the smart contract and contracts function that needs to be evaluated and the expected value. It also supported only one input from the Data Requestor - the requestor's Ethereum address.

This was insufficient as more complex constraint systems depending on multiple contracts could be possible as well as checking simple equality might not be enough for all use cases. Thus we have focused on extending the functionality.

Multi-contract constraint Our objective was to allow the data submitter to provide more complex constraints, which would depend on multiple contracts. For example, instead of owning one token, we could require that the data requestor owns two different types of tokens or owns either of two tokens. The first would require the AND of both conditions. The latter would require an OR of both conditions.

```kotlin
@Suppress( ...names: "Unused")
@Serializable
data class EthMultiContractDiscloseCondition(
    val cnf: AndList<OrList<EthContractDiscloseCondition>>
)
```

Figure 25: Code snippet of a CNF used to specify constraints [Kotlin]

To make our solution as versatile as possible, we used a Conjunctive Normal Form (CNF) [79] to be able to specify any possible combination of conditions possible. This way, the data submitter has access to the highest versatility of defining constraints, as with a CNF one can define any boolean function - a truth table of whether to allow access or not.

We have made a two-level CNF, which has been unnecessary yet proved more developer-friendly. The developer had a CNF when specifying the multi-contract constraints as well as inside the contract constraint - which value should the contract evaluate. This double level could have been reduced to a single level of multi-contract

constraints with duplicated contracts occurring with different values. Yet, as mentioned above, it would be troublesome to write such constraints.

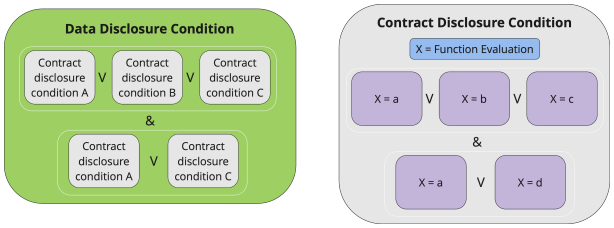


Figure 26: Diagram of the CNF layer architecture

Additional comparison functions We have also expanded the set of comparison tools of smart contract output from simple equality to greater, less than, unequal and all derived from them. This, with CNF constraints for contract evaluation, would now be enough to construct a constraint evaluating if the contract value is between two predefined numbers and more. A potential application could be requiring that the data requestor owned at least 100 tokens, thus creating a census for less invested individuals to access the secret data.

Function signature expansion Among the changes we introduced to the constraint, the system was how the smart contract function input was handled. Initially, in the second sprint, we have only supported the functions that first require the requestor's address and then any arbitrary parameters that the data submitter should provide.

Realising this is a very limiting approach, we have changed it by requiring the data submitter to provide function input data as a list, among other contract constraint information. In this list, several positions were null. The requestor's address replaced these, and we got the list of all function inputs in the correct order.

A nice part about this approach was that it allowed us to have multiple instances of the address in the function input data, which the requestor's address would substitute all. Additionally, we have saved much space as we did not require any position list to store positions of the provided input parameters in the final function signature.

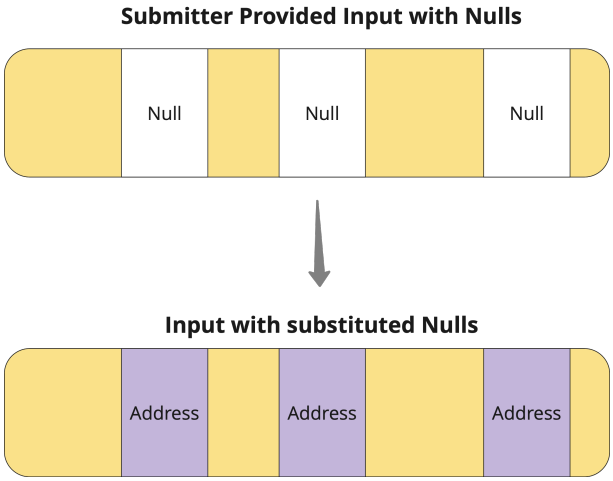


Figure 27: Diagram showing how the substitution works

Oracles and Consensus So far, we only supported one oracle, whom we trusted not to betray us and work correctly. That is not to provide inaccurate data when asked to evaluate a specific function in Ethereum. However, this might not be the case, and as we are building the decentralised solution, we can not have a single point of trust.

An example of a potential attack could be that the oracle owner decides to find the value of secret data. He requests the data from the Enclave, and then the Oracle node, which is malicious, gets the request. Let's say the request asks if the address owner has tokens. The malicious oracle will say yes,

even though, in reality, this is not the case and thus will gain access to the secret information, tricking the system.

To prevent this from happening, we will have multiple oracles, of which the majority needs to agree with the answer to arrive at a consensus.

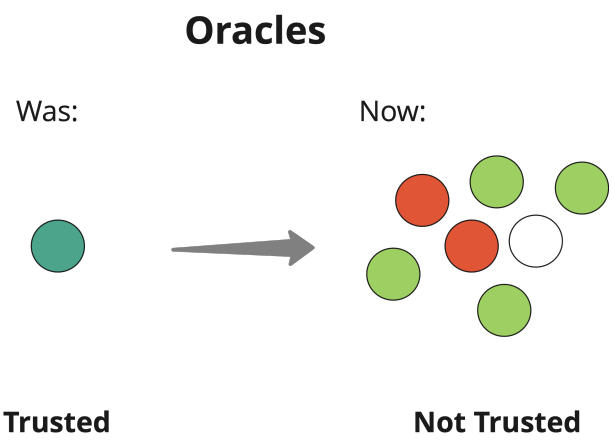


Figure 28: Diagram of moving from one trusted to multiple untrusted oracles

Multiple Oracles As one remembers, we have built an oracle registry inside the Enclave in sprint two. Their every oracle would be added once it passes registration with the enclave.

Then, whenever a data request would come that required the evaluation of the smart contracts on Ethereum, the Enclave would send an information request out to the oracles that have registered with it. Once it has received the information from them, it would aggregate the information and decide whether to share the information with the data requestor or not.

