A study of the web components for the Next Web

Author
Najib Ghadri
najibg96@gmail.com

and

Supervisor
Dr. Balázs Goldschmidt
balage@iit.bme.hu

Buapest University of Technology and Economics
Department of Control Engineering and Information Technology

May 2019

Contents

1 Introduction

There are several problems that we are facing today with the internet. Big companies like Facebook and Google hold a lot of data about us, and they are able to manipulate it without anyone noticing. Today we use a couple of web services to represent our digital identity: email addresses, a facebook account, a linked in account, instagram account, and so on. The problem with this is that: 1; a lot of data leaks out about us since we are connecting a lot of services to central ones, 2; once data has leaked it is difficult to start a new page; the services already know your personal email, they can bombard you with spam emails, sell your email address and sell your data to third parties while you are still behind the same identificators.
Today we want something new that respects our privacy and our desire for self-sovereignity. This has already been conceptualized and called Self-Sovereign Identity (SSI). We use the terminology of SSI, as the concept of individuals or organizations having sole ownership of their digital and analog identities, and control over how their personal data is shared and used. Using the words of the Rebooting the Web Of Trust organization, SSI is:

- Existence. Users must have an independent existence.
- Control. Users must control their identities
- Access. Users must have access to their own data.
- Transparency. Systems and algorithms must be transparent.
- Persistence. Identities must be long-lived.

- Portability. Information and services about identity must be transportable.
- Interoperability. Identities should be as widely usable as possible.
- Consent. Users must agree to the use of their identity
- Minimalization. Disclosure of claims must be minimized.
- Protection. The rights of users must be protected.

from Cristopher Allen's blog

This means that we want to be sure about our data in the cloud. A lot of these requirements are technically possible today, but some are more difficult. Innovations in this topica are measures to prevent bad things from happenings, however they won't yield a prefect solution. We will see concrete mathematical examples to this in this paper when we discuss consensus algorithms and proofs: the goal is always increasing probability of desired outcome, it's not possible to reach certanty, merely by the nature of the mathematical mechanisms used.

One thing that annoys me personally is email. What bothers me in it is that nowadays every web service seems to think it is an identifier, whereas it is only an electronic mail address. In the real world you aren't identified by your house address either, but by one of your identity documents.

We will see in this study that this is already the direction or today's researches: Decentralized IDs or DIDs and "verifiable claims" are coming for the rescue for self-sovereignity. Another thing bothers me in emails, is that once you give your email to a party they can send you emails no matter what. A good design decision would be to make different "proxy' identities to each service/party and once we got fed up with a certain service, we simple disable the proxy identity given to that service party.

Another flaw today is the privacy part of our data. Privacy on the internet can be achieved by 1; not providing the data in the first place, 2; encrypting it in a way so that only those can read the data whom we meant it to. This is achieved using assymetric cryptography, and one important and flawed part of it is the key exchange part. Today we rely on centralized systems to distribute the public keys of different entites such as a webiste's tls certificate using the DNS and the Public Key Infrastructure. The obvious deficiency is the centralization: corruptible by humans and prone to MITM attacks. One recent larger attack related to this is the 2016 Dyn DNS cyberattack. DNS is centralized and has suffered many attacks in the past. This system could be improved with a distributed cryptographically secure system such as the distributed ledger technology - blockchains. Blockchains - if turns out possible - could also replace centralized services too such as social networks. This would make us trust more what we see on these websites, maybe improve our digital social interactions and maybe we could see better information. In this paper we will see solution

propsals to these problems. If these problems are improved we would be able to send emails and messages to each other with more certanty about our privacy and security.

HTTP has also had competition from the distributed world, namely a system that is inspired by the torrent protocols. We will also see a new standard that encourages the use of linked data structrues, called InterPlanetary LinkedData all of these in the 6th and 7th sections.

First I will present the current authentication and authorization methods used widely then present the centralized PKI and continue with its decentralized version.

2 Authorization and authentication standards

With websites like Facebook or Google, a user can log in to one application with a set of credentials. This same set of credentials can then be used to log in to related websites or applications, like websites that ask you, "Sign up with Facebook or Google account?".

A business may have an internal employee portal with links to intranet sites regarding work-related things, or company news. Rather than requiring an employee to log in at each website, a better solution would be to have the employee log in at a portal, and have that portal automatically authenticate the user with the other intranet sites. This idea, called single sign-on (SSO), allows a user to use one set of credentials in order to access multiple applications.

The benefits are straightforward. The use of linked identities means we only need to manage one set of credentials for the related websites. The user experience is better, as they can avoid multiple logins. A user's credentials will be stored in one database, rather than multiple credentials stored across multiple databases. This also means that developers of the various applications don't have to implement authentication. Instead, they can accept proof of identity or authorization from a trusted source.

There are multiple solutions for implementing SSO. The three most common web security protocols, at the time of this writing, are OpenID, OAuth, and SAML. Implementations and libraries exist in multiple languages already, and using standards allows better interoperability.

2.1 OpenID

https://openid.net/

OpenID is an open standard for **authentication**, promoted by the non-profit OpenID Foundation. Beware, this system only deals with authentication! There are over a billion OpenID-enabled accounts on the internet, and organizations such as Google, WordPress, Yahoo, and PayPal use OpenId to authenticate users.

A *user* must obtain an OpenID account through an OpenID *identity provider* (for example, Google). The user will then use that account to sign into any

website (the *relying party*) that accepts OpenID authentication (for example Google login to Youtube). The OpenID standard provides a framework for the communication that must take place between the identity provider and the relying party.

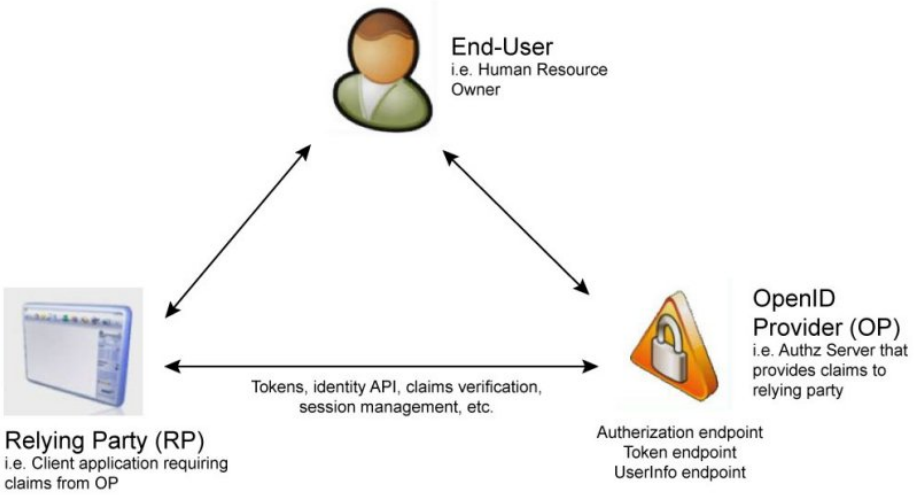


Figure 1: A picture of it's general architecture (just like a SOA architecture)

2.2 OAuth2

https://oauth.net/2/

OAuth2 is an open standard for **authorization**. It began in 2006 from Twitter. OAuth2 is also the basis for OpenID Connect, which provides OpenID (**authentication**) on top of OAuth2 (**authorization**) for a more complete security solution.

OAuth2 provides secure delegated access, meaning that an application, called a *client*, can take actions or access resources on a resource server on the behalf of a *user*, without the user sharing their credentials with the application. OAuth2 does this by allowing tokens to be issued by an identity provider *authorization server* to the third-party applications, with the authorization of the user. The client then uses the token to access the *resource server* on behalf of the user.

