BSc (Hons) Computer Science (Artificial Intelligence)
Academic Year 2018 - 2019

A Decentralized Computation Platform

Lucy Sweet

April 2019

A report submitted in partial fulfillment of the requirements for the degree of Bachelor of Science

Abstract

Traditionally, decentralized computation systems suffer from drawbacks in efficiency, making them hard to apply to any practical problem. We attempt to fix this with a new algorithm that uses psuedo-random selection to delegate expensive problems to a subset of the nodes. We introduce a novel algorithm for quickly gaining consensus across a set of nodes as to a large list known as Broadcast Consolidate Resolve Affirm and a novel algorithm for creating a random number among many non trusted nodes called Multiparty Pin Seeding. We show that this is much faster than the state of the art.

Contents

1 Acknowledgments

Thank you to Panos Louveris, whose guidance and insight have helped guide this project and ensured that it remains on track and focused.

Thank you to Pascal Terjan and Google UK for their insight and thoughts on this project that allowed me to identify and solve hard questions that arose.

Thank you to Google for the support given to me over this academic term, particularly through the Google Europe Scholar Program and Google Application Security, Vulnerability Research Grants program.

This work is inspired by the work of Colin LeMahieu in his development of the Nano cryptocurrency, which has pushed forward in leaps and bounds the speed and scalability of cryptocurrency to real world applications.

The DScript Java Client uses third party packages. All packages are used under license and the licenses for these packages are given in Appendix A.

2 Declaration

I certify that the work presented in this dissertation is my own unless referenced or acknowledged otherwise.

Lucy Sweet	Date
Signature	

3 Introduction

3.1 On the Byzantine General's Problem and Decentralized Systems

The Byzantine General's Problem (Lamport, Shostak, and Pease 1982) is a fundamental problem for distributed computing applications that do not trust a single party or handful of parties ("decentralized" systems). In crypto-currencies, this problem manifested itself in the Double Spend problem. The Double Spend Problem is a problem caused when a malicious party spends the same balance on two separate things, the network has to come to a consensus as to which transaction will be rejected and which transaction will be accepted.

Traditionally, this problem was solved with centralization through use of a central server that verified whether a token was spent (Ryan 2006). This however created a single point of failure for both denial of service and nefarious actors to influence the network. The first decentralized solution to this problem came from "Satoshi Nakamoto" in their 2008 whitepaper for Bitcoin (Nakamoto et al. 2008). Bitcoin solves the double spend problem and gains consensus across the network through the use of a chain of blocks linked together by hash references to each previous block (a "Blockchain") and an algorithm that requires computational power be expended to include a new block through forcing a user to perform many thousands of SHA256 computations searching for a value that has a certain "difficulty".

3.2 From currency to computation

Bitcoin was the first network to solve the Double Spend problem and gain Byzantine Fault Tolerant consensus through a Proof of Work Blockchain. Other networks soon followed, notably in our case, Ethereum. Ethereum was created by Vitalik Buterin in 2014 as a new cryptocurrency designed to allow the creation of applications on the network, also known as "decentralized applications" or "dApps" (Wood 2014). Ethereum introduces the Ethereum Virtual Machine ("EVM") designed with the overarching goal of deterministic execution. It is the fundamental consensus mechanism for decentralized applications on Ethereum. Ethereum decentral-

ized applications are written as "Smart Contracts". While smart contracts can be written in a variety of languages, such as Solidity (Dannen 2017) all smart contracts are eventually represented as EVM byte-code. These contracts are then included on the Ethereum blockchain. When a user wants to use a smart contract, they take the current state of the contract and their input and submit these as a transaction to the blockchain. In order for a client to verify the block this transaction is referenced in, they must take these inputs and the current state, and calculate the outputs for the block.

3.3 Performance pain

In Ethereum, all smart contracts are stored on the same blockchain, and each node must execute all the state transitions within a block to validate it. This has high Byzantine Fault Tolerance, however, it is incredibly inefficient, particularly as every client must run every program. To show how serious of a deficiency in performance this can be we can turn to a practical example of Ethereum's performance versus centralized processors. In October 2018 the Ethereum Network could process a maximum of approximately 15 transactions per second whereas the Visa network was processing an average of 45,000 transactions per second at the same time (Murphy 2018).

3.3.1 Room to Grow

In 2017 Vitalik Buterin, the creator of Ethereum, and Joseph Poon, one of the co-author's of the Bitcoin Lightning Network whitepaper (Poon and Dryja 2016) proposed a solution to the performance bottleneck called "Plasma" (Poon and Buterin 2017). In Plasma, transactions are separated into "child" blockchains that are split from the "main" blockchain to attempt to reduce the load on the main blockchain. However, this proposal however has so far failed to conclusively prove its own viability, with Vlad Zamfir, Ethereum's Lead Researcher on Proof of Stake, questioning the viability of Plasma (Hertig 2017).

Research Question Space exists for a solution that brings the abilities of Ethereum in execution of Decentralized Turing complete applications while avoiding the pitfalls of performance bottlenecks.

Our question is therefore whether we can improve the performance of the state of the art while broadly maintaining its security.

3.4 Aims and Objectives

Our aim is as follows:

To create a protocol and application that allows the execution of Decentralized Applications ("dApps") with improved performance over the state of the art.

In order to achieve this aim, we must complete the following objectives:

1. Undertake a background study to identify existing work, identify the systems used by the state of the art and identify the caveats in performance of the state of the art.
2. Identify an approach which will give us results from which we can draw rigorous conclusions.
3. Design an algorithm that allows a decentralized network of untrusted nodes to come to consensus around the end state of a state transition machine given a start state and input and that has a performance improvement over the state of the art.
4. Design and implement software that shows the algorithms described above in operation and shows the viability of the algorithms in a solution.
5. Demonstrate the security of the algorithms described above through empirical evaluation of such.
6. Demonstrate the performance improvements of the algorithms described above versus the state of the art through empirical evaluation of such.
7. Evaluate the success of this project based on the security and performance of the algorithms as described above.

3.5 Project Approach

I will approach my project by identifying five significant steps I must take.

3.5.1 Literature Review

I must undertake a study to identify existing work and the systems used by the state of the art. Particularly this study will allow me to identify systems designed by others which may be able to be applied to my problem in order to advance the state of the art. Further, this study will allow me to identify the caveats of the state of the art so that I may better focus my efforts.

3.5.2 Algorithm Design

I must design an algorithm or set of algorithms for the solving of the aim described above. These algorithms should be more efficient and faster than the state of the art but maintain high byzantine fault tolerance.

3.5.3 Proof in Theory

I must prove that the algorithms I have designed are in theory an improvement on the state of the art while also preserving the security necessary for such a network.

3.5.4 Implementation Design

I must design a software solution that implements the algorithms on a public network communicating over the internet.

3.5.5 Algorithm Implementation

I must implement the algorithms in a software package and demonstrate that the algorithms are practically viable and work in software.

3.5.6 Overall Review

I must review and evaluate the outcomes of the approach described above, particularly looking at results gained and what could be improved.

4 Background

4.1 The need for Byzantine Fault Tolerance

Byzantine Fault Tolerance is the ability of a computer system to prevent itself from falling victim to the Byzantine General's Problem (Lamport, Shostak, and Pease 1982). First discovered by Robert Shostak and dubbed the Interactive Consistency Problem (Wensley et al. 1978), the problem exhibits itself in computer systems that can not tolerate byzantine faults. Byzantine faults are faults where components fail and there is imperfect information as to whether a component has failed. In distributed systems, a Byzantine Fault can occur when any one of the nodes exhibits failure induced accidentally or by an attacker that causes it to act in a way that it is not meant to. This can lead to the network losing consensus as to a fact or changing its decision unilaterally. In Cryptocurrencies, this can exhibit itself as the Double Spend problem. The Double Spend problem is a fundamental problem of consensus across a decentralized system, where many nodes need to agree as to when some money is "spent" on a transaction or not. If the Double Spend problem is not solved, than the system cannot come to any sort of global "state" and so is rendered critically vulnerable to attack.

4.2 Fault Tolerant Data Structures

Many centralized systems have been proposed to gain fault tolerance, but as we are creating a decentralized distributed system, we will not be discussing those as they are not useful to us, instead we will focus on the decentralized data structures that are byzantine fault tolerant.

4.2.1 Blockchain

Nakamoto et al. 2008 proposed the first decentralized fault tolerant data structure known as the Blockchain. The system works by using the hash of some previous content as a time-stamp, and then including this in a newer set of content as a pointer to the previous set, Nakamoto says:

> The timestamp proves that the data must have existed at the time, obviously, in order to get into the hash.

These pieces of content each reference the last piece of content before them through this "timestamp" made of the hash of the last piece of content content can be formed together as a chain. Nakamoto then introduces an agreement system to allow the network to only agree to a single valid chain, in the case of Nakamoto et al. 2008 this is done through a simple Proof of Work system based on a goal of causing as many zero bits as possible at the start of a SHA256 hash. Agreement systems will be discussed in more detail later.

4.2.2 The Tangle

In Popov 2016 Sergie Popov introduced a novel data structure called the "Tangle". The tangle is a directed acyclic graph where each vertices represents a transaction and each edge represents an "approval". Each vertices is further attached to at least two previous vertices and is said to "approve" these vertices (this means that each new vertices is the approver of at least two previous vertices and will be approved by future vertices). A transaction is valid if the vertices itself contains a valid transaction and the edges connecting it to previous transactions (it's approvals) are also valid. This means that, for every valid transaction, every edge from that transaction references another valid transaction, or the genesis transaction (which is the single vertices at the start of the graph). Once a vertices is referenced by many future vertices it is considered agreed to, to consider why it is agreed to we should consider the nature of the vertices within the graph and what it demonstrates. The vertices in the graph demonstrates that a transaction took place. Say an attacker wishes to double spend, they may create two vertices spending the same amounts, they could append them to the graph and they would both be individually valid, but no edge could be drawn from any future vertices that connects them both as this would invalidate the edge, such, this would cause the graph to split. The network then needs to decide on which "split" to follow, and does this through a method described in Popov 2016 called Tip Selection (referring to the tips of the graph, the latest vertices included in the graph). Tip Selection is done through a random walk from the first vertices in the graph towards the tips of the graph, the walk is intentionally bias towards vertices with more "weight". Weight is a simple proof of work that is individually cheap to calculate but expensive to perform numerous times. By being bias towards more weighted

transactions, the walk will enter the side of the split with more work done on it, and over time the network will abandon the other split and it will not grow any further, invalidating one of the double spend transactions. An attacker could not prevent this without significant resources, because they would need to create more weight in the form of proof of work than the rest of the network.

4.2.3 Block-Lattice

Block Lattice was first proposed in LeMahieu 2017 as a novel data structure. In block-lattice, every account has it's own blockchain, entirely independent of every other chain on the network. The overall state of the network is therefore to each client a set of n blockchains b_0 through b_n, this is referred to as the "Ledger". Every block on an account chain must be placed there by the owner, something LeMahieu refers to as "Design-time agreement" (LeMahieu 2017). If there are two conflicting blocks on the network then they must be caused by the malicious actions of the owner of the chain they are on because they are the only person authorized to add blocks to that chain, when this happens, nodes vote using Delegated Proof of Stake for the block to keep, the block with the most votes is then retained while nodes ignore any other conflicting block.

4.3 Approaches to consensus agreement

I will now describe the two approaches to agreement I have identified as in use

4.3.1 Traditional

The traditional approach (or as I refer to it the "public matter" approach) is an approach where any node may freely be involved in the consensus building process. This is shown in nearly every cryptocurrency including Bitcoin where any node may be a miner attempting to create the work for the next block in the blockchain (Nakamoto et al. 2008) to IOTA where every node is required to approve previous transactions to add their own (Popov 2016).

4.3.2 Algorand

In Gilad et al. 2017 a new system called Algorand was introduced which introduced a new approach to consensus agreement. Algorand was the first protocol to introduce a system where a limited set of the population that is representative of the node population makes decisions on behalf of the network (I refer to this as the "delegated" approach). In Algorand, a Verifiable Random Proof is used to select a "Block Proposer" and "Certifying Committee", the Block Proposer collects transactions into a block and proposes it to the committee, who then check as to whether this would be a valid block and if so approve it. Once this is done the Certifying Committee and Block Proposer can prove that they are members of each group through use of "Winning tickets" which are mathematical proofs that they were selected by the network (that is, the identities of the certifying committee are secret until their decision has been made). The network accepts the result of the certifying committee.