Consensus To aggregate the information, we have created a class that would store all oracle answers for a particular data request. The oracle could potentially provide a new reply, thus changing the response recorded in the class instance.

```
data class OracleSubmissions (
    val notifiedOracles: Int,
    val submissions: MutableMap<String, List<Type<Any>>?>
)
```

Figure 29: Code snippet of a class definition which handles oracle submissions [Kotlin]

This way, we would aggregate oracle responses, and if enough Oracles have responded, we will finish waiting for other oracles to submit their result and evaluate the response. To know when it would be enough to stop waiting, we would see if any option has received more than 50% of the support of all notified oracles. If so, we considered that the majority voted in favour of the option and that we could accept the result as correct. Next, we would do the regular check if the result satisfied the constraints and if so, we would return the data.

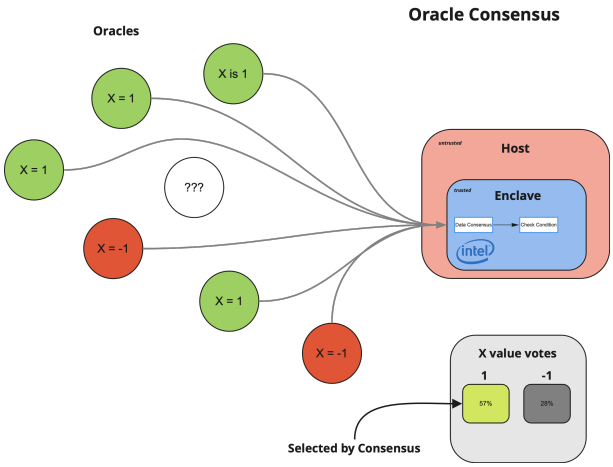


Figure 30: Diagram of oracle consensus

It is important to note here that we can not preemptively say that if any option is more popular than the rest, we are allowed to select it. This is because the host of the Enclave has adversarial power to prevent messages from honest oracles from coming through. This way, the enclave host who

owns 10% of the oracles can block the messages from the rest of the oracles. Thus, out of 10% of oracles who voted, 100% would be in favour. Our system requires that out of 100% notified, more than 50% are in favour of one option. The 50% is considered a firm security guarantee for a decentralised protocol [80].

To further secure our system, we could have oracles send messages to other oracles and then relay them to the Enclave. This would be used to avoid the Enclave censoring messages from particular oracles. We will address this in the future work section.

However, there is another big issue in the system proposed that we have not mentioned yet. In the current system, we should wait for enough Oracles to answer. However, as with any real-life system, some oracles can potentially go down or fail to receive our request. But we still would be waiting for their answer internally, thus blocking the path forward.

Our solution is to introduce time to the enclave and define how much waiting time is okay and after how long we need to cancel waiting for oracles.

```
if (voteCount < request.oracleSubmissions.notifiedOracles * MIN_ORACLE_TURNOUT)
    throw IllegalStateException("Not sufficient amount of Oracles has responded to the query.")
```

Figure 31: Code snippet of evaluating if consensus has been reached [Kotlin]

Epoch time In the background section, when discussing the TEE, we have mentioned that it works like a state machine. And there is no time in a state machine. However, here we will try to bring time to the Enclave by using external Oracles to help us with this.

Before we start, there is another reason we need time inside the Enclave. The Ethereum state changes after every new block. Every 15 seconds, the state of the Ethereum evolves forward and represents all the effects which transactions have made in that time. This implies that in one block, a requestor might have a token. In the next, he might sell it, and oracles will not be able to agree on the state of the Ethereum access condition. This is another reason why we need the notion of time for oracles to know which block to reference.

There were no common but a few potential ways we could achieve having time in the Enclave:

1. The first was to trust the host and make oracles verify that the time was correct. But as we could not extend the host due to SDK plugin limitation and this being insecure, we have disregarded this option.

2. Another option was to use VDFs, functions hard to compute and easy to verify[81]. This is the approach used in Solana DCN [82]. The host would compute such a function and let the Enclave verify it. This would prevent the host from speeding up the time inside the Enclave, which would be sufficient for our use case. However, as mentioned before, we could not modify the host behaviour, and we also desired to achieve stronger assumptions on time.

3. Finally, we could have used Oracles to bet on time in the same way as the bet on the smart contract evaluation result. This was chosen as a solution.

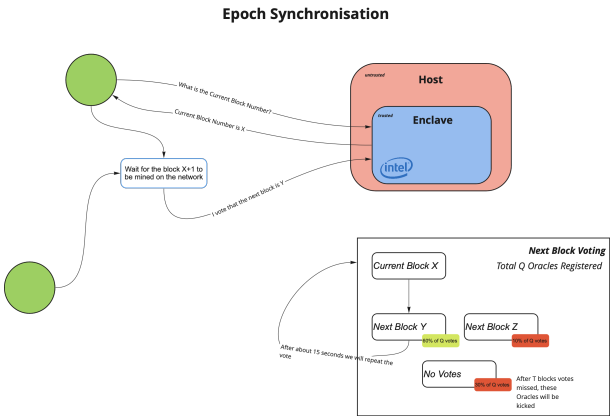


Figure 32: Diagram of how oracles provide time into the Enclave

This worked as follows. Every fixed period of time, oracles registered inside the Enclave would send the same random number. Once more than 50% of registered oracles have sent this random number, Enclave updates its state of time to this random number. Then the oracles request the current new random number and send the Enclave the following random number, repeating the process over and over. As long as more than 50% of registered oracles have agreed on a random number, we consider one epoch to pass. Thus we get a notion of time as epochs.

We decided to use the Ethereum block hash as the random number. This is for the following reasons. First, the Ethereum block hash can not be predicted in advance, meaning Oracles can not commit the block hash more often than 1 Ethereum block time (15 seconds). This is a lower bound on how fast the time can go inside the Enclave. Second, as mentioned before, the result of smart contract evaluation changes every block. That is why it is vital to know up to which block the oracle should be evaluating the smart contracts to. Finally, the block hash is easily findable for any oracle as they are already integrated with the Ethereum chain.

As a result, we had a system where oracles would update the current block hash inside the enclave every 15 seconds. And any request still present inside the Enclave after 15 seconds would have been cancelled and an error message sent to the requestor asking them to request the data again. This provided a response deadline as well, making the service more usable.

Oracle deregistration As mentioned above, to move to the next block hash, it is required that 50% of the oracles should vote on it. But as we know, oracles can go down, and the system will get blocked unless we purge them after a certain period of inactivity.

