# Decentralized Certificate Authority as a blockchain application

Course code: KISPECI1SE

Mark Rostgaard Mortensen (mrom@itu.dk)
Markus Thomsen (matho@itu.dk)

June 3rd 2019

**Abstract**

Certificate Authorities have centralized power of the issuance of TSL certificates. This makes them single points of failure, when it comes to communicating over the HTTPS protocol.
This paper explores how it may be possible to decentralize the power of Certificate Authorities using decentralized applications running on the Ethereum blockchain. We developed two approaches:

The first approach use a Decentralized Autonomous Organization, that will act as Certificate Authority enforcing multiple signatures of a requested certificate.

The second approach, named Dapp-CA, replace the membership aspect with economical incentives, to incentivize honest, open participation. Participants in the Dapp-CA stake valuable tokens on the validity of a certificates, with the risk of losing them if breaking the rules.

We conclude that it is possible to use blockchain to enforce decentralization when approving certificates. However, changes are necessary to the DNS record, for the Dapp-CA not to be vulnerable to attack. Scalability remains an issue, making real-world usage impractical, without further research.

# Contents


**1 Introduction** **5**
- 1.1 Background . . . 5
  - 1.1.1 Transport Layer Security . . . 5
  - 1.1.2 Certificate Authorities . . . 6
- 1.2 Problem . . . 6
- 1.3 Related work . . . 7
  - 1.3.1 Multi-signatures Certificate Authorities . . . 7
  - 1.3.2 HTTP Public Key Pinning . . . 8
  - 1.3.3 Certificate Transparency . . . 8
  - 1.3.4 Blockchain based proposals . . . 8
- 1.4 Hypothesis . . . 10
  - 1.4.1 Research question . . . 10

**2 Problem Analysis** **11**
- 2.1 Certificate Authorities . . . 11
  - 2.1.1 Certificate Issuance Process . . . 12
- 2.2 Decentralization as a solution . . . 13
  - 2.2.1 Decentralized Certificate Authorities . . . 13
  - 2.2.2 Blockchain . . . 13
  - 2.2.3 Certificate Transparency . . . 15
  - 2.2.4 Economical incentives . . . 16

**3 User Guide and Requirements** **17**
- 3.1 Prerequisites . . . 17
- 3.2 Smart contracts . . . 17
  - 3.2.1 Prerequisites . . . 17
  - 3.2.2 Deployment . . . 17
  - 3.2.3 DAO-CA . . . 19
  - 3.2.4 Dapp-CA . . . 21
- 3.3 Firefox extension . . . 22
  - 3.3.1 Prerequisites . . . 22
  - 3.3.2 Extension building . . . 22

**4 Method** **24**
- 4.1 Certificate Transparency Log . . . 24
- 4.2 DNS challenge . . . 26
- 4.3 DAO-CA . . . 27
  - 4.3.1 Smart contract . . . 29
- 4.4 Dapp-CA . . . 30
  - 4.4.1 Smart contracts . . . 31
  - 4.4.2 Certificate attestation . . . 33
  - 4.4.3 Governance Token . . . 34
  - 4.4.4 Forking . . . 35
- 4.5 Frontend (Firefox implementationen) . . . 37

**5 Results** 38
5.1 DAO-CA Cost . . . 38
5.2 Dapp-CA Cost . . . 40

**6 Discussion** 43
6.1 Security . . . 43
6.1.1 Blockchain censorship attack vectors . . . 44
6.1.2 Certificate Transparency Attack Vectors . . . 47
6.1.3 DAO Attack vectors . . . 47
6.1.4 Dapp-CA attack vectors . . . 49
6.1.5 Social layer . . . 51
6.2 Economical Incentives . . . 52
6.3 Scaling . . . 52
6.3.1 Plasma, Sidechains and off-chain scaling . . . 53
6.4 Comparison of the established solution, DAO-CA and Dapp-CA 54
6.4.1 Established solution . . . 54
6.4.2 Proposed Solutions . . . 55
6.5 What could have been done better . . . 57
6.5.1 Dapp-CA reliability on DNS text . . . 57
6.5.2 Gas-usage optimization . . . 58
6.6 Future work . . . 58
6.6.1 Mechanism design improvements . . . 58
6.6.2 Security Audit and formalization . . . 59
6.7 Usability . . . 60
6.7.1 UI, UX, Firefox . . . 60

**7 Conclusion** 61

**A Smart Contract Code** 65
A.1 CertificateTransparency.sol . . . 65
A.2 MinDAO.sol . . . 67
A.3 Controller.sol . . . 70
A.4 Tokens . . . 75
A.4.1 ERC20Basic.sol . . . 75
A.4.2 BasicToken.sol . . . 75
A.4.3 BurnableToken.sol . . . 76
A.4.4 StakeableToken.sol . . . 76
A.4.5 MigrateableToken.sol . . . 77
A.4.6 SafeMath.sol . . . 78

# Glossary

**Attestor** A participant in the Dapp-CA or DAO-CA that is responsible for accepting or rejecting certificates..

**Crypto currency** Native payment token of a public blockchain.

**Ether** Native crypto currency of the Ethereum blockchain.

**Ethereum** A public blockchain known for it's ability to run smart contracts.

**GOV** The ERC-20 token created for the Dapp-CA.

**Smart contract** Smart contracts are the colloquial term used to describe programmable scripts deployed on blockchains.

**Solidity** The most used programming language for programming smart contracts on Ethereum.

# Acronyms

**CA** Certificate Authority.

**CT** Certificate Transparency.

**DAO** Decentralized Anonymous.

**dApp** Decentralized Application, typically one implemented with smart contracts.

**DNS** Domain Name Server.

**DPKI** Decentralized Public-Key Infrastructure.

**HPKP** HTTP Public Key Pinning.

**HTTPS** Hypertext Transfer Protocol Secure.

**IETF** Internet Engineering Task Force.

**PKI** Public-Key Infrastructure.

**RFC** Request For Comments.

**TLS** Transport Layer Security.

# 1 Introduction

The World Wide Web have always been an excersize in decentralization. It is possible to connect to many different terminals and servers, and exchange messages between machines, without anyone's approval. The internet fundamentally relies on routing messages through many different machines, and this makes it harder to know, if you're actually communication with who you think you are. Securing and verifying the authenticity of the communication is therefore an important task that have been worked on for more than 26 years, when the certificate standard X.509 was first defined in rfc1422[33].

With this standard came the concepts of trusted third-parties known as Certificate Authorities.[35] CAs are centralized entities, responsible for certifying the identity of domain owners. The CAs doesn't come without problems. They present a single point of failure, and their power can be abused to impersonate websites at will. This have presented problems for example hackings of CAs compromising 300.000 users emails. These problems will be discussed in section 1.2.

Companies, academia and independent researcher are working on solving the problems with centralized CAs. Some of these solutions are documented in section 1.3.

In this paper we present research done, as a part of a research project, in the spring of 2019 at the IT-University in Copenhagen. The project revolves around TSL certificates and it's signing using trusted third parties known as 'Certificate Authorities'.

## 1.1 Background

### 1.1.1 Transport Layer Security

The TLS protocol exist to provide security of communication over a network. The fundamental architecture of the internet results in messages being routed through many different machines, before arriving at their end destination. A malicious actor controlling any of these machines, can change and read the message as they see fit. To avoid this, protocols can be used, one of them is the TLS protocol.
The TLS protocol aims to ensure, amongst other, the following properties:

- **Privacy:** The TLS protocol use asymmetric encryption to create a shared secret, that is the base of the symmetric encryption which is used to exchange information securely.
- **Authentication:** The identities of participants can be authenticated.
- **Integrity:** The protocol uses message authentication codes, which ensures that messages can't be tampered with or altered.

While all of these properties are important, this papers primary focus will be on the property of authentication.

Authentication of participants in TLS is carried out with public-key cryptography, and uses certificates, to attest to the identity of the party that is being communicated with. TLS uses the X.509 certificate format, which includes information about the issuer of the certificate, the subject, information about the signature, cryptographic algorithms used, and more.
A secure session starts when a browser sends a request to a server, the server will respond with its certificate which contains a public key. The public key is used to encrypt a session key, this session key is sent back to the server where the server will decrypt the session key using the servers private key. When the session key is decrypted on the server the communication between the clients browser and the webserver can now all be encrypted symmetrically.

#### 1.1.2 Certificate Authorities

When visiting a website that are using HTTPS, the issuer of the certificate will in almost all cases be a Certificate Authority. CAs are third parties, which both participants in the TLS protocol, should trust. Their purpose is to issue certificates to website owners, and other untrusted parties, which can then use said certificate to prove the authenticity of the communication. It's important to note, that certificates do not protect against any malicious behaviour by the issuer, nor by the authenticated website, but merely ensures that the connection can be trusted to actually be with who the other party says they are, I.E. the domain you are visiting are owned by the certificate owner.
The certificates issued by a CA, only guarantees authenticity of communication, as long as the CA can be trusted. Nothing in the protocol itself incentivize or stops CAs from issuing misleading certificates to malicious third parties. The threat of legal issues and financial ruin, coupled with reputation, is used to keep CAs in-check.
Since certificates are issued by a single CA, and have no other parties which attest to their veracity, a single compromised CA will compromise the security of every certificate issued by that CA. Furthermore compromised CAs will be able to issue valid certificates until it is publicly known that they have been compromised. In other words, CAs are a single-point of failure in the current implementation of TLS. This security hole has been misused by both hackers and national intelligence agencies[11].

## 1.2 Problem

CAs can sign certificates for any 'Common Name', also known as domains. For a user to trust a certificate created by a domain owner, it first needs to be signed by a CA.
Browsers and operating systems comes pre-installed with lists of trusted CAs so the user knows which CAs the software trusts.

The process to become as trusted CA is difficult and often expensive. Microsoft have a process for becoming a trusted CA for their platforms. It is a long and strenuous process[16]. Mozilla have a similarly difficult CA program[17] to be included into the Mozilla software suite, the premiere product of which being the Firefox browser.

It is necessary that the process to become a trusted CA is difficult. Because one of SSL certificates main purposes is to establish authenticity of communication, this gives the CAs the power to impersonate any website. With a critical single points of failure such as this, it becomes paramount, that no entity can compromise it, however no such thing can be guaranteed. Human failure, exploitation of software security holes and malicious insiders can all lead to compromise of the CA system.
One of the more noteworthy cases has been the hack on DigiNotar in 2011.[11]

DigiNotar was compromised and more than 500 certificates were issued and signed by DigiNotar, and considered to be valid by the browsers. The certificates were used to read the emails of more than 300,000 Iranians. It took more than a month after the breach was discovered, before DigiNotar went public with the information.

Another story was that in 2015 Symantec's subsidiary Thawte generated SSL certificates, that weren't requested by the owner of the domains, for testing purposes.[5]

CAs can also be compelled into issuing false certificates by governments. Attacks like these can be especially difficult to discover, as the certificate may only be used to target very specific users.

These examples show that without trustworthy TSL certificates, HTTPS traffic received over the Internet cannot be trusted.

## 1.3 Related work

There already exists research and proposed solutions to the problems described in section 1.2. These range from already tried and failed solution such as HTTP Public Key Pining, to currently implemented Certificate Transparency. Some of the more recent research includes blockchains, and these mostly describe how to decentralize the Public Key Infrastructure, used on the Internet today, completely.

### 1.3.1 Multi-signatures Certificate Authorities

One proposal to decentralize certificate issuance, and thereby avoiding the single point of failure, is to have multiple CAs sign every certificate. This is most effectively accomplished by using threshold encryption schemes[26]. The drawbacks to this approach, is that the certificate signing process would still be controlled by CAs, many of which may be within the same jurisdiction. To avoid compromise of certificates by nation states, it would be necessary to have certificate issuers located in multiple jurisdictions, and preferably have the governments

of these jurisdictions be non-cooperative, or even adversarial. A benefit to this approach, is that it could rather easily be extended from the current infrastructure. A drawback is that it can be hard to have transparency in who the signers are, unless made an explicit part of the certificate information.

### 1.3.2 HTTP Public Key Pinning

HTTP Public Key Pinning is a protocol to pin a public key to a users browser[25]. The first time a user visits a website that have HPKP implemented, the server sends one or more public keys with the HTTP header in the field named 'Public-Key-Pins'. This field contains information as to how long the keys are expected to be valid. After this initial pinning, the browser saves the information. The browser will then only accept requests with certificates that are signed with the pinned public key.

HPKP has been implemented in modern browser since 2015, but in 2018 Google discontinued HPKP in their Chrome browser, due to lack of use and for security measures[37]. Pinning turned out to be too much responsibility in the hands of inexperienced users. If a hacker or social engineer convince them to use his public key, and therefore correctly pin a false certificate, it becomes possible for him to make an attack, negating the upsides of pinning.

Google and other researchers has since moved to ensure that the certificates presented are also correctly signed by a CA. This effort is called Certificate Transparency and is moving towards completely replacing HPKP.

### 1.3.3 Certificate Transparency

Certificate Transperancy[22] is a protocol designed to enable anyone to monitor the activity of CAs. It's already implemented in most modern browsers and working. This paper describes CT in greater detail in section 2.2.3.

### 1.3.4 Blockchain based proposals

Decentralization of the entire PKI structure using blockchains have been proposed. One approach is Decentralized Public Key Infrastructure written by Butterin et al.[30] and suggest using blockchains to register identities, such as domains.

SCPKI: A Smart Contract-based PKI and Identity System written by Mustafa Al-Bassam[21] is another one and this propose to use a reimplementation of the web of trust model using Ethereum smart contracts to handle the signing and revocation of trust.

#### Decentralized Public Key Infrastructure

DPKI propose to decentralize the entirety of public key infrastructure. This would be a major departure from the current system, requiring every principal, the owner of an identity such as a domain name, to never lose their signing keys,

or risk losing the ability to control their identity. The paper describes rules that would need to be implemented to make it a decentralized, transparent and open system:

- "Each principal must be in complete control of their current identifier/public-key binding." [p. 6]
- "Either every principal must witness every other principal's updates to their identifier/public-key binding or else no one may observe any updates." [p. 6]
- "**Permissionless Writes:** Any principal can broadcast a message provided that it is well-formed. Other peers in the system do not require admission control. This implies a decentralized consensus mechanism." [p. 7]
- "**Fork Choice Rule:** Given two histories of updates, any principal can determine which one is the 'most secure' through inspection." [p. 7]
- "Private keys must be generated in a decentralized manner that ensures they remain under the principal's control." [p. 7]
- "Software must ensure that principals are always in control of their identifiers and the corresponding keys." [p. 7]
- "Software must ensure, to greatest degree possible, that no mechanism exists that would allow a single entity to deprive a principal of their identifier without their consent." [p. 8]

A principal, who is the domain owner, would have direct control and ownership of a domain name by registering it in a blockchain. For this reason, the system insist that each principal must be in complete control of their own key and not allow web hosting or CAs to once again centralize control, by being key holders. This way a malicious entity would need to compromise every principal they want to attack, and not just a single CA.

The paper describes how principals can manage their public-key bindings. It is crucial that the principals private keys are not lost. This would result in an irretrievable loss of access to the principals identity. It's proposed to shard the master key. This would allow multiple identities to own a piece of the master key, and when joined together, will create the master key. The sharding can use threshold cryptography, so as to only need N out of M, where $M >= N$, pieces of the shards are required to construct the master key.

A second proposal is to use smart contracts to create a multi-signature script that needs N out of M signatures to execute and recover the key.

**SCPKI: A Smart Contract-based PKI and Identity System**

This paper is written by Mustafa Al-Bassam[21] and is a research project exploring how to use Ethereums smart contracts to systemize web of trust as described by PGP 2.0.

The system is designed to let an entity create a new attribute for themselves, such as a college degree, and then another entity, like the university, will be able to sign that degree, or revoke it if the university no longer trust the validity of the degree.

The system is based on trusting participants. For example if you trust a university and they signed a degree for another entity, then you should trust the validity of that degree. Where it differ from the current CA solution, is that every participant would have to decide which organizations they trust.

Both SCPKI and DPKI proposes radical and ambitious ways to change the current PKI system. SCPKI proposes a concrete solution while DPKI is mostly a paper describing the rules of a decentralized PKI, rather than a concrete implementation.

## 1.4 Hypothesis

We believe voting and economic mechanisms, can be used to design a decentralized protocol, which allows participants to issue and verify TLS-certificates in a decentralized manner, using the blockchain and smart contracts.

### 1.4.1 Research question

Is it possible to issue TLS-certificates in a decentralized manner, while keeping the ability to resist loss of private keys, and revoking invalid certificates, using the Ethereum blockchain?

# 2 Problem Analysis

## 2.1 Certificate Authorities

When a user visits a website, they will normally type in the domain name such as 'www.facebook.com', instead of the direct IP address. The browser will make a DNS lookup and find the corresponding IP address of Facebooks server, and establish a connection with that server.

If a malicious party has placed themselves between the user and Facebook, the malicious party can impersonate Facebook. This allows them to collect the data that Facebook and the user communicate between each other. This is known as a man-in-the-middle attack. To alleviate this issue, a website can provide the user a certificate to prove its identify and use the corresponding private key to encrypt it's messages, ensuring safe communication between the user and the website.

Everyone can create and sign their own certificate, for this reason, a certificate alone is not enough information for the user to know, if it's actually Facebook or a malicious party they're communicating with.

This is the reason that Certificate Authorities (CA) exist today. A CA is a company, or another trusted third party, which exist to validate the ownership of domains and sign requested certificates, if the ownership can be proven. This way the user know to trust the certificate received by the website if the websites certificate have been signed by a trusted CA.
It's not in the scope of this paper to explain how you become a trusted CA. The important aspect is that everyone can potentially be it and it is, in the end, up to the software the user is using to decide whether a CA is trusted or not. In other words, which CAs are trusted are defined on a software by software basis. In some cases, users may have control over which certificates they trust, and can add a CAs root certificate, thereby trusting any certificate signed by that CA. The root certificate is the certificate the CA use to sign other certificates to their system.

The duties of the CA is to receive certificate signing request, where a claimed owner of a domain will ask the CA to sign their certificate so that they are able to use it to prove their identity to the users of their website. The CA can do this in different ways. One way, that the CA Let's Encrypt uses, are the ACME protocol's DNS challenge[29]. Once a request have been deemed valid, the CA will then sign the certificate, and the certificate is now ready to be used by the domain owner. The CA is also responsible for revoking certificates, though CAs at the current state have trouble enforcing this[38].

Having most of the Internets communication rely on these companies and institutions, give them a lot of power. Once they are trusted by a users system they can sign, certificate for every website. The power of a trusted CA can allow a malicious party to assume the identity of any domain, and thereby break the PKI system and the trust that follows, at least until the breach is discovered

and disclosed.

