TITLE GOES HERE!!!!!!!!!!

'SUBTITLE GOES EHRE!!!!'

UNIVERSITY OF BIRMINGHAM

Matthew Flint

1247903 - mxf203@bham.ac.uk

This work was conducted as part of the requirements of the Module 06-26587 'Computer Science Project' of the Computer Science department at the University of Birmingham, UK, and is submitted in accordance with the regulations of the University's code of conduct.

9th April 2017

Contents

1 Abstract

In abstract, you should give an overall view of the paper including the importance and necessity of the paper topic (the first line of the abstract), the previous works and difficulties on this topic(second and third lines), the big picture of your novelty in this paper (fourth and fifth lines), and verifying your results (the final part of the abstract).

1. It should not exceed 250 words
2. It should be written in one paragraph.
3. It should be written in the past tense as it refers to work done.
4. Long words should be followed by its abbreviation which would be used through out the abstract and paper.
5. It should not cite any references (except in rare cases)
6. It should never give any information or conclusion that is not stated in the paper
7. Must be accurate with respect to figures quoted in the main text.

2 Introduction

Existing electronic voting systems all suffer from a serious design flaw: They are centralized by design, meaning there is a single supplier that controls the code base, the database and the system outputs while also supplying the monitoring tools to verify the result [1]. The lack of an independently verifiable system means that, once voters mark their ballot choice, they must place their trust in the organization, that their vote is recorded and counted as intended. The lack of an independently verifiable output, makes it difficult for these centralized systems to acquire the trustworthiness required by voters, which potentially limits voter participation, or cast doubt upon the published output of an election.

Despite the digitalisation of many aspects of modern life, elections are still being largely conducted offline, on paper [2], although the use of Electronic Voting Machines has been steadily growing over recent years. Paper ballots are the traditional tool for voting and are typically marked by a human (voter) and then tallied by a machine. While costing less than most electronic systems to run they rely on physical security and trust in polling stations to not manipulate and to properly handle them [3]. Postal votes also utilize paper ballots and are used to allow voters to not have to physically attend a location in order to vote. These also suffer from the same flaws as traditional paper ballots while increasing the opportunities for attack during their traversal through the postage system.

Voting systems comprise of five main components:

1. A registration service for verifying & registering legitimate voters
2. Voter authentication stations with the task of determining a voters authorization to vote based on the completed work of the registration service
3. Voting stations where the voter makes choices on a ballot;
4. A device called the ballot box where the ballot is collected
5. A tallying service that counts the votes and announces the results

Of the five main components listed above, only `(2)`, `(3)` and `(4)` are used during an election, with `(1)` being being required for an election to take place and `(5)` happening post election [4].

Current Electronic Voting (E-Voting) takes two forms; using a machine in a polling station, rather than a ballot paper and pencil, or casting a vote over the internet. The former tends to refer to a Direct Recording Electronic system which typically displays ballot options on a screen that can be activated by the voter and then records that voting data in memory components to be processed later. However, as with many electronics there is an inherent problem of the ability to modify software and potentially insert malicious code [3]. This has been an issue raised over several recent elections and a study from 2015 concluded that 43 American states would be using Electronic Voting Machines that are at least 10 years old during the last presidential election [5]. The latter, while having to deal with issues such as privacy, fraud, voting under duress, and corruption, does nothing to improve a voter's trust. The election as the voter must assume that once they have cast their vote it will be recorded and counted honestly.

In order for our proposed E-Voting system to be a tangible challenger to more traditional voting methods it must be able to provide the current systems services, at least at the same level but preferably with improvements, while also providing substantial benefits to justify adoption. There are several standard requirements that a voting system should adhere to, each holding

equal weight: security, functionality, privacy, usability, and accessibility [6]. A "secure" voting system means one that cannot be tampered with or manipulated in any way, ensuring that votes are accurately recorded as cast. It also ensures that additional votes cannot be cast after the polls have closed or tampered with at any stage of the process. System functionality can be broad but should include; The correct registering and recording of all votes cast, permitting a voter to vote for any candidate they have the right to vote for, allowing only eligible registered voters to vote & only allowing each voter to vote once. Voters have the right to a secret ballot and to cast their vote in private [6]. This is essential to protect voters from being coerced or bribed into voting a certain way, this means that our system should not provide a receipt or any way for another person to determine the contents of a voter's ballot. On top of this the system should be easy for voters to use, meaning it's as intuitive as possible, and maintain universal vote access. It should avoid introducing bias by selecting platforms that are more available to some groups than to others as the choice of the platform, language, ballot format, or devices may seem innocuous, but it may actually prevent small factions of the voters to cast their vote [7].

Whilst maintaining these essential foundations already provided in traditional voting systems, there are several improvements and additions which I intend to explore. The first benefit in using a blockchain to log votes is in its decentralized nature. This means there is less need for trust to be placed in a centralized organization where votes are hidden behind closed doors. It also has the benefit of being significantly harder to tamper with as, once a transaction has been verified, an attacker would need to possess at least 51% of the computing power of the network to attempt to forge transactions. Any attempts to otherwise use a forged block will be noticed by the rest of the network and ignored. This decentralized system also brings in more transparency as anyone can view transactions in the blockchain leading to higher levels of trust in the elections outcome. This is further strengthened by the independent verifiability which could be performed by anyone, therefore removing the need to trust the election organizers declaration of the outcome. Once a transaction (vote) has been confirmed in the blockchain (and has further blocks built upon it) this vote for a candidate becomes immutable, meaning that the entire outcome of an election will be stored indefinitely and is able to be accessed at any time in the future.

Verifiability properties of electronic voting are divided in two categories. "Individual verifiability", which involves auditing the processes of vote creation and vote storage by the voter; and "Universal verifiability", which ensures that only votes from eligible voters are stored in the ballot box and that all stored votes are properly tallied, which can be performed by anyone [8]. Systems providing both types of verifiability are known as "end-to-end verifiable systems".

One of the individually verifiable properties is cast-as-intended verification, which is focused on the audit of the vote creation process. Another property is recorded-as-cast verification, aimed at auditing the correct reception and storage of the vote in a remote voting server [8].