Therefore, every time we cancel a smart contract evaluation request and see that an oracle has not voted for it, we add one penalty point to the oracle. The OracleManager module of the Enclave handles this. If Oracle receives more than a certain amount of penalties, it will be deregistered due to inactivity. This way, we ensure the system stays up and functioning. Current inactivity is set to 1000 missed queries and cools down if the Oracle starts to fulfil them.

SPRINT 4

The last sprint we initiated. We attempted to perform one more interaction on the prototype, yet we have failed to make progress in it due to external circumstances mentioned in the Evaluation section.

Prototype 4 Objective

1. Add support for data requestor private inputs, which would be verified by oracles using Zero-Knowledge proofs.

2. Build a Web UI for the data requestor

As we have not made enough progress in either of these objectives, we will keep this discussion to the Future work section and move forward to evaluating the project.

Project Evaluation

In this portion of the report, we will review the project from a critical point of view. In particular, we will look at our background research, technical solution, and project management skills. The ambition of this section is to identify what was done well and what could be improved.

Background Research

Overall, the background research was done on a high level. We have successfully identified technology that could be used to achieve the project ambition. This can be seen in the fact that we saw very few issues during the implementation phase due to technical limitations of our technology stack.

However, we must note that we have perhaps focused too much on the security and decentralisation components during the background research. An insufficient amount of time has been dedicated to researching and learning about the selected software building tooling, such as JVM we have used in the report. For example, the implementation phase was delayed by 16 development hours because we had to learn how polymorphic serialisation works.

Implementation

In this part, we will evaluate our software implementation. We will use a list of our original requirements, assumptions made, and potential security vulnerabilities to assess the system and the prototype we have developed.

Requirement Review

Overall the project has successfully achieved most of the initially proposed requirements. The system developed was in line with how we have conceptualised PSSC.

Functional Requirements We have achieved 8 out of 9requirements. It would be easy to adjust the system further to achieve the final requirement, with the easily adjustable prototype fulfilling the last requirement.

F1.1 - Partially achieved, the data submitter needs to run a specific application on their machine F1.2 - Not, we currently only support strings as private data, yet that could be easily extended F1.3 - Achieved F1.4 - Achieved F1.5 - Achieved F2.1 - Achieved, several attack vectors will be Functional Requirements list explored in the Security section F2.2 - Achieved, several attack vectors will be explored in the Security section F2.3 - Achieved, several attack vectors will be explored in the Security section F2.4 - Achieved

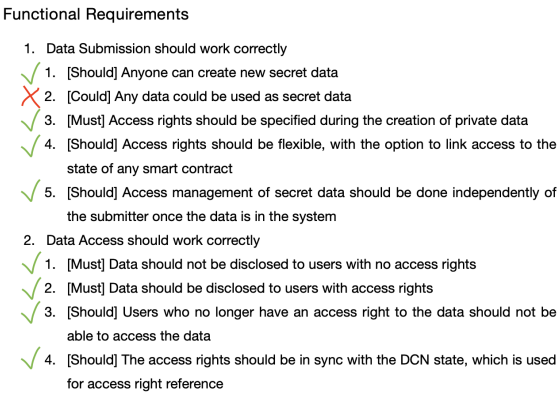


Figure 33: Reviewed functional requirements list

Non-Functional Requirements We have achieved 7 out of 8 requirements. We will review how to accomplish the last requirement in the future work section.

NF1 - Achieved. Processing time is capped at 15 seconds. NF2 - Achieved NF3 - Achieved, any user computer can run a program to access the data. We have tested with macOS and Linux. NF4 - Achieved, any user computer can run a program to access the data. We have tested with macOS and Linux. NF5 - Partially achieved. The user must still know what public and private keys are. NF6 - Partially achieved. See the security section for more details NF7 - Not achieved, as currently, the enclave size will grow with the amount of data added to it. Meaning there is a limit to the amount of data stored. NF8 - Achieved NF9 - Achieved

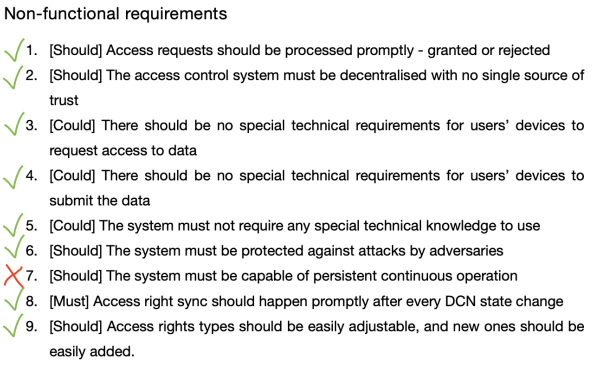


Figure 34: Reviewed non-functional requirements list

Assumption Review

Will developing this project, we have made several assumptions. We include them in the evaluation to show the current limitations of the project.

TEE Security In this project, we have selected the SGX as our TEE implementation. However, this requires us to trust Intel to verify that the SGX CPU is authentic. [83] This introduces a single point of trust in our system. For this project, we have assumed Intel to be trusted.

This assumption has been made as the TEE, in general, is secure, and our system can be reimplemented with another TEE model. The fact that we trust Intel for our prototype does not influence the achievement of the project but is purely a limitation of the prototype.

Sharing Data ID and Conditions We have assumed that the Data Submitter will share with the Data Requestors the ID of the secret data as well as the conditions required to access it.

The assumption has been made to simplify the system and provide versatility for

the data provider to communicate the data-id in any desired way.

Incentive Structure For this project, we considered there is an incentive structure present. This means the nodes are incentivised to behave correctly and are punished for malicious behaviour. This is required to add security to the project and prevent Denial of Service attacks by Oracles and the Encalve Host. We will cover the potential implementation of such an incentive structure in the future work section.

Active Community Surveillance This assumption means that we expect out of project scope communication of Oracle hosts and data requestors who monitor how often the enclave denies their messages and, if this happens too often, will confront the enclave host and stop using it.

Security Review

This section will provide an overall evaluation of project security. We will not evaluate the security of the tools used for development but rather the security of the protocol we have developed.

Overall, if we accept the assumptions that 50% of Oracles are honest and that the host is incentivised not to shut down the Enclave, the system is secure.

Data Availability Attack The host can permanently shut down the enclave and remove all the data. This can be countered by the incentive structure with the host having to stake some money to be accepted as an Enclave. Alternatively, we could have a system of multiple Enclaves that all store the same data. This solution requires further research.