### 2.1.1 Certificate Issuance Process

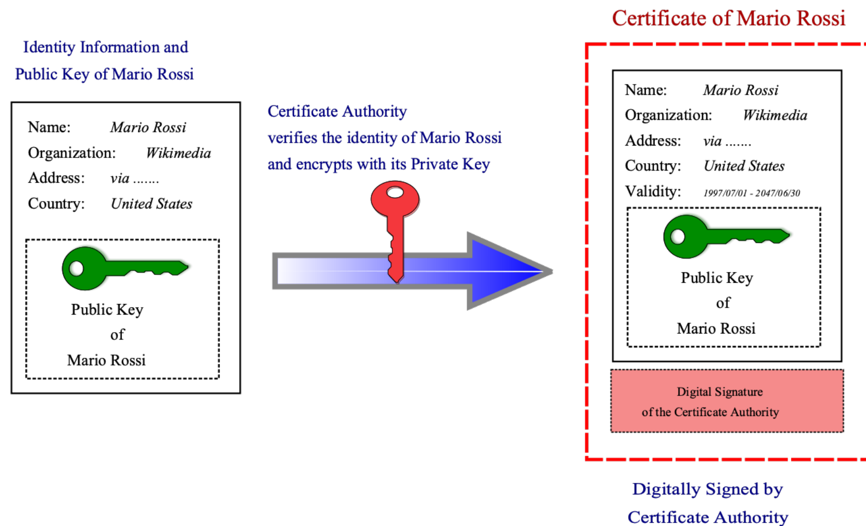



Figure 1: Diagram of the certificate issuance process[31]

Figure 1 shows the process of certificate issuance. The protocol works as follows:

A requester, Mario Rossi, needs a certificate to match his public key, so he can provide it to visitors of his website. He sends information about the certificate he wish to create, to a CA he trusts. What information is required by the CA to sign the certificate is up to the specific CA, and dependent on which verification protocol they use. Using Let's Encrypt requires at least providing[12]:

- Common Name
- Subject Alternative Names
- Subject Public Key Info

Where common name is the name of Mario's domain. Subject alternative names is a list of all the domains associated with the certificate, for example Facebook uses the same certificate for facebook.com and messenger.com. Lastly the Subject Public Key Info is the corresponding public key to Mario's private key.

This information is used to create a Certificate Signing Request (CSR), and Mario can then ask Let's Encrypt to sign the CSR, which will create the signed certificate.
Before Let's Encrypt is willing to sign the CSR, Mario needs to prove that he owns the domain. This can be done using Let's Encrypts DNS challenge as explained on their website[13] and or in the ACME protocol[29]. Once the domain validation have passed, and Mario is considered the owner of the domain, he will receive a certificate that he can use to prove his domains authenticity.

## 2.2 Decentralization as a solution

This paper proposes to decentralize the power of the CA by decentralizing parts of the certificate signing process. The paper suggests two solutions.

The first is a Decentralized Autonomous Organization that acts as CA, the description of this can be found in section 4.3.

The second proposed solution is a decentralized application that act as CA. Anyone can participate in the decentralized application, provided the have the capital necessary. This is described in detail in section 4.4.

Decentralization is not without it's obstacles. Coordination between multiple participants, is always difficult, especially when they have to arrive at consensus. This is made more difficult by the existence of potential malicious participants.

### 2.2.1 Decentralized Certificate Authorities

To alleviate the problems with centralization as mentioned in section 1.2 it seems that decentralization through blockchains, as described in section 2.2.2 might offer possible solutions.

We seek to accomplish two goals with our decentralized CAs
First, it should be an atomic action to accept a certificate, and have it directly included in a decentralized Certificate Transparency log as described in section 2.2.3. There should be no middlemen between the decentralized application and the CT log. This will ensure that newly issued certificates are broadcast immediately to anyone watching the CT log.

Secondly; Every certificate request should require more than one approval to be accepted by the decentralized application. Multiple participants in the decentralized application, would need to accept the certificate before it is considered valid. This will fix the problems of CAs being single points of failure, with more honest participants requiring more failures on part of the participants as a whole.

Smart contract based decentralized applications, running on the Ethereum blockchain, will be used in an attempt to reach this goal.

### 2.2.2 Blockchain

The Ethereum blockchain, which launched in 2015, is a generalized, trusted computing platform, which allows anyone, who holds the native cryptocurrency

Ether, to publish and run programs on the platform. These programs are colloquially known as smart contracts. A smart contract is a set of instruction to be executed when a trigger happens. A contract on the Ethereum protocol is in of itself a fully-fledged Ethereum account. This means that the contract can send and receive assets and maintain a permanent storage. The programmable nature of the platform allows building a certificate transparency protocol on top of the blockchain, which is already inherently decentralized and hard to reorder[24].

Not only do the Ethereum blockchain allow for distributed, trusted computing, but it also allows us to program economic incentives. The introduction of programmable economical incentives, which are censorship resistant, provides a whole new toolbox for protocol and application design, as the transfer of value is now as reliable as the underlying platform, as opposed to the vulnerable and non-atomic transfers between financial institutions. This means that centuries of economic game theory, can now be incorporated into protocol design, allowing for novel approaches to old problems.

Blockchains come with some challenges as well, chief of them being the scalability issue. Every transaction on a public blockchain will create data, that will have to be duplicated on every node running in the system. As such, storing data on public blockchains is an expensive operation. Because of this, we will limit the scope of certificates in this project, to what we consider a minimal viable certificate. These certificates will only contain a public key of the requester, a common name for the URL, and an expiry time.

Ethereum have been chosen as the blockchain this project will build its solutions on.

**Smart Contracts**

Smart contracts are code that executes on the Ethereum VM when blocks are mined. Smart contracts are assigned their own public keys on the network upon deployment, this is what makes it possible to identify them. Transactions can then activate public methods in the smart contract. The code can be arbitrary because the programming language is Turing complete. This open up the possibility to write sophisticated applications.

Gas is a fee payed to execute code, like transferring assets or creating a new token. For this reason, a developer should focus on the quality of the smart contract so that it can be run at a reasonable price. Furthermore, Ethereum is not optimized for large amounts of data, since blocks have a maximum size based on gas. Ethereums blocks can contain around 8 million gas, which is roughly 380 transactions. Executing smart contract code is more expensive than transferring assets, as the user pays gas for every computation and read/write call.

It is interesting to consider how economic incentives can be applied to dispute resolution and punishment of malicious participants in a decentralized system such as Ethereum. Augur, a decentralized prediction market running on Ethereum, uses economic game theory to encourage participants to report the correct outcome of events. In the case of disagreement between which outcome

is correct, or a market maker maliciously deciding a wrong outcome, a dispute resolution can be activated which essentially lets people bet on the correct outcome being chosen. Since the outcome should be publicly known at this point, it is theorized that the result of the dispute resolution will converge on the true answer, as it becomes the Schelling point.

Smart contracts will be used to create the decentralized solutions for this project.

#### 2.2.3 Certificate Transparency

The goal of the Certificate Transparency protocol is to allow interested parties to discover misissued certificates[22]. The protocol aims to achieve this by creating publicly auditable, append-only logs containing issued certificates. This makes it possible for anyone auditing the log, to make sure that the certificates are correct. It's possible for anyone to run a log service. It should be noted that the logs themselves do not contain mechanisms to prevent misissued certificates to be committed to them, they simply ensure that auditors and monitors of the log are able to detect possible misissuance.

**How Certificate Transparency works:**
The Certificates Transparency protocol is centered around the logs, the monitors and the auditors.

The log is a server that contains certificates. The log should be a publicly auditable, append-only record of the certificates submitted to the log. Anyone can submit a certificate to the log. The certificate has to be grounded in a root certificate, so it is not possible to submit self signed certificates.
The monitor is a server that periodically contacts the log servers to check for suspicious activity within the log. They also verify that submitted certificates are visible in the CT log they monitor. They should regularly fetch updates for the logs, essentially having a complete copy of the log on their system.
The auditor is usually a piece of software, that verifies that the logs are behaving correctly, and that a specific certificate exists in the log. An auditor could for an example be a part of a modern browser.

The log, the monitor and the auditor, should gossip between each other, using information gathered from other sources to make sure that everyone is behaving nicely. If that isn't the case the Log that are misbehaving should explain itself, or risk being shut down or removed from the browsers trusted list of roots.[1]

The solutions in this project implement a CT log. They do so because, the certificate attesters participating in the decentralized applications won't sign the certificate, but instead cryptographically attest to it's validity. For this reason the certificates can not be considered valid unless observed in the CT log. Any CT found in the certificate log will be considered valid, as they can only get into the CT log by successfully going through the validation process.

### 2.2.4 Economical incentives

Economical incentives are used to reward participants of the system that behave as expected and punish misbehaving activity. While many different ways to exist to motivate people, economical incentives are uniquely quantifiable, and easy to implement without middlemen, on public blockchains like Ethereum.

**Incentives in smart contracts**

Since the Dapp-CA will be designed to have open participation, it is necessary to add economical incentives. This is to promote good behaviour in Dapp-CA while punishing bad behaviour.

The Dapp-CA will charge requesters that need a certificate, a sum of money for each certificate they request. This value should be distributed evenly amongst the attesters, participating in the attestation of that certificate, no matter if they choose to include the certificate or not. This encourages requesters to request well formed certificates, to avoid losing their money for nothing.

In the Dapp-CA, malicious participants that try to include a malicious certificate into the CT-log should be punished by having some of their money slashed. Ultimately, if there's enough conflict between attesters about the validity of a certificate, the protocol should fork in two, and the market should be the arbiter of which fork carries the correct certificate.

This section can then be condensed to 3 goals the Dapp-CA should achieve:

- Reward honest attesters.
- Slash malicious or wrong attestors.
- Fork if no consensus can be reached.

This project aims to design these economical incentives into the smart contracts of the Dapp-CA.

# 3 User Guide and Requirements

## 3.1 Prerequisites

All code needed to run this project can be found on www.github.com at it's repository at 'Mrostgaard/DAO-CA'.[8]

## 3.2 Smart contracts

Usage of the smart contracts require two steps. The deployment of the contracts, and the interaction with them. Both deployment and transactions can be done on main-net Ethereum, a test-net, or simulated locally.

### 3.2.1 Prerequisites

**Wallet software**
The first step of interacting with the Ethereum blockchain, is funding a wallet with Ether. A wallet is essentially just a public-private key pair. This paper recommends using the browser extension MetaMask [15], which allows the user to manage multiple public-private key pairs and recover lost private keys using mnemonic key phrases, while also integrating with different Ethereum test-networks.

### 3.2.2 Deployment

To interact with the Ethereum blockchain, it is first necessary to obtain Ether to pay for transactions. This can be done by mining Ether using mining software, or buying Ether on one of many online exchanges. Both of these options are expensive, and we will recommend using the Ropsten testnet instead. The Ropsten testnet is a separate Ethereum blockchain intended only for smart contract testing and development, therefore the Ether currency on that network is freely given away on faucet websites like `https://faucet.ropsten.be/`.

There are many ways to execute and interact with Ethereum based smart contracts. For deploying smart contracts using a browserbased wallet, we will recommend using either Remix[19] or MyCrypto[18]. MyCrypto let's you interact with Ethereum, by constructing transactions on their website, and sending them with your local wallet. This can either be a hardware wallet or a browser based wallet like MetaMask. Once the transaction has been constructed, your wallet software will ask you to confirm the transaction, and allow you to modify gas price. `https://ethgasstation.info/` is a useful website for estimating the best price to pay for gas if using the Ethereum main-net. If using a test-net, gas price estimation is less of an issue.

The remix IDE is an online solidity editor, which lets users edit, compile and deploy smart contract code. This makes it very easy to interact with smart contracts, as there's no need to work with transaction bytecode. For extensive testing, we recommend using this tool, as it allows for both sending transactions

on main- and test-nets, along with running transactions locally in a JavaScript based virtual machine.

As the projects code is purely a proof-of-concept, we do not recommend spending real Ether on running or deploying the smart contracts. The rest of this user guide will be presented as if using the Remix IDE.

To compile any contract using the Remix IDE, it's first necessary to select any compiler version above version 0.5.0 as the projects code depends on a compiler version above this.

To deploy the DAO-CA, load *MinDAO.sol* into the Remix IDE, along with the *CertificateTransparency.sol* contract. Press 'Start to compile' as seen in figure 2.

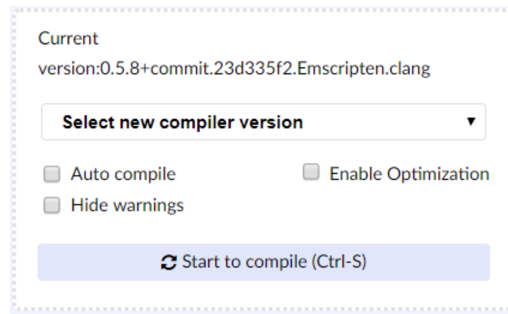



Figure 2: Compilation view in the remix editor. Compiler version can be selected from the dropdown menu.

Navigate to the 'run' tab, which is located in the top right on the page. You will be given a choice of environment as seen in figure 3.
'JavaScript VM' let's you simulate transactions locally.
'Injected Web3' let's you connect to an Ethereum network, either main-net or a test-net like Ropsten. When using the JavaScript VM, a simulated account will be created with 100 ether, while the 'Injected Web3' will try to connect to a browser wallet, like MetaMask. When deploying contracts, the gas limit sets the maximum amount of gas that can be used, if more gas than the gas limit for the transaction is needed, the transaction fails without imparting any state change on the blockchain. As such, it's important to use a gas limit higher than the deployment cost of the contract. Unspent gas will be refunded, unless the transaction fails. We recommend setting a gas limit above 6 million when deploying the smart contracts. Note that 8 million gas is currently the upper limit for gas that can be used in a block. To deploy the DAO smart contract, simply press the deploy button. The DAO-CA smart contract take no parameters.

Deploying the Dapp-CA is a little more involved.
Firstly: Load the *Controller.sol* contract, the *CertificateTransparency.sol* contract and all the token contracts into the Remix IDE. The first step is to compile the Controller.sol smart contract.

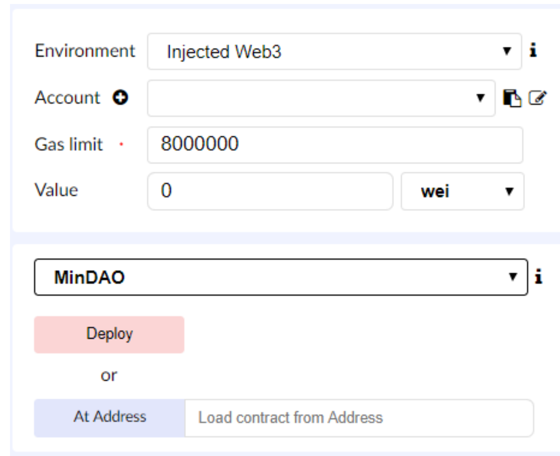



Figure 3: View of the deployment tab in Remix, when ready to deploy the DAO-CA.

Secondly, deploy the *ControllerFactory.sol* contract. This smart contract is a necessity, since smart contracts cannot recursively deploy themselves, so this contract is needed for forking.
Once deployed, the *Controller.sol* contract can be deployed. It take three parameters, one of them being _childCreator, which is the address of the ControllerFactory, one is the _parent, which should be set as the address of the deployer, and the last _isGenesis, which should be set as true, on deployment.

Once the transaction is successful, it will return a public key, which is the address of the smart contract. When the smart contracts are deployed, their methods become callable through the Remix IDE interface.

#### 3.2.3 DAO-CA

The DAO-CA has six public callable methods as seen in Figure 4. A public callable method is a method that can be called by any transaction coming from outside the contract, and it's used to interact with the contract. To call a method an Ethereum transaction is required, it should be sent to the address of the deployed smart contract and should contain the necessary data to target the specific method, using the correct parameters. Tools like the Remix IDE can generate this transaction data automatically, given it has access to the smart contract code.

- **requestCertificate**
  This method can be invoked using the URL of the specific domain, and a minimally viable certificate. A *requestID* is returned on a successful method call. This requestID will be used by attesters to attest the request.

| | |
|---|---|
| attestCertificate | uint256 requestID |
| finalizeCertificate | uint256 requestID |
| proposeKickMemb er | address kickMember |
| proposeNewMemb er | address newMember |
| requestCertificate | string _url, string _certificate |
| vote | uint256 id, bool voteYes |

Figure 4: Public callable methods of the DAO-CA and their required inputs, as seen in the Remix IDE interface.

- **attestCertificate**
  Once a certificate has been requested, members of the DAO-CA can attest to their validity. They do this by supplying the requestID as a parameter to the attestCertificate method. If a member do not find the certificate valid, they should refrain from attesting it, as the certificate will only finalize once more than 50% of the members have approved it. The finalizeCertificate method is automatically invoked when enough DAO members have attested the certificate.

- **finalizeCertificate**
  In cases where a DAO member may be kicked, reducing the threshold for invocation of the finalizeCertificate method, the function can be called separately with the requestID as a parameter, but will only complete if 50% or more of the DAO members have attested the certificate. This method finalize the certificate, pushing it into the certificate transparency log.

- **proposeKickMember**
  The proposeKickMember method takes an Ethereum address as an input, which is associated with a DAO member, and creates a vote to kick that member.

- **proposeNewMember**
  The propose KickMember method takes an Ethereum address as an input, which is not a member of the DAO, and creates a vote to include that member.

- **vote**
  The vote method takes a voteID and a boolean. If the boolean is true, the vote is voted yes on, if it's false, it's voted no on.

### 3.2.4 Dapp-CA

The Dapp-CA has six public callable methods as seen in figure 5.

| | |
|---|---|
| finalizeVote | uint256 _id |
| fork | uint256 certId |
| parentForceCert | address certOwner, bytes32 certificate, bytes32 u |
| request | string url, string certificate, uint256 expiry, uint256 |
| vote | uint256 _id, uint256 _choice, uint256 _amount |
| withdraw | uint256 _id |

Figure 5: Public callable methods of the Dapp-CA and their required inputs, as seen in the Remix IDE interface.

- **request**
  The request method is used for certificate requests. It takes the string url, and string certificate, along with an uint expiration date for the certificate, and uint value, which is the bounty paid to have the certificate attesters. This methods returns a *requestID*.

- **vote**
  The vote method is used to vote on the inclusion of a certificate. It's called with a requestID, a choice which is either 0 for acceptance, or 1 for rejection, and the uint amount, which is the amount of tokens staked on this outcome.

- **finalizeVote**
  The finalizeVote method is called with a requestId. The method will finalize the certificate request if accepted or rejected, or fork the smart contract if the fork threshold was reached.

- **withdraw**
  The withdraw method is called with a uint requestID, and is used to withdraw GOV tokens that are owed from a specific certificate finalization. In cases where a request has been made invalid by a fork, the withdraw method is used as well.

- **fork**
  The fork method can be called with a requestID. It will only be successful if the parameters of that certificate request lives up to the parameters of forking the smart contract.

- **parentForceCert**
  This method exist as a way for parent contracts to force a child to accept the inclusion of a contract. It can only be successfully called by a parent contract, and is only used upon forking, to include the disputed certificate request on the accept side of the fork.

## 3.3 Firefox extension

The Firefox extension takes most of its code from the Github user april's repository 'certainly-something'[6] and adds functionality to communicate with Ethereum through 'Infura'[10].