There has been some research conducted in this area already and several protocols for blockchain based voting have been proposed. *Pierre Noizat* proposes a system [1] where each candidate provides a unique public key, KeyC, to each individual voter along with a singular bitcoin address, AddressC, which the final sum of the voting transactions will be sent to. Each voter is also assigned an individual public key, KeyA, by the election organizers and; either they generate a Key Pair themselves (this could be done by the voting software for better usability), KeyB, or be assigned the KeyB by the organization. From these three public keys the voter can generate a `2-of-3` multi signature address which represents the vote of B in favour of C. This address is then funded with a bitcoin micropayment (around the price of a postage stamp), either funded by the voter or organization, and this is the voters confirmation of their vote. After a few hours this

ballot is securely logged on the blockchain and, as the multisignature address was funded, a voter can check that this address is represented in the blockchain and that their vote was registered. It should be noted that there is no way to guess neither the voter (B) nor the candidate (C) from a multisignature address without knowing all three public keys (KeyA, KeyB, and KeyC) and knowing to whom they belong. Once the election is concluded the organization is able to, via the `2-of-3` multi signature address, spend the coins the voter gave to the candidate to fund the address of the candidate, AddressC. This provides an unequivocal link between the vote and the candidate which can be seen and validated by anyone.

While the proposed method does provide a valuable alternative to current, proprietary electronic voting systems and has the benefits of protecting the secrecy of the ballots, allowing free, independent audits of the results and minimizing the trust level required from the organizers [1]. It does come with several drawbacks; the independent validation of votes before the organization funds a candidate's public address, AddressC, can only be done by the voting individual on their ballot choice. The protocols dependence on the Bitcoin blockchain could pose problems with subsequent elections as there is no definitive boundary between one election and the next. The currency units, although in very small denominations, could be transferred out of the election to private addresses for an individual's gain (though even if 100% of the vote currency was lost, the cost of an election would likely be less than if done using current methods).

Another proposal is that of Universal Cast-as-Intended Verifiability [8] which allows any party (not only the voter) to publicly verify that an encrypted cast vote really matches the selection of a voter. Their proposal allows a voter who's eligible to vote, to register with a registrar who then generates a pair of public-secret values for each voting option in the election. These secret values are sent to the voter, while the public ones are published, linked to the voting options they are related to. During the voting phase, the voter provides her selected voting options and a subset of the secret values she received during registration to the voting device. The voting device then encrypts the voters selections and creates a non interactive zero-knowledge proof, which will be valid only in the case that the voting device encrypted what the voter selected. Thanks to the zero-knowledge property of the proofs, they can be publicly verified while maintaining the voters privacy.

While this may seem like a vastly superior proposal externally, the additional complexity of the underlying system, that is the inclusion of zero-knowledge proofs, should not be underestimated. Furthermore, zero-knowledge protocols, despite being proposed in the late 1970s, are still in their relative infancy when compared to Blockchain technology (e.g. zCash whitepaper [9], 2014) and therefore have not been through the same level of scrutiny nor do they have the same level of development or adoption. The protocol also requires the voter to supply a secret value for each of the voting options they did not choose which, may require considerable effort if the ballot is large enough, and is counter intuitive to what would usually be expected to cast a vote.

In this paper I outline the design of an "end-to-end verifiable system" built on the Ethereum Blockchain. In this system, a voter can register an Ethereum address which is then added to a list of *allowed addresses* inside a smart contract. Upon the ballot commencing, the voter will be able to modify the allocation of their vote inside this contract up to the point of the ballot ending. Due to the strict, programmable nature of Ethereum contracts we can be sure that; no votes can be modified after the ballots end date, only the voter in control of the Ethereum address can change the vote associated with that address and only Ethereum addresses pre-registered to the contract are able to vote. Anyone can verify the number of votes for a candidate by querying the contract, and as they can be assured that only those addresses added to the contract are able to vote, we can be sure of the validity of each of these votes.

3 Design

3.1 Overview

In this paper, I present an "end-to-end verifiable system" built on the Ethereum Blockchain, i.e., a system where a voter can be assured their vote has been fairly counted, only eligible voters are allowed to vote and the tallied results of the election are publicly verifiable.

Although this system has been designed and developed with the idea of a national general election in mind, the protocols and ideas involved could be applied to smaller scale ballots which wish to provide transparency in their audit. Although we wish to minimize trust in a central authority, due to the nature of these type of elections (where there needs to be some degree of voter eligibility verification), we cannot fully decentralize this system as we need to only allow those eligible the rights to vote. Despite needing to verify an individual we still need to ensure that their votes are publicly anonymous, especially given the public transactions underpinning the blockchain concept while providing the ability for an individual to verify that their vote was correctly counted.

I do not see this system as a direct "replace all" for national election voting. I believe there will still be a need for traditional voting implementations in certain situations; for example, maintaining postal vote for the elderly who may not have the technical capability or equipment for online voting. However I do think that this could be phased in along side traditional voting, eventually replacing the pre-existing e-voting systems and ultimately becoming the main way for the majority of people to choose their government.

The designed schema for this protocol is the following:

1. Ballot creation:
    - The available ballots in the election are designed and decided upon.
    - A smart contract is created and pushed to the blockchain for each ballot containing all of the voting options.
2. Pre-election voter verification:
    - Voter registers with an external voter registrar after providing a valid ID (this could be accomplished using pre-existing government electoral registration protocols).
    - This external registrar generates a *user_id* and *nonce* which can be used by the voter to log in to the system.
    - This *user_id* is then tied to any ballots the voter is eligible for.
3. Voter registration:
    - The voter logs into the system using the received *user_id* and nonce upon which time they are immediately required to change their login credentials.
    - The voter can then register to vote through the online system for each of the ballots they are eligible for.
    - A unique Ethereum address, *voter_address*, is generated and validated (while not being linked to the *user_id*)

- The *user_address* is added to the ballots smart contract which entitles this address to vote in that ballot.
- The address is funded with enough Ether for the voter to cast their vote.