Secret Data Disclosure Attack In the system we have developed, the only way for the Enclave Host or any other party to learn the secret data is to get control over the 50% of the oracles, which is an acceptable level of tolerance for decentralised systems. We could potentially increase this even further, but this could impact the resistance to the Denial of Service attack. This is because unless 50% of oracles provide the same data response, Enclave will not share the secret data. This means we only need to consider how easy is it to perform denial of service.

DoS Attack The simplest way to DoS [84] our system is for the Host to shut down the Enclave. However, this could be mitigated by making the host stake some money, as discussed in previous attacks.

Another way is for more than 50% of oracles to go down. Yet again, the host can simply stop their messages from going through to the Enclave or someone who gets access to 50% of oracles can do this by shutting them suddenly down. Both attacks are within our tolerance and are considered acceptable.

This creates an interesting counter-host mechanism. Suppose the Enclave users notice that the host is acting maliciously. In that case, the honest oracle majority can block the Enclave by stopping replying to its messages, thus ensuring no one will get access to the secret data.

Project Management

This part will evaluate our project management technic, which issues we have faced

along the way, and how we overcame them.

Timeline

Here we will talk about how the project was paced and compare our initial timeline assumptions to the actual timeline.

We have planned the project to be done in stages:

1. Project specification
2. Background research
3. Specification refinement
4. Implementation
5. Project review

Additionally, we had four weeks of spare time if the project went off schedule, and the report is due to be compleated by early April.

In the diagram 35 We have provided the original Gant Chart next to the actual Gant Chart of how the project has gone. We have gone off schedule with the background research. However, the most considerable delay has happened with the Implementation section, delaying the project by almost a month. We will cover the reasons for this in the next section.

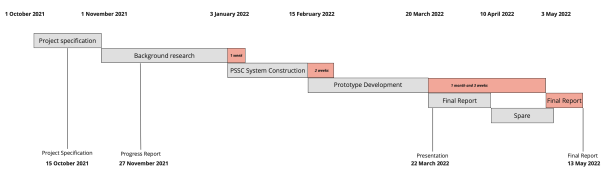


Figure 35: Diagram showing anticipated (grey) and actual work schedules (red)

Management Techniques

We have used the following techniques to manage the project better and achieve better results:

1. Weekly recaps on what has been achieved in the project
2. Weekly plans on what is to be performed in the project for the following week
3. Recurring presentations of the project progress to the Project Supervisor as well as the External Supervisor
4. Checkpoint in the calendar which followed the timeline above
5. Project Diary to take notes on essential topics and add papers to cite in the report

The first four techniques were instrumental in being on track with the project and clearly understanding where to go next. The 1st and 5th helped write the report, as most discussion topics have already been recorded in the diary.

Issues

During the project, we have found the following issues that have delayed the project and impacted the project's scope.

Issue	Why not anticipated	Impact
Learning how ZKP and other cryptography techniques is hard	Have not anticipated how hard this is	1 week delay
Misconception of how developed the cryptography techniques, including ZKP	Low space awareness before starting the project	Scope reduction
JVM complications cause by type serialisation	Not sufficient knowledge of JVM before the start of the project	Half week delay
Time conflict with other university modules	Too self confident	Unmesurable
Prototype protocol had security complications	Unknown issues	1 week delay
Eastern European war	Unpredictable	1 month delay

Figure 36: Table listing project issues and their impact

Overall, we have controlled most of the issues. We have anticipated delays and thus have scheduled the project to be completed

one month ahead of schedule. However, the Russian-Ukrainian war was an exceptional circumstance. Being directly affected, we had to request an extension and scale back on deliverables, not managing to finish the 4th Sprint. We must say that the development, structured in the form of sprints, allowed us to have a working prototype despite this colossal delay, which proved the right choice of project management strategy for our implementation.

Summary

Overall, the project has been a success. We have developed a secure and decentralised system that achieved 15 out of 17 requirements.

The project has made several assumptions to focus development time on the core concept. None of the premises is critical for the protocol developed.

The development has been put off track due to the uncontrollable issue of the Russian-Ukrainian war. However, our development methodology allowed us to still have a working prototype despite this. We will mention what we have not achieved in the Future work portion.

Future Work

This report will discuss how the project can be further expanded, improved or changed. We will explore the following improvements: usability, maintainability, security and privacy.

Usability

Here we talk about potential improvements to the usability of our system. This section has been out of scope for our project as it does not bring any research value, yet it is a crucial component of any system.

User Interface Developing a web-based user interface for both submitter and requestor would vastly improve the usability of our product's usability and remove the necessity of having a specific application installed on their machine.

QR codes This would allow us to quickly share the data-id in the form of QR codes that the data requestor could scan and automatically redirect him to the website or application where they can access the secret data. This would be useful as it addresses the issue of how data Id is shared, which has been originally out of the scope of the project.

Maintainability

This section focuses on the most needed improvements to the system, which are required to be scalable and long-living.

Data Storage Currently, all the data is stored inside the RAM in the Enclave Host machine. By default, the SGX CPU supports a maximum of 128MB [85], which would not be sufficient to store much memory.

The one possible solution is to only store encryption keys inside the Enclave and the encrypted data outside the Enclave in the host provided database. This would still offer the same security guarantees as before but

would support up to 40,000,000 secret data stored with a key size of 256 bits.

Data Submission Another way for the system to scale is to remove the data storage aspect from the Enclave application scope. Instead, the data submitter would encrypt the data using a key and store the data on a Decentralised File Storage system like Filecoin. The Enclave would be then used to store the key used for decrypting the data and sharing it with the correct participants.

The advantage of this system would be that we could leverage external systems for data storage and stop thinking about the incentive structure. Additionally, the data submitter would be free to choose any type of data storage, especially if the file is in size of Terra-bytes.

Security

Here we will look at some future work that could make our system even more secure and protect the secret data from being illegally accessed.

Data Revocation As we have mentioned, the data revocation has been out of the project's scope. It is tough to make sure one has correctly revoked the data. Yet, this is necessary, especially if someone accidentally submits the wrong private data to our system, which anyone would be able to access. That is why we need to find a solution to prevent this.

One way is to use community voting to achieve this [86]. If incorrect data is submitted, you could request to delete it, and if no one opposes, this will be done. The one thing is that it is tough to evaluate such a mechanism and guarantee its security.