4.4 Consensus agreement systems

A consensus agreement system is an underlying algorithm used to decide between two proposed states when there is a fault such as a double spend. Agreement systems seek to make it very hard to change the agreement once it has been established (for example by retroactive revocation of a transaction).

4.4.1 Proof of Work

The first agreement system was proposed in Nakamoto et al. 2008 and is known as Proof of Work. Proof of Work is a simple system whereby out of a set of proposals, the one with the most "work" is the one that the network agrees to. "Work" is an expression of computation done by a computer system and is made of an NP-hard problem (the "problem") and a "difficulty" assigned for each attempt at that problem, to demonstrate this I will describe the work problem used in the Bitcoin network.

The problem described in the bitcoin network is to attempt to combine a block's hash with some random content to produce a SHA256 hash with as many zero bits at the start of the hash before

the first one bit as possible. The "difficulty" is the amount of zero bits at the start of the given hash for an attempt at the problem. The attempt with the highest difficulty is the accepted block. Bitcoin introduces a "threshold" system to limit the amount of blocks that the network adds by requiring the next block to the chain to have a certain difficulty and a difficulty adjustment mechanism that causes the threshold to be adjusted such that across the network one block will reach this threshold approximately once every 10 minutes (O'Dwyer and Malone 2014).

4.4.2 Proof of Space

Proofs of Space are similar in nature to Proof of Works in that Proofs of Work require allocation of non-trivial amounts of processor capacity to solve a problem. In Proofs of Space, instead of processor capacity the problem requires allocation of a non-trivial amount of memory or disk space (Dziembowski et al. 2015 and Ateniese et al. 2014). Proof of Space is seen as more environmentally friendly than proof of work as problems typically require less energy consumption than Proof of Work problems (Park et al. 2015). An example of a Proof of Space problem is a challenge based around graph pebbling (Dziembowski et al. 2015 and Ren and Devadas 2016), for this example, a miner can be required to commit to labelling a very hard to pebble graph. In order to verify that the miner has labelled the graph the miner can be challenged by asking for random locations in the commitment until the network is satisfied that the miner has pebbled the entire graph.

4.4.3 Proof of Authority

Proof of Authority is limited in use but it's most notable use is in Parity Ethereum (Parity 2019). In a Proof of Authority network, some subset of the network has the "authority" to endorse or publish changes to the overall network state. Changes to the network state are agreed to by the network when some criterion level of the "authority" agree (De Angelis et al. 2018). This works for both public networks, where some algorithm may determine who is a member of the authority (such as random selection bias towards accounts with higher stakes) or on private networks where certain central authorities may form authority nodes (J. Morgan 2016).

4.4.4 Proof of Stake

Proof of Stake algorithms are based on selecting the next block based on the wealth or age of the proposer (the "stake") (Buterin 2013, King and Nadal 2012, Kiayias et al. 2017 and Saleh 2018). Proof of Stake has advantages in speed (as no work has to be done and instead those with stake in the network must simply agree) but has attracted criticism for not providing negative incentive for stakeholders to not vote for forking networks (Ray and Authors 2018). This has been however pushed back on by others. In LeMahieu 2017 Colin LeMahieu says on this topic:

> With this arrangement, those who have a greater financial investment are given more power and are inherently incentivized to maintain the honesty of the system or risk losing their investment.

Others have designed their Proof of Stake systems to introduce incentive to not vote for forking networks through punishment of those that maliciously use their stake, in Buterin and Griffith 2017 for example punishment of malicious validators is introduced:

> Accountability allows us to penalize malfeasant validators, solving the "nothing at stake" problem that plagues chain-based PoS [Proof of Stake]. The penalty for violating a rule is a validator's entire deposit.

Proof of Stake is used in a wide variety of cryptocurrencies, notably Ethereum plans to use Proof of Stake in it's Casper Protocol (Buterin and Griffith 2017).

4.4.5 Delegated Proof of Stake

Delegated Proof of Stake is very similar to Proof of Stake in that the network state changes based on the stake of nodes. In Delegated Proof of Stake nodes can further delegate the authority to act on their behalf with their stake to other nodes, networks may cause that only a certain number of nodes who have the most stake delegated to them then act to agree in order to increase the speed at which the network can come to consensus (Grigg n.d.) or may cause that nodes vote as normal, but with nodes allowed to vote not only with their own stake but with the stake delegated to them (LeMahieu 2018).

4.5 Summary of Literature Review

We have discussed the many approaches taken to problems relevant to our overall problem of Decentralized Computation, including how the network agrees on decisions through systems such as Proof of Work, Proof of Space, Proof of Authority, Proof of Stake and Delegated Proof of Stake. We've discussed how networks store their global state through systems such as Blockchain, Tangles and Block-Lattice and we've discussed the differences between the traditional approach for consensus and the Algorand approach.

5 Project Approach

Based on the results of my background review, I will now discuss my project approach. As a reminder, my aim is to:

> ... create a protocol and application that allows the execution of Decentralized Applications ("dApps") with improved performance over the state of the art.

5.1 Algorithm Design

My first step should be the fundamental design of my algorithm. In consideration of the background and algorithms used in the state of the art, I have decided that I will make use of a combination of a Proof of Authority and Delegated Proof of Stake algorithm in order to balance security and performance across the network. I was particularly inspired by LeMahieu 2017 for it's novel block-lattice data structure and will use this as part of my project.

5.2 Validation in Theory

I will validate my algorithms through empirical evaluation of their effectiveness in speed and security and will then compare them to other systems already in practical use to show my rate of improvement (if any).

5.3 Application Design

I will design my application to be written in Java and design systems that may not have been covered in the algorithms described

above, such as networking and specific information as to how the application will operate, such as an instruction set for execution of applications.

5.4 Validation in Practice

I will validate the system in practice by ensuring that the practical program operates as expected with nodes communicating on it and performing standard operations, as well as recovering effectively from byzantine faults due to mis-configured or attacking nodes.

5.5 Overall Review

I will review the project and the steps taken above to identify whether I have met the aims and objectives I set for this project.

6 DScript Protocol

I will now talk about the DScript Protocol, a collection of protocols and algorithms used to enable the DScript Consensus System.

6.1 Delegated Proof of Stake

The system uses Delegated Proof of Stake as the base mechanism to ensure consensus across the network. Delegated Proof of Stake supports the consensus of the core building blocks of the DScript network which will be described below in more detail.

6.1.1 Conflict Resolution

If some change in state is broadcast and there is no objection to this change, nothing happens and the change is silently accepted, this is an idea first proposed in LeMahieu 2017 and has advantages in terms of network capacity used. If there is an objection or conflict with a proposed state, then a voting round begins. In the voting round every online representative votes for the first valid state change they saw (if any) and the proposed change with the most votes wins and is certified by all honest representatives. So long as at least 50%+1 of the network stake is held by honest representatives, there is no way

any attacker or combination of attackers can out-vote the honest representatives.

6.2 Block-Lattice

Accounts and Applications are stored in Block-Lattice blocks, each account or application has its own, distinct blockchain, beginning with a create block which establishes the initialization parameters of the account or application, for example in accounts one initialization parameter may be the account's public key which is used to check that a transfer is signed by the real account holder. Each account and application's address is the SHA512 hash of the create block of their blockchain.

6.3 Representative Population Sampling

Representative Population Sampling is something more seen user research that in protocols designed in computer science (Krejcie and D. W. Morgan 1970). We however will be using it throughout the DScript protocol to increase the speed and efficiency of the network.

6.4 Representative Committee

A "Representative Committee" in the scope of the DScript Network is a committee of N nodes who have been selected by the network in order to accept a "problem" and return a "solution". A "problem" may for example be a start state and input into a state transition machine and a "solution" may be an end state of that same state transition machine. Representative Committees are chosen to be a random sample of the network through a psuedorandom selection algorithm [1]. The Building Blocks of Representative Committees are therefore:

- An algorithm to determine a list of nodes who may be a member of a representative committee.
- An algorithm to create a seed to be used with a psuedorandom generator to select nodes to be a member of a representative committee.

[1] It is important that the algorithm is psuedorandom rather than random so that the nodes selected can be verified in the future.

6.5 State Transition Agreement

State Transition Agreement is the overall goal of a Representative Committee. It is reached when some amount X of the N nodes in the Representative Committee agree on a solution to the problem given to them. For example nodes may agree that: $p(s0, 1) \longrightarrow s1$

6.6 BCRA

In order for our overall algorithm to work, we need a system through which every node on the network can reach consensus onto a list of accounts. If we did not have this list, then nodes would not decide on the same executors and therefore the system would fail. Therefore, this list needs to be exactly the same across all nodes in both order and contents. In order to perform this function, I have developed an algorithm that I call BCRA. BCRA [2] is one of the building blocks of representative committees and is the algorithm that is used to determine a list of nodes who may be a member of a representative committee. It is an entirely novel algorithm created by me. BCRA depends on a Consensus Agreement System and is useful for very quickly deciding upon a collection of items which are exactly the same across many nodes. BCRA depends on Time Synchronization of the nodes involved in the BCRA process to at least a level such that the skew, that is, the maximum difference in recorded time between the nodes due to bad synchronization at a given real time, satisfies $skew < roundTime/4$ where $roundTime$ is the amount of time that a single BCRA round on the network takes. As the skew becomes bigger, the BCRA algorithm becomes less efficient (in that it will likely accept less items into the list agreed on), but will still produce agreement.

6.6.1 Overall Goal

The overall goal of the BCRA protocol is to produce a list L that contains some number of items $L_1...L_n$ where n is the total number of items within L where.

- The items that are included in L are guaranteed across all nodes.

[2] BCRA is the acronym of the 4 stages of this Protocol. These are Broadcast, Consolidate, Resolve, Affirm.

- The order of the items in L is guaranteed across all nodes.
- Items that are included in L are those that are seen most widely by the network.[3]

6.6.2 Broadcast

The first stage of the BCRA process is the Broadcast phase. In the broadcast phase, nodes broadcast items that they wish to appear in the list across the network, these are forwarded across the network allowing them to be seen by more nodes. As each node receives an item, it stores the item in it's memory.

6.6.3 Consolidate

Each node takes all the items they have received and puts them into a list, each node takes each item and hashes the item such that they have a list of hashes of items, the hashes are in order as to where the original items were in the list. Each node signs their list and broadcasts it along with their representative address that has their stake. Each node remembers the items in each other representatives list and how much stake that representative has.

6.6.4 Resolve

The nodes now need to agree on a single list that they will then work towards certifying as the winner. For every record in every list we assign a weight to that record, $w(r)$ where the weight is the sum of the weight of every representative whose list it is included in, such that $w(r) = \sum_{i=0}^{count(listsContaining(r))} stake(listsContaining(r)_i)]$ and for every list the weight of that list $w(l)$ is the sum of the weight of all records in that list l_0 to l_n such that $w(l) = \sum_{i=0}^{n}[w(l_i)]$.

6.6.5 Affirm

Based on this resolution phase, nearly every honest node should now have decided on the same list. These nodes then sign this list and broadcast their signature to this list, simply now, the list which reaches a 50%+1 majority of the sake is duly agreed to. Nodes then broadcast the items that are hashed within this list by consulting

[3]This is to prevent an attacker stuffing a list with cooperating attackers.

the hashes, seeing if they hold any of the items from that hash and then broadcasting it. It is unlikely that items will not be resolved to their original un-hashed items because the winning list is the one that was most widely seen across the network, if however this does occur, the hash is retained in the list in place of the item and the item may be reintroduced by any node at any time. The hashed record is treated the same as any un-hashed record, this is necessary due to the ability of an attacker to otherwise cause a Race Condition by withholding an item and then introducing it after it has been treated differently due to being hashed.

6.6.6 Automation

This process can continue forever, with the network agreeing to new lists in advance through design time agreement as to when these lists should be agreed to.