4. Voting:
    - When the voter decides to cast their vote they are presented with an interface mirroring the options in the ballot smart contract.
    - Upon the voter selecting their options the contract is funded with the voters selected options.
    - At this point the voters choice is immutably entered into the blockchain and the tally is verifiable by all.
5. Election result:
    - Once the election is over, due to the nature of the smart contract design, no more votes can be added for any candidate.
    - The tally for each candidate is publicly verifiable by anyone along with all of the funded transactions casting votes.

3.2 Docker

Over the past few years, container technology has become increasingly promising as a means to seamlessly make software available across a wider range of platforms & allow developers to worry less about the eventual runtime environment (as this can be standardized). Docker containers provide a way to "wrap up a piece of software in a complete filesystem that contains everything it needs to run" [51].

There are several benefits to the use of Docker containers; This could substantially reduce the effort required to create and validate a new software releases, since docker containers create their own dedicated environment, testing on one OS means that the application will run the same on any OS capable of running Docker. In addition, docker containers provide a quick and easy way to install and use a software release, for our application this could mean faster patches if needed as you would simply need to swap out the docker image being used. Other benefits include faster bootup of containers (compared to just virtualization), closer development to production parity, immutable infrastructure and improved scaling (on a per-container basis) [52]. There is also a security argument to be made for using containers, as they offer a degree of isolation for each enclosed application and only expose those services which you decide. The previous point about patching also add to security, as legacy applications often forgo patches in sensitive environments due to the possibility of breakages. When using containers, changes can be fully tested in the container which can then be swapped into the production environment [53].

All of my development for this project was conducted inside of Docker containers. I decided on this because there are very distinct separations between the applications which make up my system (this is described in section LINK TO SECTION), so running each one inside a docker container seemed the logical thing to do. It also meant that I could control the startup of the system as a whole & expose (between containers) only those services necessary to communicate.

3.3 Ethereum

3.3.1 What is Ethereum

In summary, Ethereum is an open software platform based on blockchain technology that enables developers to build and deploy decentralized applications. The block chain is a decentralized network of computers who, at the most basic level, all maintain a ledger in consensus with each other. One block is added at a time, each block contains a mathematical proof that verifies it's addition to the chain and the transactions within are protected by a strong cryptography.

Ethereum is poised to become the next greatest innovation based on block chain technology so is Ethereum similar to Bitcoin? The answer is sort of, but not really. Like Bitcoin, Ethereum is a distributed public blockchain network, however there are some significant technical differences between the two. The most important distinction to note is that Bitcoin and Ethereum differ substantially in purpose and capability. Bitcoin offers one particular application of blockchain technology, a peer to peer electronic cash system that enables online payments. While the bitcoin blockchain is used to track ownership of digital currency (bitcoins), the Ethereum blockchain focuses on running the programming code of decentralized application [55].

Ethereum enables developers to build and deploy decentralized applications. A decentralized application or Dapp serves some particular purpose to its users, for example Bitcoin, is a Dapp that provides its users with a peer to peer payment system. Because decentralized applications are made up of code that runs on a blockchain network, they are not controlled by any individual or central entity. Running these Dapps on a decentralized Platform, the blockchain, they benefit from all of its properties [54]:

- Immutability, a third party cannot make any changes to data.
- Corruption & tamper proof as apps are based on a network formed around the principle of consensus, making censorship impossible.
- No central point of failure, as Dapps can be run on every node in the network.
- Secured using cryptography, applications are well protected against hacking attacks and fraudulent activities.
- Zero downtime, Dapps never go down and can never be switched off.

3.3.2 Ethereum Blockchain

The concept of the blockchain was originally outlined in a white paper [**12˙nakamoto˙2008**] authored under the pseudonym Satoshi Nakamoto in November of 2008 and was quickly followed by an open source release of the Bitcoin proof-of-concept source code in January 2009 [**13˙nakamoto˙2009**]. This is the distributed ledger which underpins the entirety of the Bitcoin and Ethereum systems. A distributed ledger is a consensus of replicated, shared, and synchronized digital data geographically spread across multiple sites, countries, and/or institutions [**24˙distributed˙ledgers˙and˙blockchain˙technology˙2016**]. This ledger is stored locally on every node in the network which is running the full version of the blockchain software [**14˙bitcoin˙2009**] and records every transaction sent and confirmed on the network (the current size of the Ethereum Blockchain is around 21GB, March 2017[**25˙blockchain˙size˙2016**]). This complete history, coupled with the fact that it is an open network means that anyone can see what is happening in the network, not just now but during all periods in the past. This is extremely powerful as it allows an individual to fully audit the entire contents of the Blockchain without relying on external parties. This process is, in fact, what happens when you first download the full version of many blockchain reliant software [**20˙developer˙guide˙bitcoin˙2016**].

While the Ethereum Blockchain is not the only most mature distributed ledger in existence, it does have several years of being a publicly proven method to achieve distributed consensus and does this via the 'proof-of-work mining' process [**24˙distributed˙ledgers˙and˙blockchain˙technology˙2016**]. This is how new information gets added to the blockchain, by nodes in the network running a special 'mining' variant of the Ethereum software which uses considerable computing resources to win the right to add another block to the Blockchain which is accompanied by a reward for the winning user. The concept of 'proof-of-work' is a method of ensuring that the information being added to the Blockchain was difficult (in terms of cost and time) to be made, though is easy for others to validate that the requirements were met [**26˙blockchain˙mining˙-˙distributed˙ledgers˙and˙blockchain˙technolo**
This means that the expenditure of computing power serves to secure the integrity of the Blockchain, while the miners themselves verify through public-private key cryptography the validity of each transaction they include in a block.

Blocks are chained together making it impossible to modify transactions included in any one block without modifying all following blocks; as a result, the cost to modify a particular block increases with every new block added to the block chain, magnifying the effect of the proof of work [**20˙developer˙guide˙bitcoin˙2016**][**38˙proof˙of˙work˙-˙masterpage˙2016**]. This is why, although a transaction is deemed clear upon its inclusion in a block on the Blockchain, best practices dictate that a user considers a transaction confirmed after its inclusion in a block and the addition of five subsequent blocks to the Blockchain [**27˙confirmation˙-˙bitcoin˙wiki˙2016**].