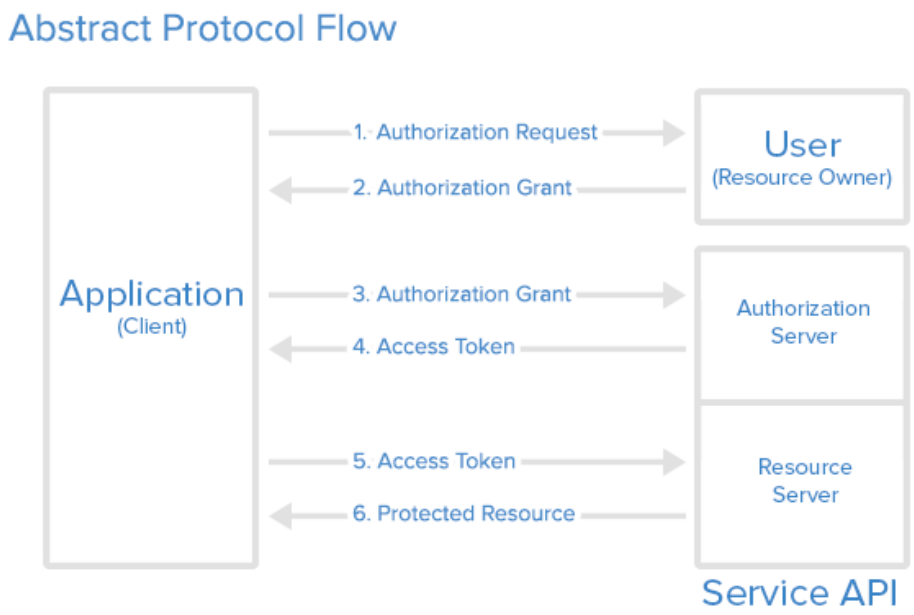


Figure 2: A general picture of the architecture and protocol flow of OAuth.

I have described and built a system called AppNet based on OAuth, which is capable of interconnecting webapplications in a similar way it happens on a mobile phone between installed applications.

2.3 SAML

https://en.wikipedia.org/wiki/SAML_2.0

SAML is the oldest standard of the three, originally developed in 2001.. SAML stands for Security Assertion Markup Language. It's an open standard that provides both authentication and authorization.

Similar to the terminology of the other two standards, SAML defines a *principal*, which is the end user trying to access a resource. There is a *service provider*, which is the web server that the principal is trying to access. And there is an *identity provider*, which is the server that holds the principal's identities and credentials.

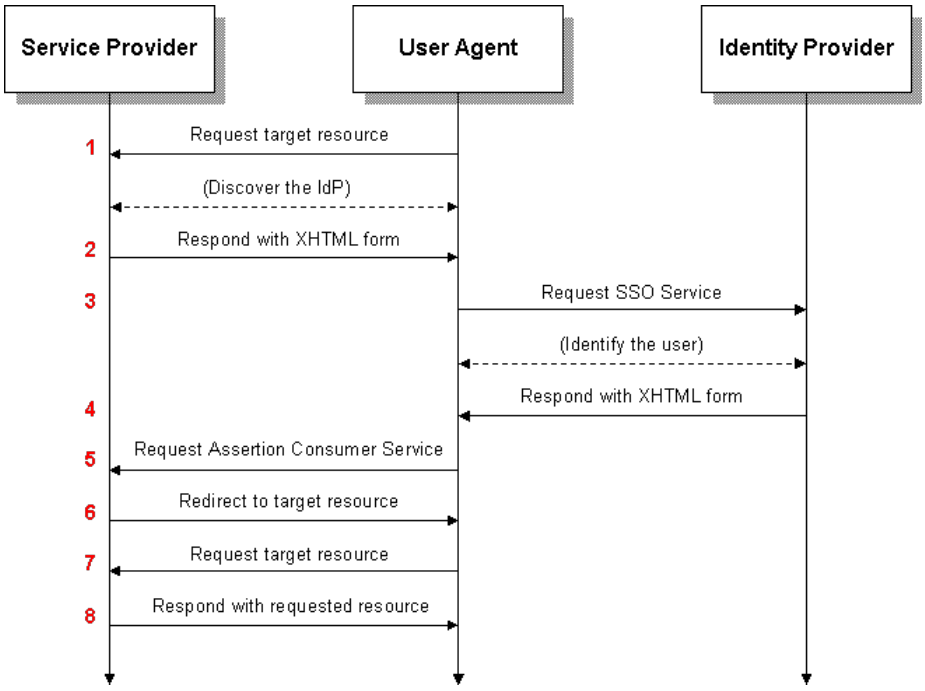


Figure 3: Protocol flow of SAML

In Budapest University of Technology and Economics we use the well-known Shibboleth infrasctructure which is based on SAM as well.

As we can see, today's authentication and authorization systems are centralized. As we will see in the following section, so is the case with one of the most important systems of the internet, the PKI, which manages public keys of real-world actors.

3 Public Key Infrastructrue and Decentralized PKI

Public Key Infrastructure (PKI) is a system designed to manage the creation, distribution, identification, and revocation of public keys. The purpose of a PKI is to facilitate the secure electronic transfer of information for a range of network activities such as, internet banking and confidential email. PKI is an arrangement that binds public keys with respective identities of entities (like people and organizations) or as I called them before real-world actors, because these can also be devices, softwares or anythings that operates with the internet.

The means of this is public key cryptography, a cryptographic technique that enables entities to securely communicate on an insecure public network,

and reliably verify the identity of an entity via digital signatures.

A PKI consists of:

- A Certificate Authority, which acts as *"the root of trust"*. The CA issues digital certificates representing the ownership of a public key.
- Registration Authority, which validates the registration of a digital certificate with a public key. Every time when a request for verification of any digital certificate is made, it goes to RA. The RA then decides whether to authenticate the request or not. RA can also issue certificates for specific use cases depending on the permissions granted to it by CA.
- A certificate database, which issues and revokes certificates.
- A certificate store where issued certificates are stored with their private keys.

Protocols such as SSL and TLS, Email signature and many other authentication and security protocols rely on PKI, so this is a critical point of the Internet.

3.1 Problems with Centralized PKIs

There are a lot of articles on the Internet expressing the factual opinion that there are some big problems with the current PKI.

The centralized PKI such as the CA-based system has its problems and limitations because it relies on a central trusted party. In a centralized PKI system, your identity is defined by trusted third parties the CAs, sometimes private companies and sometimes governments.

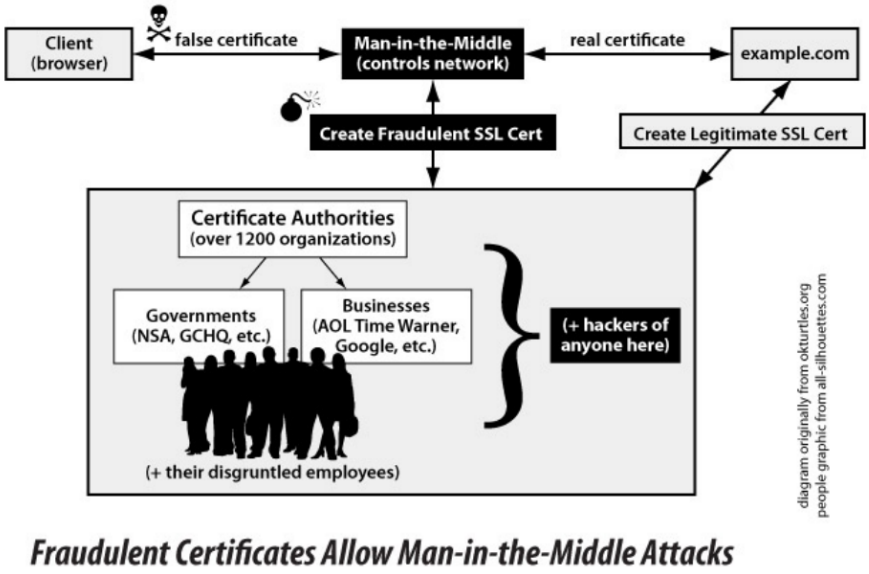


Figure 4: Illustration from DPKI documents

This is a problem because it leaves the door open for attackers to conduct MITM (Man-in-the-Middle) attacks. There are different forms of MITM attacks—ARP spoofing, IP spoofing, DNS spoofing, HTTPS spoofing, and Man in the Browser (MITB), and more. Numerous incidents have already shown that you can increase the risk of MITM attacks when you place too much trust in CAs.

In practice, attackers can trick the CA into thinking they are someone else, or they can go so far as to compromise the CA to get it to issue a rogue certificate. For instance, the that incident happened in 2017 where hackers took control of Brazilian banks DNS server and tricked a CA into issuing a valid certificate to them.

The Internet Engineering Task Force (IETF) responsible for Web PKI itself has created a memo describing current issues of PKI; a group of researchers around Rebooting the Web of Trust assessed its weaknesses in their paper, in agreement that the current implementation of Web PKI has problems that shouldn be dealt with.

3.2 Decentralized PKI (DPKI)

Decentralized Public Key Infrastructure, or DPKI, is an alternative approach to designing better PKI systems. Pretty Good Privacy (PGP), an encryption program developed by Phil Zimmermann, is a decentralized trust system that was created when blockchain didn't exist. However it did not became widespread

for minor issues. But today there is no need for the third-parties. Blockchain is a better approach to build a more competent, secure PKI system.