### 3.3.1 Prerequisites

The browser extension is based on modern web technologies and for this reason it uses Node.js and Node Package Manager (NPM) to handle the compilation of the source code into a file that can be loaded by the browser.
It is necessary to have NodeJS and NPM installed. The lastest versions of both Nodejs and NPM can be found at https://nodejs.org/en/ which is the official website of Node.js. For documentation and modification options please see the master repository, which our extension have been forked from.[6]

### 3.3.2 Extension building

To be able to use the extension in the browser it needs to be built. To do this navigate to DAO-CA/Firefox-extension/.
From the position in the folder 'Firefox-extension' run:

```
1   $ npm run-script compile
```

The command will run the package manager and create the new folder '../build/' that contains the created files.
To make the extension run in the browser, open Firefox and type

```
1   about:debugging
```

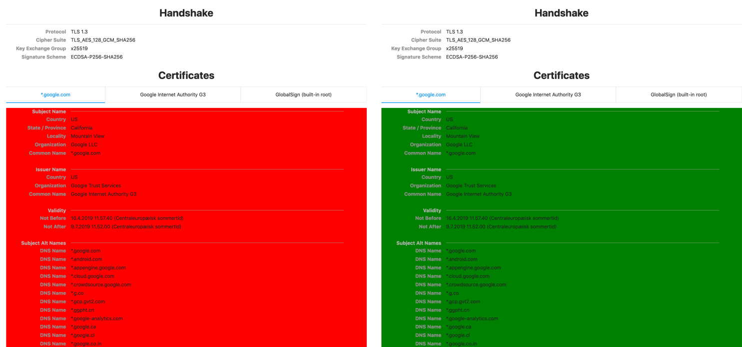
into the Firefox URL bar.

This will present the Firefox debugging page. To load the extension press 'Load Temporary Add-on'. Navigate to the DAO-CA/Firefox-extension/build/ and select the *background.js* file.
This will load the file into Firefox and a small icon on the right side of the URL bar should now be visible as seen in figure 6

To use this application, please visit any site, or refresh a given site if you have just loaded the application, press the icon as showed in figure 6. This will open up a new tab and display information regarding the certificate for that website. If the certificate exists in the decentralized CT log, the background

Figure 6: Firefox extension icon display

will be displayed as green, otherwise it will display a red background as seen in figure 7.




(a) No accepted certificate associated with the domain.

(b) Accepted certificate associated with the domain.

Figure 7: Displaying not accepted / accepted certificate view

# 4 Method

The initial idea behind decentralizing CAs was to create a Decentralized Autonomous Organization as a Certificate Authority, DAO-CA for short. During development it became clear that the solution didn't have a failure recovery mechanism, and for that reason we decided to research existing projects on the Ethereum network to see what could be adapted to fit the purpose of the project.
We found the project Augur[27], which is a decentralized prediction market using market incentives to create decentralized oracles. We decided, along with building the DAO, that we would try to modify this decentralized oracle mechanism for verifying DNS ownership. If adaptable, it would make it possible to build a completely open system, guided only by economic incentives.
A Firefox extension was created to show a proof of concept, and to provide an example as to how a user, or a piece of software could verify the certificates accepted through the DAO-CA and Dapp-CA.

While the DAO approach works, it is vulnerable to capture. Once the DAO goes into a state where, it's controlled by a malicious cartel, e.g more than 50% of the members are compromised or acting maliciously, it would be broken for good. We attempted to develop a decentralized application, with the goal of making it robust when under attack. This is the 'Dapp-CA'. In the following sections, we will go into detail of the implementation of both solutions.

## 4.1 Certificate Transparency Log

Both of the DAO-CA and the Dapp-CA, use the same basic Certificate Transparency log (CT log) smart contract. The smart contract allows a specific public key, referred to as the owner, to store certificate hashes on the blockchain. The smart contract can be owned by an individual, but is in both of the implementations, owned by the smart contract in charge of accepting or rejecting new certificates. Every smart contract on the Ethereum blockchain is assigned a unique public-key on creation, which will be given to the CT log as the owners address. The purpose of the CT log is to provide a single source of truth, in regards to which certificates have been approved by the DAO-CA or Dapp-CA. Since only the DAO-CA or Dapp-CA can accept the certificate, the inclusion of the certificate in the CT log functions as a surrogate for a signature. Essentially the DAO-CA and Dapp-CA function more as attesters to the validity of a self signed-certificate.

The implementation of the CT log is based on the work of Eiler Ka-Nid Yodla-Ong Poulsen  Martin Bøgelund Hansen[24], but have been optimized to scale better, while requiring less lines of code.

The CertificateTransparency smart contract implements the following data structure, as the minimum representation of a certificate:

```
1 struct certificate {
2     address owner;
```

```
3       bytes32 urlhash;
4       bytes32 certhash;
5   }
```

The *owner* field is the public key of the owner of the certificate. The public key will be the same one used to request the certificate from either the DAO-CA or Dapp-CA. The *urlhash* field is a byte array of length 32, containing the hash of the URL of the domain the certficate belongs to. The certhash field is a byte array of length 32 as well, containing the hash of a specific string representation of the rest of the information pertaining to the certificate, which in a minimal implementation would be the expiry date of the certificate. The certificates are saved as hashes rather than a full string representation. This is mainly to save space, which is both expensive, and a limited commodity in blocks on the blockchain. The full string representation should be the public-key as hexadecimal, common name, and unix epoch expiry, and may look like the following:

```
"19f45ea63b9d9b864ae2eee603e7b106df875754,google.com,1559500788"
```

It may be desirable in some cases to store full X.509 certificate information on the blockchain, so as to have a decentralized register, but it was decided to limit the scope for the project to purely deal with verification of certificates. In the same regard, we only hash and store the subset of certificate information, required to validate the authenticity of a minimal viable subset of certificates. Certificates will only contain three fields:

- **Common name**
  The common name is the domain name that the certificate requester claim to own. The certificates in our system do not support Subject Alternative Names, meaning that one certificate can only be linked to one specific domain.

- **Expiry time**
  The expiry time is the time in which the certificate expires, represented as a epoch time unsigned integer.

- **Owner public-key**
  This is the public key of the certificate requester, which will be used to play the DNS challenge, and is also the public-key used for the certificate.

Checking a certificate against the certificate log can be done by calling the view function *check*. Its smart contract code is exceedingly simple:

```
1       function check(bytes32 hashedUrl, bytes32 hashedCert) public
            view returns (bool) {
2           require(certificates[hashedUrl].certHash != 0);
3           require(hashedCert != 0);
4           return certificates[hashedUrl].certHash == hashedCert;
5       }
```

The first line checks whether or not a certificate for the hashedUrl exists in the certificate transaparency log. The second line makes sure the hashedCertificate isn't equal to the default value of uninitiated bytes32 fields. The third line then checks whether or not the certificate is equal to the one supplied to the check function.

The check function is called with the SHA256 hashed URL and a SHA256 hashed minimal certificate received by the website, which will be checked against the stored certificate hashes. If a match is found, the certificate is deemed valid, and the smart contract will return true, if not, it will return false.

View functions are a subset of Solidity functions, which do not cause state changes on the blockchain. This means that transactions aren't necessary to perform the calculations, but it is still necessary to have access to the active state of the Ethereum blockchain, either through a light client or a transaction relayer. The proof-of-concept Firefox extension uses Infura[10], a centralized transaction relayer, which frees us from running a local light node of the Ethereum blockchain. This introduces a security risk, as Infura becomes a single, trusted source of truth, and by extension, a single point of failure in a system designed to decentralize away that problem. Infura, or any other transaction relayer, would be capable of forging certificates and assuming the identity of the visited website. Current research[34] is on the way to reduce the computational requirements of running Ethereum light-clients in web browsers, but none are currently commercially available or practical. The ability to trustlessly access information on the blockchain, is essential for the secure operation of blockchain based certificate transparency logs. Access to data on the Ethereum blockchain, that isn't vulnerable to man-in-the-middle attacks, are crucial to eliminating the reliance on trusted third parties of the proposed systems, and only truly possible by running a node that takes part in the network.

## 4.2 DNS challenge

The DNS challenge is the specific approach used to verify the domain owner. There are many verifying games that could be played by the participants of the decentralized applications; Each attester could call the requester on the phone, and ask them to verify they have made the request, and then ask them to perform some action on the server the domain points to, like setting up a URL pointing to a page with some text specified by the verifier to further prove it. They could send a letter to the address specified in the DNS record, if any, with a series of numbers that should be put into a form on their website, which could be used to verify the address.

This paper proposes a more direct way that can be mostly automated on the verifying side and is easy to perform on the requesting side, yet it only verifies control of the DNS record.

For the requester to prove ownership of a domain, they must change the DNS text to some identifiable information, agreed to by all participants in the verification process. Public-key-pairs provide unique public keys, and the public key is known to all participants of the system at the time of the request. The

request can only be made if the requester has a public-private key pair, as the public-key is used as an address on Ethereum, and the private key is used to sign transactions. For this reason, it is easy to connect the domain owner to the requester, and confirm they are either the same person, or working together, if the DNS text for the requested domain, is changed to the public key of the requesting party. In the case that the requesting public key and the DNS text key is the same, it is fair to assume that the requesting party have control over the DNS record for that domain and by extension owns it.

This approach to proving ownership of a domain is heavily inspired by Let's Encrypt. As Let's Encrypt 'Domain Validation' protocol[13], also will demand the requesting party to update their DNS text to reflect a text given by Let's Encrypt. This domain validation is one approach, amongst a few that Let's Encrypt offer. Our approach differ, in that we never communicate with the certificate requesters server, but only verify DNS domain ownership, via the DNS text field.

## 4.3 DAO-CA

The DAO-CA implementation is an implementation of the Decentralized Autonomous Organization concept, which is a blockchain based organization controlled by on-chain voting. Each member of the DAO has one vote, and can vote in matters related to the DAO. The only way to attain more power than other DAO members, is to control more membership addresses. The DAO-CA includes includes certificate requests, by voting. 50% of the votes is enough to get a certificate included in the CT log. DAO members can also propose and vote on adding and removing new DAO members.

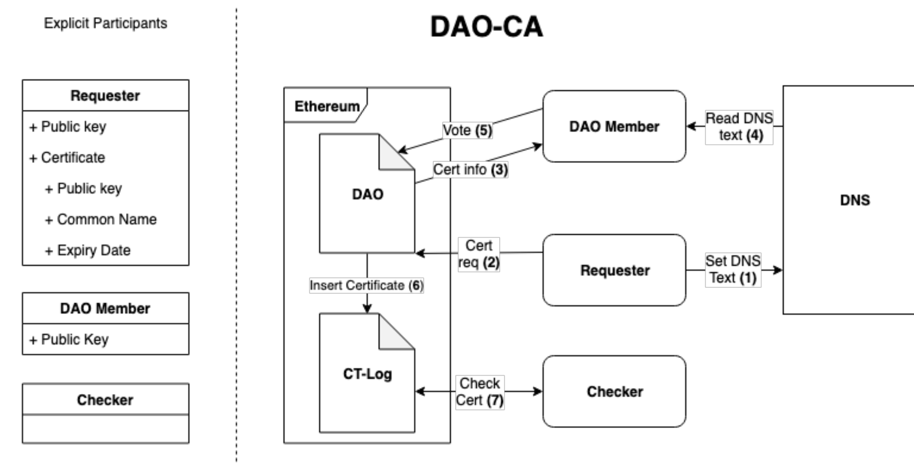



Figure 8: Overview of the DAO certificate issuance flow.

Figure 8 shows a overview of how the DAO-CA works and it's participants.

As can be seen on the left side under 'Explicit Participants' we have 3 participants actively engaging with the system; The Requester, DAO members and Checker. The Requester requests that his certificate is put into the CT log. Once enough DAO members vote to include the certificate, the certificate will be included in the CT log. The Requester has two values associated with him, his public key that he uses to send transactions to the DAO contract with, and the certificate information he wants to add to the CT log.
DAO membership gives power to attest to a certificate request, to create and vote on proposals to add other members to the DAO, and to create and vote on proposals, to remove other members from the DAO.
The last participant is the Checker. They're the user of the Firefox Extension as described in section 4.5. The Checker is a piece of software, or a person, which checks the visited website's certificate against the CT log to determine whether or not the certificate is valid. The user can safely proceed to use the site that is visited if the certificate is valid, if not, the website should probably not be trusted.

The process a certificate request goes through can be seen on the right side of the image, in the Figure 8.
The diagram has numbers representing, in order, the steps that is takes place in the protocol to accept a certificate into the CT log:

1. The Requester updates the DNS text through his DNS provider. The DNS text should be set as his Ethereum public key, as described in subsection 4.2.

2. The Requester sends a transaction to the DAO smart contract, calling the request method with the required parameters. The parameters are a string representation of the URL, and a string representation of the certificate.

3. DAO members can now read the certificate request from the DAO contract.

4. The DAO member ensures that the certificate is well formed, and will then get the DNS text for the domain, based on the certificate's common name. They will check the DNS text and verify that the DNS text is the same value as the Requesters public key, as described in section 4.2.

5. Depending on the result of step 4 the DAO member will then vote to accept the certificate request if the DNS text record is valid, if not, they will take no further action.

6. If 50% of the DAO members have gone through the process described in step 2-5 and voted to accept the certificate into the CT log, it will be accepted. There is no time-frame for when the DAO members have to decide on the validity of the certificate, and it could potentially take forever. However, every certificate request comes with a bounty, that will be paid upon inclusion.

7. A browser user visits a website that have successfully gone through step 1-6 and their certificate is now in the CT log. The browser extension will then see the certificate in the log and will tell the user of the browser that the certificate is valid.

Step 1-6 described above is the process for which a certificate requester gets their certificate request approved or denied.

### 4.3.1 Smart contract

The logic of the DAO-CA is encapsulated in the MinDAO.sol smart contract. It has two main data structures request and proposal.

```
1 struct Request {
2     address owner;
3     uint bounty;
4     string URL;
5     string certificate;
6     uint16 numAttesters;
7     bool issued;
8     mapping (address => bool) hasAttested;
9 }
```

The request data structure contains all the information necessary to attest to a certificate. The owner field is the Ethereum address of the requester. The uint bounty, is the amount of Ether paid for the inclusion of the certificate. The string URL and string certificate is the information necessary to validate the ownership of the certificate. These values will be used to play the DNS challenge game described in 4.2. The 16bit uint numAttesters is used to check whether or not 50% or more of the attesters has voted for inclusion. The boolean issued field marks whether or not the certificate has been included, and the hasAttested mapping, maps DAO member addresses, to whether or not they have already attested to the certificate.

```
1 struct Proposal {
2     address subject;
3     uint16 yays;
4     uint16 nays;
5     bool isKick;
6     mapping(address => bool) hasVoted;
7 }
```

The proposal data structure is simpler than the request, simply containing the necessary information to vote yes or no on a proposal. The subject field, is the Ethereum address of the new DAO member to be included, or the old DAO member to be kicked. The yays and nays fields are the votes for and against the specific proposal. The isKick boolean defined whether or not it's an inclusion or exclusion vote. Finally, the hasVoted field, is simply a mapping between DAO member addresses and a boolean value, showing whether or not they have voted on the specific proposal.

## 4.4 Dapp-CA

The Dapp-CA implementation tries to decentralize the certificate attestation process even further than the DAO-CA. Where the DAO is compromised as soon as more than 50% of the participants are adversarial, and will remain as such, the goal of the Dapp-CA is to be robust even in the face of continual attacks. To achieve this, the design had to get rid of the gate keeping element that is membership, and instead allow anyone to participate openly. The Dapp-CA will rely on economic incentives, to achieve desired behaviour. This means rewarding beneficial participation, and punishing malicious participation.

To incentivise particpation in the protocol, it's required by anyone who requests a certificate to post a bounty, which will be awarded to the attesters. Crucially, this bounty will be awarded to attesters no matter if they accept or reject a proposal, to avoid spam attacks on the protocol.

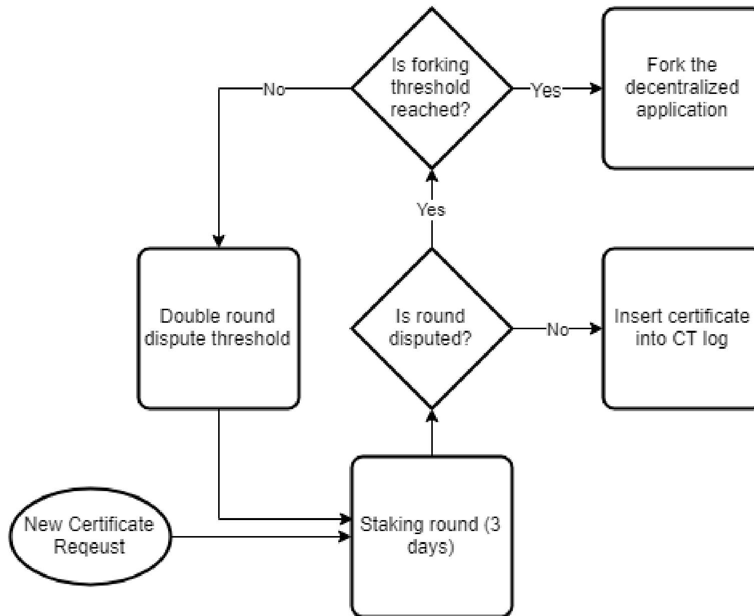



Figure 9: Flowchart of the Dapp-CA. When a certificate is requested, a staking round begins. If no consensus can be found, it goes into another staking round. If consensus can't be found before reaching the FORKING_THRESHOLD, the decentralized application will fork in two, with the conflicting sides being migrated to each fork.

The Dapp-CA introduces an economical token called the GOV token. This token is used to vote on rejecting, or accepting a certificate. Voting on either rejecting or accepting is called staking. The token can be sold and traded by anyone who owns them. As seen in figure 9, once a certificate is requested,

a staking round will begin. Within this period, holders of GOV tokens can stake their token on the validity on the request, by participating in the DNS challenge game described in section 4.2. At the end of a staking round, if the round is not disputed, the certificate can be finalized and included in the certificate log. On the hand, if it is disputed, the dispute threshold will be doubled, and the decentralized application will enter another staking round. A staking round is considered disputed, if there are staked more tokens than the *DISPUTE_THRESHOLD* on both the 'Accept' and 'Reject' option. The DISPUTE_THRESHOLD is a variable uint, which represents the amount tokens needed to be staked on both outcomes, to move to the next dispute round.

In the worst case scenario, the certificate remains disputed through 7 dispute rounds, and will reach the FORKING_THRESHOLD. The FORKING_THRESHOLD is a uint representing the percentage of tokens needed to be staked on both 'Accepted' and 'Rejected', to force the decentralized application into the forking process. Since Solidity doesn't have a decimal representation, each increase in the integer FORKING_THRESHOLD by one, increases the forking threshold by 0,1%. At this point, the dispute is considered unsolvable, and the Dapp-CA will fork in two, moving the GOV tokens that voted 'Accept' to one fork, and the GOV tokens that voted 'Reject' to the other fork. The forking procedure is described in more detail in section 4.4.4.

### 4.4.1 Smart contracts

The Dapp-CA consists of four smart contracts; Controller.sol, StakableToken.sol, MigrateableToken.sol and the aforementioned CertificateTransparency.sol:

**Controller.sol**

The controller contract is responsible for handling all logic pertaining to requesting and voting on prospective certificates to be included in the CertificateTransparency contract. Upon initialization, the Controller contract will also initialize a GOV token contract and a CertificateTransparency contract. The Controller contract act as the owner of these smart contracts, which means that only transactions going through the Controller contract are allowed to fork these owned contracts, or include certificates in the CT log.

