Invisible Ink: Blockchain for Data Privacy

ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 12 2015
LIBRARIES

by

Amir Lazarovich

B.A., The Interdisciplinary Center (2010)

Submitted to the Program in Media Arts and Sciences,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

© Massachusetts Institute of Technology 2015. All rights reserved.

Signature redacted

Author . . . . . .
Program in Media Arts and Sciences,
May 8, 2015

Signature redacted

Certified by . . . . . .
Andrew Lippman
Senior Research Scientist
MIT Program in Media Arts and Sciences
Thesis Supervisor

Signature redacted

Accepted by . . . . . .
Pattie Maes
Academic Head
Program in Media Arts and Sciences

Invisible Ink: Blockchain for Data Privacy

by

Amir Lazarovich

Submitted to the Program in Media Arts and Sciences,
on May 8, 2015, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

The problem of maintaining complete control over and transparency with regard to our digital identity is growing more urgent as our lives become more dependent on online and digital services. What once was rightfully ours and under our control is now spread among uncountable entities across many locations.

We have built a platform that securely distributes encrypted user-sensitive data. It uses the Bitcoin blockchain to keep a trust-less audit trail for data interactions and to manage access to user data [49]. Our platform offers advantages to both users and service providers. The user enjoys the heightened transparency, control, and security of their personal data, while the service provider becomes much less vulnerable to single point-of failures and breaches, which in turn decreases their exposure to information-security liability, thereby saving them money and protecting their brand.

Our work extends an idea developed by the author and two collaborators, a peer-to-peer network that uses blockchain technology and off-blockchain storage to securely distribute sensitive data in a decentralized manner using a custom blockchain protocol [40].

Our two main contributions are: 1. developing this platform and 2. analyzing its feasibility in real-world applications. This includes designing a protocol for data authentication that runs on an Internet scale peer-to-peer network, abstracting complex interactions with encrypted data, building a dashboard for data auditing and management, as well as building servers and sample services that use this platform for testing and evaluation.

This work has been supported by the MIT Communication Futures Program and the Digital Life Consortium.

Thesis Supervisor: Andrew Lippman
Title: Senior Research Scientist
MIT Program in Media Arts and Sciences

Invisible Ink: Blockchain for Data Privacy

by

Amir Lazarovich

Signature redacted

Thesis Advisor .....

Andrew Lippman

Senior Research Scientist

MIT Program in Media Arts and Sciences

Signature redacted

Thesis Reader .....

Sep Kamvar

Associate Professor of Media Arts and Sciences

MIT Program in Media Arts and Sciences

Signature redacted

Thesis Reader .....

David Clark

Senior Research Scientist

MIT CSAIL

Acknowledgments

This work would not have been possible without the help of my family, mentors, colleagues and friends.

I would like to thank my advisor Prof. Andy Lippman for believing in me, and for his guidance and support throughout this journey. I would also like to thank my thesis readers, Prof. David Clark and Prof. Sep Kamvar, for their insightful comments and suggestions.

Above all else, I am thankful for having such a wonderful, supportive and loving wife, Michal. She recently gave birth to our first child, Eitan, which is a source of inspiration for me. I would also like to thank my parents, siblings and the rest of my family, for their love and care, and for pushing me to follow my dreams - no matter what were the challenges.

I also want to thank Jeremy Rubin, Guy Satat and Yadid Ayzenberg for their continuous support and helpful advice.

This work would not have been possible without the support of the Viral Communication members and the one and only Deborah Widener.

I would like to thank the MIT Bitcoin Project and all of its founders: Jeremy Rubin, Dan Elitzer and Richard Ni for inspiring us all at MIT to conduct research related to Bitcoin. I'm very excited to see the research that will come from the MIT Digital Currency Initiative.

Thank you Oded Golan for introducing me to the world of Bitcoin in 2011.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the following sections we describe the motivation and main contributions of our work.

1.1 Motivation

Identity protection is becoming a more serious problem, since the world seems to have abandoned anonymous transaction of almost any sort. You have to create an account at any informative web site you visit whether you ultimately want to engage in a transaction with them or not. This makes the system ripe for identity theft and misuse. The former is a pervasive economic problem and the latter is a subtle social problem that erodes our sense of privacy [29]. We can envision a day when singing in the shower becomes public through the towels and faucet. This is widely recognized among researchers and watchdog organizations but hasn't yet fully registered with the public.

Technical solutions have been proposed, but they have not gained traction yet [1] [15] [25]. It is not clear why, but at least one explanation is that they are either "utopian" or cumbersome. Utopian in this context means that they require a priori agreement by a critical mass of organizations and users on an approach that may not be in their interest until everyone else adopts it. Some are cumbersome in that they merely move the problem from each commercial entity to a presumably trusted third

party–but no such party may be available or accessible in all circumstances. In other words, they move rather than solve the problem. What is needed is a more "viral" approach that can start small, scale without bound, and evolves as it does so. Such approaches have proven useful at solving the matter of "diffuse benefits and acute costs" associated with large-scale centralized efforts [71].