But how blockchain is going to improve PKI? In decentralized PKI, blockchain acts as a decentralized key-value storage. It is capable of securing the data read to prevent MITM attacks, and to minimize the power of third parties. By bringing the power of blockchain technology to the systems, DPKI resolves the issues with traditional PKI systems.

The decentralized nature of the management framework can tackle the problems with the CA systems through certificate revocation, eliminating single points of failure, and reacting fast to misuses of CAs. Blockchain is able to make the process transparent, immutable, and prevent attackers from breaking in, thus effectively avoiding the MITM attacks. The trust is established and maintained based on consensus protocols. Third-parties, the miners or validators, will have to follow the rules of the protocol, that would financially reward and punish these third-parties to effectively preventing misbehavior in the blockchain and limiting their roles.

Quoted from the Decentralized Public Key Infrastructure a paper from Rebooting the Web of Trust itself:

> The goal of DPKI is to ensure that, unlike PKIX, no single third-party can compromise the integrity and security of the system as as whole. Trust is decentralized through the use of technologies that make it possible for geographically and politically disparate entities to reach consensus on the state of a shared database. DPKI focuses primarily on decentralized key-value datastores, called blockchains, but it is perfectly capable of supporting other technologies that provide similar or superior security properties

Based on the specification also it would possible to recover keys based on key sharding, and many other functionalities, fully compatible with conventional PKI systems.

4 RWOT - Rebooting The Web-Of-Trust and the notion of Digital Identity

In the previous section we mentioned the circle called Rebooting the Web-of-Trust - RWOT. This is a group of researches who aspire to analyse and solve the many problems in the architecture of the internet, similarly to what I am doing in this very research.

It is out of the scope of this paper to talk about all of their work, as they work on many topics, but they research around the topic of Self-Sovereign Identity. Their mission, quoted from their website index:

> This facilitated design workshop ("DesignShop"), hosted by Christopher Allen, is focused on the creation of the next generation of decentralized web-of-trust based identity systems. By the way they

recently had a workshop in Barcelona on March of 2019 https://medium.com/cryptolawreview/live-coverage-8th-rebooting-web-of-trust-rwot8-b9f2f75bd4fc

Some in my opinion important papers (their papers are here):

- Creating the New World of Trust - https://nbviewer.jupyter.org/github/WebOfTrustInfo/rebooting-the-web-of-trust/blob/master/final-documents/whats-the-next-step.pdf
- Decentralized Public Key Infrastructure - https://nbviewer.jupyter.org/github/WebOfTrustInfo/rebooting-the-web-of-trust/blob/master/final-documents/dpki.pdf
- Identity Crisis - https://github.com/WebOfTrustInfo/rwot2-id2020/blob/master/final-documents/identity-crisis.pdf
- Introduction to DID Auth https://github.com/WebOfTrustInfo/rwot6-santabarbara/blob/master/final-documents/did-auth.md And specifications that became part of the W3C:
- Verifiable Credentials Data Model - https://www.w3.org/TR/verifiable-claims-data-model/
- Decentralized Identifiers (DIDs) -https://w3c-ccg.github.io/did-spec/ This is what we are going to talk about in the following section, as this is the other gem that is going to be the future of the internet, in my opinion. Here is how and why.

4.1 Decentralized Identifiers (DIDs)

For the extensive specification: https://w3c-ccg.github.io/did-spec/ A quote from Wikipedia:

> A digital identity is information on an entity used by computer systems to represent an external agent. That agent may be a person, organization, application, or device. ISO/IEC 24760-1 defines identity as "set of attributes related to an entity". https://en.wikipedia.org/wiki/Digital_identity

DIDs propose to use blockchain to register identifiers that users can use to identify themselves.

Here I will present DIDs at a high level for the reader. We start off with an overview of DIDs, DID Documents, Verifiable Claims and DIDAuth—to see how the technology works. Then we see the economics of DIDs to try and understand what problems they propose to solve, for whom, and how they go about solving them.

First of all a user can create a new DID at any time and for any reason. A DID is two things: a unique identifier and an associated DID Document. The unique identifier looks something like "did:example:123456789abcdefghi" and it's safe enough to think about that as the unique ID for looking up a DID document. A DID Document is a JSON-LD object that is stored in some central location so that it can be easily looked up. The DID is expected to be

"persistent and immutable" so that it is outside of the influence of anyone other than it's owner.

The DID Document must include a DID. It might include things like:

- a timestamp of when it was created
- a cryptographic proof that the DID Document is valid
- a list of cryptographic public keys
- a list of ways that the DID can be used to authenticate
- a list of services where the DID can be used
- any number of externally defined extensions

EXAMPLE 2: Minimal self-managed DID Document

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",
  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }],
  "authentication": [{
    // this key can be used to authenticate as DID ...9938
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:example:123456789abcdefghi#keys-1"
  }],
  "service": [{
    "type": "ExampleService",
    "serviceEndpoint": "https://example.com/endpoint/8377464"
  }]
}
```

Figure 5: An example DID Document (from the W3C DID Specification)

Note that there is no personally identifiable information in the DID Document: no username, no address, no phone number. That comes through Verifiable Claims.

4.2 Verifiable Claims

Verifiable claims is a way of how DIDs could be used. There are typically three pieces to Verifiable Claims:

- **Subject**: probably a user (like you), but it could be a company, a pet, or anything else that can be described
- **Issuer**: probably an organization of some sort, like Budapest University of Technology

- **Claim**: any statement that can be made, usually the examples are about people and include things like "my university average is above 4.5" (it's not :(..)

EXAMPLE 3: A simple verifiable claim