Lack of Normalization as a Design Feature We are aware that a reader may notice that the BCRA algorithm has no normalization of the weight of a list. This is an intentional design feature designed to thwart attackers even when they have an exceptionally high ability to influence the weight of a list. We should first consider that for an attacker, stuffing the list with executors does not necessarily increase the chances of the attacker being chosen (as we will discuss in far more detail later on) and that an attack may fail if any honest executor joins the list. Therefore, the lack of normalization is intentional, we favour lists with more nodes in every case because this increases the chance of an honest node being presented even in high attacker influence scenarios. For example, an attacker may hold a large amount of stake, but all honest nodes need to do is send 1 record onto the list, this list will inherently always have a higher weight than any list of the same but without that one record (because this one record is endorsed and weighted by the one honest representative at a minimum who broadcasted that commitment). Normalization would allow attackers to begin to attack the BCRA process in circumstances where the attackers controlled a large amount of stake. By concentrating their stake into a small list, an attacker could defeat honest nodes. Hence, we do not normalize the weight of BCRA lists.

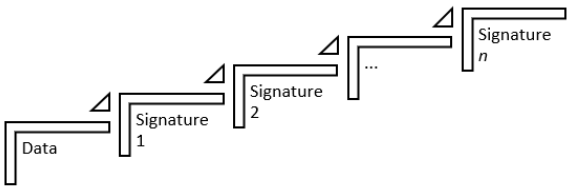

Figure 1: Visualization of a Pin in transmission which has been signed by n representatives.

6.7 Multiparty Pin Seeding

We have seen how we can create a list of nodes to select from through the use of BCRA, we now need a seed for our psuedorandom selection algorithm. I have created an algorithm called Multiparty Pin Seeding for this purpose. Multiparty Pin Seeding relies on the notion that no attacker is able to predict the signatures of other parties on a network, as to do so would require wielding that parties private key. We rely on combinations of these signatures to generate a seed. The overall goal of multiparty pin seeding is to create an algorithm whereby given some input content i the algorithm produces an output which is unpredictable for any one node or group of nodes that hold less than 50%+1 of the network stake, but is immutable in that it is the only valid output and can be verified as such. We therefore must consider that for every pin created by the network no further pin may be created for the same input in order to ensure the pin is immutable[4]. Consider some input previously agreed to d, we wish to create a pin of d for use on the network. We broadcast d across the network such that each representative see's it. When a representative observes d, they ensure that it is not similar to any previous input given to the multiparty pin seeding algorithm they are aware of[5], this protects the immutable property of pins. If the node is satisfied of this, they then take d and hash it, they sign this

[4]The immutable property of a pin is required as it is used as the seed for witness list generation, if it was not immutable an attacker could make slight changes until they received an evil seed, that is, a seed that allows them to select a list of only attackers.

[5]The similarity of two pieces of data, d1 and d2, and whether they are too similar is a decision for each node. However, we can say that if 50%+1 of the stake is held by nodes that are honest and set an appropriate threshold the system is not broken by similar data. In a practical application you may consider data similar if it has the same hash.

hash and attach it to d such that the message is now composed of $d...s_0$ where s_0 is the signature of this node. The node then broadcasts this message. For every new node that see's this message, they then (on satisfaction of the recently seen data test) sign the entire message [6] and append their signature as such, this creates a chain of repeated signatures s_0 through s_n where n is the number of nodes that have signed the pin. This is shown in Figure 1.

For every signature list, the "weight" of this list, $w(l)$ is equal to the stake of every node who has signed this list.

$|(s_0, ..., s_n)| = n$

$w(l) = \sum_{i=0}^{n}[stake(node(s_i))]$

6.7.1 Threshold and Pin Finality

When a node observes a pin where the weight of that pin is greater than or equal to 60% of the stake $w(p) >= 0.6$ they reduce the pin until if they removed the next entry the weight of the pin would be less than 55% of the stake.

$p = (d, s_0, ..., s_n)$

$where$

$n = |p^*|$

$p^* = arg\ min\ w(l|w(l) >= 0.55)$

This pin is now the conclusive pin used by the network for the piece of data d. The pin itself is composed of the d (the "initialization vector" which was signed by the first signature) and the signatures of every node thereafter.

The pin can be hashed to be used as a seed in the psuedorandom number generator:

$seed = hash(p)$[7]

So long as an attacker's stake in the network is $\leq 50\%$, they are unable to create a new pin because the honest stakeholders will not sign the same data twice, and hence the pin is immutable.

6.7.2 0.6 Threshold

Readers will notice that for every pin in Multiparty Pin Seeding, the pin is resolved at a weight of 0.6 of the total stake of the net-

[6] A node may only sign a signature list once, any signature list signed more than once by the same node is not valid and will be refused by any honest client.

[7] We use an abstract name here as in practice any qualifying algorithm could be used. Hash simply needs to be an algorithm that returns a secure one-way hash.

work, then reduced to a weight of 0.55 of the total stake of the network. To consider why these numbers were chosen, let us consider the underlying consequences of these numbers. The choice of 0.6 (the "Threshold Criterion") is a balance between the Speed of the Algorithm and it's Security. The lower the Threshold Criterion, the easier it would be for an attacker to break the immutability of an MPPS pin by creating a new pin with the same initialization vector and having cooperating attackers sign it. We have set the threshold at 0.6. Let's consider a worst case scenario where an attacker plans to create 2 pins for the same initialization vector, recall that no honest representative will sign the same initialization vector twice. Therefore, we can rule out 60% of the stake minus the attacker stake from signing the pin. Therefore, if the attacker didn't have any stake, in a worst case scenario they would only be able to convince 40% of the stake to sign their second pin, not enough to reach the threshold. We can see from this that an attacker would, in a worst case scenario for the defenders, need a 20% stake in the network to defeat the MPPS pin. This may sound quite low, but we should consider that even if the MPPS immutability property fails, it does not mean that each honest representative will now accept the new MPPS pin. That is, even if an attacker breaks the immutability of an MPPS pin, honest nodes will refuse to accept the 2nd pin even if it reaches the Threshold Criterion (so long as they have seen the first pin before). This will naturally, split the state of the network which will be resolved through Delegated Proof of Stake. While this will take time, it is still secure, and regardless, a 20% stake in the network is quite hard for an attacker to achieve. Even more unlikely is that every single node that did not sign the pin did not see it (we will discuss this later in MPPS Performance and MPPS Security).

6.8 Lightweight Distributed Database

I have created an algorithm and communications protocol called the Lightweight Distributed Database ("LDDB"). LDDB is a simple lightweight database on which state information is transferred between nodes (Sweet 2019c). Every object in LDDB is made of an Identifier and a Data Content. The Identifier is made of the hash of the Data Content [8], such that for every Data Content there is

[8] In our application we use SHA512 for this purposes.

only one valid Identifier. LDDB works so long as at least one client well connected to the network remembers the data content, every other client need only remember the identifier, reducing disk space needed. LDDB has two methods for fetching content.

6.8.1 Direct Fetching

Direct Fetching is the first method through which nodes can gain information is through Direct Fetching, in Direct Fetching a client broadcasts an Identifier, a location[9] on which to serve the Data Content and a precommitment for cancelling the request in the future. The user broadcasts this message across the network, when a node see's this message and has the Data Content, they connect to the location provided and transmit the content [10]. When the content is received, it can be easily checked as to whether it is valid by comparing it to the identifier to see if it matches. If it does, the node may then cancel their direct find request in order to ensure they do not now receive pointless transmissions of the Data Content. To do this, they simply transmit the secret portion of their precommitment.

Precommitment The precommitment is a simple way in which a user can certify their identity across the network after pre-committing that identity. To generate a precommitment, a user first generates a set of random bytes b (The "secret portion"), and then creates their precommitment as $p = hash(b)$. Any secure hashing algorithm maybe used [11]. To then execute the commitment, the user simply transmits b. When a client receives b, then hash it, notice that is has the same hash as the precommitment and can therefore be sure that the owner of the precommitment has committed to it (because assuming enough bytes were chosen at random no other person could in a reasonable time reverse the hashing process).

6.8.2 Indirect Finding

Another way that a user may fetch resources is through Indirect Finding, this is particularly useful if attackers employ tactics like

[9] This could be any protocol such as FTP or HTTP, but in our application we use HTTP.

[10] Systems may be added such as verifying the location wishes to receive the content through a simple query before transmission in order to prevent using the system for malicious purposes such as Denial of Service attacks.

[11] In our application we use SHA512.

denial of service attacks on the locations given in Direct Finding requests, but can use up more traffic. Indirect Find messages look the same as Direct Find messages minus the location parameter. Instead, when a client receives the message and has the data content, they broadcast the content across the entire network [12]. The client when they observe a broadcast with the Data Content then cancels their indirect find request in the usual way through use of their precommitment.

6.9 Execution Delegation

We've now laid out the ways in which our Representative Committee, BCRA List and Pin to use as a seed will be generated. The first thing we need to do is lay out the exact algorithm that decides on which of the candidates $(c_0, ..., c_n)$ where n is the total number of candidates to be selected, will be selected and included in the Representative Committee. I will first show a naive approach and demonstrate why it is insecure.

6.9.1 The simple approach

For avoidance of doubt this is not used in DScript but is included to show the reader why we use the method actually used by presenting the alternative and why it is not secure. One way to decide on the list is that given a list of candidates $(c_0, ..., c_n)$, a seed value p and a random number generator $r(p)$, we could simply select candidates purely at random from the list. Such that if the goal number to be selected is g:

$selected = \{c_i | i \in pulled\}$
$pulled = \{r(p), r(p), ...\}$
$|pulled| = g$[13]

The problem with this solution is it is open to a Sybil Attack. A Sybil Attack is an attack where the system is subverted by an attacker who claims to be multiple unique identities (Douceur 2002). Say I am an attacker and wish to subvert this system but I am a

[12] A suitable size threshold should be used on broadcasted messages.

[13] $|a|$ denotes the distinct elements in set a and not just each element even if equal to another element. Consider two sets, A and B, such that $A = \{1, 1, 2\}$ and $B = \{1, 2\}$. In set theory, $\{1, 1, 2\} = \{1, 2\}$ and therefore $A = B$, such for every element in A and B $x \in A \iff x \in B$. This proof adapted from and credited to to user75560 at `https://math.stackexchange.com/questions/380272/notation-for-number-of-distinct-elements-in-a-set`

minority on the network. I could cause this system to select many attackers by simply flooding the list of candidates with attackers. For each selection made from the list, the chances of an attacker being chosen are:

$$chanceAttackerChosen(l) = \frac{|\{l|attacker\}|}{|l|}$$

To attack this system, we wish to maximize the members of the committee that are chosen as attackers. To do this, we simply need to flood the list to maximize $chanceAttackerChosen(l)$. To avoid this problem, the network needs a way of skewing chances of selection based on some sort of reputation in order to counter a Sybil attack.

6.9.2 The approach taken

We will mitigate Sybil attacks through weighting the selection process to be bias towards users with a higher "stake". Stake will be decided through the Delegated Proof of Stake system and will be exclusively based on the users stake and any stake delegated to them.

Where $d(u)$ is a set of the delegations for a given account u and $b(u)$ is a number representing the balance of an account u:

$stake(u) = b(u) + \sum_{i=0}^{|d(u)|}[d(u)_i]$

Using this stake, we can now weigh the selection process in favour of accounts on the list with a higher balance, to do this, we will initialize a "selection range" r considering a set of accounts a_0 through to a_n where n is the total number of accounts.

The selection range begins at 0 and reaches the selection range ceiling c which is defined in respect of a set of accounts a as:

$c(a) = \sum_{i=0}^{|a|}[stake(a_i)]$

Given this selection range, we need to distribute the accounts in the list across the range, we do this proportionally based on the stake of each account such that an account with a 50% stake in respect of the total stake of all accounts on the list has a 50% chance of being selected from the list.

To distribute the accounts across the range, we will assign each account a segment of the range beginning at the end of the range for the previous account and of a length equal to the stake of the given account. Such that for any set of accounts a, the range of an account a_i will be a set of:

$Where\, i > 0$
$start(i) = finish(i-1) + 1$
$finish(i) = start(i) + stake(a_i)$

$Where\, i = 0$
$start(i) = 0$
$finish(i) = stake(a_i)$

$range(i) = \{start(i), finish(i)\}$

We now need to select accounts from the range. We first define the amount of selections we wish to make as s,[14] we will then make s selections across the range.

