Technik
Informatik & Medien
Hochschule Ulm
University of
Applied Sciences

Decentralized Document Management

Master's thesis at the
University of Applied Sciences Ulm
Department of Computer Science
Degree Course Information Systems

submitted by
Emmanuel SCHWARTZ
Mat.-Nr: 3119342

April 2017

1st Evaluator:	Prof. Dr. rer. nat. Stefan Traub
2nd Evaluator:	Prof. Dr. rer. nat. Markus Schäffter

Declaration of Originality

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

3rd April, 2017 Emmanuel SCHWARTZ

Abstract

Acknowledgements

I would like to express the deepest apprecetion to my first evaluator, Prof. Dr. rer. nat. Stefan Traub, who has always his office open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but didn't hesitate to put me in the right the direction whenever he thought I needed it.

I would also to give a special thanks to Prof. Dr. rer. nat. Markus Schäffter for his spontaneous agreement to act as the second evaluator. I am gratefully indebted to him for his very valuable comments on this thesis.

I owe many thanks also to my colleague and now friend, Florian Schneider for his collaboration. Some parts of our respective thesis required a lot of team work. I'm so grateful that we could achieve a fantastic job together.

Last but not least, I must express my very profound gratitude to my parents and to my girlfriend Jill, for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you!

Contents

Chapter 1

Introduction

1.1 Motivation

It is often considered that the history of electronic mail (or e-mail) begins in 1965, at a time when the Internet did not even exist yet. By that time, the first exchanges of messages was only possible between users on private networks were set up. One of the first systems to allow message exchange was the Competent Time-Sharing System (CTSS) of the famous Massachusetts Institute of Technology (MIT), although this paternity has also been claimed by System Development Corporation SDC) and its own Time-Sharing System created for the Q32, a computer specially manufactured by IBM for the US Air Force.

However, e-mail is only really born from the creation of the ARPAnet network, the ancestor of the Internet. In 1971, after writing some 200 lines of code in order to create two applications, SNDMSG nas READMAIL, the engineer named Raymond Samuel Tomlinson could sent the first email of history to himself. Some times later, Tomlinson found a way for the program to easily differentiate a local message from a network message: the symbol @ was born. It was a simple way to dissociate a user name and host name with the only character that was not used in any proper name nor, above all, in any company name. The first "netmail" test was sent with only content "QWERTYUIOP", the first line of character of the English keyboard.

The email was so successful that it quickly became unthinkable for users of the ARPAnet network to do without it. As a result, the software quickly became the "killer app" of the ARPAnet network, and developers focused either on improving the program and its transfer protocol, or creating their own solutions. In 1992, a great improvment was made: the world's first-ever email attachment, sent by the researcher Nathaniel Borenstein, where we could see a adorable photo of his barbershop quartet, The Telephone Chords. This was made possible thanks to MIME (Multipurpose Internet Mail Extensions),a internet standard that extends the data format of e-mails.

Fourty years later, despite the creation of Instant Messaging, or some years later, social networks, e-mails are still very popular: 183 billion of them are sent every day! If e-mails spam remains a major problem, e-mails has to face new challenges: **Reliability** & **Privacy**.

When a person is sending an e-mail, she expects that her message will be received successfully by the intended recipient. For most cases, it does, but sometimes, for the following reasons, it does not:

–The design of e-mail: Two users does not have to be online at the same time in order to communicate. This is called asynchronious communications. This is made possible by the mail servers, accepting messages from sources and attempt to relay them towards, or deliver them to, the recipient. In order to do so, e-mails have to jump from an server to an other: Some e-mails might get lost during these operations, for various reasons (busy servers, e-mails deferral, rejected e-mails)

–The exponential groth of e-mail spam has forced the use of e-mail rejection, intended to identify and separate legitimate e-mails from junk e-mails. Unfortunately, this has turned out into a false positive problem: Some legimate e-mails are considered as junk, this means the user thinks he did not receive the e-mail. Some solutions try to solve these problems such as RE: Reliable Email[*], which tries to create an intelligent filter for e-mails, or tools that responds automatically to undeliverability by persisting with retransmission or retransmitting to alternate recipients [*]