Another way is to implement a system where the data submitter can request to delete the data at any point in time. This is also not secure as the Enclave host can prevent such messages from reaching the Enclave, or the host can store the Enclave state before the deletion and then revert to it.

Data revocation has been a highly complex topic to solve. That is why we have not included it in the project. However, it can be done as a part of future work.

Incentive Structure The ambition is to incentivise both the Oracle and the Host to act honestly. This is usually achieved by a system of stakes slashed in case someone acts maliciously [87]. Incentive Structure has also been out of the scope of the project.

Additionally, this can be extended to a system where the Submitters and Requestors need to pay to compensate for the Oracles and Enclave host work. This would be necessary if we deploy our system as a production-ready service.

Enclave Network Currently, all secret data is stored on a single TEE machine. This means that the data can get lost if this machine crashes or the host decides to delete all data. We can address this issue by having a set of Oracles that share a state. This would mean the secret data would be copied among the Oracles, making the system a lot more resilient.

Privacy

The final component is privacy. Here, the objective is to protect the private data that

the data requestor submits to Enclave to gain access to the secret data.

ZKP As mentioned before, currently, we leak the public address of the data requestor to the Oracles. Such information leakage is tolerable, as the public address is a Zero-Knowledge Proof of a real secret - the private key. However, if the data requestor would provide confidential data, this would be intolerable for Oracle interceptions. Oracles could deanonymise the data requestor or even gain illegal access to the secret data.

Our way to protect against it is to require the Data Requestor only to provide ZKP of the Confidential data, which the Oracles would verify with zero information leakage. This has been intended for the 4th sprint, which we did not manage to complete. This would also require developing a standard for smart contracts on Ethereum to accept ZKP for verification and tooling for the Oracles to run their ZKP verification algorithms. This is a very advanced yet, promising area of research.

Secret Functions Another future work area we would like to mention is Secret Functions. As mentioned in the background section, their work is similar to a TEE, yet they do not need to trust the hardware but rather depend solely on mathematics. Exploring the tooling and building the same solution using them would be an exciting endeavour.

Conclusion

In the project, we have looked into the issue of privacy in decentralised computational networks (DCN), such as Ethereum. We have developed a data access control system that can be integrated with any DCN. We refer to such a system as PSSC. The system stores secret data and returns it only to someone who has the correct access rights.

To develop such a system, we have learned state of the art cryptography and security solutions and attended conferences and extra classes. This provided us with enough background to tackle the pressing issues of enabling privacy in decentralised settings.

The project's main deliverable has been the system and the prototype of PSSC. We have tested the prototype on a real-life use case of defining access rights in terms of token ownership on Ethereum.

Project Achievements

The project's ambition has been to explore and contribute to a presently innovative research topic of privacy in decentralised settings. Our project has focused on using currently available tooling to achieve decentralised private data storage. With no current alternative solutions, our project can be considered the first one of its kind.

We will now list other research and implementation contributions of this project.

Research Contributions

1. Solutions on how time can be implemented inside a state machine such as TEE
2. Evaluate ways how data can be revoked from the TEE
3. Solution on how to transfer information trustlessly into the TEE

4. The proposition of how ZKP can be used in an Access Control System
5. Evaluation of how ZKP can be integrated with and improve the security of TEE

Implementation Contribution

1. Decentralised application to store private data and disclose it to only specific participants based on the state of the DCN
2. The new way how TEE can be used in the DCN settings
3. Implementation of Oracle management in a TEE application
4. TEE application with trustless time

Project Limitations

We have already extensively listed our project limitations in the evaluation section of this report with ways to overcome them in the future work portion of the report. That is why we will note down only the most significant limitations.

Protocol Limitations

1. TEE requires hardware security, which can be an insufficient security guarantee
2. It is currently impossible to irreversibly delete data contributed to the TEE
3. The Enclave leaks information about secret data requestors to the Oracles
4. The data is stored on one TEE and, thus, can be corrupted or lost

Implementation Limitations

1. Not optimised for real-life deployment
2. There is currently no incentive structure for nodes to take part in the project
3. The user interface is unfriendly

Authors' Assessment of the Project

This project can be easily considered the culmination of my three years of studying at Warwick University. Both privacy and decentralisation were a long interest for mine, and combining them into one piece of work felt incredible.

I must say, the project has been very intellectually challenging. Over the course of it, I learned a lot more about the state of cryptography and how decentralised systems work. The knowledge available has been very sparse and so rapidly evolving that new developments appeared over the course of the project.

I extensively applied knowledge from different university courses during the project, such as Cyber Security, Functional Programming and Logic and Verification.

I find that this project can be helpful for both understanding the current state of the privacy computation techniques and for building real-world privacy preservation applications. I have provided all required future work to make it a reality.

Being the first project I know of which provides the desired functionality combined with research into other crucial areas, I firmly believe that the project is an achievement I

can be proud of, especially when considering the political instability that has impacted me and caused project delays and scope reductions.

Acknowledgements

1. To Dr Charilaos Efthymiou, who has helped with structuring the project work and keeping the project on track.

2. To the Aragon team for providing support and education in cryptography. Especially thank you to Arnau Cube for extensive information on ZKP and Alex Campa for helping manage the project.

3. To Dr Jose Luis Munoz for teaching how Plonk ZKP works and helping with the cryptography part of the project

4. To Dr Vincenzo Iovino for providing teaching cryptography, especially MPC and Homomorphic Encryption.

5. To the R3 team for providing information and guidance on how to use the Conclave, the TEE SDK. Especially thank you to Shams Asari, with whom we have conceptualised the time management inside the Enclave.

6. To the Obscuro team for providing details about the SGX and state of TEE in Web3 and brainstorming how the TEE can be used for private state smart contracts. Especially thank you to Cais Manai and James Carlyle.

7. To the Zero Knowledge Podcast team for organising the ZKSummit 7 provided much context about current developments in ZKP.

8. To Elizaveta Trushevskaia for helping with editing the text.

References

1 Elliott, E. (2019). A Brief History of Decentralized Computing. [online] The Challenge. Available at: https://medium.com/the-challenge/a-brief-history-of-decentralized-computing-d0d665783bcf [Accessed 11 May 2022].Chaum, D.L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, 24(2), pp.84–90. doi:10.1145/358549.358563.