The difficulty of the proof-of-work mining needs to be controlled, so that an average mining time of around 12 seconds per block is maintained. This time is somewhat arbitrary but is an attempt to find a balance between accepting transactions quickly and minimizing instability and waste in the network, as, while a new block is being distributed other miners will be working on an obsolete problem. As more miners join the network the block creation rate will increase due to the greater collective computational power. Therefore, every 2,016 blocks the difficulty of the mathematical challenge is recalculated so that the average mining time returns to normal [**20˙developer˙guide˙bitcoin˙2016**][**26˙blockchain˙mining˙-˙distributed˙ledgers˙and˙blockchain˙technology**

Despite the media often suggesting that bitcoin (and the Blockchain technology behind it) is an anonymous payment system, the Blockchain is in fact a transparent record of all user transactions on the network. Blockchain transactions are in fact pseudonymous, and your transactions in the

network are like writing under a pseudonym. If an author's identity is ever linked to their pseudonym then everything written under that pseudonym will be revealed [**28˙anonymity˙2016**]. This is particularly poignant when considering the Blockchain as every transaction is stored forever, therefore a compromised address could lead to all transactions being linked to a person. There are however ways to reduce the amount of statistical analysis which can be done on transactions that a person is a part of which help to achieve reasonable anonymity.

3.3.3 Mining & Ether

Ether is the fuel of the Ethereum system. It is the currency of the Ethereum network with which the payment of computation is achieved. Ethereum, like all blockchain technologies, uses an incentive-driven model of security where transaction consensus is based on a "proof-of-work" criterion of a given difficulty.

The block chain on which the Ethereum executes certain environment is known as the Ethereum Virtual Machine (EVM) [54]. Each participating node within the network runs the EVM and performs the proof of work algorithm called Ethash which involves finding a nonce input to the algorithm so that the result is below a certain threshold (depending on the difficulty) [57]. There is no better strategy to find such a nonce than enumerating the possibilities while verification of a solution is trivial and cheap. If outputs have a uniform distribution, then we can guarantee that on average the time needed to find a nonce depends on the difficulty threshold, making it possible to control the time of finding a new block just by manipulating difficulty [57].

This is how transactions are validated, new transactions are forwarded around the network and placed into a pool of unconfirmed transactions. These are not considered 'accepted' yet but are available for all to see almost instantaneously. Miners draw from this pool to create a candidate next set of transactions to be officially accepted which will form the next block. The full text of all of these candidate transactions, along with the hash of the previous block and a nonce, are input into the the hash function (Ethash) and miners will try different values for the nonce until the resulting hash is below a certain value. Because it's a cryptographic hash, there's no way to find a nonce that satisfies the output hashes condition other than attempting to guess [**20˙developer˙guide˙bitcoin˙2016**]. At this point, all of the miners are in a competition to find the hash first, each with a potentially different set of transactions to confirm. Once a miner succeeds they announce their solution to the rest of the network, their block becomes the next block in the Blockchain, and the transactions therein become confirmed. This strategy means that one miner will choose the next set of confirmed transactions, but the hash function effectively makes the miner a random one. All other mines then validate this new block, and the transactions held within, and can choose to accept it and start work on the next block. As the new block contains the hash of the previous block, this forms a chain of confirmed blocks securing the order of the transactions held within.

Occasionally, two miners may find a solution to the problem at the same time creating two potential next blocks in the chain. When miners produce simultaneous blocks at the end of the block chain, each node individually chooses which block to accept, this is usually the first block they see. Eventually another miner finds the solution to another block which attaches to only one of the competing blocks. This makes that side of the fork stronger and, as the general consensus is to use the strongest chain, other nodes will switch to this longer Blockchain [**20˙developer˙guide˙bitcoin˙2016**]. While this is statistically unlikely to happen, it is even more unlikely for the subsequent blocks to be solved at the same time, meaning that one fork will grow quicker than the other and the fork will resolve itself quickly. Transactions that were in the fork that wasn't chosen are not lost and are placed back into the unconfirmed transactions pool [**4˙driscoll˙2016**]. The fact that the end of the chain can be forked and rearranged means you shouldn't trust transactions at the end of the chain as much as ones further back. In Ethereum, a transaction is not considered confirmed until it is part of a block in the longest fork, and at least five blocks follow it. In this case we say that the transaction has "5 confirmations". This gives the network time to come to an agreed-upon the ordering of the blocks [**35˙nielsen˙2013**].

The successful miner of a block receives a reward for the 'winning' block, consisting of exactly 5.0

Ether along with all of the gas expended within the block, that is, all the gas consumed by the execution of all the transactions in the block. Over time, it's expected the gas reward will dwarf the block reward and become the main incentive for miners to continue working [57].

3.3.4 Transaction Costs & Gas

Ethereum does have a small transaction fee, just like Bitcoin, where users pay a relatively small amount to the executor of your transaction. The sender has to pay the fees at each and every step of the activated program, this includes the memory, storage and computation [54]. The size of the fee paid is equivalent to the complexity of the transaction, i.e. the more complex the commands you wish to execute, the more gas (and Ether) you have to pay. For example if "Alice" wants to send "Bob" 1 Ether unit, there would be a total cost of 1.00001 Ether to be paid by Alice. However if A wanted to deploy a contract or run a contract function, there would be more lines of code executable, therefore more energy consumption placed on the distributed Ether network and she would have to pay more than the 1 Gas done in the first transaction [56]. Some computational steps cost more than others, either because they are more computationally expensive or because they increase the amount of data that has to be stored in the state.

Gas is the internal pricing for running a transaction or contract in Ethereum. The gas system is not very different from the use of kilowatts in measuring electricity except that the originator of the transaction sets the price of gas, to which the miner can or not accept [56]. Ether and Gas are inversely related say for instance if the Ether price increases, than Gas price should decrease to maintain the concept of real cost [54]. With Ethereum there is also a blocksize limit, so the more space your transaction takes up the more you have to pay to get it validated. With Bitcoin miners prioritise transaction with the highest mining fees. The same is true of Ethereum where miners are free to ignore transactions whose gas price limit is too low.