Alongside with **reliability**, e-mail is facing one of the biggest challenges: **Privacy**. Since the shattering revelations of Edward Snowden, brigging to light the massive world surveillance by the U.S, people try to protect themselves by encypting their communications when possible. The famous free email application, Mozilla Thunderbird has an extension called EnigMail, which can encrypt e-mails thanks to OpenPGP. The downside is, that every user needs to have this add-ons installed, otherwise it will be not possible to read the e-mail that was intented for him. A solution could be that Thunderbird integrates directly this feature in his client, but is still not planned on the road map. But a recent piece of news[*] unviels that the IT-idustry giants such as Google, Microsoft, Yahoo!, LinkedIn and Comcast are working together to elaborate a new encrypted messaging protocol, named SMTP STS (Strict Transport Security)[*]. The idea is that a session starts in clear, and after the server announces that it supports the encrypted connection, the client can then switch to encryption mode, to avoid man-in-the-middle attack. Unfornately no released date was announced yet.

Last downside of e-mails involves the user. Nowadays, it is common practice to send business documents via e-mail or via supplier's web portal: Faster and cheaper compared to a hard copy sent via post. But, how can a user successuffly order and sort out differents kinds of documents, coming from different mailboxes? In most cases, the user does not have a clue how to establish a **document management**: local storage? cloud storage? And what if the computer has hard drive failures, or is infected by viruses: Did they make regular back-ups?

With respect to the application senario described in the next section, this thesis will try to answer these problems with new upcoming technologies such as blockchains, decentralized storage and Dapps.

1.2 Application Scenario

Even if the big actors of Internet are trying to adapt e-mails to nowadays requirements, e-mails might not be the right solution anymore to send buisiness documents. Figure 1.1

shows the path that an e-mail takes to go from the company to a client, jumping through servers via the protocol SMTP. Once the email has eached the client's mail server, the client has the choice to synchronize his email between 3 protocols: IMAP, POP3 or SMTP.

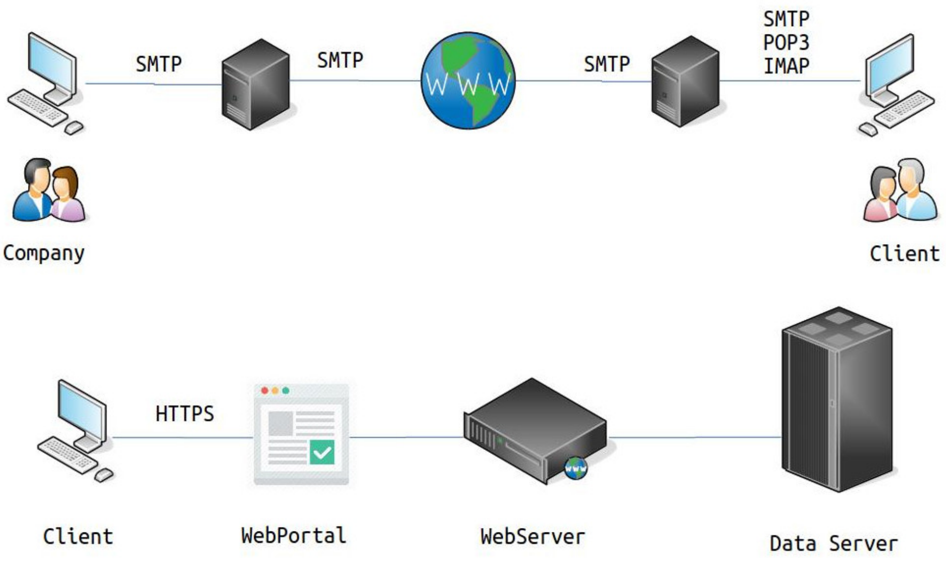

Figure 1.1:

New technologies that Bitcoin has brought, paired up with decentralized storage providers, can revolutionize the way we are going to send documents, described in the figure 1.2.

