Certificates in the Ethereum

David Šanda

Version 0.0.1

## Introduction

This text is intended to be a starting point in exploration of how to work with certificates in relation to the Ethereum blockchain and contracts (sometimes abbreviated just blockchain). Sample application for the infrastructure is to enable authentication for the insecure file transfer, but the important thing about this is the certificate management, not any particular application.

We will use the general X.509 structure and try to incorporate placement of certificates in the blockchain as a replacement for the actual signature and various other ideas.

Note that some ideas might not be worth pursuing and are only here to be effectively discussed or disproved.

## Concepts

These are very general concepts in which it is useful to think about this topic. It does not contain in practical information.

### Contracts and certificates

There is a fit of Ethereum contracts corresponding to Certificate Authorities and delegated CAs. In other words, certificate can be present in blockchain as standalone contract (with signature, pointer to signing certificate etc.), where you can work with the certificate – use it in some way to sign certificate signing requests etc. Naturally, your public key might directly be the “founder” address present in the certificate variable (Ethereum address is a hash of ECDSA public key), which would make the scheme even more elegant. Actions you might do are simply using your private key (or corresponding founder, if this is possible) in a way, that makes it easy to work with the certificate via web API. Signing the certificate (contract cannot actually hold or use private key, but it can do all the validation). One way to think about it is that apart from extra functionality, CA contracts might actually work as convenient and easily accessibly OpenSSL utilities.

### CAs and Certificates

Some certificates are just certifying to some fact and the use of the certificate is rather passive even if the private key might be used. For example certificate for https site or better yet verifying document signature against the certificate.

Others are active and there are possible actions associated with the certificates such as creating sub certificate (in other words sub certificate authority) other such action is for example signing or encrypting the document.

### Certificate signatures and certificate chain

One of the main benefits of the Ethereum involvement is that it allows us to deal with signatures and verification of certificate chains efficiently. As long as they are in the blockchain all the way. This is trivial, but useful.

What we can do:

* X.509 Issuer Unique Identifier (CA) can contain address of the contract that CA represents and itself. Meaning you can only share the address, where the certificate is and it is publicly and securely available.
* X.509 Subject Unique Identifier can contain address of itself, where it the certificate can be located in the blockchain.
* **Blockchain inclusion signatures**. Instead of RSA/ECDSA/EdDSA use whitelist, where CA lists hashes of all the certificates it issued and this list is the evidence of it. This may sound strange, but it is possible to imagine situations, where it makes sense. If the PC or simply device already interacts with Ethereum, it might be more efficient to retrieve some data from blockchain instead of validating the signature using asymmetric cryptography. I think this use is more of a theoretical exercise, because working with Ethereum is not exactly lightweight, so there might not be a scenario, where it is more practical to verify signature by checking their presence. The thing is, that while this service can be provided via some web API, you have to trust this web API.
* As above, but this is used in addition to signature.
* Revocation. As above, but this is a situation, that makes sense. Contract representing CA can easily publish revocation.

## Building blocks

In here we give practical descriptions to build blockchain enabled certificate management system. They are general and have no special purpose or extra features.

### Certificate authority contract (aka CAC)

This is the contract that that is created for each CA. It functions as the address of the CA, as a place where CA publishes revocations, offers additional functionality. If contains a link/contract/ to the CA that issued itself, so chain of trust can be quickly established.

* Both features below are privacy sensitive, so they should be strictly optional for the user.
* Consider, that this whitelist can easily serve as the signature itself. It might be thought of as a “real-time” signature, meaning that when validating given cert, you have to always check this CA and its whitelist and blacklist, which can have its role.
  + It might also provide an option to store hashes (? signatures) of issued certificates.
  + It might also provide an option to store entire certificate it issued. User then does not need to present it, and instead it can just point to the contract and some index number.

The certificates issued by CA as above should contain contract that houses the CA to make it easy to validate it.

The reason why every (sub) CA is tied to a single contract is the control. One entity usually manages one CA/contract. All of the above can obviously be implemented as a single large contract, but it would require extensive access control and the data attached to the contract would be far more cumbersome.

It is important to realize, that issuing certificate in this way is a complex process. User submits CSR (certificate signing request), CA has to consider it, sign it and publish it. Majority of this is done outside of blockchain. Publishing it means several things. Optionally putting entire certificate or just hash into the whitelist, if (sub) CA is issued, it needs to create contract for it and assign the right to it to the user and only then sign it and upload it to said contract, as naturally, contract address is not predictable and must also be signed.

### Anonymous Certificate Bulletin Board (aka ACBB)

This is a contract that simply collects and publishes certificates that could tie in with already existing certificate system. I imagine, it could be a contract, that could be either operated by anybody or by some organization and it would collect and make available certificates. A lot of implementation work can go into doing this efficiently, as certificates contain redundant data, especially if similar.

Publishing certificates in this way scales as long as blockchain lives. The only downside is revocation checking, which is not handled by this.

Alternatively, fact of having the certificate published in particular ACBB can have some meaning in itself if access to it is controlled.

Even if anonymous, there is little danger of abuse of such contract, as users can only add certificates and adding bogus one has little sense. Also, adding anything to it cost Ether, so there should be no bogus data to slow the certificates down.

### Verifier contract

“Public” contract that performs the certificate verification, including time (current time being time of last block creation). As argument, you put in the certificate, you wish to verify and list of root certificates you trust. Result would be OK or FAIL. I guess, it would be fairly complex contract, but we need it anyway.

### Certificate modifications

We need to store the issuer contract and for (Sub) CAs, subject contract. There are basically two ways. Either implement it as version 3 proprietary extension or adopt Subject Unique Identifier and Issuer Unique Identifier from version 2 for the values. I guess going the way of extension would make it easier to integrate it into current systems, but the former is also a natural fit.

If the certificate is end user or if it simply doesn’t require its own contract, then it can just as well list the CA contract where it is published.

## File authentication use case

Here we consider our current use case of authenticating files we send to another user via unsecured channel.

1. User A signs the hash of the file and sends signature and either certificate or just contract where the certificate is listed (his immediate CA).
2. User B gets on blockchain, verifies the certificate either using contract found in issuer field or by following the link and matching identity of the sender.
3. If the user wishes to pass a secret message to the receiver, natural way to do it, would be to encrypt this message using receiver public key and send it along the link. Secret message is in effect ephemeral and has no place in blockchain.
4. If point 3 is not to your linking, user can have its own contract (as CAs need to), where he can store such messages etc.

## API

API level description of contracts described above will be here, when we discuss it and agree, that we move further.

## Measurements

Contract scaling, gas price predictions and other feasibility questions come here. To be done by University team.