2 Chaum, D.L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. Communications of the ACM, 24(2), pp.84–90. doi:10.1145/358549.358563.

3 Torproject.org. (2019). The Tor Project — Privacy Freedom Online. [online] Available at: https://www.torproject.org/.

4 Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. [online] Available at: https://www.ussc.gov/sites/default/files/pdf/training/annual-national-training-seminar /2018/Emerging_Tech_Bitcoin_Crypto.pdf.

5 Cerf, V. and Kahn, R. (1974). A Protocol for Packet Network Intercommunication. IEEE Transactions on

Communications, 22(5), pp.637–648. doi:10.1109/tcom.1974.1092259.

6 Tribis, Youness El Bouchti, Abdelali Bouayad, Houssine. (2018). Supply Chain Management based on Blockchain: A Systematic Mapping Study. MATEC Web of Conferences. 200. 00020. 10.1051/matecconf/201820000020.

7 Mehboob Khan, K., Arshad, J. and Khan, M. (n.d.). Secure Digital Voting System based on Blockchain Technology. [online] Available at: https://core.ac.uk/download/pdf/155779036.pdf.

8 Buterin, V. (2013). Ethereum Whitepaper. [online] ethereum.org. Available at: https://ethereum.org/en/whitepaper/.

9 cypherpunks-core.github.io. (n.d.). Chapter 13: The Ethereum Virtual Machine · GitBook. [online] Available at: https://cypherpunks-core.github.io/ethereumbook/13evm.html [Accessed 11 May 2022].

10 ethereum.org. (n.d.). Smart contracts. [online] Available at: https://ethereum.org/en/smart-contracts/.

11 Ethereum (n.d.). Decentralized finance (DeFi). [online] ethereum.org. Available at: https://ethereum.org/en/defi/.

12 DeFi Market Cap. (n.d.). DeFi Market Cap — Top DeFi Tokens by Market Capitalization. [online] Available at: https://defimarketcap.io/: :text=Total

13 Bandyopadhyay, T. (2021). Bank nationalisation: 52 years and ticking. Business Standard India. [online] 18 Jul. Available at: https://www.business-standard.com/article/opinion/bank-nationalisation-52-years-and-ticking-121071800755_1.html: :text=Fourteen

14 www.moneyland.ch. (n.d.). Hidden Order. [online] Available at: https://www.moneyland.ch/en/hidden-order-definition [Accessed 11 May 2022].

15 Investopedia. (2019). Dark Pool. [online] Available at: https://www.investopedia.com/terms/d/dark-pool.asp.

16 Shipp, A. (2021). Privacy Without DeFi Is Boring, DeFi Without Privacy Is Predatory. [online] www.coindesk.com. Available at: https://www.coindesk.com/markets/2021/08/16/privacy-without-defi-is-boring-defi-without-privacy-is-predatory/ [Accessed 11 May 2022].

17 Reiff, N. (2021). What Is a DAO? [online] Investopedia. Available at: https://www.investopedia.com/tech/what-dao/.

18 www.constitutiondao.com. (n.d.). ConstitutionDAO. [online] Available at: https://www.constitutiondao.com/ [Accessed 11 May 2022].

19 Guida, P. (n.d.). Council Post: Privacy Is The Biggest Challenge Preventing DeFi Lift-Off. [online] Forbes. Available at:

https://www.forbes.com/sites/ forbes-financecouncil/2021/08/31/ privacy-is-the-biggest-challenge-preventing-defi-lift-off/ [Accessed 11 May 2022].

20 Ethereum Foundation (2016). Privacy on the Blockchain. [online] Ethereum.org. Available at: https://blog.ethereum.org/2016/01/15/privacy-on-the-blockchain/.

21 ethereum.org. (n.d.). Ethereum Virtual Machine (EVM). [online] Available at: https://ethereum.org/en/developers/docs/evm/.

22 Product plan (2018). What is MoSCoW Prioritization? — Overview of the MoSCoW Method. [online] Productplan.com. Available at: https://www.productplan.com/glossary/moscow-prioritization/.

23 the Guardian. (2021). Apple's plan to scan for child abuse images 'tears at heart of privacy'. [online] Available at: https://www.theguardian.com/world/2021/oct/15/apple-plan-scan-child-abuse-images-tears-heart-of-privacy.

24 ADVANCED TRUSTED ENVIRONMENT (n.d.). [online] Available at: http://www.omtp.org/OMTP_Advanced_Trusted_Environment_OMTP_TR1_v1_1.pdf.

25 Intel. (n.d.). Intel® Software Guard Extensions (Intel® SGX). [online] Available at: https://www.intel.co.uk/content/www/uk/en/architecture-and-technology/software-guard-extensions.html.

26 Cryptographic checksum [online] Available at: https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095651597

27 Daor, Joa Daemen, Joan Rijmen, Vincent. (1999). AES proposal: rijndael.

28 Rivest, R.L., Shamir, A. and Adleman, L. (n.d.). A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. [online] Available at: https://people.csail.mit.edu/rivest/Rsapaper.pdf.

29 sconedocs.github.io. (n.d.). Updating CPU microcode - Confidential Computing. [online] Available at: https://sconedocs.github.io/microcode/ [Accessed 11 May 2022].

30 Nilsson, A., Bideh, N. and Brorsson, J. (n.d.). A Survey of Published Attacks on Intel SGX. [online] Available at: https://arxiv.org/pdf/2006.13598.pdf [Accessed 11 May 2022].

31 docs.conclave.net. (n.d.). Maximising the security of your enclave - Conclave documentation. [online] Available at: https://docs.conclave.net/security.html [Accessed 11 May 2022].

32 Conclave. (n.d.). Conclave — Leading Confidential Computing Platform For Regulated Markets. [online] Available at: https://www.conclave.net [Accessed 11 May 2022].

33 Schneier, B. (1996). Applied cryptography : protocols, algorithms, and source code in C. New York: Wiley.

34 GitHub. (2022). darkwallet/darkleaks. [online] Available at: https://github.com/darkwallet/darkleaks/blob/master/EXPLANATION [Accessed 11 May 2022].

35 A Few Thoughts on Cryptographic Engineering. (2014). Zero Knowledge Proofs: An illustrated primer. [online] Available at: https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/.