Bitcoin and other distributed technologies have been suggested for many situations where legacy centralized approaches are failing to meet current needs to adapt to modern technologies [49] [69]. Uber, for example, bypasses much of the baggage of the organized taxi industry [72]. Many of the restrictions of such regulated public transport date from an era where the goal was to control and minimize the number of vehicles on the road, whereas now the problem is that the public interest would be better served by fewer private cars and more on-demand ones. The industry could not evolve to suit this change in climate and is being supplanted in many circumstances. The distributed version shows promise. Bitcoin itself has also been suggested for audit functions, for exchanges, and for a host of other applications where an often-monopolistic central organization has become inefficient or untrustworthy [2] [59] [12].

The main advantage of Bitcoin technology here is that it has shown that a decentralized, durable, public and irreversible recording method can be created and work in the real world.

There are a number of cases where researchers and entrepreneurs have proposed applying Bitcoin technology to identity [40] [55]. There are many possible approaches; no single solution has emerged as the obvious "best one".

Our work extends an existing idea of a framework for controlling access to files and ensuring that ultimate access can be eliminated securely [56]. It involves a cryptographic key system where the registration of the core keys is simple and can be duplicated yet still be controlled. In essence, the basis for our extension is making that key system public and decentralized.

The further advantage of our approach is that it can be extended to provide for continuous control over who can access the files and data.

This work is demonstrated in a real world system and assessed for the thoroughness

of the solution.

1.2 Novelty and Main Contributions

This thesis makes an important contribution to the field of identity privacy by giving people the ability to be in control of their personal data.

a) **We developed a platform that securely distributes sensitive data while keeping it available on-demand.**

b) **We created a *certified mail* service built on top of our platform.** This service allows one to communicate with others and keep their conversations "on record". By building this service we were able to test our platform and evaluate its feasibility.

c) **We presented an alternative usage for the *blockchain* technology.** Almost all other applications for this technology target the financial industry. We hope to open up opportunities for future research and innovation using this technology in many other fields.

Chapter 2

Background and Related Work

In the following sections we describe the history of Bitcoin, highlight innovations in similar fields and discuss related work to our system.

2.1 From Bitcoin to privacy

2.1.1 Before Bitcoin

For many years people around the world have been attempting to create digital currencies. Two well known attempts were E-Gold and Liberty Reserve [26] [6]. Both services allowed instantaneous transactions of digital tokens with minimal fees. E-Gold tokens were backed by real gold and other precious metals, whereas Liberty Reserve tied its tokens to the US dollar and euro. Both of these services were shut down by the United States after many years of operation under the allegations of fraud, money laundering and more.

Similar to Napster, the biggest technology crime all these digital currency innovations had was of being centralized [22]. That means they were orchestrated and controlled by a single source of authority. One might be tempted to assume that if these services were not operated by a single entity, then people wouldn't be allured to tamper with the funds going through their service and therefore illegal consequences would be prevented.

In 1999 Confinity launched a project called PayPal which allowed individuals to send ("beam") money to anyone from their Palm Pilots, pagers and other smart devices [41]. You could either upload your credit card or send money to your account and be credited. PayPal bridged the gap between the financial world and emerging smart devices, making it easy and immediate to transfer money from point A to point B. However, these transactions came with costs related to handling the money, for example by Merrill Lynch.

Opposite to centrality, there were also conceptual attempts to create decentralized digital currencies. B-Money and Bit-Gold are great examples of this [14] [60]. One might argue that the main reason these suggestions weren't implemented is due to their flaw of being incomplete [61]. One example is how B-Money handled its token creation and that Bit-Gold was prone to certain attacks, such as a Sybil attack[1] [18]. Another noteworthy innovation is called Peppercoin [46] [35]. They created a micro-payment scheme that significantly reduced bank processing costs.

Chaum's Blind Signatures in 1983 paved the way for many future attempts to make digital currencies, such as OpenCoin and DigiCash [9] [63] [8]. Nicholas Negroponte, the MIT Media Lab co-founder, was also the director of DigiCash [67] [10] [64].

In 1997, Adam Back introduced an innovation that was intended to solve a growing problem: email spam. He called it Hashcash [3]. To simplify its complexity, for Hashcash to work it requires emails to include a signature that could be easily verified but hard to produce. Emails that weren't verified were considered spam. Therefore, spammers were required to narrow down their mailing list in order to send all their emails in a reasonable time. Back's invention is the foundation of all future attempts and implementations of decentralized digital currencies [37] [44].

