PoWcert: Decentralised Proof-of-Work Certificates

Koen van Hove

July 29, 2018

Abstract

Certificates provide proof of the authenticity and integrity of a document. The problem we are currently facing is that all of our trust is put in a handful of certificate authorities. This paper will propose a decentralised alternative, based on proof-of-work and circles of trust, allowing any user to sign a document, thereby removing the need to rely on one central authority.

1 Introduction

Current document signing methods depend on a central certificate authority (or pre-established trust) and authority of one signer. PoWcert requires neither. It alleviates the problem of trust in a single authority by employing a decentralised herd protection system, allowing a single document to be signed by many.

2 A New Way to Trust

Although current certificate systems work when the signer can be trusted, they provide little more guarantee than a hash if the signer is unknown. Additionally, they rely on a few select intermediate and root certificates. This is a very hierarchical structure, where we as a collective have decided to trust certain vendors to take care of their root certificates. The problem that arises is that we are forced to trust them, even if we do not know them.

2.1 Circle of Trust

For this reason, we propose a radically different system based on actual trust. You have individuals that you trust, they have individuals that they trust and so on. This in turn links individuals together based on the documents they sign, a *Circle of Trust.* This removes the need for a forced trust relationship with some root or intermediate certificate authority, as the structure is fully flat.

Of course a single link between two individuals may be insufficient, and one may impose more prerequisites to establish a trust relationship. Likewise, when someone signs a document that you distrust, this can negatively impact your trust in that individual. These conditions are to be decided by the user.

2.2 Herd Protection

Just like when moving to a new city where everyone is unknown, it is very feasible that one comes across a document that has not been signed by anyone they already know and trust. This is why signing a document follow the proof-of-work model, by requiring some CPU power to sign a document. The idea being that *honest* documents will be signed more often than *dishonest* documents (as dishonest individuals often have differing motives, whereas honest individuals have the same goal). If a document is signed by more *keys*, then the chance of it being honest increases.

3 Decentralised

The system works by letting any user sign a document, by generating a key based on their user id and the document they sign. The user id is not centrally registered, but uses the same system as email for identifying the user. The advantage is that this makes the system fully decentralised, and requires no central ledger to keep track of all the users.

4 Design

4.1 Generating

The user provides the SHA-1 hash of a document (the *doc_code*, for example `b55bfb...`), their user identifier (the *user_code*, for example `koen@koenvh.nl`) and their private key (ECDSA NIST192p SHA1). The *user_code* and *doc_code* should be in lower case.

The CRC32 of the *user_code* will be calculated, which will be converted to hexadecimal (for example `2e1009db`). It will then generate a set of potential keys (where a key is a number), by taking the SHA-512 hash of the *doc_code* + *user_code* + *key* (for example `b55bfb...koen@koenvh.nl512522`). In this example, this would result in. `214ec9...`. If this sequence of characters contains the CRC32 calculated earlier (in this example `2e1009db`), it is a valid key, and it will be added to a list of valid keys.

A signature will be generated using the private key, based on the *doc_code*, *user_code* and the set of *keys* (in that order, each item separated by `\n`).

4.2 Verifying

In order to verify the keys from a user, the user's public key needs to be retrieved. This can be done by making a DNS query to the domain from the *user_code*. A TXT record with a value `PoWcert=<server>` will point to the server where the public key for this user can be retrieved (for example `PoWcert=https://example.org/certs/{user_code}.pem`). The client will then replace the `{user_code}` placeholder with the user_code, and request the public key.

The *doc_code*, *user_code* and *keys* will then be verified using the retrieved public key and the signature provided.

For the *keys* provided by the user, the SHA-512 and CRC32 hash are calculated in the same way as described above, and it is checked that hashes of the keys do indeed contain the CRC32 of the *user_code*.

5 Repository

The circle of trust can differ per person, but it can also be more formalised by erecting a repository. A repository is a central place where individuals can download certificates, and add their own keys to existing certificates. The exact process and regulations are entirely up to the owner of the repository.

6 Incentive

The current design has, due to its fully decentralised design, no means of providing an incentive. Repositories may provide an additional incentive to sign documents. This is an area that is to be researched further.

7 Proof of Concept

A Python implementation of the specification described above has been implemented, and can be found on `https://github.com/Koenvh1/PoWcert`. It requires Python >3.5 to run, along with the following packages: `pip install dnspython requests ecdsa`

Then it is possible to run either the generator or the verifier. The generator calculates new keys, the verifier verifies the keys in the document.

As an example, one can run `python generate.py johndoe@example.org example_doc.bmp example_doc.bmp.powcert` to generate more keys, and `python verify.py example_doc.bmp example_doc.bmp.powcert` to verify the keys (add `--verify` to verify signatures as well).

In order to sign keys, please generate a private/public key pair by running `python generate_key.py`. Upload the public key to some location according to the specification described in "Verifying". It is then possible to add `--sign private.pem` to your generate command.

An example of a repository can be found on `https://home.koenvh.nl/powcert` (the source can be found on `https://github.com/Koenvh1/PoWcert-directory`).

8 Statistics

The current implementation is capable of calculating around 10^6 hashes per second on average systems. Given that the probability of generating a valid hash is roughly $1/10^8$, a valid key will on average be generated once per 100 seconds.

9 Potential Use Cases

9.1 Community Selection

When two or more individuals bring out their version of a document, the community can use the proposed system to "vote" for the proper one by signing it. Creating a herd around the proper one. This uses the *Herd Protection* and *Circle of Trust* as described above. A central certificate authority signed by one would not allow the community to pick out the right one.

9.2 Decentralised Ad Hoc Peer-to-Peer Networks

Ad Hoc Peer-to-Peer Networks do not always feature a master that dictates the actions by others. Imagine that there is a group of 200 individuals currently in the network, and a new individual decides to join, and broadcasts a message asking whether a name is available. If one individual does not want to allow people to join, they would always respond with "no". Using this system, given that the majority will be honest, the answer will be honest.

Using a system that just asks for everyone's opinion allows others to falsify the amount of votes by joining the network multiple times, or forging the opinion of others.