36 H, T. (2009). Introduction to algorithms. Cambridge, Mass.: Mit Press.

37 Bitansky, N., Canetti, R., Chiesa, A. and Tromer, E. (2012). From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Proceedings of the 3rd Innovations in Theoretical Computer Science Conference on - ITCS '12. doi:10.1145/2090236.2090263.

38 Kate, A., Zaverucha, G. and Goldberg, I. (2010). Polynomial Commitments *. [online] Available at: https://cacr.uwaterloo.ca/techreports/2010/cacr2010-10.pdf [Accessed 11 May 2022].

39 Kate, A., Zaverucha, G. and Goldberg, I. (n.d.). Constant-Size Commitments to Polynomials and Their Applications. [online] Available at: https://www.iacr.org/archive/ asiacrypt2010/6477178/6477178.pdf [Accessed 11 May 2022].

40 Blake, I.F., Gadiel Seroussi and Smart, N.P. (2005). Advances in Elliptic Curve Cryptography. Cambridge: Cambridge University Press.

41 Mannak, R. (2020). Comparing General Purpose zk-SNARKs. [online] Coinmonks. Available at: https://medium.com/coinmonks/comparing-general-purpose-zk-snarks-51ce124c60bd [Accessed 11 May 2022].

42 Groth, J. (2016). On the Size of Pairing-based Non-interactive Arguments. [online] ePrint IACR. Available at: https://eprint.iacr.org/2016/260 [Accessed 11 May 2022].

43 Bowe, S., Grigg, J. and Hopwood, D. (2019). Recursive Proof Composition without a Trusted Setup. [online] ePrint IACR. Available at: https://eprint.iacr.org/2019/1021 [Accessed 11 May 2022].

44 Gabizon, A., Aztec, Z., Williamson, A. and Ciobotaru (2022). PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. [online] Available at: https://eprint.iacr.org/2019/953.pdf [Accessed 11 May 2022].

45 Wikipedia. (2020). Quantum supremacy. [online] Available at: https://en.wikipedia.org/wiki/ Quantum_supremacy.

46 Wohlwend, J. (n.d.). ELLIPTIC CURVE CRYPTOGRAPHY: PRE AND POST QUANTUM. [online] Available at: https://math.mit.edu/ apost/courses/18.204-2016/ 18.204JeremyWohlwend finalpaper.pdf.

47 Ben-Sasson, E., Bentov, I., Horesh, Y. and Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. [online] Available at: https://eprint.iacr.org/2018/046.pdf.

48 Yao, A. (n.d.). Protocols for Secure Computations. [online] Available at: https://research.cs.wisc.edu/areas/sec/yao1982-ocr.pdf [Accessed 11 May 2022].

49 Medhi, B. (2019). Privacy-Preserving Computation Techniques FHE from Ziroh Labs. [online] Medium. Available at: https://medium.com/@bhaskarmedhi/privacy-preserving-computation-techniques-fhe-from-ziroh-labs-8814e88044a [Accessed 11 May 2022].

50 Chakarov, D. and Papazov, Y. (2019). Evaluation of the Complexity of Fully Homomorphic Encryption Schemes in Implementations of Programs. Proceedings of the 20th International Conference on Computer Systems and Technologies. doi:10.1145/3345252.3345292.

51 Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. Proceedings of the 41st annual ACM symposium on Symposium on theory of computing - STOC '09. [online] doi:10.1145/1536414.1536440.

52 Shentu, Q. and Yu, J. (n.d.). Research on Anonymization and De-anonymization in the Bitcoin System. [online] Available at: https://arxiv.org/pdf/1510.07782.pdf [Accessed 11 May 2022].

53 darkrenaissance.github.io. (n.d.). The DarkFi Book. [online] Available at: https://darkrenaissance.github.io/darkfi/zkas/examples/sapling.html [Accessed 11 May 2022].

54 https://pantherprotocol.io Panther Protocol [online] Available at: https://pantherprotocol.io

55 Malene, T. (2022). DeFi on Bitcoin and private ZK Proofs. [online] Obscuro. Available at: https://obscuro.substack.com/p/defi-on-bitcoin-and-private-zk-proofs?utm_source=urls=r [Accessed 11 May 2022].

56 scrt.network. (n.d.). Secret Network - Bringing Privacy to Blockchains, Smart Contracts Web3. [online] Available at: https://scrt.network [Accessed 11 May 2022].

57 rekt. (n.d.). Rekt - Wormhole. [online] Available at: https://rekt.news/wormhole-rekt/ [Accessed 11 May 2022].

58 obscu.ro. (n.d.). Obscuro — An Ethereum Layer 2 Blockchain Privacy Protocol. [online] Available at: https://obscu.ro [Accessed 11 May 2022].

59 Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., Nazarov, S., Topliceanu, A. and Tramèr, F. (n.d.). Chainlink 2.0: Next Steps in

the Evolution of Decentralized Oracle Networks. [online] Available at: https://research.chain.link/whitepaper-v2.pdf.

60 SecureCoding. (2021). How Kotlin is More Secure Than Java? [online] Available at: https://www.securecoding.com/blog/kotlin-vs-java-nullpointerexception/.

61 GitHub. (2022). Kotlin multiplatform / multi-format reflectionless serialization. [online] Available at: https://github.com/Kotlin/kotlinx.serialization [Accessed 11 May 2022].

62 nvd.nist.gov. (2021). NVD - CVE-2021-44228. [online] Available at: https://nvd.nist.gov/vuln/detail/CVE-2021-44228.

63 Wrike.com. (2015). Complete Collection of Project Management Statistics 2015. [online] Available at: https://www.wrike.com/blog/complete-collection-project-management-statistics-2015/.

64 Wong, Whee Lee, Chan Tshai, Kim Yeow. (2013). The Importance of a Software Development Methodology in IT Project Management: An Innovative Six Sigma Approach - A Case Study of a Malaysian SME Organization. 10.13140/2.1.5106.0807.

65 Beynon-Davies, P., Carne, C., Mackay, H. and Tudhope, D. (1999). Rapid application development (RAD): an empirical review. European Journal of Information Systems, 8(3), pp.211–223. doi:10.1057/palgrave.ejis.3000325.

66 Spring. (n.d.). Spring makes Java simple. [online] Available at: https://spring.io.

67 GitHub. (2022). Talking to 25% of the Web: an in-depth report and analysis on the WordPress REST API. [online] Available at: https://github.com/humanmade/rest-api-white-paper/blob/master/en/03-what-is-a-rest-api.md [Accessed 11 May 2022].