```
{
  "@context": "https://w3id.org/security/v1",
  "id": "http://example.gov/credentials/3732",
  "type": ["Credential", "ProofOfAgeCredential"],
  "issuer": "https://dmv.example.gov",
  "issued": "2010-01-01",
  "claim": {
    "id": "did:example:ebfeb1f712ebc6f1c276e12ec21",
    "ageOver": 21
  },
  "revocation": {
    "id": "http://example.gov/revocations/738",
    "type": "SimpleRevocationList2017"
  },
  "signature": {
    "type": "LinkedDataSignature2015",
    "created": "2016-06-18T21:19:10Z",
    "creator": "https://example.com/jdoe/keys/1",
    "domain": "json-ld.org",
    "nonce": "598c63d6",
    "signatureValue": "BavEll0/I1zpYw8XNi1bgVg/sCneO4Jugez8RwDg/+
    MCRVpjOboDoe4SxxKjkCOvKiCHGDvc4krqi6Z1n0UfqzxGfmatCuFibcC1wps
    PRdW+gGsutPTLzvueMWmFhwYmfIFpbBu95t501+rSLHIEuujM/+PXr9Cky6Ed
    +W3JT24="
  }
}
```

Figure 6: An example Verifiable Claim (from the W3C Verifiable Claims specification)

Note that a Verifiable Claim is just a JSON document and the specification only defines the data model. There is no specified protocol for how to document gets from one party to the next. The Verifiable Claims specification defines two more roles:

- **Holder**: Someone that receives and holds on to a Verifiable Claim, probably a user (like you) that got a Verifiable Claim from some issuer.
- **Inspecter-Verifier**: Probably a service of some sort, like Facebook or a Scholarship organization, that receives Verifiable Claims and uses them as part of their service.

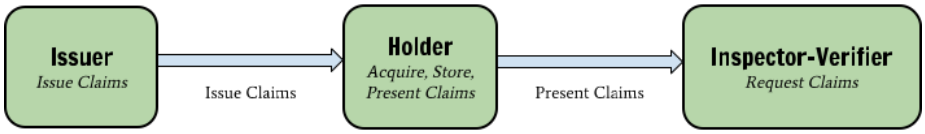


Figure 7: The various actors in Verifiable Claims (from the W3C Verifiable Claims specification)

Perhaps an example would help clarify all these terms. Let's say that I would like to apply to Erasmus, but I need to verify that my average is at least 4.5. I will use a Verifiable Claim from the BME (the Issuer) that says Najib Ghadri (the subject) has at least 4.5 average (the claim). I will download that Verifiable Claim document from BME (I am now the holder) and upload it to the Scholarship organization (the Issuer-Verifier). The organization will validate the claim and then allow me to go to Erasmus and register me.

Like putting together LEGO pieces we could see that the Issuer of the Verifiable Claim used a DID for the URI, enable the Issuer-Verifier to resolve that DID to a DID Document containing their public key.

How does the Issuer know that the Holder has the right to request a claim? And how does the Inspector-Verifier know that the Holder is associated with the ID in the claim? For that there is DID-based Authorization, known as DIDAuth.

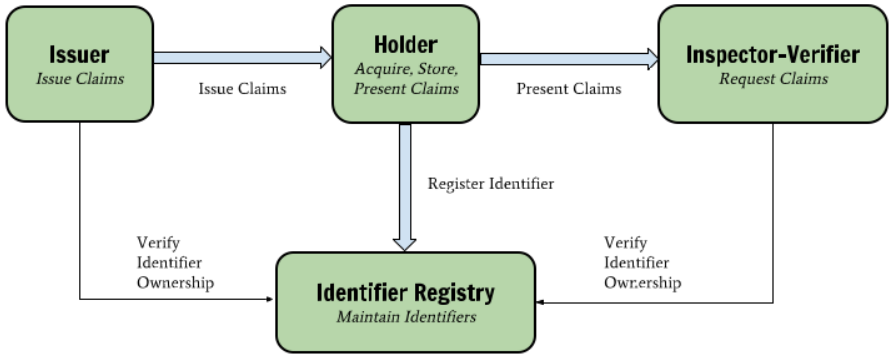


Figure 8: Identity ownership is required for Verifiable Claims (from the W3C Verifiable Claims specification)

At it's heart, DIDAuth is a challenge-response authentication protocol: the service that you are logging in to sends a random challenge, which you sign with your private key, and then send the challenge, the signature, and DID back to the server.

From where do they get the public key of your id? From the DID blockchain as previously mentioned.

Of course the Verifiable claims is an unnecessary part of the system, but one that could operate. It is all about making sure that a piece of information provided by a party is related to a certain identity behind an identifier. But this is the same thing as in OAuth or SAML. If a third-party (Inspector-Verifier) want to verifiy or actuall access a piece of information, they ask for permission from you, then through the Issuer they get the information. This is basically the flow happening here.

4.3 Conclusion

Implementations of the above idea, DID, are already existing, one of them, promising, namely Hyperledger Indy, which is part of a bigger framework initiated by the Linux Foundation. In this semester work I did not have the time to dive in deeper into that framework.

Now we have replacements for PKI, OAuth/SAML, and LDAP systems that are secure and without central point of failure.

5 Blockchain technologies

The distributed ledger technology is a disruptive technology today. We have Bitcoin a billion dollar phenomenon, many "Altcoins" that also strive to make money, and lately application based on blockchains. Among them most known the Ethereum, a decentralized secure computer, Hyperledger a framework to build blockchain application. Many are still echoing that this technology is the future of everything in the Web. This might be close to the truth; we have seen in the previous section that blockchains indeed can be a solution to decentralized trust when using them for DIDs, DPKI. But can it be more than that? Can we build whole applications on top of blockchains?

Here I will talk mostly about the main mathematical problem and it's solutions, the problem os consensus in a distributed system. Then I will show some different applications of blockchains.

5.1 Byzantine Fault Tolerance

Blockchains are inherently decentralized systems which consist of different actors who act depending on their incentives and on the information that is available to them.

Whenever a new transaction gets broadcasted to the network, nodes have the option to include that transaction to their copy of their ledger or to ignore it. When the majority of the actors which comprise the network decide on a single state, **consensus** is achieved.

> A fundamental problem in distributed computing and multi-agent systems is to achieve overall system reliability in the presence of a

> number of faulty processes. This often requires processes to agree on some data value that is needed during computation.[19]

These processes are described as consensus. - What happens when an actor decides to not follow the rules and to tamper with the state of his ledger? - What happens when these actors are a large part of the network, but not the majority?

In order to create a secure consensus protocol, it must be *fault tolerant.*

Now, I will talk briefly about the *Byzantine Generals' Problem and discuss Byzantine Fault Tolerance* in distributed and decentralized systems. Then, we will discuss how this relates to blockchain thecnologies.

5.2 The Byzantine Generals Problem

We have to know that this problem is an extension of the *Two Generals Problem* [20], which describes a scenario where two generals are attacking a common enemy. General 1 is considered the leader and the other is considered the follower. Each general's army on its own is not enough to defeat the enemy army successfully, thus they need to cooperate and attack at the same time. This seems like a simple scenario, but there is one caveat:

In order for them to communicate and decide on a time, General 1 has to send a messenger across the enemy's camp that will deliver the time of the attack to General 2. However, there is a possibility that the messenger will get captured by the enemies and thus the message won't be delivered. That will result in General 1 attacking while General 2 and his army hold their grounds.

Even if the first message goes through, General 2 has to *acknowledge (ACK,* notice the similarity to the 3-way handshake of TCP *)* that he received the message, so he sends a messenger back, thus repeating the previous scenario where the messenger can get caught. This extends to infinite ACK's and thus the generals are unable to reach an agreement.

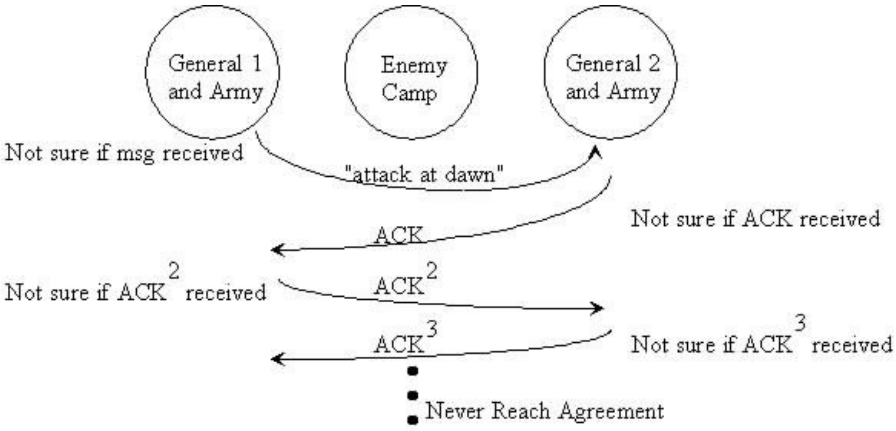


Figure 9: Since the possibility of the message not getting through is always > 0, the generals can never reach an aggrement with 100% confidence.

The Two Generals Problem has been proven to be **unsolvable.**

Following to Byzantine Generals Problem, described in 1982 by Lamport, Shostak and Pease, it is a generalized version of the Two Generals Problem with a twist.

It describes the same scenario, where instead more than two generals need to agree on a time to attack their common enemy. The added complication here is that one or more of the generals can be a traitor_,_ meaning that they can lie about their choice (e.g. they say that they agree to attack at 0900 but instead they do not).

The leader-follower paradigm described in the Two Generals Problem is transformed to a commander-lieutenant setup. In order to achieve consensus here, the commander and every lieutenant must agree on the same decision (for simplicity *attack* or *retreat).*

The algorithm to reach consensus in this case is based on the value of majority of the decisions a lieutenant observes. The algorithm is as seen below.

Algorithm OM(0).

(1) The commander sends his value to every lieutenant.
(2) Each lieutenant uses the value he receives from the commander, or uses the value RETREAT if he receives no value.

Algorithm OM(m), $m > 0$.

(1) The commander sends his value to every lieutenant.
(2) For each i, let v_i be the value Lieutenant i receives from the commander, or else be RETREAT if he receives no value. Lieutenant i acts as the commander in Algorithm OM($m-1$) to send the value v_i to each of the $n-2$ other lieutenants.
(3) For each i, and each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step (2) (using Algorithm OM($m-1$)), or else RETREAT if he received no such value. Lieutenant i uses the value *majority*($v_1, \ldots, v_{n-1}$).

Figure 10: The algorithm, 7th page of LSP1982

The important thing to remember is that the goal is for the **majority** of the lieutenants to choose the **same** decision, not a specific one. For a good explanation and a hands on explanation I suggest this article: https://marknelson.us/posts/2007/07/23/byzantine.html

Byzantine Fault Tolerance is the characteristic which defines a system that tolerates the class of failures that belong to the Byzantine Generals' Problem. How does this all relate to blockchain?

Blockchains are decentralized ledgers which, by definition, are not controlled by a central authority. Due to the value stored in these ledgers, bad actors have huge economic incentives to try and cause faults. So a solution to the Byzantine Generals' Problem for blockchains is necessary. In the absence of such a countermeasure, a peer is able to transmit and post false transactions removing the blockchain's reliability. To make things worse, there is no central authority to take over and repair the damage. However the BFT algorithm is not enough, and its variations have constraints, namely that for correct functioning at most less than a third of the network can be dishonest only.

The big breakthrough when Bitcoin was invented, was the use of Proof-of-Work as a probabilistic solution to the Byzantine Generals Problem.

Blockchains use consensus algorithms to elect a leader who will decide the contents of the next block. That leader is also responsible for broadcasting the block to the network, so that the other peers can verify the validity of its contents.

5.3 Proof of Work (PoW)

This is the most popular algorithm being used by currencies such as Bitcoin and Ethereum, each one with its own differences. We have to know about hash functions to understand the basic idea of the algorithm, which is not so complicated. Shortly put, a hash function is secure enough if changing one bit in the input, will likely change half the bits in the outputs relative to the original input.