To select an account, we generate a random number r and consult the range, if an accounts boundaries are such that $startBoundary >= randfinishBoundary <= r$ then the account is selected. An account can be selected more than once, and if it is selected more than once then we ignore the selection, such that if an account is selected twice to be included in c and $s = 5$, then $|c| = 4$ because the account will not be added the 2nd time but neither will another account. This is an intentional design choice in order to prevent degradation of the honest subset of the network if there is a case where very honest nodes hold very large stakes.

Based on this, we can select the committee, to recall the definitions given previously:

> ... of the candidates $(c_0, ..., c_n)$ where n is the total number of candidates to be selected

We can say:
$selected = \{c_i | i \in pulled\}$
$a(i) = accountAtPositionInRange(i)$
$pulled = \{a(i), ...\}$
$|pulled| = s$
Where:
$accountAtPositionInRange(i) = \{a_x | startBoundary(a_x) >= i \cap endBoundary(a_x) <= i\}$

[14] In the practical implementation of DScript, we always attempt to select 10 members of the committee.

6.10 Execution

We now need to discuss how this committee will decide on the result of a problem presented to it. Recall that our overall aim is:

> To create a protocol and application that allows **the execution of Decentralized Applications ("dApps")** with improved performance over the state of the art.[15]

We will therefore discuss this portion of the protocol in narrower terms, that is, in terms of the execution of a decentralized application. To do this, we must first lay out the anatomy of a Decentralized Application on the DScript Network as well as the language used to execute Decentralized Applications on the DScript Network.

6.10.1 Deterministic Distributed Language

The Deterministic Distributed Language ("DDL") is a programming language designed and implemented by me for this dissertation. The DDL design document (Sweet 2019a) begins by saying:

> This is a fast, low memory, highly deterministic, backwards compatible and platform neutral language designed for the execution of decentralized applications (DApps).

All DDL programs are referred to in the design document in pure binary form. Every DDL program begins with a 2 byte version identifier, the purpose of this identifier is to allow backwards compatibility, for example by emulating deprecated behaviour when executing old programs on new clients. From there, every DDL program declares how long it is (the "instruction set length") using 4 bytes, these guarantee the exact amount of instructions that are within the program.[16] In order to preserve state, DDL provides a "permanent memory" which is a map of 64 byte indexes onto 64 byte values. For more temporary storage during execution, DDL provides a stack with a maximum size of 2048 64 byte items, DDL also provides a single value called the temporary value, which can be used to store a single item independent of the stack (which is also not preserved at the end of execution). The commands for interaction

[15] Bold added for clarity.

[16] A DDL program is not valid if the declared amount of instructions is not consistent with the actual amount of instructions.

with permanent memory are `0x29`, `0x30` and `0x31`. The commands for interaction with the temporary value are `0x32`, `0x33` and `0x34`. Most commands in DDL are routine and common across many low level languages, such as `0x2` "push" and `0x5` "stacksize". But DDL has some notable commands related to its use on the network such as `0x19` "getaddress" or `0x20` "maximumcost". Additionally, there are some other commands to do with extensions to the DScript network such as `0x35` "setsubchain".

6.10.2 Anatomy of a Decentralized Application

A DScript Decentralized Application is composed of two components, the "Memory" component which stores the memory that makes up the current "state" of the program and a "Method" component which stores the logic of the program.

The Method Component The method component describes the callable methods available to the program which accept input. Each method is indexed by 64 bytes and contains three parts:

- The Input and Output Definition
- The Visibility
- The Source Code

Input and Output Definition The Input and Output Definition defines the inputs required to invoke the method and specifies whether the method may return an output. The input definition may specify the amount of bytes that the input must satisfy as a range, this is formatted in the format `a,b` where a is the minimum bound (inclusive) and b is the maximum bound (inclusive). It is illegal to call a method with an input that does not satisfy this range, if a single number is supplied as the input definition, it is both the minimum and maximum bound. The input definition may be zero, in which case no input is acceptable. The program may read the input via instructions `0x24` "getinputlength" and `0x25` "getinput". The output definition is a single Boolean that may either be `true`, which would mean that the program can send output, or `false`, which would mean that the program cannot send output. If the output

definition is false, then it is illegal for the program to call instruction `0x28` "output", if the definition is true, then the program may output up to 1024 bytes of output, as shown in Sweet 2019a:

> If the output exceeds 1024 bytes only the first 1024 bytes of output will be included, the rest will be ignored.

Visibility Visibility controls whether the method may be invoked as the initial call when the committee is executing a problem (that is, whether an outside user may advance a programs state through a call directly to this method). Visibility is a boolean, if it is `true` then the method may be called initially, if `false` it may not. The committee will refuse to execute a non-visible method as the first method, but another DDL program may call the method, while not mentioned in the DDL design docs, calling `0x101` will cause the program to attempt to invoke the method with the address given in the temporary value, this is a behaviour exclusively seen in DScript.

Source Code The Source Code of every method is an independent, self contained DDL file that follows the standard set out in the Deterministic Distributed Language section.

6.10.3 Cost

Now that we have described the language on which applications will be executed and the structure of these applications, we can now delve into how a committee will actually execute these applications. The first thing to mention is "cost". Cost is a way in which the network prevents abuse of the execution functionality while incentivizing executors. Every request to advance a program's state (which contains the program's address, the method to call and the input) also contains an "commitment". A commitment is similar to a basic "transaction" in that it has an authorization of an underlying account and specifies an "amount" except that a commitment does not instantly "spend" the amount specified within. To spend the amount specified the commitment must be converted to a series of transfers. To convert the commitment into transfers, the committee must come to a decision, this decision can jointly be used to advance the state of the program and claim some amount of the "commitment". The commitment is claimed up to the "real cost", this being

the actual cost incurred for the execution of the application. Cost is based exclusively upon instructions called, each instruction in DDL and the extended instructions for DScript (those from `0x101` onwards) have a set "cost" to call them. The total cost of executing i instructions of n length is $\sum_{x=0}^{n}[cost(i_x)]$. The real cost is the total cost of advancing the entire state specified in the message to advance the programs state. In order to reward the executors, we perform the following reward strategy:

- We set the amount to be distributed a to be the total cost rounded down to the nearest multiple of n where n is the amount of members of the committee.
- We create a transaction for each member of the committee that transfers a to that member, the authorization for this transaction is the commitment itself. All the transactions are bundled together and pushed in full to each chain of each member of the committee.

If at any point the cost of executing the program exceeds the maximum amount given in the commitment, the program shall halt as if it had reached an exception and output exception `0x402`.[17] This is similar in nature to authorization holds used by banking services in that tokens are not spent immedietally but are authorized in advance for spending.

6.10.4 Gossip

Gossip is an ability of the committee to communicate messages to each other during execution, to implement gossip, we leverage the existing network broadcast messaging system. To prevent non-committee members from reading gossip it is encrypted so only members of the committee may view it. Each gossip message is made of two components, the "data" component and the "key recovery" component. To create the data section, a client takes the data they wish to transmit and generates an AES-256 key k to encrypt the data. The data section is then this encrypted data. The key recovery section is composed of n sections where n is the number of members of the committee c, each section is created by taking the

[17] This is a nod to HTTP status code 402 (payment required).

public key of each member of the committee and using it to encrypt k, such that:

$keyRecovery = \{\forall i \in c[encrypt(k, publicKey(c_i))]\}$

This gossip is then sent across the network, when a member of the committee recieves the gossip, they can reveal its contents by finding the relevant section of the key recovery segment, decrypting that section to reveal the AES key and using that AES key to decrypt the data segment.

6.10.5 Execution by the Committee

We will first clarify as previously alluded to in the cost section that there are certain instructions added on top of the standard DDL instructions by DScript, these are instructions `0x101` onwards (as DDL will never create an instruction outside the range of `0x0` to `0x100`). These instructions are "methodcall" `0x101` which can call another method in a DScript program and is described above, as well as "random" `0x102` which pushes an entirely random 64 bytes into the temporary value and "httpget" `0x103` which attempts to fetch some bytes over HTTP. We will describe exactly how these methods are performed later as they are special in that in the case of `0x102` and `0x103` they require the communication of the committee to some degree. In the basic case of a decentralized application, each member of the committee takes the current program state, the method to be called, and the input, they verify this is a legal action to take and then execute the method until completion or an exception. So long as there is no exception, they then hash the permanent memory contents and gossip them to the committee, if the entire committee gossip the same hash they know they have come to the same agreement, each committee member then generates the serialized end state and signs it, these signatures combined with the serialized end state form the proof of execution and final state. This can then be converted into a block that advances the application's chain, the advance of the chain is made of the committees decision, the proof of who was selected to be a member of the committee and the initial input. If the committee is unable to agree to a final state then the state does not advance and no transfers of compensation may take place (that is, a condition for using a confirmation transaction is the completion of some valid state transition). This is so that the user may then take another attempt at causing a transition.

This is not an attack vector as for any honest network, the attacker will eventually encounter a committee of fully honest nodes and will then lose their tokens when they advance the state and claim the funds.

6.10.6 Extra Instructions

There are some instructions unique to DScript that we shall now describe. Extra Instructions are not included in Sweet 2019a because they extend the language in ways only possible in the DScript network.

Method Call `0x101` `0x101` (aka "methodcall" or "Method Call") is a special instruction that allows the application to call other methods in the application regardless of their visibility. Method Call does not support specific input arguments or any output (although an output is acceptable it will not be available to the caller). Importantly however, a method called by method call will share the same environment as the caller, such that it will share the same stack, temporary value and permanent memory of the caller, this allows transfers of information and inputs to be sent via the stack. The address of the method to be called is always whatever value occupies the temporary value holder (if the temporary value is empty, the address is 0x0).

Random `0x102` `0x102` aka "random" is a special instruction that pushes a completely random 64 byte value to the stack. While in a normal program this would be easy, it requires more work in DScript due to the fact that many nodes are performing the same computation, we need to ensure the randomly generated value is deterministic across the committee, while making sure it is also unpredictable. To do this, we will use the gossip system to conclude a seeding value to feed into a Psuedorandom Number Generator on the first call of random. When a node first see's that `0x102` is called, they announce this in a gossip to the committee. Each member of the committee then computes a "randomness vector" v which is a random number of any value. Each member also computes a "randomness padding" x which is a random number of any value. Such that $-\infty \leq v \leq \infty$ and $-\infty \leq x \leq \infty$. Based on these two val-

ues each member computes a "randomness commitment" c which is made of $sha512(v + x)$. The reason for the randomness padding is to prevent a rainbow table attack against the commitment by making the hash different even for the same number. Each committee member then signs their commitment as their own and transmits it as a gossip message. Once each member has done this and all messages have been received by all witnesses, the committee enters the "reveal stage", normally, the last member to transmit would be able to influence the seed as their final number would set the overall number of the seed, but due to their randomness commitment, only v is acceptable to the rest of the committee. Each committee member reveals v and x to the network, and this is only accepted if $sha512(v + x) = c$. Once each committee member has accepted all randomness vector values as a set V, they create the randomness seed s as:

$s = \sum_{i=1}^{|V|}[V_i]$

This seed can be fed into the Psuedorandom Number Generator every time the instruction `0x102` is seen and used to input the same random numbers each time across each run of the committee.

HTTP Get `0x103` `0x103` aka "httpget" is a primitive example showing the networks ability to interact with off chain facts, in this case, websites transmitted over HTTP. When the instruction is called, a URL is created by encoding the item in the temporary value into a US-ASCII string. Each member of the committee makes a request to this URL and hashes the response, if all committee members get the same response they may continue with execution by providing the response to the program. To do this, they divide the HTTP response body only into 64 byte chunks and add these one by one, from the end of the body to the start, to the stack, they then add a single value to the stack which indicates how many items of response were added to the stack.

7 DScript Java Client

As part of my Final Year Project, I have created a Java application that implements the DScript Protocol described above. The application runs a node that can communicate with any other node

over a network and can perform the functionality described in the DScript Protocol.

7.1 Networking

I designed my own networking solution for connecting nodes together which is described in detail in the DScript Technical Design Document (Sweet 2019b). As part of this design, I decided that communication should be done over HTTP and I created 3 networking layers on top of the Application layer of the OSI Model. These are the Cluster, Message and Application[18] layers.