The reason for the inclusion of a gas price per transaction or contract is to deal with the Turing Complete nature of Ethereum and its EVM essentially to guarantee that code running in the network will terminate. So for example, 0.00001 Ether or 1 Gas can execute a line of code or some command. If there is not enough Ether in the account to perform the transaction or message then it is considered invalid. This aims to stop denial of service attacks from infinite loops, encourage efficiency in the code and make any potential attacker pay for the resources they use (whether that be bandwidth, CPU calculations or storage) [56].

3.3.5 Smart Contracts

Smart contracts are the key element of Ethereum. In them, any algorithm can be encoded, they can carry arbitrary state and can perform any arbitrary computations even being able to call other smart contracts. This gives the scripting capabilities of Ethereum tremendous flexibility [59]. When run a smart contract becomes like a self-operating computer program that automatically executes when specific conditions are met and because they run on the blockchain, they run exactly as programmed without any possibility of censorship, downtime, fraud or third party interference. While all blockchains have the ability to process code, most are severely limited. Ethereum is different in this respect as rather than giving a set of limited operations, Ethereum allows developers to create whatever operations they want allowing developers to build thousands of different applications that go far beyond anything seen previously [55].

The Ethereum Virtual Machine is where smart contracts are run. It provides a more expressive and complete language than bitcoin for scripting and is Turing Complete. A good metaphor is that the EVM is a distributed global computer where all smart contracts are executed [58]. There are several higher level languages used to program smart contracts, but Solidity is the most mature and widly adopted. Its syntax is similar to that of JavaScript, its statically typed, supports inheritance, libraries and complex user-defined types among other features.

Smart contracts are run by each node as part of the block creation process and, just like in Bitcoin, this is the moment where transactions actually take place. An important part of how smart contracts work in Ethereum is that they have their own unique address in the blockchain. In other words, contract code is not carried inside each transaction that makes use of it. Instead contracts are "deployed" to the blockchain in a special transaction that assigns an address to a contract. This transaction can also run code at the moment of creation. After this initial transaction, the contract becomes forever a part of the blockchain and its address never changes. Whenever a node wants to call any of the methods defined by the contract, it can send a message to the address of the contract, specifying data as input and the method that must be called. The contract will then run as part of the creation of newer blocks up (subject to the gas limit or completion) and can return a value or store data [59].

3.3.6 Why Choose Ethereum

Ultimately the decision to use Ethereum for this project was an easy one. Ethereum is not just a digital currency, it is a blockchain based platform with many aspects desirable when designing and creating distributed applications. There simply is no other technology (at the time of writing) that can offer the same level of customization of decentralized programming and has a similarly substantial user base.

I initially investigated using the Bitcoin protocol as a method to store data (votes) immutably and reviewed several papers proposing voting solutions [60]. All of these proposals were however 'clunky' in design due to the inescapable fact that that is not what Bitcoin was designed for. Bitcoin was written in a stack based language that isn't Turing Complete as it was designed as a distributed value transfer ledger.

Ethereum, on the other hand, has contracts written in a Turing Complete Language meaning that anything can be done with it given enough time and enough computing power. This means that Ethereum was built specifically to handle smart contracts over simple currency transactions and although Bitcoin could be built upon to allow the functionality that Ethereum has it would seem unnecessary and be likely cause more problems when programming the application.

Ethereums block confirmation time is also much shorter than Bitcoins. Bitcoin is currently at around 10 minutes whereas Ethereum is around 12 seconds. So consequently, while bitcoin transactions normally take a few minutes to be cleared, Ethereum transactions are cleared almost instantly and at most in a matter of seconds.

The choice to use Solidity as the programming language for my Smart Contracts was mostly governed by maturity. The simple fact is, that there is no other competing languages that have sufficient levels of publicly tested development to justify their use. The closest competitor looks to be Viper [61] though this is still in the very early stages of development and lacks many features I would require to be able to use it for this application.

3.4 System Design

There were several points to consider when designing the high level plan of this voting system. The most important of which being the need to ensure separation of a voters account (tied to an individual) and the address they use to vote in the ballot contracts. This directly affected how I designed the system and lead to me splitting the system into distinct sections to take on specific roles; the *Application server* (voter interaction), the *Online Account Verifier* (verify the legitimacy of an account to vote) and the *Online Ballot Regulator* (manages the ballot contracts).

Figure 1: Outline of system showing the basic interactions between nodes.

Splitting the system like this adds both a layer of security and increased scalability. AS all of the data is not centralized on one node (or in one database) this would make it more difficult for a potential attacker to breach the system as they would have to get past the security of at least two nodes to obtain anything useful. The big benefit here is scalability, as if this was scaled up to a general election, more traffic would be seen on some nodes than others (e.g. more logins through the *Application Server* than ballot queries in the *Ballot Regulator*). We could then independently scale each node accordingly.

As each of these nodes are effectively self contained, they only need to expose a select number of services to allow inter-node further decreasing the possibilities for attack. As many of these services only need to be called from other nodes, we can add authentication to these connections to ensure that is upheld. In fact, there are only two external points of contact with this system, the web interface for the voter to use and the blockchain interface to contact other Ethereum nodes, meaning we could black box our system from the outside world fairly completely (see networking between nodes section LINK SECTION).

Finally, I chose python as the main language for this system as it has strong frameworks for building web applications (Django) which I heavily utilized these for the *Application Server* & *External Voter Registration* nodes. There is also strong development of Web3.py [62] which is a python implementation of web3.js, heavily used when interacting with Ethereum through the Geth client. I also discovered the Twisted python networking package [63] which includes an Asynchronous Messaging Protocol (AMP) implementation for calling remote methods and the Pycrypto library which allowed me to perform the RSA blind signature verification which is crucial to separating a user from their Ethereum address.

3.4.1 External Voter Registration

The "External Voter Registration" node is meant to represent *some external registrar* who is not directly a part of the designed voting system, but plays a crucial role in the verification of a voters validity (e.g. the UK governments register to vote system). Their role should be purely during the "pre-election registration" stage where the voting populous registers their intent to vote in the upcoming election.