The Controller contract has two main data structures, the *Vote* structure and the *Cert* structure. The Vote structure has the following data fields.

```
1  struct Vote {
2      uint certId;
3      uint accepted;
4      uint rejected;
5      uint voteStart;
6      uint voteEnd;
7      uint round;
8      uint bounty;
9      bool isFinalized;
10     mapping(address => uint) stakedAccepted;
11     mapping(address => uint) stakedRejected;
```

```
12 }
```

The vote structure contains all information necessary to conduct the voting process, for whether or not to include a certificate. The *certId* refers to a specific certificate request. The *accepted* and *rejected* integers count how many GOV tokens have been staked on either choice. These counts will ultimately be used to decidedwhether or not a certificate is included. The *voteStart* and *voteEnd* integers are epoch timestamps, representing the start and end of each voting round. Whenever a voting round is over, if the vote result cannot be finalized, the timestamps are updated by adding the *VOTE_PERIOD_TIME* constant, which is the number of seconds a dispute round takes. There is also a field keeping a count of the current dispute round. This field is necessary to know what the dispute threshold is, and thus, whether or not a vote can be finalized or will need to go through another dispute round. The *bounty* field merely keeps a count of what the amount of GOV tokens the winners of the vote will be awarded. Once a vote is finalized, the amount of staked tokens of the losing side is added to the bounty field. The *isFinalized* field is a simple boolean indicating whether or not the vote has completed its life cycle. Finally, the two fields *stakedAccepted* and *stakedRejected* are mappings, where public keys or addresses in Ethereum vernacular, maps to the amount of GOV tokens that has been staked on either the accepted or rejected outcome. These values are ultimately used to calculate the share of the bounty a successful attester will receiver, or the amount of GOV tokens a losing vote will cost the losing attester.

```
1 struct Cert {
2     uint certId;
3     uint expiry;
4     string memory url;
5     string memory certificate;
6     address certOwner;
7 }
```

The Cert data structure contains all the information necessary to validate a certificate. The *certId* is just an incremental count used to keep track of the different votes and certificates being voted on. The *expiry* integer field is an epoch timestamp of the certificates expiry time. The *url* field is a string memory field containing the string representing the certificates common name. The *certificate* field, is string format representing the required information for validating the certificate, but is not used for any computations on-chain. The *certOwner* field is the public key, more specifically the Ethereum address, of the requester of the certificate.

**StakableToken.sol & MigrateableToken.sol**

The StakableToken and MigrateableToken contracts are extensions to the ERC-20 standard token. A token is just a mapping between addresses and a uint256, representing the number of atomic units of that token the address owns. The ERC-20 interface implements methods for transferring tokens between addresses, and checking the balance of addresses. The StakableToken extends the ERC-20 standard with the ability to lock an amount of tokens as a 'Stake' until an

expiry, where the tokens can be withdrawn or moved again. This is used for locking tokens up in the Controller.sol contract.

The MigrateableToken extends the StakeableToken, which inturn extends the ERC-20 token. This token allows migrating tokens from it to child contracts. These are the MigrateableToken contracts created by a fork in the associated controller. Once the fork is activated in the controller, the fork method is called in the parent MigrateableToken, which freeze all transfers in the parent MigrateableToken. The freezing of tokens force token holders to use the *migrate* method to migrate their tokens to one of the child tokens, to resume transfers. The ERC-20 token also implements two string fields, one for the token name, in our case we will use "Governance Token" and a symbol, in this case, "GOV".

#### 4.4.2 Certificate attestation

Malicious behaviour should be punished. There are three kinds of malicious behaviour when it comes to interacting directly with the Dapp-CA smart contracts.

1. Requesting false certificates
2. Staking to accept a false certificate
3. Staking to reject a true certificate

To punish the request of a false certificate, it's only necessary for the attestation process to work as intended. Any certificate which is deemed invalid, either wrongly constructed, or not in fact owned by the requester, should be rejected by the attesters. A standard protocol for checking the validity of a certificate can be followed as described in section 4.2. This behaviour will incur an economic loss in form of the certificate bounty. Along with the standard Ethereum transaction cost, this will result in great expense to the attacker, if repeated false certificate requests were made.

To punish false voting, either voting for accepting an invalid certificate or voting to reject a valid certificate, we introduce strict slashing conditions on voting against the majority. Any staked coins which are in the minority on a vote, will be given to the winners of the vote. As long as there's an honest majority of active stakers, the malicious stakers will be slashed and have their tokens transferred to the honest stakers.

A system like this alone is subject to hostile takeover, by any entity rich enough to buy a majority stake in the active staker pool. An incentive to oppose these hostile takeovers exist, as winning a vote will grant you the tokens of the opposing side, but it cannot be guaranteed that a malicious agent cannot amass the necessary amount of tokens to successfully control the protocol. To combat this worst case scenario, we allow the protocol to fork in case of a large enough disagreement between token holders.

The staking game of the Dapp-CA differs slightly from the voting of the DAO-CA, as there's a chance of economic loss. As seen in figure 10, the attesters

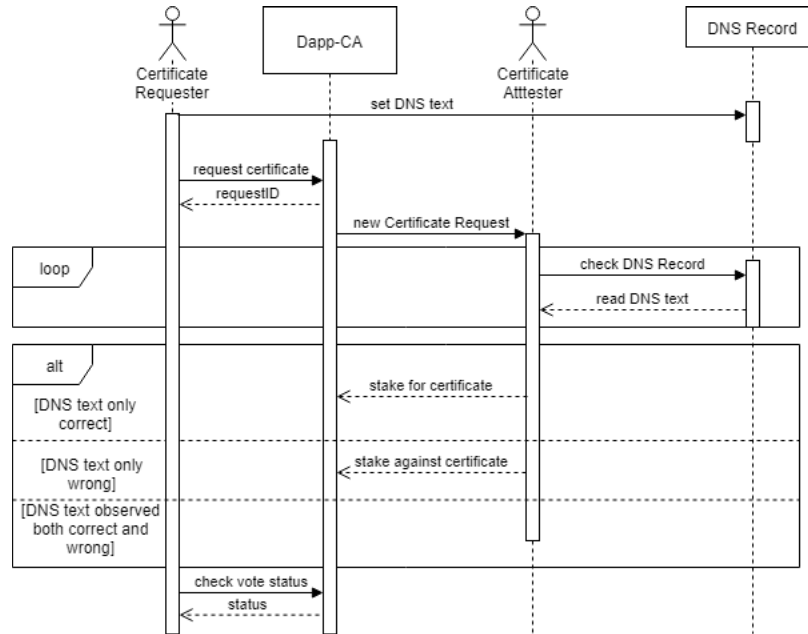



Figure 10: Overview of the attestation process in the Dapp-CA.

participating in the Dapp-CA should check the DNS text record multiple times. This is to reduce the chance of being victim of a bait attack, discussed in section 6.1.4. If the attester only sees correct information after many DNS reads, they can feel more confident that they're not the victim of bait attack and vote for acceptance. If they only see wrong information, they can too can feel secure in voting for rejection. However, if they spot both a correct and incorrect DNS text, they should refrain from participating in the staking process, as a bait attack may be occurring.

### 4.4.3 Governance Token

Any fees in the protocol are paid with the GOV tokens. A token on the Ethereum network refers to tradeable units, either fungible or non-fungible, which can be owned by addresses, and transferred to other addresses. The token is implemented as a standard ERC-20 token[7]. The ERC-20 token is an open token standard for fungible tokens, which essentially behave in the same way the native Ether crypto-currency behaves, with the notable exception of not being able to pay for transaction costs. The ERC-20 standard is implementable by other smart contracts, and allow the tokens to easily be transferred and accepted in the greater Ethereum smart contract ecosystem. It is assumed that reducing the friction of trading the token, will help it attain value in the market.

The reasons for using a governance token are two-fold:

1. To ensure that holders, and by extension, stakers of the token are financially motivated to act in the best interest of the Dapp-CA protocol.

2. To have control of the issuance of the token, so as to be able to fork, issue, duplicate and burn tokens when required by the protocol.

We believe the utility of having a certificate issued in the certificate transparency log will confer economic utility on the requester, which in turn will create demand for GOV tokens to pay the attestation bounty, and the attestation bounty will spur further demand for attesters, to buy and stake the GOV tokens to get a piece of the GOV token pie. This becomes a chicken-and-egg problem when having to bootstrap the initial value of the GOV token, but it may be possible to accomplish with some initial, charitable attestation work, or a guarantee to buyback GOV tokens for a minimum amount.

#### 4.4.4 Forking

In the controller contract, responsible for logic pertaining to certificate attestation, a certificate vote can reach a point where the amount of tokens staked eclipse the *FORK_THRESHOLD* ratio, the protocol will then fork into two protocols. This is similar to the forking of a crypto-currency, but not an exact analogy.

As seen in figure 11, if the amount of staked GOV tokens are pushed above the *FORKING_THRESHOLD*, the finalizeVote**(1)** method will automatically activated the forking procedure, by calling the Fork method**(2)**, and two new controller contracts, with associated certificate transparency logs and GOV token contracts, are created. Every non-finalized certificate attestation vote will automatically go to refund mode, allowing the requester to withdraw their bounty. The forked contracts will have a shared history of approved certificates, both of them pointing back to the same parent address, with the two new contracts representing alternate histories. One of the histories will have the disputed certificate accepted, and the other where it was rejected. The governance tokens staked on each result can only be migrated to the specific branch they were staking, forcing them to choose that specific history, and claiming the coins of the other votes. Thus the acceptors will claim the rejectors GOV tokens on the Accepted certificate fork, while the rejectors will claim the acceptors tokens on the Rejected certificate fork. Every other GOV token holder will have to choose which fork to migrate their tokens to. On the surface, this may look like an impossible choice, as choosing the wrong fork could have grave economic consequences. However, we can assume that one branch will always be malicious, while the other will always be honest. By playing the same game as the stakers, it should be trivial to verify which fork is the malicious one, and which is the correct one. There exist an attack that can subvert this, which we discuss in section 6.1.4.

Assuming it is trivial to verify which branch is the valid one, and which is

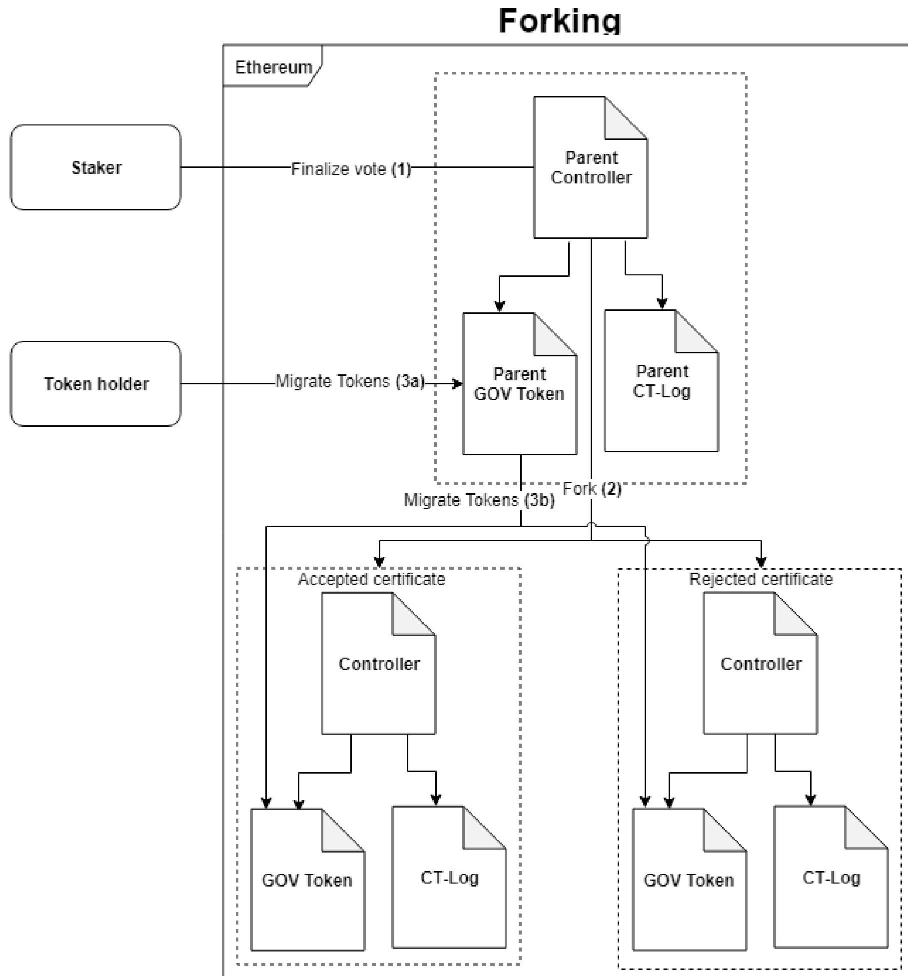



Figure 11: Overview of the forking process.

the malicious one, the market is likely to value the tokens of the valid fork much higher than the other, as the economic utility of having a certificate included in a certificate log controlled by a malicious actor is much lower. This will likely lead to the economically inconsequential fork to be abandoned, while the valid fork is likely to remain economically viable, maybe even more so, as there's now potentially a lower overall supply of tokens. At the same time, this attack is expensive for the attacker, as they will lose:

$$_TOKEN_SUPPLY * FORK_THRESHOLD$$

per attack, with the result of only pausing operations for a few weeks. Considering that multiple certificate logs of this design can exist in parallel, on both the same blockchain or multiple independent blockchains, an attack like this would even have to be executed multiple times, to bring the attestation of new certificates to a halt.

## 4.5 Frontend (Firefox implementationen)

As a proof of concept of for real world usage, a FireFox browser extension was created. This extension will run in the browser at all times and is located next to the URL bar. When pressed, it will open up a new tab and display information regarding the certificate for that page. The extension is mostly a copy of the Github user 'april's 'certainly-something' Firefox extension.[6] The implementation for this project extends the functionality of 'certainly-something' to make contact with the Ethereum test-net 'Ropsten', and will then make a call to the contract we created for this project, asking if the certificate for that domain is valid. In the case where the certificate is valid the background of the tab will be green, else the background will be red instead.

The Firefox implementation is agnostic about which smart contract it uses, as long as the contract implements the methods of CertificateTransparency smart contract. It will work with both the DAO-CA, and Dapp-CA. The extension calls the check method of the CertificateTransparency smart contract in the *utils.js* file. The method **verifyCertificate** takes 3 parameters: 'publicKey', 'expiryDate' and 'value'.
The publicKey is the certificate's public key. expiryDate is to ensure that certificates that are out of date aren't approved and value contains the common name. This information is collected, cleaned and a string is created from it.
The contract is contacted through the Infura transaction relayer, calling the check method of the CT log on the blockchain. If the result of the call is false, the method will return 'red' as a string, if the result is true then it will return 'green' as a string. The result of the function will be sent to the HTML file for the website and will then set the background to green or red as seen in figure 7.

# 5 Results

In the Ethereum blockchain, *gas* is purchased with Ether every time a transaction is conducted. The gas is payed to the miner who mined the block, and the amount of gas depends on the complexity of the transaction. The Ethereum Virtual Machine assigns every assembly operation a constant gas cost. Purely computational operations tend to be cheaper, than operations that require permanent state. The gas cost of operations change as Solidity and Ethereum is updated, and as such, the gas price of the following transactions are subject to change in the future. Currently there's a plan to introduce state-rent, only letting users store data as long as it's paid for, instead of allowing data to be stored in perpetuity. This might positively impact the price of storing a certificate, as certificates would only have to be stored until they expire, and do not require to be stored in perpetuity.

Every Ethereum block has a gas limit, which represents the total amount of gas that can be sold in a block. The gas-limit is variable, being dynamically decided by the Ethereum miners. At time of writing, the average gas-limit is roughly 8 million gas per block. As the gas limit increase, so do the average size of blocks, along with the computational requirements of validating them. As such the amount of uncle blocks, blocks which are correct under the consensus algorithm but are too late to be included in the blockchain, increase as the propagation time increases. Recent optimizations to the client software of Ethereum has sharply reduced the uncle rate, sparking proposals for dramatically increasing the gas limit.

## 5.1 DAO-CA Cost

In this section, the cost of running the DAO-CA protocol on the Ethereum blockchain is investigated. All gas costs are calculated by the Ethereum Virtual Machine.

- Deploy MinDAO contract gas cost: 2,486,398
  The MinDAO contract has less state than its Dapp-CA counter part, and only has the CertificateTransparency smart contract as a dependency, as such the deployment cost is fairly low.

- Certificate request gas cost: 127,614
  Requesting a certificate is relatively expensive, as it requires a fair bit of storage on the blockchain. The quoted price is the minimum cost, which is likely to be bigger for large certificates, and long URLs, as every character in the string incurs an additional cost.

- Attest certificate gas cost: 64,053
  Attesting a certificate is cheaper than an ERC-20 token transfer. This is due to the only state changes being the flipping of a boolean and the increment of an integer.

- Finalize certificate gas cost: 109,741
  Finalizing the certificate bears the cost of transferring state from the DAO contract to the CT log. It is only slightly more expensive than a token transfer, as it only uses five 256bit words of memory. Using the *delete* function upon finalization might reduce the cost of this transaction, as the function has a negative gas cost as a reward for signalling that the state is no long necessary to remember.

The minimum cost of requesting a certificate, voting on it, and including the certificate in the CT log ends up being equal to:

$$127,614 + 109,741 + N/2 * 64,053 \quad (1)$$

Where N is the number of DAO members. 127,614 is the cost of a certificate request, 109,741 is the cost of finalizing the certificate and 64,053 is the cost of voting on a certificate. The minimum cost with two DAO members is then 301,408 gas.

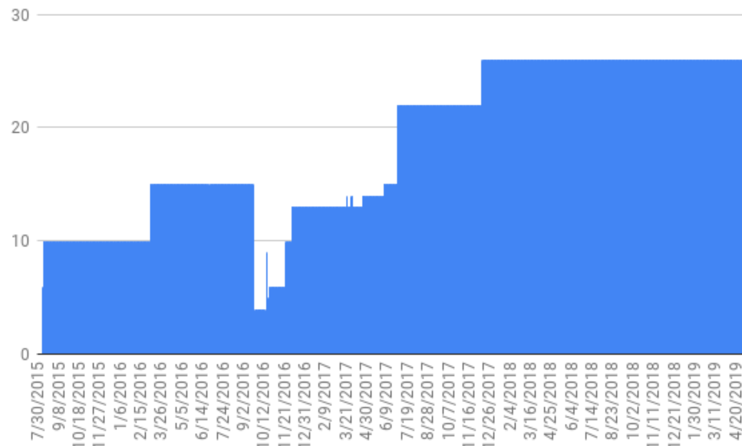



Figure 12: Maximum number of certificates that can be included into the certificate log through the DAO-CA, given the historic gas-limit.

With only two members, the DAO has no advantages over the established CA system, and would need at least three participating members to give any real advantage over the current system. Only one person would have to attest to a certificate, to include it in the CT log, with two members in the DAO.

With a minimum price of 301,408 gas per certificate request, it is currently possible to include 26 certificates per block using the DAO-CA, as can be seen in figure 12. This corresponds to approximately 1.9 certificates per second. This is a maximum of 164,160 certificates per day, if no other transactions takes place on the Ethereum network.