7.1.1 Cluster

The Cluster Layer is responsible for ensuring that the node remains connected to the DScript Peer to Peer network, it does this by discovering nodes through queries to already known nodes and by announcing itself to nodes. Principally, the Cluster Layer is responsible for ensuring the network is as well connected and aware of other devices on the network as possible.

7.1.2 Message

The Message Layer is responsible for the transport of messages across the entire network, all messages sent over the Message layer are broadcast-like in nature. The Message Layer handles the small proof of work system used to ensure Denial of Service cannot be performed over the message layer, it routes messages from other nodes and it passes messages for our node to the respective components that need to be aware of them.

7.1.3 Application

The Application Layer is responsible for complex operations described in the DScript Protocol and has a wide range of responsibilities. Any system built on top of the DScript messaging system sits here, including the real-time conflict detection system, the proof of work voting system and the BCRA list generation system.

[18] Application as it relates to DScript

7.2 Cryptography

The DScript network requires cryptography for making references to previous blocks by hashing them, for use in proof of work systems and for use in authenticating an action taken on an account (such as through use of a key to prove said action). We use SHA-2 512 in hashing on the network, SHA-2 512 is an industry standard hash algorithm with wide supported, recommended by the United States Federal Government through Federal Information Processing Standards Document 180-4 (Dang 2015).

7.2.1 Encryption

For encryption, we use Google Tink. [19] Tink is a library that Google describes as "secure, easy to use and harder to misuse". Tink is good for our use because it is easy to use and makes it harder for us to make mistakes that would expose our application to critical security vulnerabilities. For encryption, we also need to select an algorithm to use, for Public-Private key encryption we use the Elliptic Curve Digital Signature Algorithm (ECDSA) with a 256 bit key. This is comparable in security to an RSA key of approximately 2048 bits (Blake-Wilson et al. 2006). For symmetric encrpytion we use AES 256 which is comparable in security to an RSA key with 15,360 bits (Blake-Wilson et al. 2006).

7.3 Bootstrapping

Bootstrapping is a hard problem with block lattice networks, we took notice of how this is done in the Nano network to create our bootstrapping solution (LeMahieu 2018). In our system, there are two "systems" running at all times, the Real-time network which is watching transactions across the network and looking for conflicts to vote on and resolve and the bootstrap network which runs in the background. It is important to note that if a node was not there at the time to observe a transaction, the only way to verify it is legitimate is to announce it to the network again to see if any node objects. When a client wishes to bootstrap, they select some nodes at random and ask for their "frontiers", frontiers are the heads of recently updated chains on the block-lattice. The nodes reply with

[19] `https://github.com/google/tink`

proposed "frontiers" for the client. From each of these, the client then asks nodes on the network for the head blocks on these frontiers. The client then verifies these blocks by broadcasting them on the network to check for objection. If there is no objection, the client can then confirm these head blocks, importantly, if these head blocks are valid, then that also shows that every block under these head blocks is also valid, and hence the client can now download every block under this head block and as long as those blocks are properly referenced by the head they can accept them as valid without having to verify them.

7.4 Interface

Our network needs a User Interface, for this purpose we take advantage of the HTTP Server used by the Cluster layer network in order to display a web panel in a browser. There is no other user interface.

7.4.1 Web Panel

The web panel is accessed by the user by navigating to `my.dscript.site:port`.[20] my.dscript.site is a DNS A record that points solely to 127.0.0.1 (the local machine), hence a connection to my.dscript.site:port is a connection to 127.0.0.1:port. The web panel is then located under the path /ui. In order to prevent attackers accessing the panel, authentication is needed, when the user is sent to the panel by the application opening their browser, the application includes an authorization parameter in the query string which is also saved in the application. Once this is seen by the panel a cookie is inserted with another random token to maintain authorization and the user is redirected to remove the authorization token from their query string. Inside this panel the user can access features of the node such as account creation, transfers and the Name Service through clicking on links in the panel as shown in Figure 2. There are also certain hidden pages allowing debugging of the application, such as visualizing certain chains on the block-lattice as shown in Figure 3.

[20] We use this domain name rather than localhost because the Chrome browser does not allow cookies on localhost but does on a domain, hence this is a workaround to that problem.

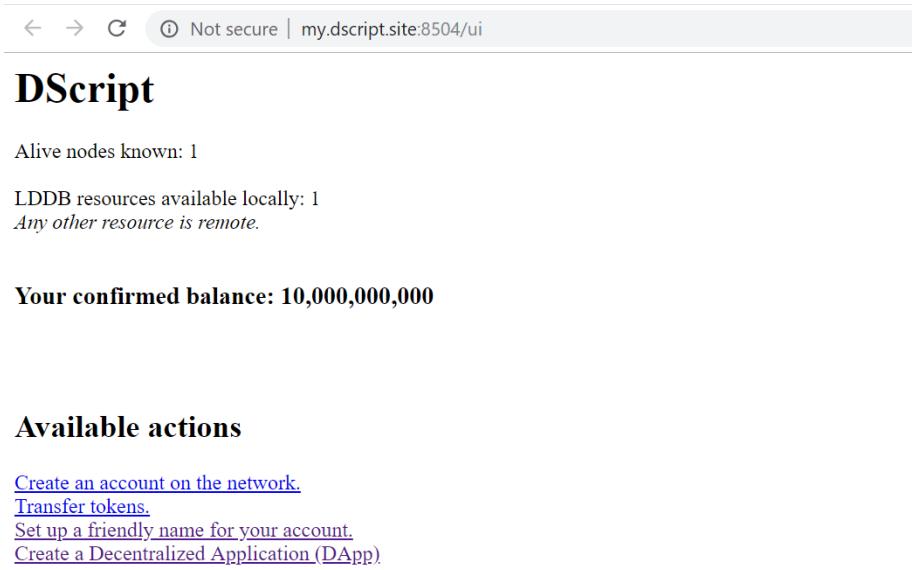

Figure 2: Screenshot from the Web Panel home page showing some of the features available.

Figure 3: Screenshot from the Web Panel showing a hidden page that allows exploring the chain of certain accounts on the block lattice. In this example we are the genesis account and have transferred tokens to 3 others.

7.5 Caveats

There are certain caveats to the current application, the most noticeable of which is that every node must be able to forward a port to run on otherwise they will not be able to receive messages. This is a feature that we could improve in the future through systems such as web sockets or holding messages for nodes to query for at a later time.

8 Evaluation

We have described how the DScript Protocol and Consensus Mechanism operates on the network, we will now evaluate the performance and security of the protocol.

8.1 Committee Consensus Performance

We will now discuss the performance of the committee consensus process of the DScript Protocol by itself, to clarify, this is the part of the protocol that deals with how a committee can agree a decision after that committee has been selected. The Committee is composed of n members who each have their own copy of the problem p_0 through p_{n-1}. In a basic sense, to come to agreement, each member of the committee computes $r_i = solve(p_i)$ which over the entire network has a rate of complexity of $O(n)$ where n is the committee population. In general, we can say that for every action of the committee, given the traditional cost of that action (that is, the cost for a single computer to execute it) c, the cost for the committee to execute it is nc, because every member of the committee must execute the action to verify the truthfulness of the other members of the committee. Indeed, it is important that we make the point that no member of the committee inherently trusts another member of the committee solely due to their membership of the committee. Hence, every action must be verified and incurs a performance penalty equal to the committee population.

8.2 Committee Consensus Security

One of the most important mechanisms that is required to balance availability of the committee when faced with attack versus the trust

that can be placed in decisions in the committee is the decision criteria. The decision criteria are the set of criteria that must be met for the network to accept a decision of the committee. In DScript, we lean heavily towards more security, our decision criteria is **unanimity**. That is, every member of the committee must agree with a decision for it to be accepted by the network. We recognize that this will allow an attacker to prevent a committee reaching a decision even if they have one member, but we counter it as necessary for these reasons:

- The same principle applies in a reverse, even if every member of a committee bar one is a cooperating attacker, that one member will be able to veto actions of the attack.
- An attacker cannot hope to prevent a decision on a fact forever, only delay that. Recall that we have previously said:

  > To convert the commitment into transfers, the committee must include come to a decision

  because of this, a user may simply repeat their request and form a new committee until the attacker is excluded. An attacker can't do this in reverse, because if an honest committee forms and agrees, then they cannot request a new one as the funds they used are now spent.[21]

8.3 BCRA Performance

The rate of performance of BCRA cannot be described in terms of algorithm run time because BCRA rounds run for exact set intervals. Therefore, the best way to describe BCRA performance is in terms of how early an item must be broadcast in order for it to be included in the list. Let's measure the performance therefore in relation to the amount of times a message must be transmitted between two nodes before it is accepted into the network. Such that the cost c of the message is $c = \sum[transmission]$ where a *transmission* occurs when any node transmits the message to any other node.[22] For all

[21] This is covered in more detail in the Overall Security section. It is important to note that an attacker with a very large stake may be able to overcome this, but the stake required is exceptionally hard to obtain.

[22] For the avoidance of doubt, this means that if a node broadcasts a message to n nodes, they perform n transmissions

cases, the amount of transmissions done increases at a rate in line with the population of the network such that for c, the rate of complexity is $O(n)$ where n is the network population. We will, for the purposes of this evaluation, assume that the stake of the network is distributed approximately equally across the network, we believe we are reasonable in making this conclusion because in a decentralized system using Delegated Proof of Stake those delegating their stakes would not wish for any one user or group of users to hold disproportionate power in the network because this would increase the risk to those user's stakes that those nodes could be convinced to act in a malicious way. We would also point out that any while it is unlikely there will be perfect balance across the network, slight variations on this level of stake would have only a minor effect on the performance described herein. Therefore, we say that for all n nodes $stake(n) = t/n$ where t is the total stake across the entire network. We should now consider when we can confidently say that a list will be selected. In all cases, an object is guaranteed to be selected if across the network the stake attributed to the object is at least 50%+1 of the entire stake of the network. As we have distributed the stake equally, we can therefore say that the object needs to reach 50%+1 of the nodes or $(n/2) + 1$, to transmit this far, we can remove 1 transmission as it is the first node to broadcast the object and therefore inherently knows of it, and therefore we must make $n/2$ transmissions. We should also consider how quickly these transmissions will be made, to consider this let's represent the nodes as a connected graph where the edges attached to every node are about equal. We can say that in this graph the rate at which the message propagates from any node is exponential, that is, it increases by a power of 2 for every round where a round is 1 interval where every node that is aware of the message transmits it across every edge they have, therefore, the number of rounds needed to propagate this message to half the graph, given the amount of nodes on the graph n is $(log_2 n) - 1$.

8.4 BCRA Security

We should now consider whether BCRA is secure. In order to determine whether BCRA is secure we should determine the ways in which an attacker could hope to gain a material advantage with the

BCRA system, there are two ways this could happen:

- The attacker causes a list to be populated with nobody except attackers.
- The attacker invalidates a previous list.

For the first attack this is impossible without a 100% stake, this is because regardless of the stake held by the attacker assuming that stake is less than 100%, any honest node may take the attackers list, append some honest items with a stake and broadcast it. This list will, obviously, have a higher stake than the attackers because it will have the attackers stake as a minimum and then any stake held because of any honest nodes being known by any honest nodes. The second attack is possible once the attacker holds a 50%+1 stake in the network and can occur during the resolution phase where the attackers confirm a list which wasn't decided on, that is, in a worst case scenario BCRA security fails when an attacker has a 50%+1 stake.

8.5 Multiparty Pin Seeding Performance

We will now discuss the performance of Multiparty Pin Seeding ("MPS"). Pin Seeding completes when a list is created where the weight of the list is at least 60% of the total weight, therefore, Pin Seeding is dependant on how fast a pin is signed by a cumulative amount of nodes that hold a 60% stake of the network. We will make the same assumptions made when analyzing BCRA performance, that is:

> We will, for the purposes of this evaluation, assume that the stake of the network is distributed approximately equally across the network
>
> ...
>
> let's represent the nodes as a connected graph where the edges attached to every node are about equal.

Our justifications for these assumptions are given in the BCRA Performance section. Based on this, we can express the performance of the Multiparty Pin Seeding algorithm as an expression of how many transmissions must be made until a seed is created that is valid. A

transmission has the same meaning as is given in the BCRA Performance section, that is a transmission is the sending of one message from one peer to another peer.[23] As the stake is distributed equally, we can say that the object needs to reach 60% of the nodes or $0.6n$. To transmit this far, we remove 1 transmission for the first node to create the pin and we therefore can say we make $0.6n - 1$ transmissions.