I envision this as being very similar to registering to vote with GOV.UK, where you send your uniquely identifying information (e.g. date of birth, national insurance number, etc) and are then, if verified as a valid voter, registered for the appropriate ballots for your area. As such there would be no need to write this application from scratch as i would make sense to utilize

the existing verification systems already in place and modify them to interact with the Ethereum voting system as appropriate.

For the purpose of demonstrating my application I have written the software for the "pre-election registration" node so that I can register a new voter in the system easily (as there needs to be multiple database entries created) and so I can easily deploy a new ballot contract to the blockchain from a specific Ethereum address (necessary so that we have permission to register a voter to a ballot contract). The "External Voter Registration" application runs on a Django base, this is because I needed to present a web interface for interaction when registering a new ballot & user.

3.4.2 Application Server

The "Application Server" node is the main place for voter interaction with the system. Here, the Django backend provides the web interface for users to log in, register to a ballot & ultimately vote.

User interaction is centred around a dashboard page which displays core system information to the user (ballots they are eligible for, whether they have voted, etc). From here the user can invoke all voting operations available to them such as registering for a ballot & voting, to account management options such as changing passwords. When displaying the voting page after a user requests to vote in a ballot, the application server will query the blockchain contract for the voting options & also display current statistics about the ballot (such as number of registered voters, the current votes per candidate & voter turnout).

3.4.3 Online Account Verifier

The "Online Account Verifier" holds the role of authenticating that a user is eligible for a ballot while retaining the anonymity of the final Ethereum address the user casts their vote with. This is accomplished with the use of blind token signing, the process of signing a message without seeing the contents. This means we can verify a voter and return them a signed token which can be sent in future along with an Ethereum address to verify that the address is legitimate.

The first table is used to verify that a user account has not requested to register for a particular ballot previously with a blind token. The second table is used to keep track of registered Ethereum addresses and which ballot they are registered to. Both of these tables are only used for internal state storage, they are used to determine if a user has already registered for a ballot and show the 'register' or 'vote' options accordingly. The data stored here is not used by the application server to retrieve any voter credentials. These tables store the token hashes so that we can ensure that tokens cannot be reused by an attacker.

3.4.4 Online Ballot Regulator

The "Online Ballot Regulator" manages everything to do with the voting ballots. This includes all created ballots in the system and their corresponding Blockchain addresses through to which user accounts are registered to which ballots.

The "Online Ballot Regulator" is where the "External Voter Registrar" sends the available ballots for a user account to be stored. This node is the most queried in the entire system as it holds

Database table holding token requests.

token_request_id	blind_token_hash	user_id	ballot_id	created_on
1	54baa883f28a768d6bd352d12c7307d1592d394acea4337a018ee2d0f143642a	68401	1234	2017-04-08 18:08:49.527817
2	4451458fe4d479387fdca5dde405a16fd2fca2c4f92c516fea78b2f0d2727f7c	67173	1234	2017-04-08 18:27:25.984678

Database table holding token registrations.

register_vote_id	signed_token_hash	voter_address	ballot_id	created_on
1	19a91bc3d91b5873e5b0e4687a988a08b362bd357a85ce3bc109cba4ed984407	0x79957083494aa13895ae6bad9f04f2bb99f0ad39	4321	2017-04-08 18:30:25.736455
1	13jad238sjkgfn39n3asd882nd2d877ad327dnk3lq9afv4b234akd3nd898hd82	0x925A8e765F9563D979b576A68210903a9968B8Be	5432	2017-04-09 12:23:41.162341

Figure 2: Database table used to store the requests to register an Ethereum address to a ballot for a registered user.

information about ballots the user account is tied to along with the address of ballots in the blockchain.

3.4.5 Blockchain Ballot Contract

Database table showing which ballots a userID is allowed to vote in.

ballot_register_id	user_id	ballot_id	created_on
1	1234	1234	2017-04-08 18:26:23.440678
2	1234	6543	2017-04-08 18:26:23.440678
3	2345	6543	2017-04-08 18:26:23.440678
4	67173	1234	2017-04-08 18:26:44.067895
5	67173	4321	2017-04-08 18:26:44.134928

Database table holding information about each ballot in the system.

ballot_id	ballot_name	ballot_address	created_on	ballot_interface	ballot_end_date
1234	Election of the Member of Parliament for the Harborne Constituency	0x127c73Af1F9E0efF8226 Db6bdf04310fDEe674F6	2017-04-08 18:26:23.440678	x800358071002e	1603238400
4321	Election of Police and Crime Commissioner for Edgbaston area	0x8C872c720DF854a058C3 D1DD54e4CEE51d798B6A	2017-04-08 18:26:23.440678	x800358071002e	1603238400
6543	Referendum on the United Kingdoms membership of the European Union	0x7654EC4067e8fA04184D 68fF08169A29A3B20F19	2017-04-08 18:26:23.440678	x800358071002e	1603238400

Figure 3: Database table used to store the requests to register an Ethereum address to a ballot for a registered user.

3.5 Pre-election setup

3.5.1 Creating a new ballot contract

The first thing which needs to be done before any other aspect of the election can take place, is publish the smart contract for each ballot in the election to the blockchain (a deeper analysis of the contract is presented in LINK TO SECTION. The smart contract I created can be seen as a 'template' of sorts, allowing different sets of voting options to be added to a similar core structure. The system requests the 'ballot name', 'ballots voting options' (provided as a comma separated list) and the 'end date' of the ballot. Because all of the Blockchain interactions are handled by the "Online Ballot Regulator" we wrap up this information and send it in a network call to the "Online Ballot Regulator".

Figure 4: Registering a new ballot in the blockchain.

The *Online Ballot Regulator* does two things, first registering the ballot contract into the Blockchain with the information received from the remote call. This is funded (and therefore deployed) by the *Ballot Regulators* private key and corresponding Ethereum address which, due to the programming of the ballot contract, means that the ballot regulator has exclusive rights to modify the contract. Deploying the contract happens in three stages in the ethereum/ethereum.py class of the *Ballot Regulator*. First, the contract 'template' is deployed to the Blockchain via the Ethereum software run on the server. This is done by sending the compiled contracts bytecode in an Ethereum transaction along with the contract parameters needed to initially setup the contract (the ballot name & ballot end time). Once the contract is deployed and confirmed into the blockchain we can access the contract at a specific address which we will use from here on out to interact with the contract (e.g. 0x127c73af1f9e0eff8226db6bdf04310fdee674f6).