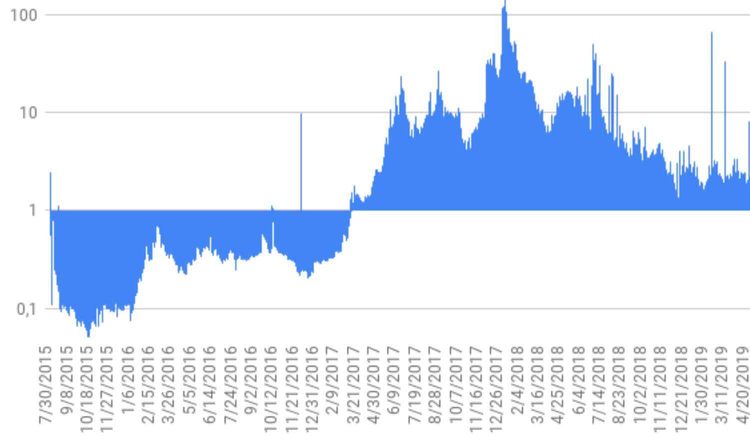



Figure 13: Cost of including a certificate in the DAO-CA in $ cents, based on the average gas price and Ether price of the day.

## 5.2 Dapp-CA Cost

In this section, the cost of running the Dapp-CA protocol on the Ethereum blockchain will be investigated. All gas cost are calculated by the Ethereum Virtual Machine.

- ControllerFactory contract deployment gas cost: 5,219,733
  The gas cost is high due to the necessity of deploying the entire contract code of the Controller, and all it's dependent smart contracts.

- Controller contract deployment gas cost: 5,731,708
  Deploying the controller is expensive as well, as it deploys the contract code of it dependencies, but also instantiates them, populating the blockchain with state beyond just the code.

- Certificate request gas cost: 198,647
  Requesting a certificate is cheaper than deploying a smart contract, but more expensive than a standard token transaction. This results from the state needing to be saved on the blockchain. The transaction cost isn't fixed either, increasing with the length of the URL and certificate needing to be saved.

- Voting gas cost: 112,301
  Voting is only slightly more expensive than doing an ERC-20 token transfer, and approximately 5 times the cost of an Ether transaction. It also nearly twice as expensive as voting in the DAO-CA.

- finalizeVote resulting in the inclusion of a certificate gas cost: 130,407
  The high cost of finalizing a vote is problematic, as the finalization of the vote is necessary for the completion of the process, by the upfront cost isn't shared evenly among the beneficiaries. Every staker in the system has an incentive to wait for another staker to do the finalization.

- finalizeVote resulting in fork gas cost: 7,731,045
  Finally, the cost of forking the contract is very expensive. This was to be expected, as two copies of all the major contracts involved have to be created. Luckily, the transaction comes in just under the current gas limit of Ethereum blocks. This is a very expensive transaction, likely to cost 10's if not 100's of Dollars in Ether. However, the protocol is specifically designed to avoid forking, only necessitating it if under attack.

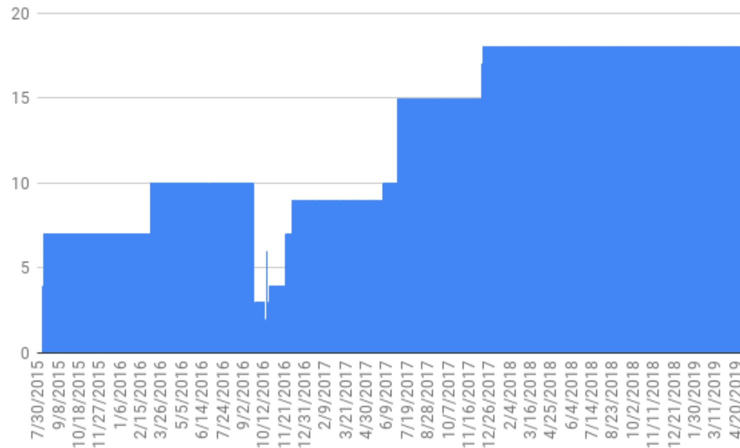



Figure 14: Graph showing the maximum number of certificates the Dapp-CA can include per block, based on avg. block gas limit since the launch of Ethereum.[4]

The absolute minimum price of a certificate is the cost of a request transaction, a vote, and a finalization. This gives us a minimum cost of 441,355 gas in the Dapp-CA. As can be seen in figure 14, the current maximum throughput for certificates on the Ethereum blockchain is 18 certificates per block, or 1.3 certificates per second. This is a maximum of 112,320 certificates per day, a rather paltry number compared to the 1.3 million certificates issued by Let's encrypt in a single day.[14]

Transactions on blockchains are expensive compared to operations in centralized systems. Predicting the price of transactions are further complicated by the dynamic fee market, which works on an auction model, and is priced in an asset which isn't pegged to a relatively stable value like USD. In 15 we see

how the minimum cost of issuing a certificate have swung quite a bit, since the start of Ethereum, reaching a high of approximately 2.50$ USD per certificate, while costing a more reasonable 0.07$ USD at time of writing.

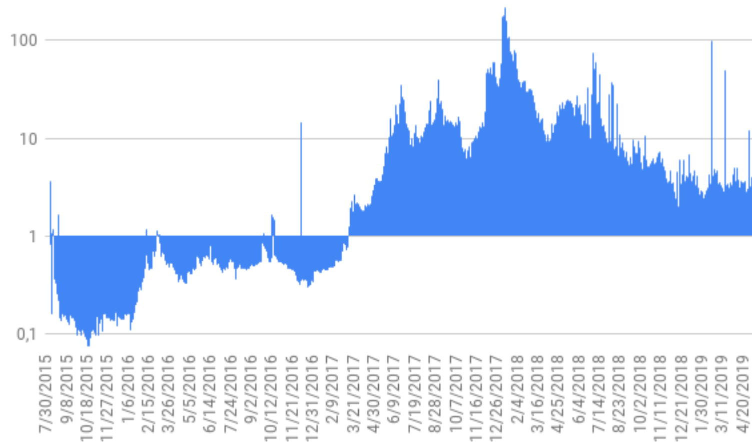



Figure 15: Cost of including a certificate in the Dapp-CA in $ cents, based on the average gas price and average Ether price of the day.

# 6 Discussion

## 6.1 Security

To get a full picture of the security of the DAO-CA and Dapp-CA, it behooves us to look at known attack vectors against each of the underlying systems and protocols, and how a successful attack against one of the subsystems, may influence the other subsystems and the protocol as a whole.

**Assumptions:**

- Attackers will act with their own economic interest in mind, unless the opposite is explicitly stated.
- The underlying blockchain is sound, with no unknown exploitable security holes. The blockchain is still assumed to be vulnerable to known exploits, such as selfish mining and 51% attacks.
- The operators of necessary 3rd party infrastructure, such as DNS service providers, will not act maliciously.

When it comes to attacking the DAO-CA and Dapp-CA, there are different degrees of severity of an outcome. There are the following possible outcomes of attacks:

**Issuance of false certificate.**
The worst outcome of an attack is the issuance of a false certificate. Not only does it mean that there have been a malicious capture of the protocol, but it also further endanger ordinary users, by making man-in-the-middle attacks possible.

**Censoring of valid certificate.**
While not as critical as the issuance of a false certificate, it remains important that the issuance of valid certificates cannot be censored, when honest requesters participate in the process. Indiscriminate censorship risks degradation of the economic function of the protocols, as the economic utility of requesting a certificate disappears when you're denied ownership, while targeted censorship risks failing the people who need decentralized technology the most.

**Punishment of honest attesters.**
Punishment of honest participation should be so unlikely, that the overall estimated value of honest participation is still positive. Any attacks that put honest attesters at risk of losing money, may degrade the economic incentives of the protocols. Any attack that can reliably and cheaply steal or reduce value from honest attesters, will lead to non-participation.

**Degradation of attesting process.**
Not all attacks necessarily lead to protocol failure, but may just degrade the quality of service. Degradation in the context of the DAO-CA and Dapp-CA can

mean delay or cancellation of the certificate issuance. While not catastrophic for the protocol, repeated degradation of the process may result in a slow death, as the utility of using the protocol lessens.

### 6.1.1 Blockchain censorship attack vectors

**Block reordering long-range attacks**

Block reordering attacks are very problematic in the context of both the DAO-CA and Dapp-CA, as they may remove or delete already validated and used certificates. In blockchains that don't have transaction finality, this risk is ever present, but the more blocks that are built on-top of the chain, the more expensive an attack will be. A one hour reordering of the Ethereum blockchain would cost 143,000$[3] at the time of writing, assuming that enough hashing power could be purchased through services like `https://www.nicehash.com/`. Nicehash only have capacity to offer 6% of the needed computational power to attack Ethereum though, so other services would need to supply the remaining 94%, making the attack more expensive if the remaining hashing power could not be sourced from similar sources, but instead would require the attacker to run their own mining hardware.

It may make sense to use a blockchain with transaction finality for the protocols, to ensure that certificates of a certain age can't be reordered. Transaction finality is achieved by checkpointing the blockchain at certain intervals, and while no major blockchain currently implements transaction finality, Ethereum is expected to adopt that feature, with the change to a Proof of Stake consensus protocol in the future.

The likelihood of a block reordering attack against the DAO-CA protocol is minuscule, since it is unlikely to change the outcome of a vote. Even if someone were to successfully rollback the blockchain enough blocks to interfere with a vote, the attack would only be able to extend a vote, merely requiring the voting DAO members to place their votes again after the reordering. Since there is no time-limit to the requesting process of the DAO-CA, it is impossible to force a change in the outcome of a vote, without changing the way DAO members vote.

Block reordering attacks are much more problematic against the Dapp-CA, as they effectively allow the attacker to decide the outcome of a dispute round. This can be guaranteed by mining a long enough hidden chain, to essentially fill the entire voting period with the attackers own votes, denying any other voting transactions to become a part of the block. An attack like this would be exorbitantly expensive, but smaller reordering attacks may tip the balance of more even votes. It is unlikely that an attack like this would take place, as it is probably cheaper to simply buy all available blockspace in the same interval of the attack, rather than mining your own blocks. We will explore an attack like that in the next section.

**Block-stuffing censorship Attacks**

The Ethereum blockchain, and any blockchain with a transaction-fee market, is susceptible to someone pushing up the price of transactions to the point, where

some transactions become economically unfeasible, or if taken to the extreme, miners will include no other transactions due to the value of the malicious transactions.

The Fomo3D lottery-ponzi-scheme hybrid was a project, where the last person to buy a ticket would win the lottery. It was famously drained for 10,469.66 Ether, by an attacker which outbid every transaction going to the contract for 5 minutes, allowing the attacker to win the main prize[39]. A similar attack could be employed against the Dapp-CA, letting a certificate request be submitted for inclusion in the CT log, before spamming the transaction pool with high-fee and high-gas transactions, until the certificate request voting period has passed. Every voting transaction in every Ethereum block within the voting period would have to be outbid.

| Address | Transaction | Gas Cost | Gas Price |
|---|---|---|---|
| 0xaf0206A5632ebd5df2dA0Cd2206c5402A1Cc6662 | honest DAO vote | 128000 | 40 Gwei |
| 0x1234bB130F80859d326C3CeF58C5aE5E7dFC9644 | send Ether | 21000 | 30 Gwei |
| 0xbad15716E6B55797b4DFdF74D54F18E94ce054a9 | transfer GOV | 86000 | 25 Gwei |

(a) Example of how a normal block might look.

| Address | Transaction | Gas Cost | Gas Price |
|---|---|---|---|
| 0xbadacc7126D45feFc764E96E5DD14eFAB21438fD | malicious DAO vote | 128000 | 42 Gwei |
| 0xbadacc7126D45feFc764E96E5DD14eFAB21438fD | recursive gas burn | 7900000 | 41 Gwei |
| ~~0xaf0206A5632ebd5df2dA0Cd2206c5402A1Cc6662~~ | ~~honest DAO vote~~ | ~~128000~~ | ~~40 Gwei~~ |

(b) Example of a block undergoing a blockstuffing attack. The malicious transactions are highlighted with red. The third transaction cannot be run, as all the gas of the block has been used.

As long as the voting period is somewhat long, this would be a very expensive attack, as it'd require the attacker to keep transaction fee levels over the point where it wouldn't be financially gainful for an honest attester to dispute the vote. The malicious attester would have to push the transaction gas price to where the cost of voting is greater than the economic utility of what would be gained from taking the attackers stake, plus the amount of honestly staked GOV tokens and the bounty from the inclusion of the certificate.

We wish to define a function $g(x)$ for calculating the minimum price of performing a single block stuffing attack, in block $x$.

We can calculate the maximum economically viable gas cost as $MGC$

$$MGC = \frac{S_a + S_h + b}{112,301} \tag{2}$$

Where $S_a$ is the stake of the attacker, $S_h$ is the stake of the honest participants, and $b$ is the bounty rewarded for certificate inclusion. The constant 112,301 is the gas needed for calling the vote method in the Dapp-CA.

We then define a function $f(x)$ as the gas used by transactions, with a gas cost higher than the $MGC$ in block $x$. The attacker do not have to censor these

transactions, as they help keep out cheaper transactions from the block. The function $GAS_LIMIT(x)$ returns the total units of gas available in block $x$.

$$g(x) = MGC * (GAS_LIMIT(x) - f(x)) \tag{3}$$

$g(x)$ ether will have to be paid for every block, for it to be economically unviable for the honest stakers to vote. We can define the cost of the entire attack as the following:

$$\sum_{n=0}^{n=m} g(x_n) \tag{4}$$

Where $n$ is the current block number, 0 is the first block of the attack, and $m$ is the block number at which the certificate can be finalized. This will make it economically unviable for honest stakers to contest a malicious certificate being included in the CT log.

This attack could only ever prolong the voting period in context of the DAO-CA, but could be used to issue a malicious certificate in the Dapp-CA.

An interesting question is whether a block reordering or block stuffing attack is the cheapest. Only looking at the capital expenditure, a block stuffing attack is likely to be the cheapest. Looking at the past three months of transaction fees paid on the Ethereum network, the average daily fee expenditure is between 250 and 500 Ether, or around 125,000$ at time of writing. Were someone to initiate a block stuffing attack, it is likely that prices would be inflated heavily, as competition for block space intensifies. As such the true cost of a block stuffing attack is hard to estimate, as it's limited by how much money other participants in the decentralized applications are willing to spend to get their transactions through.

The attacker doesn't necessarily have to fill out the entire block, even though that would be the only way to make sure no transactions go through to the Dapp-CA, if they just keep transactions above the threshold where it wouldn't be economically viable for the other participants in the Dapp-CA to band together.

**Miner Censorship Attacks**

Any blockchain is beholden to it's miners. Usually, miners will opt to organize in mining pools, which are coordinated mining groups. Each individual miner is paid proportionally to how much hashing power they provide the pool, every time the pool is the first to discover a new block. Miners partake in these pools, due to the increased predictability in payouts. However, this creates the problem of miner centralization, where the coordinators of these pools, the people or software which decides the composition of the next block to mine, have sizable power over the blockchain. If enough mining pools collude to censor certain addresses or smart contracts, or even refuse to mine on top of blocks that include these addresses, then the address is effectively censored from partaking in the blockchain. For an attack like this to be successful, the colluding miners would

need more than 33%[32] of the hashing power, to have a chance of succeeding, and above 50% of the hashing power to succeed in the long run.

A mining pool doing selfish mining[40], controlling 33% or more of the hashing power, could effectively censor certificates from being included in the DAO-CA, by not including transactions that would vote to include the specific certificate. The censorship could also be used to force any certificate through the Dapp-CA, as the attacker would merely have to censor the attesters staking against inclusion.

**Impact of censorship attacks**

For the DAO-CA, censorship will only result in a halting of function, until the censorship attack seizes. The only way to impact the DAO-CA forever, is to keep up the censorship attack in perpetuity. It can never be used to punish honest partakers in the DAO, or to issue fraudulent certificates.

For the Dapp-CA, the situation is different. If honest voters can be censored from partaking during the voting period of a certificate, it will require a very small amount of GOV tokens to push a fraudulent certificate through the voting process. This is however still unlikely to happen for most certificates, as all of the mentioned attacks require a heavy capital investment. Miner censorship attacks require more than 33% of miners to collude and selfishly mine. Blockchain reordering has the same requirements. While a block stuffing attack might be the cheapest, it is still likely to cost an attacker thousands of Ether to keep up, over a voting period of 3 days. As such, the entire category off censorship attacks fall within the category of critical impact, but very low likelihood.

### 6.1.2 Certificate Transparency Attack Vectors

**Man-in-the-middle Attacks**

If a browser runs a light node, it should be incredibly hard to conduct a man in the middle attack. Currently there are no light clients able to run in a browser, but they're being worked on[34]. The alternative is to use a service like Infura, which relays transactions and data, to and from the Ethereum blockchain. However, relying on a third-party middleman is inherently vulnerable, as any successful attack against Infura, would allow the attacker to successfully insert himself between the user and the blockchain.

An attacker who successfully manages to insert themselves between the certificate transparency log, and the certificate checker, can effectively impersonate any website he chooses.

### 6.1.3 DAO Attack vectors

**Bribery Attacks**

Any public voting infrastructure on the blockchain is vulnerable to bribery attacks, as there exist no plausible deniability and the briber can get cryptographic proof of whether a participant voted the way he bribed him to. Even if votes can't be traded on the blockchain, it's still possible to write code, either on-

chain smart contracts or off-chain using trusted hardware, which ensure that a bribery attack will only be executed, if it's going to be successful[28]. The best way to mitigate bribery attacks, is to build plausible deniability into the decentralized application. If the briber can't prove the DAO member voted the way he wanted, the bribed DAO member is free to take the bribe, and vote honestly any way. Ideally, it should only be possible to verify whether or not someone voted, and not what they voted for. While some blockchains like Monero and ZCash implement advanced anonymity features, this is not currently the case for Ethereum.

Due to the lack of anonymity in our DAO voting implementation, there is nothing hindering attackers from bribing members. Bribing a single member to vote for inclusion of fraudulent certificate is unlikely to be effective, and would likely result in the removal of the bribed DAO member. The real issue is, if the bribery attacks are used to either vote in new malicious DAO members, or kick honest members. Any member letting themselves be bribed into this behaviour, could be considered to already be a malicious member, and if behaviour like this persist, is likely to lead to a malicious capture of the DAO.

**Capture**

There's no need for any kind of bribery, if a party or cartel simply controls 50% of the voting power in the DAO, since they will be able to control the outcome of all votes. When building a system, where participants are only identified by a unique public key, it's impossible to know how many of these keys are controlled by the same person or group. This in turn makes it difficult to ensure, that a single party isn't slowly amassing control of the votes. Block explorers like Chainalysis have tools and products for de-anonymizing addresses[2], but these tools only work on a subset of address, which at some point linked their transaction history to identifiable information. Another solution is to integrate a decentralized identity protocol like uPort[20] to verify the identity of DAO members, however none of these solutions protect against voting cartels, made up of multiple parties.

Nothing is done on a protocol level, to combat capture of the DAO-CA. Instead, every new member of the DAO should be carefully vetted before inclusion.

Functionality that let any DAO member create a fork of the DAO-CA and CertificateTransparency smart contract could be useful. This gives an honest minority the option to fork away from the DAO and create their own continuation of the CertificateTransparency log. It wouldn't prevent capture of the initial DAO, but would let honest attesters continue off-of the CertificateTransparency they've already build, while leaving the malicious fork of the DAO to hopefully be forgotten.