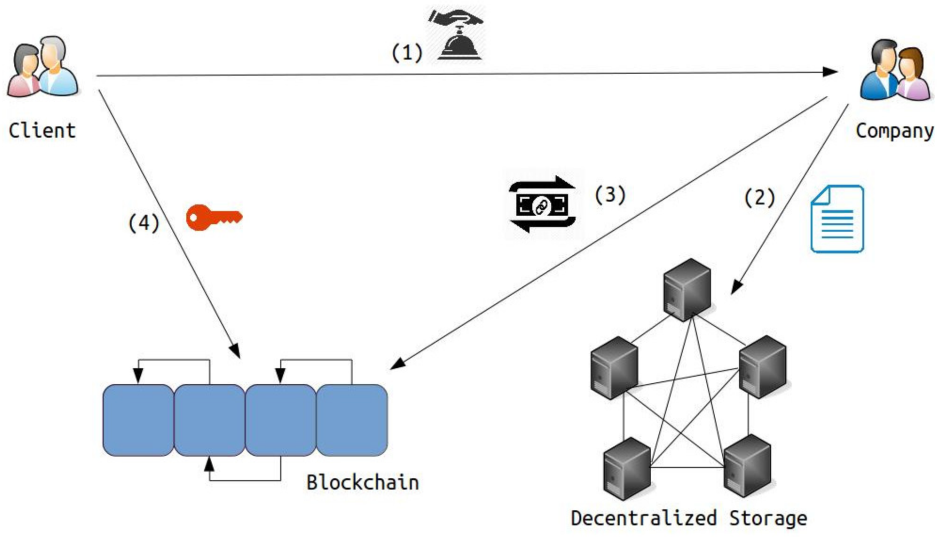

Figure 1.2:

The customer is creating a request to the company for a specific document (1). Then, the document is stored using a distributed storage provider (2). A link is created, which is stored directly in the blockchain by doing an transaction (3). The blockchain transaction provides the evidence, that the document has been sent and keeps a permanent not alterable link to the document. In order to open a document, the customer only needs its private key (4). (For further explanation, see 2.2)

1.3 Objective

The main goal of this thesis is to see if there is a possible implementation of a proof of concept (poc) using the new technologies such as blockchains, decentralized storage and Dapps to send documents between a company and a customer. Pros and Cons will be also discussed. Additionnally, MetaData can give useful information such as the author of a document, the timestamp, some comments etc etc. This thesis will show if metadata can be added during a transaction. Besides these constraints, a document should be also signed Within the proposed architecture. Furthermore, a flexible approach is desirable that supports multiple storage providers such as Amazon, Azure, IPFS, StorJ. Last but not least, this implemantation should be able to be combined with traditional cloud systems.
In summary, the objective is to find a proof of concept that can send documents between two users with the following points:

–A dvantages and limitations of this architecture

–Signing a document with this architecture?

–Combination with traditional cloud systems

–Flexible regarding storage providers

–Easy & intuitive GUI for users

–Metadata

1.4 Overview

!!!!!!!!!!!!!!!!!!!!!!!!! TBD !!!!!!!!!!!!!!!!!!!! To achieve the objectives stated in section 1.3, chapter 2 - Related Work and Basic Principles - contains an in-depth overview of state of the decentralized document management. TDB: 2.1, 2.2, 2.3 etc Chapter 3 presents the method. Chapter 4: evaluate the proposed approaches Finally, chapter 5 concludes the work

Chapter 2

Related Work and Basic Principles

2.1 Related Work

2.1.1 Towards Cloud-Based Decentralized Storage for Internet of Things Data

2.1.2 Prototype of cloud based document management for scientific work validation

2.2 Basic Principles

2.2.1 Reminder in Cryptography: Hash and Signature

2.2.1.1 Hash Function

A hash function is an algorithm that generates a numeric, or fixed-size character output from a variable-sized piece of text or other data (Figure 2.1). An essential property is that it is practically impossible to reverse it: the hash of a datum by the function is calculated very efficiently, but the inverse calculation of an input hash to find a datum is infeasible. For this reason, such a function is said to be one-way.The input data is often called the message, and the output (the hash value or hash) is often called the message digest or simply the digest.

An ideal cryptographic hash function has the following four properties:

–The hash value of a message is calculated very quickly

–Two messages can't have the same footprint

–It is impossible for a given hash value to construct a message having that hash value except by trying all possible messages

–A small change to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value

The most common hash functions are listed below:

–MD4 and MD5 (Message Digest) were developed by Ron Rivest. MD5 produces 128-bit hashes by working the original data in blocks of 512 bits.