Next we send another transaction for each ballot option, calling an internal method of the contract, to add each of the options to the deployed contract (you can see examples as the 2nd and 3rd transactions in the above link). These options are then immutably added as choices of the ballot.

The final transaction to the contract is to the internal 'finalize()' method. After which, no more ballot options can be added and any registered voters are able to cast their votes.

Once the ballot has been deployed to the blockchain the *Ballot Regulator* confirms its validity and then stores internally the ballots name & Blockchain address. This allows us to query the *Ballot Regulator* later to obtain the correct address for a specific ballot.

3.5.2 Registering voters in our system

Once we have a voter who wishes to register, and is eligible to voter in a specific ballot (or set of ballots) we need to add this user to our system so they can log in & vote.

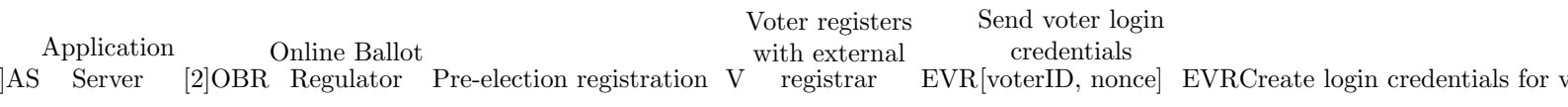

Figure 5: Sequence diagram showing order of calls when a voter is registered with our system.

The first step is validating the user requesting to register is eligible to vote. The validation process is out of the scope of this project but you could imagine this being a similar process to current election registration schemes. Therefore our system has no verification built in and allows anyone to sign up for any ballot of their choosing.

Next we request a new user account is created in the *Application Server* for our voter. A network request is sent and handled by the accounts/remote_user_add.py class which generates a new userID & random secure password which are then passed back to the caller.

We now register the userID for any ballots they are eligible for. This is done in another network call to the *Online Ballot Regulator* and is handled in the onlineballotregulator/network_request.py class. A database entry is created linking the userID to a ballotID which is used later to verify which ballots a logged in user is eligible for.

Finally the login credentials are sent securely to the user using an applicable method. For a more traditional registration system, this could be sent in the post similar to how you receive a credit card & pin number (separate letters). It would be possible to encode this information into a QR code format so that the end user need simply scan their received credentials to first log into the system. If the voter validation process was online based, i.e. allowing users to upload their identity documents for automatic processing, we could respond with the users login details almost instantly just like signing up to any secure website (the risks here are reduced as the user is required to change their password on first login anyway).

Figure 6: Screenshot of the web interface to the *External Voter Registration* showing the previously created ballots at the top & the ability to register a new user (to a set of ballots) at the bottom.

3.6 During the Election

3.6.1 First login

Once a user has requested to be registered into the system, and received their initial login credentials, they are able to access the service and log in. Before they are able to access any site content, we enforce a password change and basic user information to be entered. This is for security reasons and is good practice for web applications.

This enforcement is handled by some extra Django middleware in the "Application Server" (accounts/middleware.py) which will not allow access to any internal pages unless the user has already entered their initial information. It does this by checking, before the display of any internal page, if there is a 'requires initial information' flag set in the user database and if so refuses to display the requested page and redirects to the initial information request page..

Figure 7: Initial login information entry page.

As the system I produced is only proof-of-concept I have not enforced entry of much information besides an email address and the password reset. If this system was deployed in an actual election you could request for more options to be set here such as contact preferences, display preferences (visually impaired mode) or further security options (maybe setting up two-factor authentication). Once the user has entered the required information they are able to proceed further into the system and access the user dashboard.

3.6.2 Online Registration

The user dashboard page is the starting place for any user interaction with the system. Here, a list of ballots the user has been approved to vote in is shown along with their corresponding Ethereum address & associated information such as the current user registration state. Users are presented with a link to an external blockchain viewer showing the transactions of each contract that they can use to independently verify a contracts validity if they wish.

Figure 8: User dashboard showing ballot information along with the users registration state for each ballot.

In order for the user to engage in voting on a ballot we require them to 'register' with that ballot. This will start the process of generating a new Ethereum address that will be allowed to interact with the blockchain contract. Note that we could do this automatically without user interaction (possibly after the initial login information has been entered) but I have chosen to offer this as a distinct step which must be invoked in this proof-of-concept system. This is because its easier to demonstrate the separate process of registering a user to a system account to that of registering an address to a deployed ballot contract. In a real world system there would be no need to show this step to the user, as it could cause confusion about what is being registered, and it could easily be abstracted away.

When the user clicks the button to register on a ballot contract we begin the process of generating an Ethereum address for the voter to use, giving it some Ether and allowing it to vote on the selected contract.

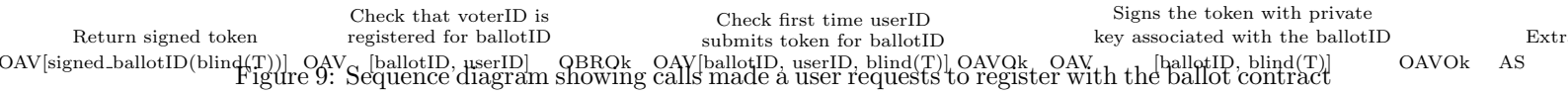

Figure 9: Sequence diagram showing calls made a user requests to register with the ballot contract in the blockchain.

The process of allowing a voter to interact with a deployed ballot contract is quite involved and is designed to anonymize (to the system and external parties) a voterAddress while still being able to verify that the address is being supplied by a user who is allowed to vote. This verified yet anonymous status is achieved through the use of blind signatures on tokens.

0.5cm

Blind Signatures

A "Blind signature" is a signing scheme where the signer doesn't know the content of the message he/she is signing but the resulting blind signature can be verified against the original unblinded message just like a regular digital signature [64].