### 6.1.4 Dapp-CA attack vectors

The Dapp-CA is vulnerable to the same kind of 51% attacks that the DAO is. Infact, an attacker only needs enough GOV tokens equal to the FORKING_THRESHOLD to force the Dapp-CA into a fork. The FORKING_THRESHOLD is a constant set in the smart contract, denominating the percentage of GOV tokens that needs to be staked on an unsuccessful staking outcome, to force a fork. However, where the DAO is potentially compromised forever, when an attacker gains control of 50% of the seats, the attacker will lose a significant amount of their GOV holdings every time they successfully attack the Dapp-CA. He will either lose his GOV tokens by forcing the Dapp-CA into a fork, or attesting to the losing outcome of a vote. This makes repeated attacks against the Dapp-CA impossible, unless the attacker owns more than

$$total_token_supply * (1 - FORK_THRESHOLD) \tag{5}$$

of the total token supply, as this would be enough GOV tokens to make sure the opposing side never gain enough voting power to fork the Dapp-CA.

**Bait attacks**

The most devious attack against the Dapp-CA design, is an attack type we've dubbed "Bait attacks". The goal of a bait attack is to bait a minority of the honest attesters to vote for one outcome, and then convince the remaining majority to vote for the other outcome. The attack requires a malicious certificate requester, who also participates in the attestation process. Since the certificate requester is in control of the single source of truth, the DNS text field, they can change the contents of the text to make their request appear either valid or invalid at will. Unfortunately, the attacker can almost always guarantee the success of a bait attack.

An example of a bait attack, depicted in figure 16, would be to request a valid certificate, but leave the DNS text unchanged. The attacker would then wait for some stakers to reject the certificate, putting a stake of GOV tokens on the rejection outcome. Once the honest stakers have put down their stake, the attacker can change the DNS text to a valid proof of ownership, and have future honest stakers, stake to accept the certificate, while himself also staking for an acceptance outcome. This would slash the GOV tokens of the honest stakers, who staked to reject, giving the attacker a portion of their slashed tokens, proportional to the amount he staked on acceptance.

For that reason, if attesters continue to check in on the DNS text field, and see differing results, they're encouraged to not participate in the attestation of the certificate. Even if they see a proof of ownership, they're not guaranteed the certificate requester won't change it later. This proof of ownership may only be available for a few seconds, leaving any attester who approves the certificate immediately, to vote against a majority of rejecters, and thus vulnerable to having his honest stake slashed.

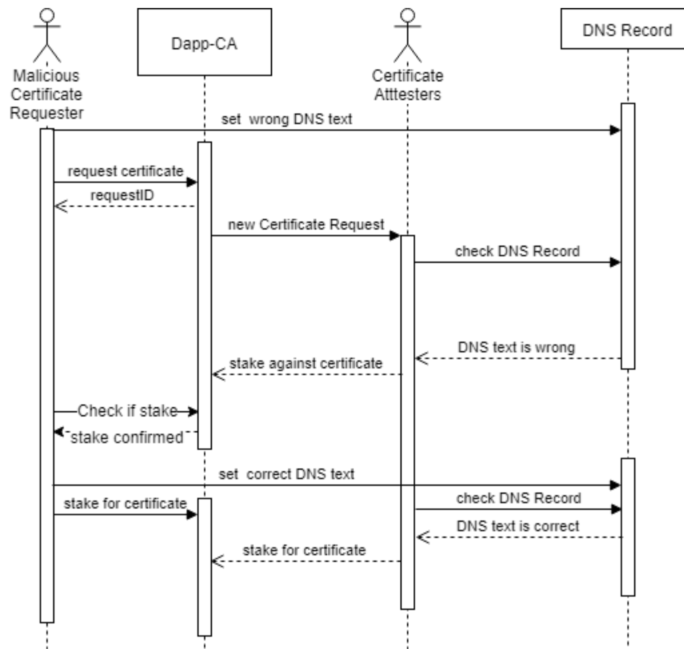



Figure 16: Sequence diagram of a bait attack. Note that the certificate attesters are a broad group, with no clear way of communicating. After the initial honest stake against the certificate, the attacker changes the DNS record to be correct, baiting the initial staker to make a false stake.

**Fork bait attacks**

Bait attacks can also be performed in conjunction with a fork attack. The attacker will first force the Dapp-CA into a forking situation by presenting an invalid DNS text when requesting a certificate, while voting for said certificate to be accepted. Once the Dapp-CA is in forking mode, the malicious DNS owner and certificate requester, will change the DNS text to a valid proof of ownership, in an attempt to bait GOV token holders to migrate their tokens to his maliciously controlled fork. This attack would effectively slash the stakes of honest token holders, and can only be combated by playing politics on the social layer, and presenting additional proof of the malicious activity to the GOV token holders. Unless the fork was forced through on a clearly malicious certificate, getting people to migrate to the right fork is likely to be a political process. These are both messy solutions, and are not guaranteed to work, not even among perfectly rational actors.

The only way to mitigate bait attacks, would be if DNS records supplied a timestamp, incremental nonce or history, showing stakers that a change had taken place since the initial certificate request. With this hypothetical DNS record, it would be possible to see that a change had taken place, immediately

invalidating the certificate request.

### 6.1.5 Social layer

With self-governing systems like the DAO-CA and Dapp-CA, a certain amount of coordination will happen outside of the software. We refer to this coordination, outside of the smart contract code, as the social layer.

**DAO**

The goal of the DAO is to have a self regulated environment in which a majority of the DAO members act honestly. It is important that the participants of the DAO has it's long-term survival as a priority. This is partly achieved by rewarding participants a bounty for voting on certificate requests, but ideally participants would also have social reputation on the line. A good starting point would be to have organisations or individuals with some social capital at stake to participate in the DAO. These could be comprised of organizations involved in software activism like EFF, FSF or Mozilla, institutions working for the public good like universities, or just individuals known for their unshakeable principles like Edward Snowden or Richard Stallman.

The key would be to distribute these organizations and individuals across political spectrums and jurisdictions, as to avoid the chance of ideologically fueled censorship or state coercion. An added benefit of the public nature of the DAO and blockchains, would be that any fraudulent behaviour exhibited by said individuals and organizations, would be immediately transparent and harm their social status.

In the case where the majority conspire together and approve a request for a clearly malicious certificate, the DAO has been compromised beyond repair, and it should be deemed corrupt and untrustworthy. All the certificates in the CT log should be questioned and the requesters should get new certificates right away from a trusted CA. The guilty parties involved in the fake certificate can be recognised and publicly shamed, and any fraudulent certificate could be traced back to the organizations who voted for it's inclusion. This would make it easy to build a new DAO, without the members who voted for the malicious certificate.

**Dapp-CA**

The social layer has a smaller influence over the Dapp-CA than the DAO-CA, as it is open for everyone to participate in it's operation.

The reliance on governance by forking however, will almost surely bring the social layer into play, as GOV token holders will have to make the choice between which fork to migrate their tokens to. If there is no clearly malicious fork, for an example as a result of a forking bait attack, it will be up to the community of GOV holders to figure out which fork will be the right one to migrate to.

## 6.2 Economical Incentives

Mentioned in section 2.2.4 the Dapp-CA should contain economical incentives to enforce good behaviour. While the incentives can be expected to reward good behaviour under normal conditions, this is not the case during bait attacks.

**Dapp-CA**

The idea for the Dapp-CA came from trying to improve the shortcomings of the DAO-CA. Namely the Dapp-CA attempts to be robust in face of attack. Its incentive structure is build to punish misbehaving attesters, and attesters that are wrong in their assessment of the validity of a requested certificate. The goals for the economic incentives defined for the Dapp-CA from section 2.2.4 were:

- Reward honest attesters.
- Slash malicious or wrong attesters.
- Fork if no consensus can be reached.

The process of forking and the staking mechanism have been described in great details in section 4.4.4 and section 4.4.2 respectively. To reward participation, every successful inclusion or rejection of a certificate, is paid a bounty. The winners of a bet, are also rewarded with the stake of the opposing site. The idea is that the Schelling point for the staking, will always be the honest result of the DNS challenge. This would ideally reward honest participation, but unfortunately, bait attacks can result in honest attesters getting slashed, as a malicious certificate requester can change the outcome of the DNS challenge at will.

In the same vein, as long as there's no malicious certificate requester, bad behaviour should be punished by the Dapp-CA, and thus discourage malicious or wrong attesting.

Stakes are locked until the certificate expire. The idea is, that malicious GOV token holders will be incentivized to not ruin the value of the GOV tokens they have locked into the staking system, as they will not be able to be withdrawn until after a period of time. This helps protect against short-term bad behaviour, by limiting the ability to dump maliciously obtained GOV tokens immediately, but it doesn't protect against attacks where someone might have taken a short position on public markets, or where the the attacks don't result in a degradation of GOV token value.

## 6.3 Scaling

When scaling a blockchain based decentralized application, there are a few different options.

The first decision is which blockchain to choose. If the blockchain is like Bitcoins blockchain, a block is only created roughly every 10 minute[36]. This

means that there will only be allowed 2,100 transactions every 10 minutes or about 3.5 transactions per second, and this will obviously limit the amount of certificate requests possible over a given time period.

The Ethereum blockchain, which has a blocktime of 14 seconds and about 380 transactions per block. This comes out to 25 transactions per second, or 15,000 transactiosn every 10 minutes. This is an improvement on the maximum transactions per second of around 700%. Of course, not every transaction is created equal. Moving around Bitcoin or Ethereum is fairly limited in it's data use, while including information about certificates, can quickly balloon the transaction size allowing for fewer transaction in a block.

The scaling issue have been widely debated in the past few years. Ethereum offers the options for scaling such as sidechains and more specifically Plasma to handle the scaling issues that arrive with the popularity of a given blockchain. Other digital ledgers promise a much larger transaction bandwidth, however centralization and mutability is the price to pay for those improvements. If we're willing to make those kind of tradeoffs, it may be more interesting to look at sidechains, where the validator set is decided and secured, by logic on the mainchain.

### 6.3.1 Plasma, Sidechains and off-chain scaling

Plasma is a framework proposed by Vitalik Butterin and Joseph Poon. Plasma is a proof-of-stake sidechain design, build to interact with Ethereum. The idea behind sidechains, at least in the sense of Plasma, is that it is itself a complete blockchain. Every computation necessary to run the blockchain app will be made on the sidechain, and then propagated up into the main chain when necessary. The main chain will only be used to settle disputes regarding the state of the sidechain. If a fraudulent blockheader is submitted to the main chain, and a fraud claim against that block is made, and proven correct, the creator of that block will be punished by having their staked Eth slashed. The Plasma design can be implemented in a way allowing token transfers as well, allowing tokens necessary for the plasma chain, to interact with the broader Ethereum ecosystem.[23].

**Plasma for the DAO-CA:**
Implementing Plasma into the DAO-CA could be done as follows:
First the decentralized app would be deployed to the Ethereum blockchain and a Plasma sidechain would be created as part of the smart contracts. This Plasma smart contract would contain information about how the DAO-CA sidechain would be run, such as when and how state hashes should be confirmed by the root chain. A choice of consensus algorithm would have to be made. A proof of authority (PoA) protocol might be good as the consensus algorithm for the DAO-CA, because the majority of the DAO members are needed to be trustworthy, which is also the requirement for PoA to remain secure[41]. PoA protocols offers a quicker blocktime than proof of work, and less computational overhead, making the burden of running the sidechain as small as possible.

All the interaction with the DAO-CA would take place on the sidechain, with certificate requests being submitted to the DAO members who serve as validators of the sidechain. The two main cost factors when running a public blockchain, is computation and storage cost. By employing a PoA sidechain, it is possible to eliminate the computation cost associated with Proof-of-Work, and sharply reduce the storage, as the data duplication factor is reduced to the number of DAO members. The only logic necessary to run on Ethereum's main chain, is to handle membership of the DAO, and potential fraud claims. Whenever a fraudulent block is created, a plasma fraud claim can be made and if the claimer can prove the fraud, then the creator of the block will have his ether slashed.

**Plasma for Dapp-CA:**
Plasma chains could also be used for the Dapp-CA, in this case it would be necessary to use Proof of Stake (PoS) consensus, instead of PoA. PoS would allow the GOV token to be used for staking, allowing anyone with a financial stake in the Dapp-CA to become a validator of the sidechain. If fraudulent behaviour is detected, their stake of GOV tokens would be slashed.

## 6.4 Comparison of the established solution, DAO-CA and Dapp-CA

In this subsection, we will explore the benefits and drawbacks of our blockchain based solutions, both compared to each other, and to the established solution.

### 6.4.1 Established solution

The established solution is the reliance on trusted CAs. This solution has worked for many years, and by and large is a safe-option. The internet haven't had a decentralized way to agree on the state of a registry until the advent of blockchain technology[30]. CAs helped move the Internet from using HTTP to HTTPS, with CA issued SSL certificates. Projects such as Let's Encrypt, let's you obtain a certificate easily, and is completely free. Furthermore the centralization of CAs also makes the system incredibly scalable. There is no crisis of certificate insecurity, which must quickly be solved. However there do exist some downsides that should be mitigated.

CAs are extremely centralized with all the downsides that comes with it. One of these downsides are the single point of failure in fundamental Internet security, every trusted CA represent. Even if just one CA is hacked, the attacker will be able to sign a valid certificate for any site, such as Facebook, Google or an online bank. The centralization also makes them vulnerable to disgruntled employees, who may misuse their power to create fake, but valid, certificates.

Furthermore, there is currently no oversight, not by any governments or organisations. This can result in certificates being generated for websites that

haven't requested them, and there is no way for these websites to know that these certificates have been created, as mentioned in section 1.2.

There have been RFCs from the IETF[22] proposing to enforce a Certificate Transparency log, which would impose a somewhat decentralized oversight on the CAs. In this case only certificates in the log would be considered valid, even if the certificate not in the log, had been signed by a trusted CA. Our implementations natively work with CT logs, atomically including new certificates in CT logs upon acceptance.

### 6.4.2 Proposed Solutions

**DAO-CA**

The first idea was to create the DAO-CA. It made sense to create a decentralized organisation, where multiple participants could attest to the validity of a certificate. The DAO solution comes with some advantages and disadvantages:

The DAO-CA runs on the Ethereum blockchain, which gives us cryptographical proof, that an attester has approved a certificate. The option to add and demand multiple attesters of a certificate can be written into the smart contract, and pushing a signed certificate to the certificate transparency log is an atomic step in the signing process. The CT log will, by the security of the blockchain, be append only and close to immutable. Another bonus of using the blockchain, is the transparency it brings to the process of certificate issuance. Since every transaction is saved in a block, it becomes easy to spot bad behaviour, both in the certificate transparency log and DAO. If we were to compare it with the Dapp-CA it is not vulnerable to bait-attacks, and as long as a majority of the DAO members are honest, certificates will be accepted or rejected according to the rules. It is also likely to use less gas, having a smaller minimum gas cost needed for approval.

There are also some drawbacks with the DAO-CA. It's only somewhat decentralized, still relying on gate keeping to keep the system functioning. The DAO is vulnerable to capture, having no effective way to escape the clutches of anyone who controls a majority of the membership. Anyone controlling a majority of votes would be capable of deciding the inclusion or exclusion of all requested certificates, rendering the DAO pointless if that should happen.

Furthermore there is no options for requesters to obtain free certificates, unless the DAO decided to run as a charity. The established system allows for centralized CAs such as Let's Encrypt to provide domains with free certificates, and while the security of these certificates are questionable, all websites do require a certificate to be trustworthy. Most browsers will not display the web page properly, unless specified by the user to trust the website, if it doesn't use a certificate. It might not make sense, if you run a hobby website, to pay the bounty to receive a certificate, and for this reason the solution would alienate potential users, especially the less resourceful.

Scalability is a massive issue compared to traditional CAs. Where Let's Encrypt can issue 1.3 million certificates a day, the DAO-CA can only issue a few

hundred thousand, at a much steeper cost than it's centralized counterparts.

**Dapp-CA**

We believed that there was a way to design a completely open system, which could avoid capture of the voting process, while incentivizing honest validation through economic incentives. The resulting system is the Dapp-CAS.

The Dapp-CA runs on Ethereums blockchain and will, after a certificate have been accepted, put that certificate into the certificate transparency log. The main advantage of Dapp-CA over the DAO-CA is the ability to fork when voting power is captured. This way the Dapp-CA is able to fork away malicious participants that won't accept a valid certificate or will try to include a certificate that the domain owner haven't requested. The decentralize application will fork into two new decentralized applications, one for each of the disagreeing parties. One will be with the malicious users, and anyone who believes they voted correctly, and one with the well behaved users, that now have a smart contract free of the malicious stakers.

To participate as an attester or request a certificate, you will need the token associated with the Dapp-CA, the 'GOV' token. GOV tokens are tradable, and allow anyone to buy a large enough stake to fork the protocol if they're are wealthy enough. This isn't a big problem, as it's an attack that can only be performed once, but if it's coupled with a bait-attack, it can be used to fork off honest users from the protocol as described in section 6.1.4 description of 'Fork bait attacks'.

Same as with the DAO-CA, there are no options to receive a free certificate, due to the expensive nature of transactions on the Ethereum network.

**Comparison**

In this paper we have proposed two different solutions to mitigate the problems with centralized CAs.

The Dapp-CA and DAO-CA decentralize the process and makes sure that there are no single points of failure, beyond the DNS. When centralized CAs are hacked, they are not incentivized to report and fix it. Employees of CAs may misuse their power, and the abuse can be hard to track. These issues with centralized CAs do not exists in either the Dapp-CA or DAO-CA. The protocols they implement enforce inclusion of accepted certificates directly into the certificate transparency log, making the system fully transparent. Furthermore they both run on Ethereums blockchain, making every transaction created by the protocol participants public record. It is impossible for participants of the Dapp-CA or DAO-CA to hide the inclusion of fraudulent certificates.

However, it is hard to argue that the DAO-CA or Dapp-CA are better than the current way of doing things. Smaller websites or non-profit sites might abandon using certificates and HTTPS all together, if a free or lower price option isn't available. This would have a negatively impact the security of the

Internet in general. The scaling of the DAO-CA and Dapp-CA is also abyssmal, being a magnitude lower of that of Let's Encrypt, which is just one of many centralized CAs. A DAO consisting of free internet activists and organizations might still be an interesting thing, as the scalability would be less of an issue, if the DAO-CA focused primarily on issuing certificates to websites, which have powerful enemies, such as Wikileaks.

The Dapp-CA, while an interesting experiment, falls apart under scrutiny. The complete control malicious actors have over their DNS record, keep it from being a viable contender compared to centralized CAs or the DAO-CA. Unless DNS records are updated to keep an immutable count of changes, it will not be possible to built a decentralized oracle service for certificates in the style of the Dapp-CA.

## 6.5 What could have been done better

We set out to explore whether or not it would be possible to bridge the gap between the established solution of centralized CAs, and the all-or-nothing world of self-signed certificates. The current solution of CAs centralizes power, risk systemic failure, because of a few big corporations, while fully decentralized proposals like DPKI, make certificates impossible to be reversed, if the private key is lost or stolen. While we believe we've found some novel approaches, which fit in-between the two extremes, they're not without their own issues.

### 6.5.1 Dapp-CA reliability on DNS text

The Dapp-CA protocol was designed to be a completely open system, where anyone who can afford an economic stake, can participate. The design emphasizes open participation under the assumption that the economic incentives alone are enough to facilitate honest behaviour, and that the open design will result in the biggest amount of participators. This is all in the pursuit of decentralization, and through that, the removal of single-points of failures. However, the protocol still rely on DNS text as a single source of truth, which results in serious problems. One of the issues is that a malicious DNS owners, who can control the DNS text, can switch it between honest and malicious text. It's impossible for the protocol not to be limited by it, due to the information sparseness of the DNS records. A timestamp of when the DNS text was last changed, or even better, a history of changes to the DNS text, would allow the attesters to avoid the bait attacks6.1.4 which are currently possible against the Dapp-CA protocol.