We should now further consider how quickly the transmissions are made, we shall introduce the same idea of a *round* given in the BCRA Performance section, that is, a period in which every node that wishes to make a transmission makes a transmission. Based on the assumptions we have made, we can say that for every *round*, the amount of transmissions increases exponentially as does the amount of nodes who have seen the pin. Importantly, unlike BCRA however, the pin must be passed node to node, such that in reality every node must be aware of every change to the pin. This is a limitation on the performance of Multiparty Pin Seeding which is necessary because each signature signs every signature before it as well as the initialization vector. Therefore, we must first determine how fast a single update to the pin can be seen by a node on the network that can make its own update. To determine this, we must first consider that a node can only sign a pin once, that is, as the pin grows certain nodes will be unable to sign it as they have already signed the pin. We can express the proportion of nodes able to sign a pin with a stake s as $available(s) = 1 - s$ because every node that has stake may sign the pin. We should now express how many rounds it will take for a network of n nodes to, on average, find a node that is available. To do that, we should consider the graph of the network itself, in this graph, nodes nearest to the current longest pin (the "head") are the most likely to themselves have signed this pin (because the holder of the head likely saw the updated pin from them as the nearest neighbours). Therefore, we should express the nodes that may sign the pin as, in a worst case scenario, the nodes furthest away from the holder of the head. To do this, we must first express how many rounds can be performed over a network by considering the exponential propagation of transmissions over rounds. In each round, the approximate number of nodes aware of any message increases by a power of 2. Therefore, we can say

[23] A broadcast to n peers is n transmissions, one for each peer.

that, given a network of n nodes, the amount of rounds to reach the entire network is $log_2 n$. We should now identify the population of a network of size n which is likely able to sign the pin given the stake already in the pin s. Based on the assumption of about equal stakes given above, this is $a = n * available(s)$. Given the total population and the population available to sign the pin we can express the number of rounds needed to reach this subset of the population as $\lceil log_2(n-a) \rceil$. Finally, we can say that assuming around equal stake, the amount of signatures in the pin should be $0.6n$ as this represents around 60% stake.

Therefore, we can say that the total number of rounds, given an amount of nodes n, is:

$k = 0.6n$

$totalRounds = \sum_{i=0}^{k} [\lceil log_2(n - (n * (1 - i))) \rceil]$

8.6 Multiparty Pin Seeding Security

Recall that the objectives of Multiparty Pin Seeding are:

> to create an algorithm whereby given some input content i the algorithm produces an output which is unpredictable for any one node or group of nodes that hold less than 50%+1 of the network stake, but is immutable in that it is the only valid output and can be verified as such

We can break this down into two objectives of the algorithm and then assess them independently, these are:

- (that) the algorithm produces an output which is unpredictable for any one node or group of nodes that hold less than 50%+1 of the network stake.
- (that the algorithm is) immutable in that it is the only valid output and can be verified as such.

8.6.1 Unpredictable output

The first part we should consider is whether the output is unpredictable. It is important to first specifically define what we mean by unpredictable, it is true that an attacker may be able to influence part of a pin, naturally, by including their signature in it. However, what we mean by unpredictability is that at least one bit in the pin

will be completely unknown to an attacker. We acknowledge that a single bit is not enough unpredictability to deter an attacker, hence we shall first define a parameter known as the "bit threshold" used to test whether a pin is unpredictable, that is, an attacker should not be able to predict at least "bit threshold" bits.

8.6.2 Bit Threshold

To decide on this threshold, we should consider the objective of the attacker. An attacker wants to predict the output of the pin and then use this knowledge to change the portion of the pin they can influence in order to produce a pin that seeds the psuedorandom number generator to select attackers, making the selection process rigged in favour of the attacker. Hence, in order to do this an attacker needs to conclude with certainty the contents of **every bit** within the pin. This is because the attacker needs to publish their segments of the pin, influenced to produce the seed they wish, during pin construction. Hence, an attacker cannot wait for each node where there is an unknown value to send that value or the attacker will be unable to add to the pin themselves. We should first therefore consider the degree of certainty an attacker can have in their conclusion given a certain bit threshold. Every bit the attacker is unsure of has a 50% chance of being a 0 or a 1 respectively, hence, the chance of the attacker guessing the entire bit threshold b, is 0.5^b. We can graph the chance of an attacker guessing the entire bit threshold therefore as shown in Figure 4. From this, we can say that the chances of an attacker to predict the output if 8 bytes are unknown is 0.5^{64}, which is a 1 in 18,446,744,073,709,551,616 chance. We consider this a reasonable enough threshold to deter any attacker. Therefore, our bit threshold shall be 64.

Pin Definitions Now that we have defined how many bits we must protect from being predictable by the attacker we must determine the parts of the pin that could be made unpredictable. We will explicitly rule out the initialization vector as a source of unpredictability, because a system could use a predictable initialization vector (such as counting upwards from 1, or a time-stamp). Therefore, the only source of unpredictability is the signatures on the pin. To determine how many bits each signature contains we will need

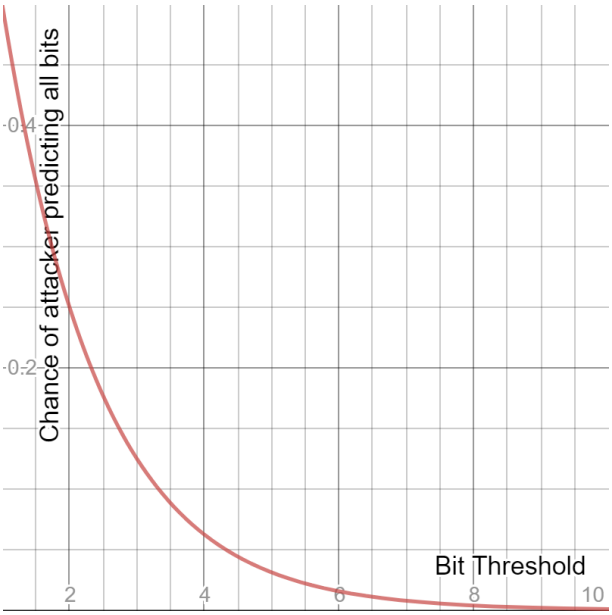

Figure 4: Graph of the chances of an attacker's success in predicting every bit within the bit threshold for a given pin.

to specify a cryptography system that these signatures use. To do this, we will define the system used as the system that is used in the DScript Java Client, that is ECDSA_P256 [24]. In ECDSA_P256 the length of the signature is 64 bytes (Adalier et al. n.d.), far beyond our bit threshold of 64 bits. Therefore, we can say at least one signature in the final pin must be unpredictable to the attacker in order for the system to be secure.

Necessary stake to defeat Recall that for a pin to be valid it must have at least a 55% stake, therefore, an attacker cannot defeat Multiparty Pin Seeding without holding at least a 55% stake in the network.

8.7 Overall Security

First, let us review the security of the DScript protocol. To do this, we will consider the scenario for an attacker when attacking the

[24] ECDSA_P256 means Elliptic Curve Digital Signature Algorithm (Adalier et al. n.d.) and is implemented through the Google Tink libraries.

DScript protocol. Let's introduce a state transition machine s, in state 0, where the state transition function, f, contains $f(s_0, 1) \rightarrow \{s_2, 1\}$. Now, let us say an attacker wants to trick the network into certifying a state transition of $f(s_0, 1) \rightarrow \{s_1, 1\}$,[25] even though no honest committee should ever conclude this. First, let us review the overarching Delegated Proof of Stake system. This system is vulnerable to attack if the attacker holds any more than 50%+1 of the stake of the network, as at this point the attacker may attempt to retroactively revert legitimate state transitions by voting them away in conflicts, they may do this until they manage to create the circumstances they desire. This is similar to the 50%+1 attack on Proof of Work systems described in Nakamoto et al. 2008. Assuming that honest nodes hold at least 50%+1 of the stake, the avenue for the attacker to break the system is through an evil committee, an evil committee is a committee of nodes where every node is a cooperating attacker. This committee may certify any decision they wish, including the otherwise illegal $f(s_0, 1) \rightarrow \{s_1, 1\}$. To consider whether an attacker will succeed with this attack, we must define the state in which an attacker succeeds. An attacker succeeds in their attack if every member of the committee is a cooperating attacker and fails if any member of the committee is not an attacker, this is because regardless of attacker population, any one member of the committee may block the agreement of the committee. We should now consider the chances of the attacker to fill the committee with exclusively cooperating attackers. To do this, we must consider the stake of the attacker in proportion to the entire stake of the network, s, where $0 \leq s \leq 1$ and the size of the committee. In DScript, the size of the committee is always up to 10 nodes, so that is what we will be considering. It is also important to note that nodes are not withdrawn from being selected when included in the committee, but may be included any number of times, but will only be a member once, this is described above in the random selection algorithm section, and is to prevent an attack when the network is in a state where few honest nodes hold large stakes. Therefore, the chance of an attacker succeeding in populating the committee with only cooperating attacks is:

$$chance = s^n$$

[25] This is an illegal state transition but as not every node executes the state transition function and instead relies on the result of the committee, the network may not be aware of the fact that this state transition is illegal.

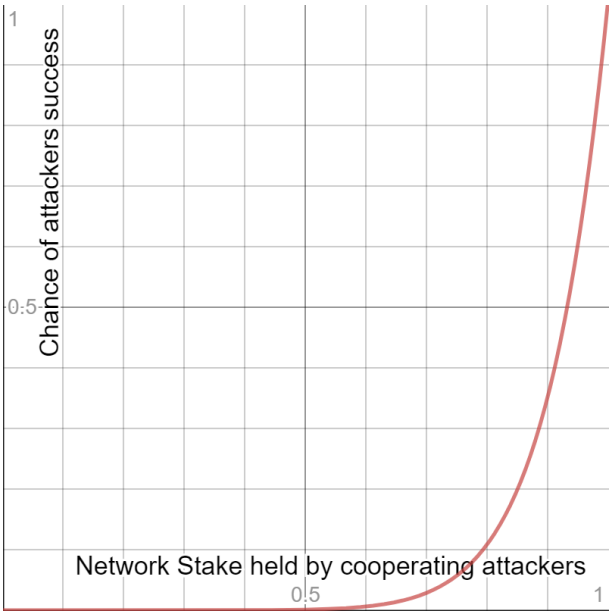

Figure 5: Graph of the chances of an attacker's success in gaining an evil committee based on stake.

Where n is the amount of members to be selected for the committee.[26]

Therefore, we can graph the chances of an attacker succeeding given their stake in the network as shown in Figure 5.

We can reverse this calculation in order to consider how much stake is necessary for a given chance at an evil committee, c, as:

$stake = \sqrt[10]{c}$

Based on this, we can see the stake needed for a 50% chance of an evil committee is $\sqrt[10]{0.5}$ which is 93.3% of the network. Further, if the attacker holds a 50% stake in the network the chances they produce an evil committee are 0.5^{10} which is a 0.097% chance. The lower the stake the lower the chances, with a 10% stake the chance is 0.00000001%. This is an unreasonable barrier for any attacker, as to break this system would require a sizable stake in the network. Further, if this stake was obtained, an attacker may be harming themselves by attacking the network with it, in LeMahieu 2018 Colin

[26]The amount of members is set to a goal of 10, but if any node is chosen twice, they are only included once, hence the committee may be any number of nodes from 1 to 10. Every time a node is selected after it's first selection the committee size decreases by 1.

Lemahieu stated on the possibility of an attacker taking a large stake of the network:

> [those with more stake] ... are inherently incentivized to maintain the honesty of the system or risk losing their investment.

8.8 Overall Performance

We will now discuss the performance of the protocol. To consider the performance of the overall protocol, we should recall how the overall protocol operates. The DScript Protocol operates by selecting a set absolute number of nodes (in our Java client, 10) who will certify a result on behalf of the entire network. One of the most important points to make in this regard is how our algorithm does not grow in complexity as the network population increases, in comparison to many other cryptocurrency networks. The amount of nodes selected to perform expensive tasks in our network is always 10, hence, for DScript, performance is indicated as $O(1)$. This means our network can scale practically forever without consequence to the underlying protocol of the network.