This can be analogized to an individual, Alice, placing a letter inside a carbon paper lined envelope. This is then handed to a trusted third part, Bob, who (without opening the letter) signs the outside of the document & hands it back to Alice. Due to the carbon paper inside the envelope, Bobs signature is also transferred to the letter within. Alice can then extract the letter which now contains Bobs signature despite him never having seen the letter contents.

0.5cm

Figure 10: Blind signature analogy showing how Bob never sees the contents of Alice's message despite being able to sign it.

Now we can try to translate this to the language of cryptography. Suppose Alice has a message m that she wishes to have signed by Bob, and she does not want Bob to learn anything about m. Let (n, e) be Bob's public key and (n, d) be his private key. Alice generates a random value r such that $gcd(r, n) = 1$ and sends $x = (r^e m) \bmod n$ to Bob. The value x is "blinded" by the random value r; hence Bob can derive no useful information from it. Bob returns the signed value $t = x^d \bmod n$ to Alice. Since $x^d \equiv (r^e m)^d \equiv rm^d \bmod n$, Alice can obtain the true signature s of m by computing $s = r^{-1}t \bmod n$. Now Alice's message has a signature she could not have obtained on her own [65].

I've used the concept of blind signatures in this system to send a randomized Token to the "Online Account Verifier" for them to sign. The singing in my system is accomplished using RSA keys and the PyCrypto library [66] which abstracts away most of the mathematics and allows the generation of a blind message (see client implementation in "Application Server" user_ballot_registration/
views.py and the coresponding server code at "Online Account Verifier" at signatures/token_request.py). The "Online Account verifier" has a unique key pair for each ballot that is registered in the system, this means that any tokens signed are valid only as identification for the specific ballot they were requested for.

The order of events for a user to register an Ethereum address to vote is as follows: Firstly, the "Application Server" generates a random token to be used in the interaction. This is then blinded with a randomly generated number (as explained in the *Blind Signatures* section) and the public key of the ballot we are requesting to add the address to. Next we send this blinded token across to the "Online Account Verifier" to be signed.

When the request to sign the blind token is received by the "Online Account Verifier" we initially run a few checks. First, we check to see if the voter requesting to be registered for a ballot is indeed eligible to vote. Secondly, we check that this is the first time we are seeing this user request a token signature for this particular ballot. These checks ensure that users can only register for ballots they are eligible for and each address can only register one address per ballot. If all of our checks pass then we sign this blinded token with the associated ballots private key & return it to the "Application Server".

The "Application Server" now has a signed, blinded token (the contents of which have not yet been seen by the "Online Ballot Regulator") which we can unblind to reveal a signature of the raw token by the "Online Ballot Regulator". We now generate and store an ECDSA keypair [67] which we can use to derive the Ethereum address the voter will use to interact with the ballot in the Blockchain. The token, token signature & voterAddress are then sent back to the "Online Account Verifier" to be verified before being added to the ballot contract.

This is now the first time the "Online Account Verifier" is seeing the token and, as the message is not accompanied by any userID, is unable to link this request to register into the ballot contract to a user. The system can however verify that this request is legitimate by checking the signature of the token against the token, as only a valid voter is able to obtain a signature through the system we can verify that this request is from a genuine voter and should be allowed to proceed. First we check that this is the first time we are seeing this token & signature (so that the same token cannot be used multiple times) before sending a request to the "Online Ballot Regulator" to insert the voterAddress into the ballot contracts list of eligible voters.

As the "Online Ballot Regulator" is in control of the Ethereum address that created the ballot contract the node is the only one with the ability to add new voters to the contact (see section LINK TO BALLOT SECTION). We create a new transaction calling the *giveRightToVote()* method of the ballot contract with the voterAddress as a parameter. Once this transaction is confirmed the state change in the ballot contract will mean that, when the voter with the keys to the voterAddress sends a transaction to call the *vote()* method in the ballot, they will be allowed to add their cast vote to the contract. At the same time we also fund the voterAddress with enough Ether to be able to fund their voting transactions (see ethereum/ethereum.py for the relevant code).

In summary, this is how we are able to verify that an Ethereum address is eligible to vote without revealing the user behind the private key.

3.6.3 Voting

3.7 Post Election results

3.7.1 Retrieving results

anyone can, openly verifiable

References

[1] Pierre Noizat. *Blockchain Electronic Vote*. 2016. URL: http://www.the-blockchain.com/docs/blockchain-electronic-vote.pdf (visited on 17/11/2016).

[2] Adam Ernest. *The Key To Unlocking The Black Box: Why The World Needs A Transparent Voting DAC*. 2014. URL: https://followmyvote.com/wp-content/uploads/2014/08/The-Key-To-Unlocking-The-Black-Box-Follow-My-Vote.pdf (visited on 15/12/2016).

[3] Willem Wyndham, Spencer Chen and Saurav Das. *Proposal For Secure Electronic Voting*. 2016. URL: http://www.economist.com/sites/default/files/maryland_cyber_ctr.pdf (visited on 15/12/2016).

[4] Safevote. *Voting System Requirements*. 2001. URL: http://www.thebell.net/papers/vote-req.pdf (visited on 19/12/2016).

[5] Briony Holmes. *Blockchain voting systems offer transparency and security Brave New Coin*. 2016. URL: http://bravenewcoin.com/news/blockchain-voting-systems-offer-transparency-and-security/ (visited on 16/12/2016).

[6] 2016. URL: http://www.ncsl.org/research/elections-and-campaigns/voting-system-standards-testing-and-certification.aspx (visited on 17/12/2016).

[7] Sabina Petride. *Security Properties for Electronic Voting*. 2016. URL: http://www.cs.cornell.edu/courses/cs513/2002sp/proj.00.StuSolns/sp2580.htm (visited on 18/12/2016).

[8] Alex Escala et al. *Universal Cast-as-Intended Verifiability*. 2015. URL: https://fc16.ifca.ai/voting/papers/EGHM16.pdf (visited on 21/12/2016).

[9] Eli Ben-Sasson et al. *Zerocash: Decentralized Anonymous Payments from Bitcoin*. 2014. URL: http://zerocash-project.org/media/pdf/zerocash-oakland2014.pdf (visited on 21/12/2016).