68 Costa, D., Andrzejak, A., Seboek, J. and Lo, D. (2017). Empirical Study of Usage and Performance of Java Collections. Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering. doi:10.1145/3030207.3030221.

69 Liu, Dapeng Xu, Shaochun. (2015). Comparison of Hash Table Performance with Open Addressing and Closed Addressing: An Empirical Study. International Journal of Networked and Distributed Computing. 3. 60. 10.2991/ijndc.2015.3.1.7.

70 Research, C. (2009). Standards for Efficient Cryptography SEC 1: Elliptic Curve Cryptography. [online] Available at: https://www.secg.org/sec1-v2.pdf.

71 Singh, A.K. and Misra, A.K. (2012). Analysis of Cryptographically Replay Attacks and Its Mitigation Mechanism. Advances in Intelligent and Soft Computing, pp.787–794. doi:10.1007/978-3-642-27443-5_90.

72 docs.openzeppelin.com. (n.d.). ERC20 - OpenZeppelin Docs. [online] Available at: https://docs.openzeppelin.com/contracts/4.x/erc20.

73 ethereum.org. (n.d.). Anatomy of smart contracts. [online] Available at: https://ethereum.org/en/developers/docs/smart-contracts/anatomy/environment-variables [Accessed 11 May 2022].

74 Richardson, C. (2017). Microservices.io. [online] microservices.io. Available at: https://microservices.io/patterns/microservices.html.

75 ethereum.org. (n.d.). JSON-RPC API. [online] Available at: https://ethereum.org/en/developers/docs/apis/json-rpc/ [Accessed 11 May 2022].

76 docs.soliditylang.org. (n.d.). Solidity — Solidity 0.8.13 documentation. [online] Available at: https://docs.soliditylang.org/en/v0.8.13/.

77 docs.oracle.com. (n.d.). Type Erasure (The JavaTM Tutorials ¿ Learning the Java Language ¿ Generics (Updated)). [online] Available at: https://docs.oracle.com/javase/tutorial/java/generics/erasure.html [Accessed 11 May 2022].

78 Veracode. (n.d.). Veracode. [online] Available at: https://www.sourceclear.com/vulnerability-database/security/remote-code-execution-through-object-deserialization/java/sid-1710/ [Accessed 11 May 2022].

79 Shultz, T.R., Fahlman, S.E., Craw, S., Andritsos, P., Tsaparas, P., Silva, R., Drummond, C., Ling, C.X., Sheng, V.S., Drummond, C., Lanzi, P.L., Gama, J., Wiegand, R.P., Sen, P., Namata, G., Bilgic, M., Getoor, L., He, J., Jain, S. and Stephan, F. (2011). Conjunctive Normal Form. Encyclopedia of Machine Learning, pp.209–210. doi:10.1007/978-0-387-30164-8_158.

80 Sayeed, S. and Marco-Gisbert, H. (2019). Assessing Blockchain Consensus and Security Mechanisms against the 51% Attack. Applied Sciences, 9(9), p.1788. doi:10.3390/app9091788.

81 Boneh, D., Bonneau, J., Bünz, B. and Fisch, B. (2019). Verifiable Delay Functions. [online] Available at: https://eprint.iacr.org/2018/601.pdf.

82 Solana — News. (2021). Proof of History: How Solana brings time to crypto. [online] Available at: https://solana.com/news/proof-of-history.

83 Fei, S., Yan, Z., Ding, W. and Xie, H. (2021). Security Vulnerabilities of SGX and Countermeasures. ACM Computing Surveys, 54(6), pp.1–36. doi:10.1145/3456631.

84 kumarasamy, S. (2011). Distributed Denial of Service (DDOS) Attacks Detection Mechanism. International Journal of Computer Science, Engineering and Information Technology, 1(5), pp.39–49. doi:10.5121/ijcseit.2011.1504.

85 Enclave Memory Measurement Tool

for Intel® Software Guard Extensions (Intel® SGX) Enclaves Enclave Memory Measurement Tool (EMMT). (n.d.). [online] Available at: https://www.intel.com/ content/dam/develop/external/us/en/ documents/enclave-measurement-tool-intel-sgx-737361.pdf [Accessed 11 May 2022].

86 Gemini. (n.d.). The DAO: What Was the DAO and How Was it Hacked? [online] Available at: https://www.gemini.com/cryptopedia /the-dao-hack-makerdao.

87 Education Proof-of-Stake: A crypto path to lower energy consumption and yield. (n.d.). [online] Available at: https://content.ftserussell.com/sites /default/files/education_proof_of_ stake_paper_v6_0.pdf [Accessed 11 May 2022].

Sources of aspiration

1 Chen L. (2017). Elliptic Curve Cryptography. https://csrc.nist.gov/Projects/Elliptic-Curve- Cryptography. [Date Accessed 03-03-22]

2 Zhang, F., He, W., Cheng, R., Kos, J., Hynes, N., Johnson, N., Juels, A., Miller, A. and Song, D. (2020). The Ekiden Platform for Confidentiality-Preserving, Trustworthy, and Performant Smart Contracts. IEEE Security Privacy, 18(3), pp.17–27. doi:10.1109/msec.2020.2976984.

3 Raval S. (2016). Decentralised Applications. First Edition. Canada: O'Reilly Media. p.1.

4 Frankenfield J. (2018). Proof of Elapsed Time (Cryptocurrency). https://www.investopedia.com/ terms/p/ proof-elapsed-time-cryptocurrency.asp.

5 Lamport, L. (1983). The Weak Byzantine Generals Problem. Journal of the ACM, [online] 30(3), pp.668–676. doi:10.1145/2402.322398.

6 Mathforum.org. (2019). [online] Available at: http://mathforum.org/dr.math/ faq/faq.birthdayprob.html.

7 GitHub. (2022). Getting Started. [online] Available at: https://github.com/ethereum/vyper [Accessed 11 May 2022].

8 Ethereum: A secure decentralised generalised transaction ledger. (n.d.). [online] Available at: https://ethereum.github.io /yellowpaper/paper.pdf.

9 GitHub. (2022). About ==¿circom. [online] Available at: https://github.com/iden3/circom [Accessed 11 May 2022].

10 GitHub. (2022). snarkjs. [online] Available at: https://github.com/iden3/ snarkjs [Accessed 11 May 2022].