8.9 Network Performance vs Ethereum

We will now compare the performance of our system versus the performance of a State of the Art system, in this case Ethereum. Ethereum is a decentralized computation system that uses a Proof of Work Blockchain to secure itself. [27] Ethereum Decentralized Applications are written in EVM byte code and run on the Ethereum Virtual Machine, a highly deterministic environment designed specifically for the Ethereum network. The system works largely the same as systems like bitcoin described in Nakamoto et al. 2008 for consensus, that is, using Proof of Work and difficulty to choose a chain. In Ethereum, a Decentralized Application is created by including it in a block on the blockchain and is made of a "state" which comprises the state of the Decentralized Application at a given point in time and is made of the most recent transaction that modified the state of the Decentralized Application as well as the EVM Source of the

[27] We would be amiss to point out that Ethereum is currently introducing a protocol called Casper that uses Proof-of-Stake.

application. For example if I have a state transition machine with state transition function $f(s0,1)->(s1,1)$, my decentralized application is in state $s0$ and I wish to input 1, I create a transaction for the decentralized application that references my already published DApp on the network and includes my input, in this case 1. This is then included in a block. When any node gets this block they compute all inputs to decentralized applications contained therein, that is, every node then computes that $f(s0,1)->(s1,1)$ and they update the state of the application in their memory accordingly. The problem with this is efficiency, naturally, we want to avoid running a piece of code more times then necessary yet in Ethereum every single node must run every single input to every single DApp, this is, predictably, very inefficient. The efficiency can be expressed in reference to the amount of times the state transition function is executed by the network, given a number of nodes n, as $t = n$. Importantly from this, we can see that the rate of complexity of execution of a DApp increases at a rate of $O(n)$ on the Ethereum network.

Now let us talk about the performance of the DScript system. DScript applications are written in their own language, based off the Ethereum EVM, called DDL (Deterministic Distributed Language), while DDL has some changes from the EVM they are in essence achieving the same objectives, creating a low level deterministic environment for execution of software. DScript applications further have the same idea of state being recorded onto the blockchain. Where DScript differs however is how state is recorded, let's recall our example with Ethereum we gave above of a state transition machine with the function $f(s0,1)->(s1,1)$. In DScript, the execution of this state transition function is delegated to a committee of 10 nodes broadly representative of the nodes in the network by stake. The block recorded in the blockchain is the inputs and unlike Ethereum, the outputs of the execution of those inputs, certified by the members of the committee and with the proofs with regards to selection of those members (a reference to the BCRA List and the pin from Multiparty Pin Seeding). Instead of executing the state transition function themselves, each node instead simply verifies that the committee is properly formed and unanimously decided, and if so they accept the output without executing the input themselves. In our system, no state transition will be executed more than

10 times, $t \leq 10$.[28] For the purposes of this document, let us assume that the state transition is executed the maximum number of times, that is, 10.

We can now use the values we have calculated that reference the efficiency of each solution by calculating the amount of times that state transition function is executed overall by the entire network, t in both cases, to determine whether DScript has performance improvements over Ethereum and if so, by how much (and this value shall be referred to from hereon as the "score" for each network). Recall that we stated that the rate of increase of complexity and therefore the score in Ethereum is $O(n)$, therefore, we will compare the rate of complexity at different baselines. We shall assume the least efficient scenario for DScript, that is, 10 executions (except where this is impossible in which case we assume every node). The score therefore for Ethereum is $ethScore(n) = n$ and for DScript is $dscriptScore(n) = min(n, 10)$. In both networks, the goal is increased efficiency and so to minimize the score. We can see the results of the calculation of the scores for each network in Figure 6.

We can see that efficiency is about equal until we exceed 10 nodes, at which point the rate of increase of efficiency of DScript versus Ethereum is equal to the amount of nodes in the network minus 10. We can now calculate the improvement in efficiency in DScript as $improvement = \frac{ethScore(n)}{dscriptScore(n)}$ and we can graph that as shown in Figure 7.

As of the time of writing this document, the Node Population of Ethereum is 8,718 nodes, [29] based on this, we can see the increases of performance based on the current Ethereum node population is $\frac{ethScore(8718)}{dscriptScore(8718)}$ which is $\frac{8718}{10}$ which an improvement of 871.8 times. This is a significant improvement in efficiency versus the Ethereum network.

8.10 Network Security vs Ethereum

We will now describe the security of our network versus Ethereum. Ethereum Security is based fundamentally on Proof of Work in the same way as bitcoin, it fails if an attacker holds more than 50%+1 of

[28] A state transition may be executed less than 10 times as the same node may be selected to be included in the committee many times as explained previously in this document.

[29] Sourced from `https://www.ethernodes.org/network/1` at 08:54 GMT on the 20th of March 2019.

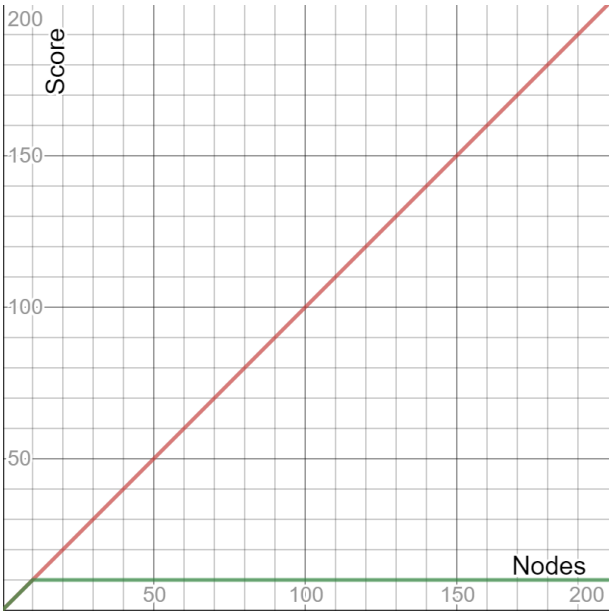

Figure 6: Comparison of the efficiency "score" of Ethereum versus DScript, the lower line is DScript and the higher line is Ethereum. The higher the score the lower the efficiency.

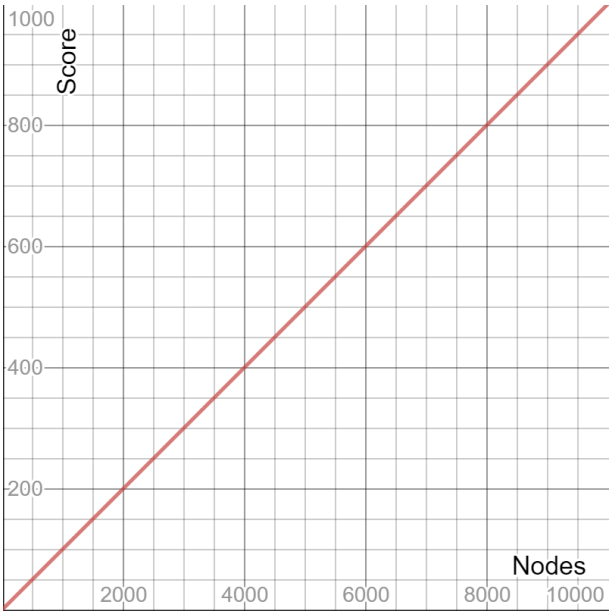

Figure 7: The rate of improvement in efficiency of DScript versus Ethereum based on the efficiency score values described in Network Performance vs Ethereum.

the computation power of the network. Ethereum however can never suffer an attack where an attacker manages to convince the network of an illegal state transition because every node on the network executes the state transition themselves in isolation. In DScript, we have introduced a tiny chance of this happening and have used it to reap performance benefits, in Ethereum the chance of an illegal state transition being accepted is 0%. In DScript, the network would accept an illegal state transition if an entire committee randomly selected agreed to it, assuming that our random seed algorithm and list population algorithm are secure (as described in previous sections) we can express the chance of an illegal state transition being accepted as the cooperating attackers stake in the network s to the power of the amount of nodes in the network n, such that the chance c is $c = s^n$, we previously described above how n is always 10, hence the chance is $c = s^{10}$, as shown in Figure 8. We can therefore say that the stake needed by an attacker for a given chance c is $s = \sqrt[10]{c}$. From this, we can see that for an attacker to gain a 50% chance of defeating the network they would need to hold 93.3% of the stake

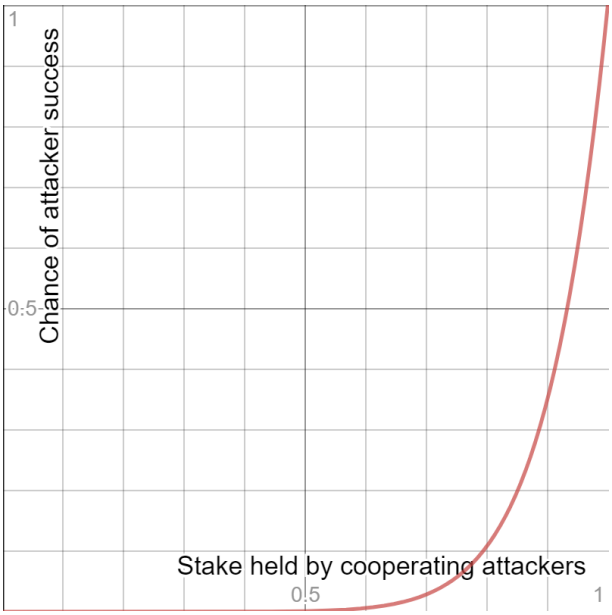

Figure 8: Chance of an attacker succeeding in convincing a committee of an illegal state transition.

in the network, this is an unreasonable barrier for any attacker. By introducing and then mitigating a small risk into the algorithm, we have gained very large performance benefits.

9 Conclusion

We will consider whether we have met our overall aim by individual evaluation of our objectives, we will then take into account our progress on each objective to identify whether we have met our aim. First, let us consider what we have achieved with this project, we have independently developed 4 algorithms. The first algorithm we developed was BCRA, which is novel and for the first time allows nodes to come to rapid consensus around very large structures of lists. The second algorithm we developed is Multiparty Pin Seeding which joins a list of many algorithms that occupy the area of Multiparty Coin Toss algorithms such as Beimel, Omri, and Orlov 2010. The third algorithm is Lightweight Distributed Database, a simple algorithm that saves nodes on our network disk space and our fourth

algorithm, is the DScript Algorthim and Protocol that allows creating a Decentralized Computer with **a rate of complexity increase in comparison to network population of O(1).**

We will now review each of our objectives.

> Undertake a background study to identify existing work, identify the systems used by the state of the art and identify the caveats in performance of the state of the art.

We undertook a background study and identified existing work, this is shown in depth in the Background section of this report. We further identified where systems were used by the state of the art and the caveats to their use.

> Identify an approach which will give us results from which we can draw rigorous conclusions.

In our Project Approach section we laid down the approach we would take to this project and how we would validate our approach throughout.

> Design an algorithm that allows a decentralized network of untrusted nodes to come to consensus around the end state of a state transition machine given a start state and input and that has a performance improvement over the state of the art.

We designed the DScript Protocol in the DScript Protocol section, a protocol for communication and agreement in a decentralized network based on Delegated Proof of Stake and Psuedorandom Selection to ensure security. The DScript Protocol is one of few protocols that achieve consensus in O(1) based on network population through a combined system that designates an authority for a Proof of Authority execution (other protocols that do this include Gilad et al. 2017). And we are the first protocol to apply this to decentralized applications in a practical setting.

> Design and implement software that shows the algorithms described above in operation and shows the viability of the algorithms in a solution.

We wrote the DScript Java Client, a program that implements the DScript Protocol, the DScript Java Client worked as expected across

a simple network and this demonstrates the algorithms as viable. While the client still requires some work to be completely finished, we are satisfied that it implemented the core of our protocol.

> Demonstrate the security of the algorithms described above through empirical evaluation of such.

We evaluated the Security of the DScript Protocol in the Performance and Security in Theory Section of this report, we considered each major component and the overall security of the protocol.

> Demonstrate the performance improvements of the algorithms described above versus the state of the art through empirical evaluation of such.