Additionally, the one who control the DNS text server can themselves interact maliciously with the Dapp-CA protocol. As long as the Dapp-CA protocol relies on a centralized party, it can never boast to be fully decentralized.

### 6.5.2 Gas-usage optimization

Every bit of storage, and every compute cycle is expensive on blockchains. This is inevitable since every transaction, and it's effects, have to be computed on every node participating in the network. The Ethereum network gives every operation a fixed gas-cost. Storage operations are usually expensive, and compute operations are a bit cheaper, while a few operation, which remove the need of storage, have a negative gas cost. Every block has a gas-limit, putting a limit on the amount and size of transactions that can occur in a single block. We've done no work to optimize for gas cost. It would definitely improve both the DAO-CA and Dapp-CA to take a look at the efficiency of the code, and pare down the amount of compute and storage needed to complete transactions. It's likely that some computations, such as hashing URLs could simply be performed off-chain, rather than happening as part of the on-chain logic.

## 6.6 Future work

### 6.6.1 Mechanism design improvements

There are many areas of the mechanism design in the Dapp-CA and DAO-CA, which could likely be improved by doing a thorough evaluation of the game theory underpinning the systems. Mechanism design is a subfield of economics, and as such falls slightly outside the purview of Computer Science, as such, it is likely that experts within this field may improve upon our design in significant ways. We will outline a few areas which we think might be particularly interesting to look at.

One should always be careful when designing economic incentives, as an incentive that may look sound on the surface, may incentivize bad behaviour later. A famous example is that of rat-catchers in Hanoi, who were paid one cent per rat-tail they brought to the office of the colonial government. At the beginning, the program seemed successful, with thousands of rat tails being delivered, until curious sightings of tailless rats started to be reported around Hanoi. The rat-catchers were smart enough to realise, that simply amputating the tail of the rat and releasing it to breed again, would increase their future revenue.[9]

**Economic incentives**
A obvious place to start, would be to look at how further economic incentives may be able to increase the chance of honest participation, while disincentivizing malicious participation. Interesting avenues to explore might be incentivizing participants to quickly commit to a fork, so as to avoid a situation where the game theoretic optimum is to wait for a clear winner to emerge, and then only migrate ones tokens when the market has chosen which fork is the correct one.

It may also be worth adding an extra price to requesting a certificate, requiring any certificate request to carry a deposit, which will be paid back to the requester if the certificate is accepted, and the tokens being burned if not. This

is an incentive that we've deemed not to implement, as it's not clear whether or not it would incentivise attesters to skew towards rejecting certificates, so as to reduce the supply of GOV tokens, thereby increasing the value of their own tokens.

Finally, a thorough investigation of the already present economic parameters would be of interest. Analyzing and answering questions like what is the ideal supply of GOV tokens? What should the minimum bounty of a certificate request be? What is the ideal *FORKING_THRESHOLD*?

**Vote mechanisms**

Voting and voting mechanism is also a field studied immensely. Our DAO works by a simple majority in all matters, requiring more than 50% of the votes in matters of certificate inclusion and membership. This is not necessarily the best way to go about it. It may be beneficial to increase the success threshold of the exclusion and inclusion of members to a super majority, to avoid small majorities consolidating power over the DAO.

The only identifier of a member in the current implementation is a public key. This means that there's nothing inherent to the protocol, stopping a single entity from controlling multiple attester addresses. Whether or not a single entity wind up controlling more than a single vote, depends entirely on how the DAO members participate in the elections of new members, and more dastardly, whether or not DAO members can acquire control of the private keys of other membership addresses. One way to bring transparency to the DAO would be to look at implementing a decentralized identity protocol like uPort[**?** ]. Different parameters for identity could be experimented with, ranging from using full legal identities, to have third-party, anonymous proof of personhood or organization. A benefit of using legal identities is that it may be possible to pursue legal action in the case of malicious behaviour, however it goes against the spirit of decentralization and trustless systems. It also opens up participants in the DAO to coercion, adding a new attack vector for malicious actors.

Use of decentralized identity might also be interesting in the case of the Dapp-CA, as it'd require more work of malicious actors to hide collusion between malicious stakers, and malicious certificate requesters. This would especially be useful for convincing GOV token holders to choose the right fork.

### 6.6.2 Security Audit and formalization

While care has been given not to write code that is buggy or insecure, every smart contract written in this project would benefit from a proper security audit, and ideally an attempt at formalization. The reference code function primarily as start-off point for further development and a proof of concept, and as such would need both auditing and verification before being used in production.

## 6.7 Usability

### 6.7.1 UI, UX, Firefox

Comparing the DAO-CA implementation to the Dapp-CA implementation shows some caveats for the Firefox extension. In DAO-CA the user of the extension would only have to concern themselves with one, continuous, CT log that would never fork. When comparing that to the Dapp-CA the concern now spans over, in worst case, N amounts of possible CT logs, where N is the amount of forks that have happened on the Dapp-CA protocol. The purpose of the forks in the Dapp-CA is to fork off anyone with ill intentions or malicious motives. Attacks can happen, and honest users want to join the honest fork. For this reason, it makes sense to have options and recommendation as to how and which fork the end user should switch to. It might make sense to have 'oracles' from which the extension could make a decision of which fork to pick.

Not all users are interested or even capable of understanding the nuances in the forks and by extension, will not be capable of picking the right fork. The oracles might help with this feature and should be a standard. Organisations or private individuals could be oracles. A prime candidate for a oracle could be EFF.

In the end, the user would rely on the market to be the ultimate oracle, as GOV tokens on a malicious fork would likely be worth much less than the fork the market trusts.
In the current Firefox extension there is no way for a user to switch to another fork, this would need to be implemented in the future work for the Dapp-CA.

Ideally the CT log of the either the DAO-CA or Dapp-CA would be an integrated part of the browser experience so the user wouldn't rely on any third party extensions. This way the CT logs for the established CAs and the CT log for these proposed solutions could work in tandem.

# 7 Conclusion

The research question of the project is: "Is it possible to issue TLS-certificates in a decentralized manner, while keeping the ability to resist loss of private keys, and revoking invalid certificates, using the Ethereum blockchain?"

The DAO-CA solution satisfies the research question in that it is decentralized, and private key loss could happen without critical failures. The requester simply need to create a new request and go through approval process again, if they lose access to their private key.
The DAO isn't without problems, as it fails to scale certificate acceptance to the level that's necessary for large-scale practical use. It also has the problem of vote capture, where a voting cartel may end up taking complete control of the membership roster of the DAO, and the certificates they issue.

To remedy the risk of failure by capture, we set out to design a system that would be completely open for certificate attesters to participate. Instead of existing members gatekeeping new participants with a voting mechanism, it would use economic incentives to encourage truthful attestation of certificates. This endeavour resulted in the Dapp-CA decentralized application, described in 4.4. This system sadly fail to accomplish it's goals, due to the possibility of bait attacks.
We identify that a small change to the DNS record, could remedy this problem as described in section 6.5.1. The Dapp-CA also suffer the same problems of scalability, as the DAO-CA. We do not believe that blockchain based solutions to certificate transparency and issuance are fundamentally unscalable, as discussed in section 6.3.
The Certificate Transparency log is what allows the protocols to list accepted certificates without signing them. CT logs being controlled by either the Dapp-CA or DAO-CA, could work in tandem with CT logs containing certificates issued by traditional CAs, offering a smooth integration of the decentralized applications into the established solution.

In conclusion the DAO-CA solution worked as expected, wasn't vulnerable to bait attacks but wasn't robust and would be captured if 50% of the participants were to be corrupt and colluding.
The Dapp-CA works in the sense that it can accomplish the same tasks as the DAO-CA. The problem the Dapp-CA faces are with the DNS protocol which, at its current state, exposes the Dapp-CA to bait attacks. Without any chance of knowing if changed occurred in the DNS record, since the certificate request were made, the Dapp-CA will remain flawed.
The CT-log is essential to the design of the DAO-CA and Dapp-CA, as certificates can only be issued to the CT-log if approved by the decentralized applications. This allows the DAO-CA and Dapp-CA to skip signing the certificate, and instead just attest to it's validity.
We believe that blockchain based certificate issuance may become reality in the

future, if the key problem of scalability can be solved, hopefully heralding a future of truly trust-minimized certificate authorities.

All url in reference have been checked and works on *July 2nd 2019.*

# References

[1] Certificate transperancy org - how certificate transparency works. https://www.certificate-transparency.org/how-ct-works.

[2] Chainalysis website. https://www.chainalysis.com/.

[3] Crypto51 - ethereum (eth). https://www.crypto51.app/coins/ETH.html.

[4] Ethereum (eth) blockchain explorer. https://etherscan.io/.

[5] Fuming google tears symantec a new one over rogue ssl certs. https://www.theregister.co.uk/2015/10/29/google_symantec_dodgy_certs/.

[6] Github - april/certainly-something. https://github.com/april/certainly-something.

[7] Github - erc-20 token standard. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md.

[8] Github - mrostgaard/dao-ca. https://github.com/Mrostgaard/DAO-CA.

[9] The great hanoi rat massacre of 1902 did not go as planned. https://www.atlasobscura.com/articles/hanoi-rat-massacre-1902.

[10] Infura - your access to the ethereum network. https://infura.io/.

[11] Inside 'operation black tulip': Diginotar hack analysed. https://www.theregister.co.uk/2011/09/06/diginotar_audit_damning_fail/.

[12] Let's encrypt - certificate signing request (csr). https://letsencrypt.org/docs/glossary/#def-CSR.

[13] Let's encrypt - domain validation. https://letsencrypt.org/how-it-works/#domain-validation.

[14] Let's encrypt stats - certificates issued per day. https://letsencrypt.org/stats/#daily-issuance.

[15] Metamask - brings ethereum to your browser. https://metamask.io/.

[16] Microsoft trusted root certificate: Program requirements. https://docs.microsoft.com/en-us/previous-versions//cc751157(v=technet.10).

[17] Mozilla root store policy. https://www.mozilla.org/en-US/about/governance/policies/security-group/certs/policy/.

[18] Mycrypto. https://mycrypto.com/.

[19] Remix - ethereum ide. https://remix.ethereum.org/.

[20] uport - open identity system for the decentralized web. https://developer.uport.me/.

[21] Mustafa Al-Bassam. Scpki: A smart contract-based pki and identity system. http://www0.cs.ucl.ac.uk/staff/M.AlBassam/publications/scpki-bcc17.pdf.

[22] E. Kasper Google B. Laurie, A. Langley. Certificate transparency. 2013. https://www.rfc-editor.org/rfc/pdfrfc/rfc6962.txt.pdf.

[23] Joseph Poon Vitalik Buterin. Plasma: Scalable autonomous smart contracts. 2017. https://plasma.io/plasma.pdf.

[24] M. B. Hansen E. K. Y. Poulsen. *Decentralized Certificate Transparency*. 2018.

[25] C. Evans et. al. Public key pinning extension for http. Technical report, 2015. https://tools.ietf.org/html/rfc7469.

[26] Jack Doerner et. al. *Threshold ECDSA from ECDSA Assumptions: The Multiparty Case*.

[27] Jack Peterson et. al. Augur: a decentralized oracle and prediction market platform. 2018. https://www.augur.net/whitepaper.pdf.

[28] Philip Daian et. al. On-chain vote buying and the rise of dark daos. 2018. http://hackingdistributed.com/2018/07/02/on-chain-vote-buying/.

[29] R. Barnes et al. Automatic certificate management environment (acme) - dns challenge. Technical report. https://tools.ietf.org/html/draft-ietf-acme-acme-18?fbclid=IwAR3okE6qMG9cQxpBhXWwVcBANxv7Dd0QUX6VJp1loyncxu2aacH8hUjzdiA#page-64.

[30] Vitalik Buterin et. al. Decentralized public key infrastructure. https://github.com/WebOfTrustInfo/rwot1-sf/blob/master/final-documents/dpki.pdf.

[31] Giaros. Wikipedia - the procedure of obtaining a public key certificate. https://en.wikipedia.org/wiki/Certificate_authority#/media/File:PublicKeyCertificateDiagram_It.svg.

[32] Emin Gün Sirer Ittay Eyal. Bitcoin is broken. 2013. http://hackingdistributed.com/2013/11/04/bitcoin-is-broken/.

[33] S. kent et al. Privacy enhancement for internet electronic mail: Part ii: Certificate-based key management. https://tools.ietf.org/html/rfc1422.

[34] Jason Lee. Metamask labs presents mustekala — the light client that seeds data. Medium, 2018. https://medium.com/metamask/metamask-labs-presents-mustekala-the-light-client-that-seeds-data-full-nodes-vs-light-clients-3bc785307ef5.

[35] Michael Melone. Microsoft technet blog - basics and history of pki. https://blogs.technet.microsoft.com/option_explicit/2012/03/10/basics-and-history-of-pki/.

[36] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf.

[37] Chris Palmer. Google - intent to deprecate and remove: Public key pinning. Technical report. https://groups.google.com/a/chromium.org/d/msg/blink-dev/he9tr7p3rZ8/eNMwKPmUBAAJ.

[38] Alexey Samoshkin. Ssl certificate revocation and how it is broken in practice. https://medium.com/@alexeysamoshkin/how-ssl-certificate-revocation-is-broken-in-practice-af3b63b9cb3.

[39] SECBIT. How the winner got fomo3d prize—a detailed explanation. 2018. https://medium.com/coinmonks/how-the-winner-got-fomo3d-prize-a-detailed-explanation-b30a69b7813f.

[40] Ittay Eyal Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. https://dl.acm.org/citation.cfm?id=3212998.

[41] Parity Tech. Parity - proof-of-authority chains. https://wiki.parity.io/Proof-of-Authority-Chains.

# A Smart Contract Code

## A.1 CertificateTransparency.sol

```
1  pragma solidity ^0.5.1;
2  contract CertificateTransparency{
3      struct Certificate {
4          address owner;
5          bytes32 urlHash;
6          bytes32 certHash;
7      }
8
9      //Owner contract which can operate the inner workings of the CT
           contract
10     address owner;
11
12     //Certificates are indexed by their hashed URLs
13     mapping(bytes32 => Certificate) certificates;
14
```

```
15      //Current number of certificates
16      uint certNum;
17
18      modifier onlyOwner {
19          require(owner == msg.sender);
20          _;
21      }
22
23      constructor(address _owner) public{
24          owner = _owner;
25          certNum = 0;
26      }
27
28      //Might be easier to hash url on-chain, but this is cheaper,
            and arguably more private
29      function ownerSetCertificate(address certOwner, bytes32
            certHash, bytes32 hashedUrl) external onlyOwner returns(
            bytes32){
30          return setCertificate(certOwner, certHash, hashedUrl);
31      }
32
33      function setCertificate(address certOwner, bytes32 certHash,
            bytes32 hashedUrl) internal onlyOwner returns (bytes32) {
34          if(newCertificate(hashedUrl)){
35              certNum++;
36          }
37          certificates[hashedUrl] = Certificate(certOwner, hashedUrl,
                 certHash);
38          return hashedUrl;
39      }
40
41      function newCertificate(bytes32 hashedUrl) public view returns
            (bool) {
42          if(certificates[hashedUrl].owner == address(0x0)){
43              return true;
44          } else {
45              return false;
46          }
47      }
48
49      function add(string memory _url, string memory _certificate,
            address certOwner) public onlyOwner returns(bytes32){
50          bytes32 hashedInput = keccak256(abi.encode(_toLower(_url)))
                ;
51          bytes32 certHash = keccak256(abi.encode(_certificate));
52          address oldOwner = certificates[hashedInput].owner;
53          if(oldOwner != address(0)){
54              require(oldOwner == certOwner);
55          }
56          return setCertificate(owner, certHash, hashedInput);
57      }
58
59      function check(bytes32 hashedUrl, bytes32 hashedCert)public
            view returns (bool) {
60          require(certificates[hashedUrl].certHash != 0);
61          require(hashedCert != 0);
62          return certificates[hashedUrl].certHash == hashedCert;
```

```
63      }
64
65      function transferCertificateOwnership(address _newOwner,
            bytes32 hashedUrl) external {
66          require(certificates[hashedUrl].owner == msg.sender);
67          certificates[hashedUrl].owner = _newOwner;
68      }
69
70      function transferOwnership(address _newOwner) public onlyOwner
            {
71          owner = _newOwner;
72      }
73
74      // Changes a string from uppercase to lowercase.
75      // https://gist.github.com/thomasmaclean/276
            cb6e824e48b7ca4372b194ec05b97
76      function _toLower(string memory str) private pure returns (
            string memory)  {
77          bytes memory bStr = bytes(str);
78          bytes memory bLower = new bytes(bStr.length);
79          for (uint i = 0; i < bStr.length; i++) {
80              // Uppercase character...
81              if ((uint8(bStr[i]) >= 65) && (uint8(bStr[i]) <= 90)) {
82                  // So we add 32 to make it lowercase
83                  bLower[i] = bytes1(uint8(bStr[i]) + 32);
84              } else {
85                  bLower[i] = bStr[i];
86              }
87          }
88          return string(bLower);
89      }
90  }
```