```
Example (a character is more the one bit of course):
sha1("hello") = aaf4c61ddcc5e8a2dabede0f3b482cd9aea9434d
sha1("hello1") = 88fdd585121a4ccb3d1540527aee53a77c77abb8
```

In Proof of Work, in order for an actor to be elected as a leader and choose the next block to be added to the blockchain they have to find a solution to a particular mathematical problem. Let that mathematical problem be:

> Given data X, find a number n such as that the hash of n appended to X results is a number less than Y.

```
Example - hash is a hypothetical hash function
Y = 10, X = 'test'
hash(X) = hash('test') = 0x0f = 15 > 10
hash(X+1) = hash('test1') = 0xff = 255 > 10
hash(X+2) = hash('test2') = 0x09 = 9 < 10 OK, Solved.
```

Given that the hash function used is cryptographically secure the only way to find a solution to that problem is by bruteforce (trying all possible combinations). In other words, probabilistically speaking, the actor who will solve the aforementioned problem first the majority of the time is the one who has access to the most computing power. These actors are also called **miners**

It has been widely successful primarily due to its following properties:

1. It is hard to find a solution for that given problem
2. When given a solution to that problem it is easy to verify that it is correct

In other words it is an NP problem if not NP-hard or NP-complete. Due to the limited supply of computational power, miners are also incentivized not to cheat. Attacking the network would cost a lot because of the high cost of hardware, energy, and potential mining profits missed.

5.4 Proof of Stake (PoS)

Proof of Stake takes away the energy and computational power requirement of PoW and replaces it with stake. Stake is referred to as an amount of currency that an actor is willing to lock up for a certain amount of time. In return, they get a chance proportional to their stake to be the next leader and select the next block. There are various existing coins which use pure PoS, such as Nxt and Blackcoin. The main issue with PoS is the so-called nothing-at-stake problem. Essentially, in the case of a fork, stakers are not disincentivized from staking in both chains, and the danger of double spending problems increase.

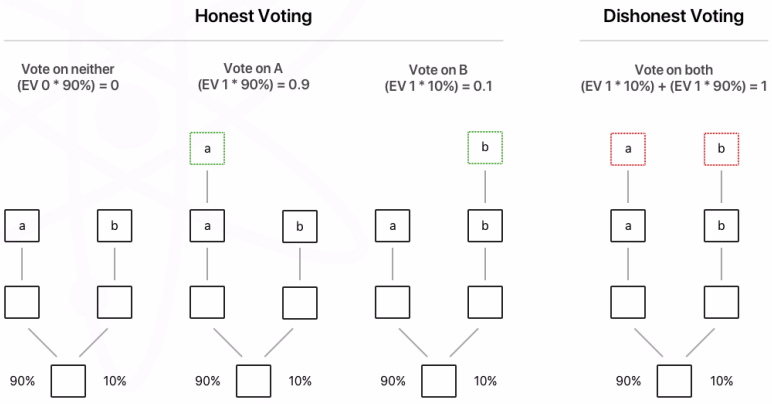


Figure 11: Expected value for voting on competing chains is greater than expected value for voting on a single chain in naive PoS design.

PoS consensus algorithms are still researched, both Ethereum and Bitcoin. However in Ethereum 2.0 it is claimed to use PoS.

https://blog.cosmos.network/consensus-compare-casper-vs-tendermint-6df154ad56ae
https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/

5.5 Delegated Proof of Stake (DPoS)

DPoS is nothing else than PoS but restricted to a limited number of block producer nodes. DPoS is a system in which a fixed number of elected entities (called *block producers* or *witnesses*) are selected to create blocks in a round-robin order. Block producers are voted into power by the users of the network, who each get a number of votes proportional to the number of tokens they own on the network (their *stake*). However DPoS is a protocol that sacrifices decentralization for throughput due to the low number of block producers.

There is a trilemma that claims that blockchain systems can only at most have two of the following three properties:

- **Decentralization** (defined as the system being able to run in a scenario where each participant only has access to O(c) resources, i.e. a regular laptop or small VPS)
- **Scalability** (defined as being able to process O(n) > O(c) transactions)
- **Security** (defined as being secure against attackers with up to O(n) resources)

Phrased by Vitalik Buterin here: https://github.com/ethereum/wiki/wiki/Sharding-FAQ

For applications like a social network or game, every comment or in-game movement doesn't require the full security of a fully decentralized network, but they do require high throughput. For those, DPoS seems like a good fit.

5.6 Ethereum

https://github.com/ethereum/wiki/wiki/White-Paper

The Ethereum blockchain is essentially a **transaction-based state machine**. In computer science, a *state machine* refers to something that will read a series of inputs and, based on those inputs, will transition to a new state.

To cause a transition from one state to the next, a transaction must be valid. **For a transaction to be considered valid, it must go through a validation process known as mining.** Mining is when a group of nodes (i.e. computers) expend their compute resources to create a block of valid transactions. The same as we have seen previously. The Ethereum also works on PoW, for now.

On Ethereum developers can deploy *Smart Contracts*, which are codes that use the Ethereum Virtual Machine. With these Smart Contracts we can build simple logic that makes application logic possible. Hence we can deploy truly decentralized applications.

5.7 Sidechains

This takes us to the notion of Sidechains. The first viral DApp on Ethereum was enough to bring the network to its knees. It was not scaleable. The number of pending transactions reached all time highs and transaction fees also reached absurd amounts, with one user also (mistakenly) paying $11.000 in transaction fees. The need for scaling solutions is becoming more urgent every day, as transaction fees and the transaction backlog increase.

The term "sidechains" was first described in the paper "Enabling Blockchain Innovations with Pegged Sidechains". The paper describes "two-way pegged sidechains", a mechanism where by proving that you "locked" some coins that were previously in your possession, you were allowed to move some other coins within a sidechain.

A sidechain is defined by a custom "rule-set" and can be used to offload computations from another chain. Individual sidechains can follow different sets of rules from the mainchain, which means they can optimize for applications that require extremely high speeds or heavy computation, while still relying on the mainchain for issues requiring the highest levels of security.