Text	Hash(text)
Here is a long text	f272bcf903
Hello World!	d1be9c0ff4
Hello World.	0084a53e9d
Hello World.	0084a53e9d

Figure 2.1: Example of an hash function

–SHA-1 (Secure Hash Algorithm 1), like MD5, is based on MD4. It also operates from blocks of 512 bits of data and produces condensed bits of 160 bits at the output. It therefore requires more resources than MD5.[*]

–SHA-2 (Secure Hash Algorithm 2) has been published recently and is intended to replace SHA-1. The main differences are in the possible chopping sizes: 256, 384, or 512 bits. It will soon be the new reference in terms of hash function.[*]

–RIPEMD-160 (Ripe Message Digest) is the latest version of the RIPEMD algorithm. The previous version produced 128-bit digits but presented significant security flaws. The current version remains safe for now; It produces as the name indicates the condensed 160 bits. A final point concerning it is its relative greediness in terms of resources and in comparison with SHA-1 which is its main rival.

2.2.1.2 Signature

The digital signature[*] is a mechanism to guarantee the integrity of an electronic document and to authenticate the author by analogy with the handwritten signature of a paper document. It must allow the reader of a document to identify the person or organization that has affixed his/her signature. Moreover, a digital signature mechanism must have the following properties:

–Authentic: the identity of the signatory must be able to be found with certainty

–Forgery: the signature can not be falsified. Someone can not pretend to be another

–Not reusable: the signature is not reusable. It is part of the signed document and can not be moved to another document.

–Unalterable: A signed document is unalterable. Once it is signed, you can not change it

–Irrevocable: the person who signed can not deny it

In order to sign a document, the common way to do it is to use keys. Each user has a pairs of keys: a public one, called PubKey, and a private key, nammed PrivKey.