## A.2 MinDAO.sol

```
1   pragma solidity ^0.5.1;
2   import "./CertificateTransparency.sol";
3
4   contract MinDAO {
5       address initialOwner;
6       CertificateTransparency ct;
7       mapping (address => bool) addressIsMember;
8       mapping (uint => Proposal) proposals;
9       mapping (uint => Request) requests;
10      uint numMembers;
11      uint numProposals;
12      uint numRequests;
13
14      struct Request {
15          address owner;
16          uint bounty;
17          string URL;
18          string certificate;
19          uint16 numAttesters;
20          bool issued;
21          mapping (address => bool) hasAttested;
22      }
```

```
23
24      struct Proposal {
25          address subject;
26          uint16 yays;
27          uint16 nays;
28          bool isKick;
29          mapping(address => bool) hasVoted;
30      }
31
32      constructor() public {
33          initialOwner = msg.sender;
34          numMembers = 1;
35          numProposals = 0;
36          addressIsMember[initialOwner] = true;
37          ct = new CertificateTransparency(address(this));
38      }
39
40      modifier onlyMember(){
41          require(addressIsMember[msg.sender]);
42          _;
43      }
44
45      function setCertificate(address _certOwner, bytes32 _certHash,
            bytes32 _hashedUrl) internal onlyMember {
46          ct.ownerSetCertificate(_certOwner, _certHash, _hashedUrl);
47      }
48
49      function toBytes(uint256 x) internal pure returns (bytes memory
             b) {
50          b = new bytes(32);
51          for (uint i = 0; i < 32; i++) {
52              b[i] = byte(uint8(x / (2**(8*(31 - i)))));
53          }
54      }
55
56      //found @ https://ethereum.stackexchange.com/questions/32003/
            concat-two-bytes-arrays-with-assembly
57      function MergeBytes(bytes memory a, bytes memory b) public pure
             returns (bytes memory c) {
58          // Store the length of the first array
59          uint alen = a.length;
60          // Store the length of BOTH arrays
61          uint totallen = alen + b.length;
62          // Count the loops required for array a (sets of 32 bytes)
63          uint loopsa = (a.length + 31) / 32;
64          // Count the loops required for array b (sets of 32 bytes)
65          uint loopsb = (b.length + 31) / 32;
66          assembly {
67              let m := mload(0x40)
68              // Load the length of both arrays to the head of the
                    new bytes array
69              mstore(m, totallen)
70              // Add the contents of a to the array
71              for {  let i := 0 } lt(i, loopsa) { i := add(1, i) } {
                    mstore(add(m, mul(32, add(1, i))), mload(add(a, mul
                    (32, add(1, i))))) }
72              // Add the contents of b to the array
```

```
73              for {  let i := 0 } lt(i, loopsb) { i := add(1, i) } {
                    mstore(add(m, add(mul(32, add(1, i)), alen)), mload
                    (add(b, mul(32, add(1, i))))) }
74              mstore(0x40, add(m, add(32, totallen)))
75              c := m
76          }
77      }
78
79      function requestCertificate(string memory _url, string memory
            _certificate) public payable returns(uint) {
80          //We require no minimum payment, as it is up to the
                attesters, whether or not they want to sign something.
81          requests[numRequests] = Request(msg.sender, msg.value,
                _url, _certificate, 0, false);
82          numRequests += 1;
83          return numRequests-1;
84      }
85
86      function attestCertificate(uint requestID) external onlyMember
            {
87          //If the member has alreay attested the CSR, they cannot
                attest it again.
88          require(!requests[requestID].hasAttested[msg.sender]);
89          requests[requestID].hasAttested[msg.sender] = true;
90          requests[requestID].numAttesters += 1;
91          //If half or more of the DAO members have attested a
                certificate, it is finalized on the spot.
92          if(requests[requestID].numAttesters > numMembers/2){
93              finalizeCertificate(requestID);
94          }
95      }
96
97      function finalizeCertificate(uint requestID) public {
98          require(requests[requestID].numAttesters > numMembers/2);
99          bytes32 hashedURL = keccak256(bytes(requests[requestID].URL
                ));
100         bytes memory certificateBytes = bytes(requests[requestID].
                certificate);
101         bytes memory urlBytes = bytes(requests[requestID].URL);
102         bytes32 certificateHash = keccak256(MergeBytes(
                certificateBytes, urlBytes));
103
104         setCertificate(requests[requestID].owner, certificateHash,
                hashedURL);
105     }
106
107     function proposeNewMember(address newMember) external
            onlyMember returns (uint) {
108         Proposal memory prop = Proposal(newMember, 0, 0, false);
109         proposals[numProposals] = prop;
110         numProposals++;
111         return numProposals - 1;
112     }
113
114     function proposeKickMember(address kickMember) external
            onlyMember returns (uint){
115         Proposal memory prop = Proposal(kickMember, 0, 0, true);
```

```
116         proposals[numProposals] = prop;
117         numProposals++;
118         return numProposals - 1;
119     }
120
121     function vote(uint id, bool voteYes) external onlyMember {
122         require(proposals[id].subject != address(0x0));
123         require(!proposals[id].hasVoted[msg.sender]);
124
125         proposals[id].hasVoted[msg.sender] = true;
126         if(voteYes){
127             proposals[id].yays = proposals[id].yays+1;
128         } else {
129             proposals[id].yays = proposals[id].nays+1;
130         }
131
132         if(proposals[id].yays > numMembers / 2){
133             addressIsMember[proposals[id].subject] = !proposals[id
                    ].isKick;
134         }
135     }
136 }
```

## A.3 Controller.sol

```
1   pragma solidity ^0.5.1;
2
3   import "./token/MigrateableToken.sol";
4   import "./CertificateTransparency.sol";
5
6   contract ControllerSupplier {
7       function createChild(address _parent) public returns (address
            _newController);
8   }
9
10  contract ControllerFactory {
11      function createChild(address _parent) public returns (address
            _newController) {
12          return address(new Controller(_parent, address(this), false
                ));
13      }
14  }
15
16  contract Controller{
17
18      MigrateableToken public token;
19      CertificateTransparency public certificateLog;
20      Controller public parent;
21      Controller public acceptedChild;
22      Controller public rejectedChild;
23      ControllerSupplier public childCreator;
24      bool isForked = false;
25      mapping(uint => Vote) votes;
26      mapping(uint => Cert) certs;
27      uint disputeId;
28      uint FORK_THRESHOLD = 10;
29      uint certCount = 0;
```

```
30      uint BOUNTY_THRESHOLD = 1000;
31      uint MAX_ROUND = 7;
32      uint MINIMUM_STAKE = 1;
33      uint VOTE_PERIOD_TIME = 3 * 24 * 60 * 60;
34
35      struct Vote {
36          uint certId;
37          uint accepted;
38          uint rejected;
39          uint voteStart;
40          uint voteEnd;
41          uint round;
42          uint bounty;
43          bool isFinalized;
44          mapping(address => uint) stakedAccepted;
45          mapping(address => uint) stakedRejected;
46      }
47
48      struct Cert {
49          uint certId;
50          uint expiry;
51          string url;
52          string certificate;
53          address certOwner;
54      }
55
56      constructor(address _parent, address _childCreator, bool
            _isGenesis) public{
57          if(_isGenesis){
58              parent = Controller(address(0));
59              token = new MigrateableToken(_parent, true);
60              certificateLog = new CertificateTransparency(address(
                    this));
61          } else {
62              require(_parent != address(0));
63              certificateLog = new CertificateTransparency(address(
                    this));
64              parent = Controller(_parent);
65              token = new MigrateableToken(address(Controller(_parent
                    ).token), false);
66          }
67          isForked = false;
68          childCreator = ControllerSupplier(_childCreator);
69      }
70
71      function request(string memory url, string memory certificate,
            uint expiry, uint bounty) public returns(uint){
72          require(token.balanceOf(msg.sender) >= bounty);
73          require(bounty >= BOUNTY_THRESHOLD);
74          require(token.transfer(address(this), bounty));
75          certs[certCount] = Cert(certCount, expiry, url, certificate
                , msg.sender);
76          votes[certCount] = Vote(certCount, 0, 0, now, now +
                VOTE_PERIOD_TIME, 0, bounty, false);
77          certCount += 1;
78          return certCount-1;
79      }
```

```
80
81      function vote(uint _id, uint _choice, uint _amount) external{
82          require(now < votes[_id].voteEnd);
83          require(now > votes[_id].voteStart);
84          uint personalStake = votes[_id].stakedAccepted[msg.sender]
                + votes[_id].stakedRejected[msg.sender];
85          require(personalStake + _amount > MINIMUM_STAKE);
86          if(personalStake == 0){ //If nothing has been stked on this
                 outcome yet.
87              token.incrementStakes();
88          }
89          token.stake(_amount + personalStake, certs[_id].expiry);
90          if(_choice == 0){
91              // Choice 0 = accept certificate
92              votes[_id].accepted += _amount;
93              votes[_id].stakedAccepted[msg.sender] += _amount;
94          } else if(_choice == 1){
95              // Choice 1 = reject certificate
96              votes[_id].rejected += _amount;
97              votes[_id].stakedRejected[msg.sender] += _amount;
98          } else {
99              revert(); //If people don't make a deliberate correct
                    choice, they will not stake.
100         }
101     }
102
103     function finalizeVote(uint _id) external {
104         require(now > votes[_id].voteEnd);
105         require(!votes[_id].isFinalized);
106         if(canFinalize(_id)){
107             votes[_id].isFinalized = true;
108             if(votes[_id].accepted > votes[_id].rejected){
109                 votes[_id].bounty += votes[_id].rejected;
110                 certificateLog.ownerSetCertificate(
111                     certs[_id].certOwner,
112                     keccak256(abi.encode(certs[_id].certificate)),
113                     keccak256(abi.encode(certs[_id].url))
114                 );
115             } else {
116                 votes[_id].bounty += votes[_id].accepted;
117             }
118         } else if(canFork(_id)){
119             fork(_id);
120         } else {
121             votes[_id].round += 1;
122             votes[_id].voteStart = now;
123             votes[_id].voteEnd = now + VOTE_PERIOD_TIME;
124         }
125     }
126
127     function withdraw(uint _id) external {
128         require(votes[_id].isFinalized);
129         uint staked = 0;
130         uint bounty = 0;
131         //If contract is forking or forked
132         if(disputeId == 0){
133             if(votes[_id].accepted > votes[_id].rejected){
```

```
134                 staked = votes[_id].stakedAccepted[msg.sender];
135                 bounty = votes[_id].bounty * (staked / votes[_id].
                        accepted);
136                 votes[_id].stakedAccepted[msg.sender] -= staked;
137                 token.incrementStakes();
138                 token.transfer(msg.sender, bounty);
139              } else {
140                 staked = votes[_id].stakedRejected[msg.sender];
141                 bounty = votes[_id].bounty * (staked / votes[_id].
                        rejected);
142                 votes[_id].stakedRejected[msg.sender] -= staked;
143                 token.decrementStakes();
144                 token.transfer(msg.sender, bounty);
145              }
146         // If contract is forking, but not on this specific vote,
                let participants withdraw.
147         } else if(disputeId != _id){
148             staked = votes[_id].stakedRejected[msg.sender] + votes[
                    _id].stakedAccepted[msg.sender];
149             votes[_id].stakedAccepted[msg.sender] = 0;
150             votes[_id].stakedRejected[msg.sender] = 0;
151             if(certs[_id].certOwner == msg.sender){
152                 staked += votes[_id].bounty;
153                 votes[_id].bounty = 0;
154             }
155             token.transfer(msg.sender, staked);
156         //If disputed certificate, force migration
157         } else {
158             staked = votes[_id].stakedAccepted[msg.sender];
159             if(staked > 0){
160                 bounty = votes[_id].bounty * (staked / votes[_id].
                        accepted);
161                 votes[_id].stakedAccepted[msg.sender] -= staked;
162                 MigrateableToken(address(acceptedChild.token)).mint
                        (msg.sender, staked + bounty);
163             }
164             staked = votes[_id].stakedRejected[msg.sender];
165             if(staked > 0){
166                 bounty = votes[_id].bounty * (staked / votes[_id].
                        rejected);
167                 votes[_id].stakedRejected[msg.sender] -= staked;
168                 MigrateableToken(address(rejectedChild.token)).mint
                        (msg.sender, staked + bounty);
169             }
170         }
171     }
172
173     function canFinalize(uint _id) public view returns (bool){
174         //Can finialize if the less votes option is below the
                threshold.
175         if(min(votes[_id].accepted, votes[_id].rejected) <
                roundThreshold(votes[_id].round)){
176             if(now > votes[_id].voteEnd){
177                 return true;
178             }
179         }
180         return false;
```

```
181     }
182
183     function canFork(uint _id) public view returns (bool){
184         //If the more than the maximum roundThreshold disputed the
                 choice, fork.
185         if(min(votes[_id].accepted, votes[_id].rejected) >
                roundThreshold(MAX_ROUND)){
186             return true;
187         }
188         return false;
189     }
190
191     function fork(uint certId) public {
192         require(canFork(certId));
193         //1. Create new controller
194         token.fork();
195         acceptedChild = Controller(childCreator.createChild(address
                (this)));
196         rejectedChild = Controller(childCreator.createChild(address
                (this)));
197         //2. Create new accepted certificate in acceptedChild
                history
198
199         acceptedChild.parentForceCert(
200             certs[certId].certOwner,
201             keccak256(abi.encode(certs[certId].certificate)),
202             keccak256(abi.encode(certs[certId].url))
203         );
204
205         disputeId = certId;
206     }
207
208     function parentForceCert(address certOwner, bytes32 certificate
            , bytes32 url) external {
209         require(msg.sender == address(parent));
210         certificateLog.ownerSetCertificate(certOwner, certificate,
                url);
211     }
212
213     function min(uint a, uint b) internal pure returns (uint){
214         if(a < b){
215             return a;
216         }
217         return b;
218     }
219
220     function getBalance() public view returns (uint){
221         return token.balanceOf(msg.sender);
222     }
223
224     function roundThreshold(uint round) public view returns (uint){
225         return token.totalSupply()*FORK_THRESHOLD/100/2**(MAX_ROUND
                -round);
226     }
227 }
```

## A.4 Tokens

### A.4.1 ERC20Basic.sol

```
1  pragma solidity ^0.4.23;
2
3  /**
4   * @title ERC20Basic
5   * @dev Simpler version of ERC20 interface
6   * @dev see https://github.com/ethereum/EIPs/issues/179
7   */
8  contract ERC20Basic {
9    function totalSupply() public view returns (uint256);
10   function balanceOf(address who) public view returns (uint256);
11   function transfer(address to, uint256 value) public returns (bool
       );
12   event Transfer(address indexed from, address indexed to, uint256
       value);
13 }
```

### A.4.2 BasicToken.sol

```
1  pragma solidity ^0.4.23;
2
3  import "./ERC20Basic.sol";
4  import "../../math/SafeMath.sol";
5
6  /**
7   * @title Basic token
8   * @dev Basic version of StandardToken, with no allowances.
9   */
10 contract BasicToken is ERC20Basic {
11   using SafeMath for uint256;
12
13   mapping(address => uint256) balances;
14
15   uint256 totalSupply_;
16
17   /**
18   * @dev total number of tokens in existence
19   */
20   function totalSupply() public view returns (uint256) {
21     return totalSupply_;
22   }
23
24   /**
25   * @dev transfer token for a specified address
26   * @param _to The address to transfer to.
27   * @param _value The amount to be transferred.
28   */
29   function transfer(address _to, uint256 _value) public returns (
       bool) {
30     require(_to != address(0));
31     require(_value <= balances[msg.sender]);
32
33     balances[msg.sender] = balances[msg.sender].sub(_value);
34     balances[_to] = balances[_to].add(_value);
```

```
35      emit Transfer(msg.sender, _to, _value);
36      return true;
37    }
38
39    /**
40    * @dev Gets the balance of the specified address.
41    * @param _owner The address to query the the balance of.
42    * @return An uint256 representing the amount owned by the passed
        address.
43    */
44    function balanceOf(address _owner) public view returns (uint256)
        {
45      return balances[_owner];
46    }
47  }
```

### A.4.3 BurnableToken.sol

```
1   pragma solidity ^0.4.23;
2   import "./BasicToken.sol";
3
4   /**
5    * @title Burnable Token
6    * @dev Token that can be irreversibly burned (destroyed).
7    */
8   contract BurnableToken is BasicToken {
9
10    event Burn(address indexed burner, uint256 value);
11
12    /**
13     * @dev Burns a specific amount of tokens.
14     * @param _value The amount of token to be burned.
15     */
16    function burn(uint256 _value) public {
17      _burn(msg.sender, _value);
18    }
19
20    function _burn(address _who, uint256 _value) internal {
21      require(_value <= balances[_who]);
22      // no need to require value <= totalSupply, since that would
          imply the
23      // sender's balance is greater than the totalSupply, which *
          should* be an assertion failure
24
25      balances[_who] = balances[_who].sub(_value);
26      totalSupply_ = totalSupply_.sub(_value);
27      emit Burn(_who, _value);
28      emit Transfer(_who, address(0), _value);
29    }
30  }
```

### A.4.4 StakeableToken.sol

```
1   pragma solidity ^0.5.1;
2   import "./BurnableToken.sol";
3
4   contract StakeableToken is BurnableToken {
```

```
5
6     mapping(address => uint) public staked;
7     mapping(address => uint) public release;
8     mapping(address => uint) public activeStakes;
9     address controller;
10
11    function stake(uint amount, uint release_time) external{
12        uint current_balance = this.balanceOf(msg.sender);
13        require(current_balance + staked[msg.sender] >= amount);
14        if(amount > current_balance){
15            staked[msg.sender] += amount - current_balance;
16            balances[msg.sender] -= amount - staked[msg.sender];
17        }
18        if(release[msg.sender] < release_time){
19            release[msg.sender] = release_time;
20        }
21    }
22
23    function withdraw(uint amount) public returns(bool){
24        require(staked[msg.sender] > amount);
25        require(release[msg.sender] < now);
26        require(activeStakes[msg.sender] == 0);
27        staked[msg.sender] = staked[msg.sender] - amount;
28        balances[msg.sender] += amount;
29    }
30
31    function incrementStakes() public{
32        activeStakes[msg.sender] += 1;
33    }
34    function decrementStakes() public{
35        activeStakes[msg.sender] -= 1;
36    }
37
38    function amountStaked(address staker) public view returns (uint
          ){
39        return staked[staker];
40    }
41    function releaseTime(address staker) public view returns (uint)
          {
42        return release[staker];
43    }
44 }
```

### A.4.5 MigrateableToken.sol

```
1  pragma solidity ^0.5.1;
2
3  import "./StakeableToken.sol";
4
5  contract MigrateableToken is StakeableToken {
6
7      address parent;
8      address owner;
9      bool isForked;
10     bool isGenesis;
11
12     constructor(address _parent, bool _isGenesis) public{
13         owner = msg.sender;
```

```
14          if(_isGenesis){
15              parent = address(0);
16              isGenesis = _isGenesis;
17              mint(_parent, 10*10**12);
18          } else {
19              parent = _parent;
20          }
21          isForked = false;
22      }
23
24      function transfer(address _to, uint256 _value) public returns (
            bool){
25          require(!isForked);
26          return super.transfer(_to, _value);
27      }
28
29      function fork() public {
30          require(msg.sender == owner);
31          isForked = true;
32      }
33
34      function mint(address migrater, uint amount) public{
35          require(msg.sender == parent);
36          totalSupply_ += amount;
37          balances[migrater] += amount;
38      }
39
40      function migrate(MigrateableToken child, uint amount) external{
41          //Call mint on child and burn tokens
42          super.burn(amount);
43          child.mint(msg.sender, amount);
44      }
45  }
```

### A.4.6 SafeMath.sol

```
1   pragma solidity ^0.5.1;
2
3   /**
4    * @title SafeMath
5    * @Developed by OpenZeppelin
6    * @https://github.com/OpenZeppelin/openzeppelin-solidity/blob/
         master/contracts/math/SafeMath.sol
7    * @dev Math operations with safety checks that throw on error
8    */
9   library SafeMath {
10
11    /**
12    * @dev Multiplies two numbers, throws on overflow.
13    */
14    function mul(uint256 a, uint256 b) internal pure returns (uint256
          c) {
15      if (a == 0) {
16        return 0;
17      }
18      c = a * b;
19      assert(c / a == b);
20      return c;
```

```
21     }
22
23     /**
24     * @dev Integer division of two numbers, truncating the quotient.
25     */
26     function div(uint256 a, uint256 b) internal pure returns (uint256
           ) {
27       // assert(b > 0); // Solidity automatically throws when
             dividing by 0
28       // uint256 c = a / b;
29       // assert(a == b * c + a % b); // There is no case in which
             this doesn't hold
30       return a / b;
31     }
32
33     /**
34     * @dev Subtracts two numbers, throws on overflow (i.e. if
           subtrahend is greater than minuend).
35     */
36     function sub(uint256 a, uint256 b) internal pure returns (uint256
           ) {
37       assert(b <= a);
38       return a - b;
39     }
40
41     /**
42     * @dev Adds two numbers, throws on overflow.
43     */
44     function add(uint256 a, uint256 b) internal pure returns (uint256
            c) {
45       c = a + b;
46       assert(c >= a);
47       return c;
48     }
49   }
```