6 Security concerns with DNS and prospects

Domain Name Servers are distributed in nature but all have to rely on servers to get the right information. This convoluted process could allow a query to be hijacked.

There are several ways for your query to be hijacked:

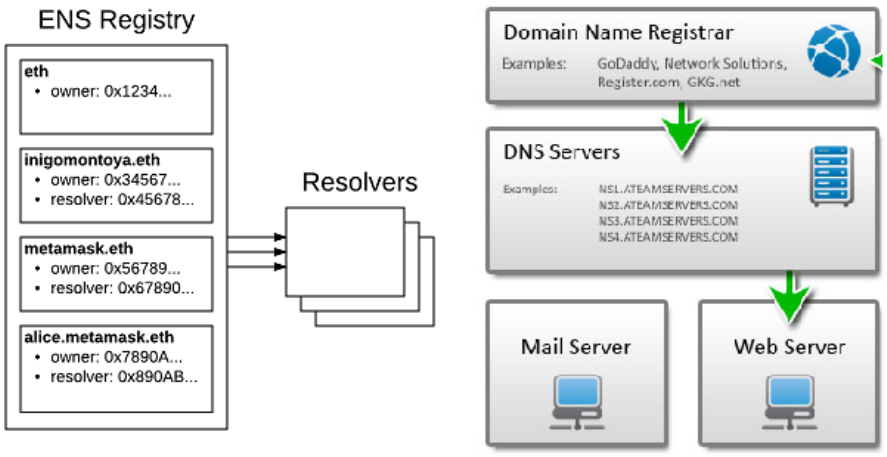
1. The first is when a hijacker takes control of one or more of the authoritative DNS servers for a domain.
2. A second way is called cache poisoning attack, which essentially injects malicious data to the recursive server.
3. A third way is to take over the registration of the domain itself and reroutes users.
4. A fourth way is Distributed Denial of Service (DDoS) which can turn small queries into much larger ones, taking down the user's server.

The attacks above could essentially reroute a user to a site that the hijacker controls.

What can we do about it? As we have seen before in the DIDs and DPKI there is the obvious solution of using blockchains as a tool of decentralization. On blockchains we could store DNS data, such as hosts.txt and every DNS record type that we have today the same way. There are already implementations for this namely two more significant: Ethereum Name Service which resolves and records domains under the .eth tld, and the other Namecoin which manages the .bit tld.

6.1 ENS

https://ens.domains/ DNS is similar to ENS which converts machine-readable address to human-readable address. IP addresses would be replaced by Ethereum addresses, IPFS/Swarm hashes. The architecture between both DNS and ENS are different due to the nature of the Ethereum blockchain. As you can see below, ENS significantly reduces its architecture by allowing the smart contract to be the registrar instead of Go daddy. By reducing this function to a smart contract anyone can create a subdomain based on the rules of the registrar. Finally, the resolver acts as a translator that converts names to hashes and addresses.



In order to use any Ethereum DApp, such as ENS, we need to access the Ethereum blockchain, even as clients. Since the browsers are based on conventional DNS and HTTP, we need to alter the browser. For that we use plugins, and for our case MetaMask is the plugin that lets us connect to the Ethereum blockchain. In order to manage the names we also need to have the MetaMask wallet, however we can inspect the free domain names here: https://manager.ens.domains/

6.2 NameCoin

https://www.namecoin.org/ Namecoin is a blockchain system and cryptocurrency which allows users to protect domain name servers (DNS) by embedding them on a distributed ledger.

The stated purpose of Namecoin is to protect free speech by making the Internet resistant to censorship. It accomplishes this by providing decentralized DNS domains (.bit) which are not controlled by any entity. It is promoted and developed by the Namecoin Foundation.

The .bit domain is served via the cryptocurrency Namecoin infrastructure, which acts as an alternative, decentralized domain name system. Use of the .bit domain requires a copy of the Namecoin blockchain, a supporting public DNS server, or a web browser plug-in. In February 2014, a beta version of a Windows/Linux Firefox plug-in called FreeSpeechMe was released that allows automated resolution of .bit addresses, by downloading the Namecoin block chain and running it in the background. This is similar to ENS where we also had to bypass the browsers way of doing DNS queries, and instead downloading the MetaMask plugin to use any Ethereum DApp.

In order to manage the NameCoin names, we have to use the NameCoin wallet that looks like this:

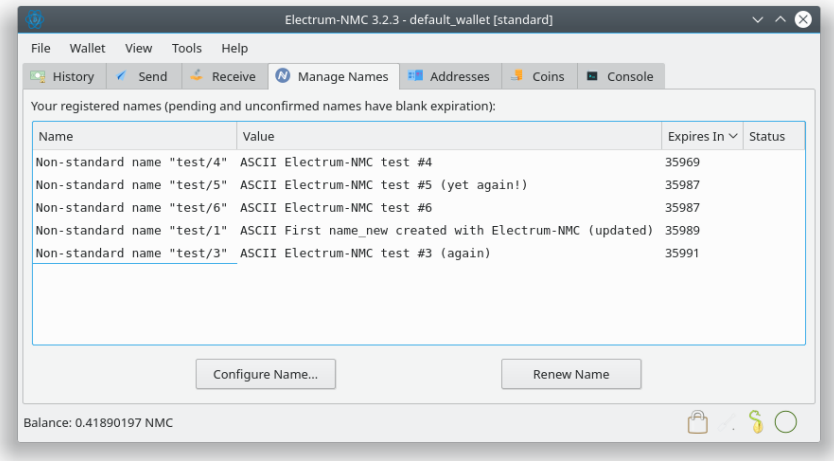


Figure 12: NameCoin wallet

7 IPFS - InterPlanetary File System

https://protocol.ai/projects/

Whitepaper: https://ipfs.io/ipfs/QmV9tSDx9UiPeWExXEeH6aoDvmihvx6jD5eLb4jbTaKGps

InterPlanetary File System (IPFS) is a protocol and network designed to create a content-addressable, peer-to-peer method of storing and sharing hypermedia in a distributed file system. Similar to a torrent, IPFS allows users to not only receive but host content. As opposed to a centrally located server IPFS is built around a decentralized system of user-operators who hold a portion of the overall data, creating a resilient system of file storage and sharing.

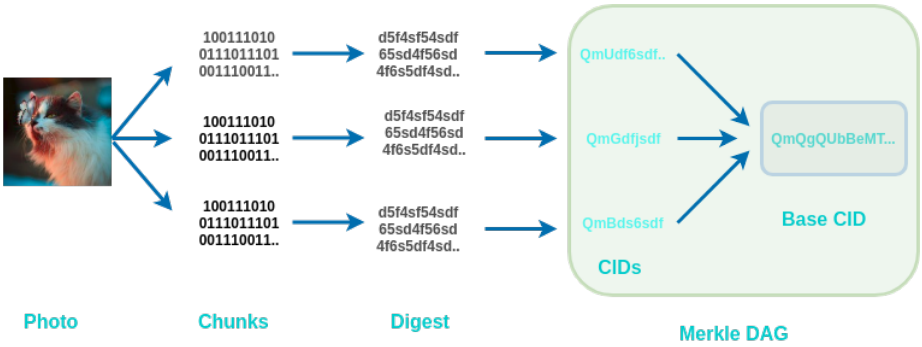
IPFS is a peer-to-peer distributed file system that seeks to connect all computing devices with the same system of files. IPFS could be seen as a single BitTorrent swarm, exchanging objects within one Git repository. In other words, IPFS provides a high-throughput, content-addressed block storage model, with content-addressed hyperlinks.

When we add the photo to IPFS, this is what happens:

The final hash for example: `QmQgQUbBeMTnH1j3QWwNw9LkXjpWDJrjyGYfZpnPp8x5Lu`
When we added the image, we converted the image into the *Raw* data which computer can understand. Now we make it *content-addressable* by using hash functions.

We pass the *Raw* image data into SHA256 hash function and get the unique Digest. Now, we need to convert this Digest into a CID(Content Identifier). This CID is what IPFS will search for when we try to get back the image. This shows its *tamper-proof* property, hence making IPFS a *Self-certifying File System*.

Well, now we have added our photo to IPFS, but this was not the whole story. What is actually happening is something like this:



If the files are bigger than 256 kB, then they are broken down into smaller parts, so that all the part are equal or smaller than 256 kb. We can see the chunks of our photo using this command:

`ipfs object get Qmd286K6pohQcTKYqnS1YhWrCiS4gz7Xi34sdwMe9USZ7u`

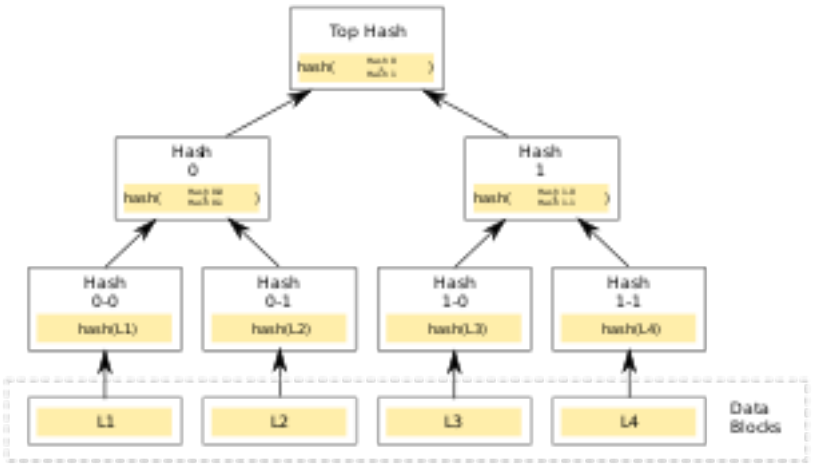
This gives us 15 chunks, each of which is smaller than 256kb. Each of these chunks is first converted into a digest and then into CIDs: The JSON-LD below is also an example for InterPlanetary Linked Data.