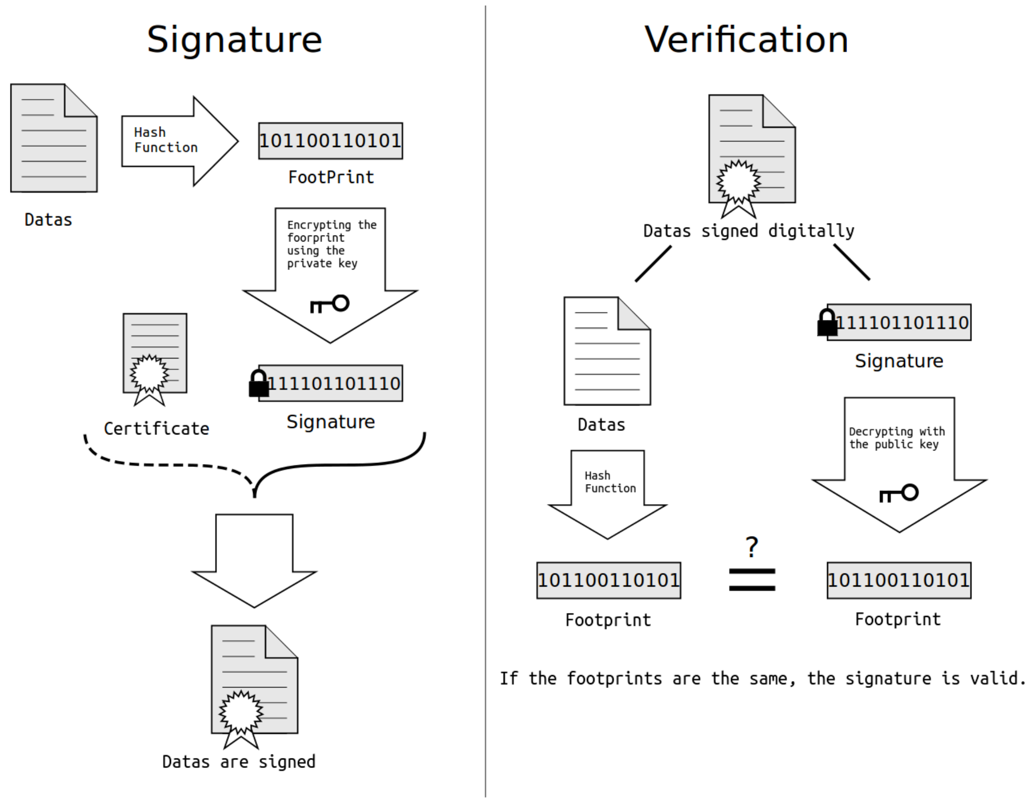

Figure 2.2: Digital signature diagram

2.2.2 Blockchains

2.2.2.1 What is a blockchain?

This section will dive into blockchain through the famous cryptocurrency: Bitcoin. As a reminder, Bitcoin was created in 2008 and brought at the same time a new concept: The system operates without central authority or single administrator, but in a decentralized way thanks to the consensus of all the nodes of the network.
-problem #1: give coin at the same time
-problem #2: How can we split a coin
-problem #3: problem of the double spent

2.2.2.2 Transactions

- propagation of a transaction to all the nodes

2.2.2.3 Block & Proof of Work

- detail of a block
- example of a block
- pseudo code of what are all the steps to confirm one block
- Fork
- How can be the the transaction is valid
- transaction every 10 mins

-how much transaction per block

2.2.2.4 Merkle Tree

- scheme merkel tree
- explanation

2.2.2.5 Nounce

- complexity varies
- calculation of hashes
- random hash
- pseudo code how to resolve a hash
- example of a nounce

2.2.2.6 Payment

- Evolution of the difficulty
- problem of pool mining

2.2.3 Ethereum

2.2.3.1 What is Ethereum?

- blockchain 2.0
- blockchain that can store
- faster blockchain but engender new problems
- really expensive to store data
- everything is public

2.2.3.2 GHOST

- orphan blocks - godfather blocks - scheme

2.2.3.3 Accounts

- ID
- balance
- storage root
- code hash
- one account per user / smart contract
- storage : non accessible outside account

2.2.3.4 Smart Contracts

- stored in blockchain
- can modify the state of the blokchain
- associted to one unique account
- conditions/ loops / unlimited stacks & memory/
- example of transaction with ethereum
- smart contract call talk to other smart contracts
- properties - constrainsts
- solidity, serpent, LLL
- Dapps

2.2.3.5 Oracle

- smart contracts that can provide pieces of information of the external world
- synchrone mode
- asynchrone mode

WHY DO WE DO THIS ?

2.2.4 Decentralized Storage Providers

2.2.4.1 IPFS

2.2.4.2 StorJ

2.2.5 Metadisk: Blockchain-Based Decentralized File Storage Application

2.2.5.1 Dat-data

2.2.5.2 Sia

2.2.6 Access control

Chapter 3

Method

3.0.1 Data Structure for Smart Contract

Information to Store	Types in Solidity
Storage Provider	String
Status	String
Name of the Document	String
Hash	bytes32 (SHA 256)
Signature	bytes32 (Pub Key)
Metadata [Date,author]	[unit (unix time), String]
Content Hash	bytes32
Verification of valid copy	???

3.0.2 EBNF/BNF

```
<urn>::= <prefix>:<provider>
<prefix>::= urn:x-docs
<provider>::= <storj>|<ipfs>|<azure>|<amazon>|<GDrive>|<Dropbox>|<OwnCloud>

<storj>::= ???
<ipfs>::= ipns:<DHTHash>:<FilePath>
<azure>::= ???
<amazon>::= arn:aws:s3:::mybucket/photo.png
<GDrive>::= ???
<Dropbox>::= ???
<OwnCloud>::= ???
```

3.0.3 URN/RFC

2 ways to register: formal ID:
are assigned by the IETF consensus / reviewed by a lot of people to be standartized / takes a long time informal ID:
are assigned with a number as an identifier (eg: "urn-¡number¿")
takes 2 weeks disccusion after the sending of the registration paper (urn-nid@apps.ietf.org)

use X- (eg: usn:x-docs:)

Chapter 4

Results

Chapter 5

Conclusion and Future Work

List of Figures

List of Tables