We evaluated the Performance of the DScript Protocol in the Performance and Security in Theory Section of this report, considering each major component and the overall performance improvement of the protocol versus a state of the art protocol.
We have demonstrated the DScript Protocol's Performance and Security through the Performance and Security evaluation we performed, based on this, as well the fact that we have met all of our objectives [30]. Further, we have created an algorithm that is much faster than the State of the Art (see Network Performance vs Ethereum). Based on all of this, conclude that we have met our overall aim.

9.1 Future Work

I believe that there are many further areas to explore to increase the performance and security of this system and it's privacy. I will now talk about a few of the different areas I did not have the opportunity to fully explore during this project but I plan to explore in the future.

9.1.1 Verifiable Random Proofs

In Gilad et al. 2017 it is shown that random committees can be formed through Verifiable Random Proofs, this has a benefit over our system in that in this system the members of the committee are a secret until they have made their decision. This protects against

[30] Our last objective, to "Evaluate the success of this project based on the security and performance of the algorithms as described above." is performed in this section.

attackers influencing or attacking members of the committee as they make their decision. Although I have done no analysis of it as it is outside the scope of this project, I think this would increase overall security.

9.1.2 Privacy

One of the caveats of our system is that in our system, every node sees every invocation or execution of a program. This means that the system offers no privacy to it's users. I think that there may be some approaches to ensure that systems can be run on the network without disclosing information about them, and I will describe the two areas I am interested in exploring in this area below.

Homomorphic Encryption In 2019 Microsoft introduced SEAL. [31] SEAL is an "easy-to-use homomorphic encryption library" (Chen, Laine, and Player 2017), Homomorphic Encryption is described by Microsoft as "a new type of encryption technology that allows computation to be directly on encrypted data, without requiring any decryption in the process". This could be used in our network to protect user privacy by allowing operations to be performed on inputs that are encrypted in their entirety before being run on the network. In this way, user privacy is preserved while we can prove that the program acted in a certain way on the encrypted data.

Compartmentalization I had the idea for this sort of approach when reading about the US Town of Oak Ridge, Tennessee. During the Second World War, Oak Ridge was the site of part of the development of the first ever nuclear weapons under the United States Manhattan Project (Olwell 2004). Due to the immense secrecy around the creation of this weapon, the United States Federal Government needed to ensure that information about what was being constructed at Oak Ridge was not revealed to the enemy. In order to make revealing this information harder, the United States instrumented a policy of Heavy Compartmentalization. Heavy Compartmentalization at Oak Ridge was a policy that each worker by themselves knew very little about the overall work of the system due to their role being exceptionally limited in both task and information granted to

[31] `https://www.microsoft.com/en-us/research/project/microsoft-seal/`

them. For example, one employee was simply given the instructions of "hold this device up to these clothes and if you hear many clicks tell this person". The worker knew not why they were doing this or what the device was, of course, we now know the device was measuring radiation. This idea of giving many people small jobs that hold no meaning by themselves but collectively cause something complex to happen is an approach I believe could be explored for Execution Secrecy. I believe it would be possible to develop an algorithm where so long as a certain proportion of the nodes failed to cooperate in discovering the overall secret, the overall operation being performed by the nodes would remain a secret to everyone except the person who wished it to be executed (the "coordinator").

References

Adalier, Mehmet et al. (n.d.). "Efficient and secure elliptic curve cryptography implementation of Curve P-256". In:

Ateniese, Giuseppe et al. (2014). "Proofs of space: When space is of the essence". In: *International Conference on Security and Cryptography for Networks*. Springer, pp. 538–557.

Beimel, Amos, Eran Omri, and Ilan Orlov (2010). "Protocols for multiparty coin toss with dishonest majority". In: *Annual Cryptology Conference*. Springer, pp. 538–557.

Blake-Wilson, Simon et al. (2006). *Elliptic curve cryptography (ECC) cipher suites for transport layer security (TLS)*. Tech. rep.

Buterin, Vitalik (2013). *What Proof of Stake is and why it matters*.

Buterin, Vitalik and Virgil Griffith (2017). "Casper the friendly finality gadget". In: *arXiv preprint arXiv:1710.09437*.

Chen, Hao, Kim Laine, and Rachel Player (2017). "Simple encrypted arithmetic library-SEAL v2. 1". In: *International Conference on Financial Cryptography and Data Security*. Springer, pp. 3–18.

Dang, Quynh H (2015). *Secure hash standard*. Tech. rep.

Dannen, Chris (2017). *Introducing Ethereum and Solidity*. Springer.

De Angelis, Stefano et al. (2018). "Pbft vs proof-of-authority: applying the cap theorem to permissioned blockchain". In: *Sapienza University of Rome, University of Southampton*, pp. 3–6.

Douceur, John R (2002). "The sybil attack". In: *International workshop on peer-to-peer systems*. Springer, pp. 251–260.

Dziembowski, Stefan et al. (2015). "Proofs of space". In: *Annual Cryptology Conference*. Springer, pp. 585–605.

Gilad, Yossi et al. (2017). "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, pp. 51–68.

Grigg, Ian (n.d.). *EOS-An introduction*. URL: `https://cryptomonday.de/wp-content/uploads/2018/04/EOS_An_Introduction.pdf`.

Hertig, Alyssa (2017). *Ethereum + Lightning? Buterin and Poon Unveil 'Plasma' Scaling Plan*. URL: `https://www.coindesk.com/ethereum-lightning-buterin-poon-unveil-plasma-scaling-plan`.

Kiayias, Aggelos et al. (2017). "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer, pp. 357–388.

King, Sunny and Scott Nadal (2012). "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake". In: *self-published paper, August* 19.

Krejcie, Robert V and Daryle W Morgan (1970). "Determining sample size for research activities". In: *Educational and psychological measurement* 30.3, pp. 607–610.

Lamport, Leslie, Robert Shostak, and Marshall Pease (1982). "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3, pp. 382–401.

LeMahieu, Colin (2017). "Raiblocks: A feeless distributed cryptocurrency network". In: *URL https://raiblocks. net/media/RaiBlocks_Whitepaper__English. pdf*.

— (2018). "Nano: A feeless distributed cryptocurrency network". In: *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*.

Morgan, JP (2016). "Quorum whitepaper". In: *New York: JP Morgan Chase*.

Murphy, Hannah (2018). *The rise and fall of Ethereum*. URL: https://www.ft.com/content/a8d2c280-d2b6-11e8-a9f2-7574db66bcd5.

Nakamoto, Satoshi et al. (2008). "Bitcoin: A peer-to-peer electronic cash system". In: *bitcoin.org*.

O'Dwyer, Karl J and David Malone (2014). "Bitcoin mining and its energy footprint". In: *25th IET Irish Signals Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014), 2014 p. 280 – 285*.

Olwell, Russell B (2004). *At Work in the Atomic City: A Labor and Social History of Oak Ridge, Tennessee*. Univ. of Tennessee Press.

Parity (2019). *Proof-of-Authority Chains*. URL: https://wiki.parity.io/Proof-of-Authority-Chains.

Park, Sunoo et al. (2015). "Spacecoin: A cryptocurrency based on proofs of space". In: *2015: 528, Tech. Rep.*

Poon, Joseph and Vitalik Buterin (2017). "Plasma: Scalable autonomous smart contracts". In: *White paper*, pp. 1–47.

Poon, Joseph and Thaddeus Dryja (2016). *The bitcoin lightning network: Scalable off-chain instant payments*.

Popov, Serguei (2016). *The tangle*. URL: http://www.descryptions.com/Iota.pdf.

Ray, James and The Ethereum Authors (2018). *Hard Problems of Cryptocurrencies*. URL: https://github.com/ethereum/wiki/wiki/Problems.

Ren, Ling and Srinivas Devadas (2016). "Proof of space from stacked expanders". In: *Theory of Cryptography Conference*. Springer, pp. 262–285.

Ryan, Mark (2006). *Digital Cash*. https://www.cs.bham.ac.uk/~mdr/teaching/modules06/netsec/lectures/DigitalCash.html. (Accessed on 03/04/2019).

Saleh, Fahad (2018). *Blockchain without waste: Proof-of-stake*. URL: https://ssrn.com/abstract=3183935.

Sweet, Lucy (2019a). *Deterministic Distributed Language*. URL: https://docs.google.com/document/d/1hsDyAzdCx4PSHdfn86SM1sGgxkFhJmilMnYF5Pfjxnc/edit.

— (2019b). *DScript Network Design*. URL: https://docs.google.com/document/d/1Fzb32T_Bn5XNLGWLbL5koCKj3gD2YgAJHRsrXKvAjQU/edit.

— (2019c). *LDDB*. URL: https://drive.google.com/file/d/16Y4zzqieoLA9y6pCzmyTPPBFThurICB4/view?usp=sharing.

Wensley, J. H. et al. (1978). "SIFT: Design and analysis of a fault-tolerant computer for aircraft control". In: *Proceedings of the IEEE* 66.10, pp. 1240–1255. ISSN: 0018-9219. DOI: 10.1109/PROC.1978.11114.

Wood, Gavin (2014). *Ethereum: A secure decentralised generalised transaction ledger*. URL: https://ethereum.github.io/yellowpaper/paper.pdf.

10 Appendix A - Personal Reflection

10.1 Reflection on Project

I have undertaken a project jointly of research into new and innovative consensus mechanisms and development of practical software. My specific knowledge of this topic area pushed my decision to choose this for my Final Year Project as I have been working on this idea for quite some time, formerly under the name of "Sequestered Witness". Most of the problems I have encountered throughout this project have been technical in nature, at one point I have had to manually patch a third party library to ensure I could use it in my project. [32] If I had a chance to change how I undertook this project, I would have likely spent longer designing the project, as of the time of writing a team at MIT including Silvio Micali have proposed that they have found a solution for random selection that requires no consensus mechanism. I am very interested in this system and I regret not contacting them directly when I first started this project to see if there would be a chance to have an exchange of ideas with them about this area of research. The DScript Java Client does not implement the entirety of the DScript Protocol, however this would have taken too long for the timescale of this FYP. The Client already has 153 classes and around 10,000 lines of code. Further the client supports the fundamental consensus systems of the protocol. I intend to improve the client in the future to support the entire protocol.

10.2 Personal Reflection

While overall I am happy with how I have worked on this project and what I have delivered, I would have preferred to have delivered some of the extensions to the DScript Protocol in a working fashion which I have not had the chance to add to the software. While the core of the DScript Protocol works in the software, subchaining among other systems is highly unstable. I unfortunately did not have enough time to stabilize this feature. Further, I would have liked to have revisited how networking works on the DScript network, while I really like the system I made with the layers of Cluster,

[32] `https://github.com/perwendel/spark/pull/1092` - This patch is forced into my version of spark through Gradle by including my modification in my project files on compile time.

Message and Application I think there is room to make the Cluster and Message layers more efficient and to use less traffic in their operation. I should have done more research into Multiparty Coin Toss algorithms before making MPPS, MPPS is inferior to systems such as Beimel, Omri, and Orlov 2010. In the future, I may move to the system described in that paper.

11 Appendix B - Licenses

Uri locations of the licenses for all third party packages are given below:

- Google Guava
  `https://github.com/google/guava/blob/master/COPYING`
- JSON in Java (org.json)
  `https://github.com/stleary/JSON-java/blob/master/LICENSE`
- Spark - a tiny web framework for Java 8
  `https://github.com/perwendel/spark/blob/master/NOTICE`
- Apache HttpClient
  `http://hc.apache.org/httpclient-3.x/license.html`
- Apache Commons IO
  `http://www.apache.org/licenses/LICENSE-2.0.html`
- Apache Commons Validator
  `http://www.apache.org/licenses/LICENSE-2.0.html`
- Google Tink
  `https://github.com/google/tink/blob/master/LICENSE`
- Apache Freemarker
  `https://freemarker.apache.org/docs/app_license.html`
- Zxing ("Zebra Crossing") Barcode Scanner
  `https://github.com/zxing/zxing/blob/master/LICENSE`
- JUnit
  `https://junit.org/junit4/license.html`

- JUnit Toolbox

  `https://github.com/MichaelTamm/junit-toolbox/blob/master/LICENSE.md`

These locations were correct at time of submission.

12 Appendix C - Ethical Approval

[graphic redacted from public copy]

This was my Bachelors dissertation / Final Year Project to conclude my 3 year Computer Science degree at Brunel University London. I recieved an overall degree class of First Class Honours.