```
{
  "Links": [
    {
      "Name": "",
      "Hash": "QmZ5RgT3jJhRNMEgLSEsez9uz1oDnNeAysLLxRco8jz5Be",
      "Size": 262158
    },
    {
      "Name": "",
      "Hash": "QmUZvm5TertyZagJfoaw5E5DRvH6Ssu4Wsdfw69NHaNRTc",
      "Size": 262158
    },
    {
      "Name": "",
```

```
  "Hash": "QmTA3tDxTZn5DGaDshGTu9XHon3kcRt17dgyoomwbJkxvJ",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmXRkS2AtimY2gujGJDXjSSkpt2Xmgog6FjtmEzt2PwcsA",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmVuqvYLEo76hJVE9c5h9KP2MbQuTxSFyntV22qdz6F1Dr",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmbsEhRqFwKAUoc6ivZyPa1vGUxFKBT4ciH79gVszPcFEG",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmegS44oDgNU2hnD3j8r1WH8xZ2RWfe3Z5eb6aJRHXwJsw",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmbC1ZyGUoxZrmTTjgmiB3KSRRXJFkhpnyKYkiVC6PUMzf",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmZvpEyzP7C8BABesRvpYWPec2HGuzgnTg4VSPiTpQWGpy",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmZhzU2QJF4rUpRSWZxjutWz22CpFELmcNXkGAB1GVb26H",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmZeXvgS1NTxtVv9AeHMpA9oGCRrnVTa9bSCSDgAt52iyT",
  "Size": 262158
},
{
  "Name": "",
  "Hash": "QmPy1wpe1mACVrXRBtyxriT2T5AffZ1SUkE7xxnAHo4Dvs",
```

```
      "Size": 262158
    },
    {
      "Name": "",
      "Hash": "QmcHbhgwAVddCyFVigt2DLSg8FGaQ1GLqkyy5M3U5DvTc6",
      "Size": 262158
    },
    {
      "Name": "",
      "Hash": "QmNsx32qEiEcHRL1TFcy2bPvwqjHZGp62mbcVa9FUpY9Z5",
      "Size": 262158
    },
    {
      "Name": "",
      "Hash": "QmVx2NfXEvHaS8uaRTYaF4ExeLaCSGpTSDhhYBEAembdbk",
      "Size": 69716
    }
  ],
  "Data": "\b\u0002\u0018\u0001 \u0010 \u0010 \u0010 \u0010 \u0010 \u0010 \u0010 \u00
}
```


Merkle DAG In cryptography and computer science, a hash tree or Merkle tree is a tree in which every leaf node is labelled with the hash of a data block, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Hash trees allow efficient and secure verification of the contents of large data structures. Hash trees are a generalization of hash lists and hash chains.



IPFS uses IPLD Merkle DAG, or directed acyclic graph for managing all the

chunks and linking it to the base CID.

IPLD (objects) consist of 2 components:

- `Data`—blob of unstructured binary data of size < 256 kB.
- `Links`—array of Link structures. These are links to other IPFS objects.

Every IPLD Link(in our case the 15 links that we got above) has 3 parts:

- `Name`—name of the Link
- `Hash`—hash of the linked IPFS object
- `Size`—the cumulative size of linked IPFS object, including following its links

IPLD is built on the ideas of Linked Data that was conceptualized in the ideas of Web 3.0

As we said before, if we change the content of the input, we get a different digest, which will mean that the final "Base CID" will be different. So chainging the content of the website changes the address of it. Everybody who has the link to the previous website will not be able to see the new website

To address this problem, IPFS uses InterPlanetary Naming System(IPNS). Using IPNS links point to CIDs. If we want to update my website CID, we will just point a new CID to the corresponding IPNS link (similar to DNS).

```
ipfs name publish QmYVd8qstdXtTd1quwv4nJen6XprykxQRLo67Jy7WyiLMB
```

This can take up to a few minutes. You will get an output like this:

```
Published to Qmb1VVr5xjpXHCTcVm3KF3i88GLFXSetjcxL7PQJRviXSy: /ipfs/QmYVd8qstdXtTd1quwv4nJen6
```

Now, if we want to add an updated CID, we will just use the same command

```
ipfs name publish <new_CID>
```

IPFS as a replacement for HTTP protocol. For example if there are more client who want the same data, instead every client asking the same server multiple times for the same resource, the clients could share the data with each other like in a torrent system.

HTTP also presents a big problem if there is some problem in the networks and the client is unable to reach the server. This can happen if by netwrok problems, 404 problems, or even in censorship situations.

The location-based addressing model of HTTP is based on centralization. With IPFS and its part Libp2p peers can discover each other on a netwrok and and using the IPFS protocol exchange data with signinficant speed and load improvement. Hence;

- IPFS is content-addressable. The data on IPFS is identified using CIDs.
- These CIDs are unique with respect to the data referenced by it.
- IPLD also adds De-duplication property to IPFS.
- IPFS uses IPNS for linking CIDs to a fixed IPNS link, analogous to today DNS of the centralized Internet.
- IPFS uses Libp2p to communicate data and discovering other peers

7.1 IPLD - InterPlanetary Linked Data

https://github.com/ipld/ipld

The Semantic Web or Linked Data is a term coined by Sir Tim Berners Lee in a seminal article published in Scientific American in 2001. Berners Lee articulated a vision of a World Wide Web of data that machines could process independently of humans, enabling a host of new services transforming our everyday lives. While the paper's vision of most web pages containing structured data that could be analyzed and acted upon by software agents has NOT materialized, the Semantic Web has emerged as a platform of increasing importance for data interchange and integration through the growing community implementing data sharing using international semantic web standards, called Linked Data.

We will see more examples in the section of DID Documents and Verifable Claims in Rebooting the Web-Of-Trust, where the DID documents follow a scheme of IPLD.

7.2 IPNS - InterPlanetary Name System

https://docs.ipfs.io/guides/concepts/ipns/

We have seen an example above of how IPNS works on the hand. As we have seen IPFS produces *immutable links*, which is not ideal for the web world, where resources with URIs get updated often. Here is where IPNS comes in.

By using IPNS you can generate a mutable link, which:

- will be human-readable and easy to remember.
- points to the latest version of your website, profile photo, video etc.

A name in IPNS(the hash follows `/ipns/` in a link) is the hash of a public key. It is associated with a record containing information about the hash it links to that is signed by the corresponding private key. New records can be signed and published at any time.

IPNS can be implemented in many ways, but its current implementation uses Distributed Hash Table (DHT). As a consequence, only the most recent mapping of each URI to its corresponding hash is available for resolution, forgetting any historical mappings. This is not good from the archival perspective as the previous versions of a file might still exist in the IPFS store, but their corresponding URI mappings are lost.

7.3 FileCoin

https://filecoin.io/

Filecoin is an open-source, public, cryptocurrency and digital payment system intended to be a blockchain-based cooperative digital storage and data retrieval method. It is made by Protocol Labs and builds on top of InterPlanetary File System, allowing users to rent unused hard drive space.
There are 3 types of entities in the Filecoin system:

- Client—Pay to store data and to retrieve data in the DSN, via Put and Get requests
- Miner—Storage Miners provide data storage to the network. Storage Miners participate in Filecoin by ordering their disk space and serving Put requests. To become Storage Miners, users must pledge their storage by depositing collateral proportional to it.
- Retrieval Miner—Retrieval Miners provide data retrieval to the Network. Retrieval Miners participate in Filecoin by serving data that users request via Get. Unlike Storage Miners, they are not required to pledge, commit to store data, or provide proofs of storage.

FileCoin provides a data storage and retrieval service via a network of independent storage providers that does not rely on a single coordinator, where: (1) clients pay to store and retrieve data, (2) Storage Miners earn tokens by ordering storage (3) Retrieval Miners earn tokens by serving data.

The protocol uses *Proof-of-Replication* (PoRep), a novel Proof-of-Storage which allows a server (i.e. the prover P) to convince a user (i.e. the verifier V) that some data D has been replicated to its own uniquely dedicated physical storage, and *Proof-of-Spacetime*, where a verifier can check if a prover is storing her/his outsourced data for a range of time.

FileCoin builds on multiple novel and previous proof techniques related to Proof-Of-Storage among them Proof-of-Replication, most well described in this paper: https://eprint.iacr.org/2018/678.pdf

Broadly speaking, proofs of storage are interactive protocols between a server (prover) and client (verifier) with different goals relating to statements about the server's storage.

These proofs are never 100 percent proofs, rather they are $\epsilon-Nashequilibrium$ proofs, so a rational prover/node/player does not lose so much by following the expected behaviour of a system's rules. This is just like in any Proof-of-... concept!

For example;

Provable Data Possession (PDP) scheme sallow user V to send data D to server P, and later V can repeatedly check whether P is still storing D. PDPs are useful in cloud storage and other storage outsourcing settings. PDPs can be either privately-verifiable or publicly-verifiable, and static or dynamic. A wide variety of PDP schemes exist. The simplest PDP based on a hash function and precalculating k calls then being able to prove the posession at P server by making the k calls with the k different keys then comparing with the precomputed expectations. This has the obvious disadvantage of being limited to k proofs, however as said before there are many, still on-going improvements; Proof-of-Storage techniques: https://filecoin.io/proof-of-replication.pdf

8 BitTorrent P2P protocol

BitTorrent is a communication protocol for peer-to-peer file sharing (P2P) which is used to distribute data and electronic files over the Internet. The protocol is

well described in the BEP - BitTorrent Enchancement Proposal published on bittorrent.org: http://www.bittorrent.org/beps/bep_0005.html.

BitTorrent uses a "distributed sloppy hash table" (DHT) for storing peer contact information for "trackerless" torrents. In effect, each peer becomes a tracker. The protocol is based on Kademila and is implemented over UDP.

Please note the terminology used in this document to avoid confusion. A "peer" is a client/server listening on a TCP port that implements the BitTorrent protocol. A "node" is a client/server listening on a UDP port implementing the distributed hash table protocol. The DHT is composed of nodes and stores the location of peers. BitTorrent clients include a DHT node, which is used to contact other nodes in the DHT to get the location of peers to download from using the BitTorrent protocol.

8.1 Overview

Each node has a globally unique identifier known as the "node ID." Node IDs are chosen at random from the same 160-bit space as BitTorrent infohashes. A "distance metric" is used to compare two node IDs or a node ID and an infohash for "closeness." Nodes must maintain a routing table containing the contact information for a small number of other nodes. The routing table becomes more detailed as IDs get closer to the node's own ID. Nodes know about many other nodes in the DHT that have IDs that are "close" to their own but have only a handful of contacts with IDs that are very far away from their own.

In Kademlia, the distance metric is XOR and the result is interpreted as an unsigned integer. distance(A,B) = |A xor B| Smaller values are closer.

When a node wants to find peers for a torrent, it uses the distance metric to compare the infohash of the torrent with the IDs of the nodes in its own routing table. It then contacts the nodes it knows about with IDs closest to the infohash and asks them for the contact information of peers currently downloading the torrent. If a contacted node knows about peers for the torrent, the peer contact information is returned with the response. Otherwise, the contacted node must respond with the contact information of the nodes in its routing table that are closest to the infohash of the torrent. The original node iteratively queries nodes that are closer to the target infohash until it cannot find any closer nodes. After the search is exhausted, the client then inserts the peer contact information for itself onto the responding nodes with IDs closest to the infohash of the torrent.

The return value for a query for peers includes an opaque value known as the "token." For a node to announce that its controlling peer is downloading a torrent, it must present the token received from the same queried node in a recent query for peers. When a node attempts to "announce" a torrent, the queried node checks the token against the querying node's IP address. This is to prevent malicious hosts from signing up other hosts for torrents. Since the token is merely returned by the querying node to the same node it received the token from, the implementation is not defined. Tokens must be accepted for a reasonable amount of time after they have been distributed. The BitTorrent

implementation uses the SHA1 hash of the IP address concatenated onto a secret that changes every five minutes and tokens up to ten minutes old are accepted.

8.2 Routing Table

Every node maintains a routing table of known good nodes. The nodes in the routing table are used as starting points for queries in the DHT. Nodes from the routing table are returned in response to queries from other nodes.

Not all nodes that we learn about are equal. Some are "good" and some are not. Many nodes using the DHT are able to send queries and receive responses, but are not able to respond to queries from other nodes. It is important that each node's routing table must contain only known good nodes. A good node is a node has responded to one of our queries within the last 15 minutes. A node is also good if it has ever responded to one of our queries and has sent us a query within the last 15 minutes. After 15 minutes of inactivity, a node becomes questionable. Nodes become bad when they fail to respond to multiple queries in a row. Nodes that we know are good are given priority over nodes with unknown status.

The routing table covers the entire node ID space from 0 to 2160. The routing table is subdivided into "buckets" that each cover a portion of the space. An empty table has one bucket with an ID space range of min=0, max=2160. When a node with ID "N" is inserted into the table, it is placed within the bucket that has min <= N < max. An empty table has only one bucket so any node must fit within it. Each bucket can only hold K nodes, currently eight, before becoming "full." When a bucket is full of known good nodes, no more nodes may be added unless our own node ID falls within the range of the bucket. In that case, the bucket is replaced by two new buckets each with half the range of the old bucket and the nodes from the old bucket are distributed among the two new ones. For a new table with only one bucket, the full bucket is always split into two new buckets covering the ranges 0..2159 and 2159..2160.

When the bucket is full of good nodes, the new node is simply discarded. If any nodes in the bucket are known to have become bad, then one is replaced by the new node. If there are any questionable nodes in the bucket have not been seen in the last 15 minutes, the least recently seen node is pinged. If the pinged node responds then the next least recently seen questionable node is pinged until one fails to respond or all of the nodes in the bucket are known to be good. If a node in the bucket fails to respond to a ping, it is suggested to try once more before discarding the node and replacing it with a new good node. In this way, the table fills with stable long running nodes.

Each bucket should maintain a "last changed" property to indicate how "fresh" the contents are. When a node in a bucket is pinged and it responds, or a node is added to a bucket, or a node in a bucket is replaced with another node, the bucket's last changed property should be updated. Buckets that have not been changed in 15 minutes should be "refreshed." This is done by picking a random ID in the range of the bucket and performing a find_nodes search on it. Nodes that are able to receive queries from other nodes usually do not need

to refresh buckets often. Nodes that are not able to receive queries from other nodes usually will need to refresh all buckets periodically to ensure there are good nodes in their table when the DHT is needed.

Upon inserting the first node into its routing table and when starting up thereafter, the node should attempt to find the closest nodes in the DHT to itself. It does this by issuing find_node messages to closer and closer nodes until it cannot find any closer. The routing table should be saved between invocations of the client software.

9 Email protocols and prospects

This section was originally meant to provide a basic picture of how email works today, however there is no big magic. POP3, SMTP and IMAP are the currently used protocols, they all share fundamentals: a server where a recipient's emails are stored, and they obviously rely on DNS when looking up MX records to find out where to send it the email.
We have seen how DNS could be fixed, however what do we do with the privacy of the emails? First of all using DPKI we could solve the servers end-to-end email sending, however using DIDs we could even achieve user end-to-end encryption. This is something I will discuss in my future works.

10 Conclusions and prospects

We have seen many new technologies and they are all promising. It has become clearer how to make applications with high secruity and privacy requirements. Hopefully the following goals are going to be achieved in the upcoming period:

- Decentralized DNS
- Decentralized PKI
- Decentralized IDs with self-sovereignity and wallet controllability
- Distributed social applications and reputation applications
- More secure email system
- Faster, secure distributed storage and storage retrieval system such as IPFS
- As the Web 3.0 is envisioned: linked data

I will work on this study in the upcoming research work. Thank you for reading.

References

[1] Arvid Norberg arvid@bittorrent.com Andrew Loewenstern drue@bittorrent.com. Dht protocol.

[2] Vitalik Buterin Jon Callas Duke Dorje Christian Lundkvist Pavel Kravchenko Jude Nelson Drummond Reed Markus Sabadello Greg Slepak Noah Thorp Christopher Allen, Arthur Brock and Harlan T Wood. Decentralized public key infrastructure.

[3] Ethereum Github. Ethereum white paper.

[4] Yahsin Huang. Decentralized public key infrastructure (dpki): What is it and why does it matter?

[5] Preethi Kasireddy. How does ethereum work, anyway?

[6] Imran Khan. Handshake, ens and decentralized naming services explained.

[7] Georgios Konstantopoulos. Million user dapps on ethereum - an introduction to application-specific sidechains.

[8] Georgios Konstantopoulos. Understanding blockchain fundamentals, part 1: Byzantine fault tolerance.

[9] Georgios Konstantopoulos. Understanding blockchain fundamentals, part 2: Proof of work proof of stake.

[10] Georgios Konstantopoulos. Understanding blockchain fundamentals, part 3: Delegated proof of stake.

[11] Jaime Lightfoot. Authentication and authorization: Openid vs oauth2 vs saml.

[12] Christian Lundkvist Cedric Franz Alberto Elias Andrew Hughes John Jordan Dmitri Zagidulin Markus Sabadello, Kyle Den Hartog. Introduction to did auth.

[13] Adam Powers. Understanding decentralized ids (dids).

[14] vasa. Understanding ipfs in depth(1/6): A beginner to advanced guide.

[15] vasa. Understanding ipfs in depth(2/6): What is interplanetary linked data(ipld)?

[16] vasa. Understanding ipfs in depth(3/6): What is interplanetary naming system(ipns)?

[17] W3C. Decentralized identifiers (dids) v0.12.

[18] W3C. Verifiable credentials data model 1.0.

[19] Wikipedia. Consensus (computer science).

[20] Wikipedia. Two generals' problem.