A lot of interesting suggestions and implementations can be found in the archive of the cypherpunks mailing list [13]. Hal Finney's Reusable Proof of Work is one for example [23]. But it was not until 2008 that there was one that was highly adopted. In 2008, one attempt showed a huge potential for its technology and features. It

[1] A Sybil attack in computer science is when a malicious actor in (mostly) peer-to-peer networks tries to subvert the reputation of the network by controlling a large percentage of nodes. The term "Sybil Attack" was coined by Brian Zill at Microsoft Research in or before 2002 [16].

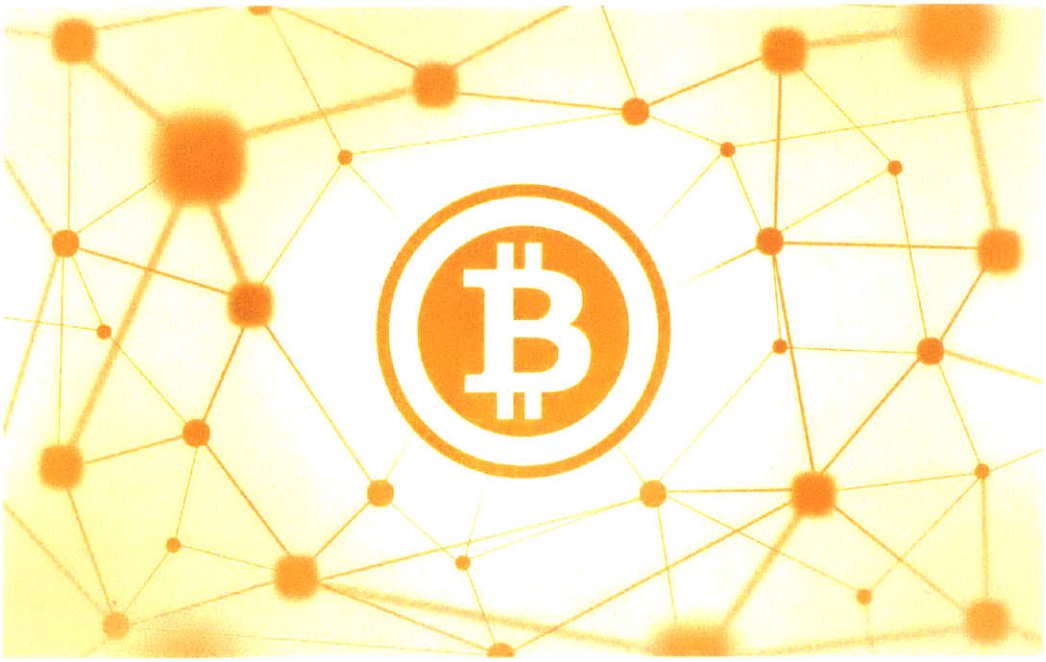

Figure 2-1: Bitcoin - the first decentralized peer-to-peer digital currency

presented a working peer-to-peer decentralized, Byzantine-fault-tolerant and Sybil-attack-tolerant digital currency that is protected against double spending [38] [70]. That project was called Bitcoin [50] [51].

2.1.2 The Bitcoin era

Bitcoin was first introduced in 2008 in a private mailing list called cypherpunks [50]. It was announced by Satoshi Nakamoto. As of now, no one knows who Satoshi really is or whether it is a group of people or an individual.

Bitcoin was created to solve mainly one problem: digital payments go through financial institutions that are supposed to keep our transactions, funds and privacy secure and therefore require us to trust them [51]. Trust is the key word here. Many attempts have been made before Bitcoin, as mentioned in Section 2.1.1, but none succeeded to remove entirely the need of trust. Bitcoin, to some extent, succeeded for the first time in history in doing so.

At a high level, Bitcoin makes it possible for a peer-to-peer network to share a database, organized by time, without having any single central source of management.

Every node in the network can verify by itself the correctness of that shared database. This database is called the blockchain. It is a specialized linked list such that each of its items contain a list of transactions, a time-stamp and other important information that are not mentioned for brevity [52]. Nodes in the network verify transactions based on previous transactions that exist in previous blocks inside the blockchain. To add a new block it is required to find a magic number that along with other data inside the block itself, if processed by a specific function, will produce a string that follows certain guidelines. This process is an implementation of Adam Back's Hashcash algorithm, or in more common terms, the "proof-of-work" required to produce a new block [3]. Bitcoin adjusts the difficulty of finding such string every two weeks to ensure block creation takes on average 10 minutes. Nodes in the network choose blocks that are attached to the longest chain. These characteristics protect against an important attack called double spending [70]. In a nutshell, double spending is the ability to use more than once the same input source by manipulating the network state. In Bitcoin, input sources point to bitcoins and our network state is the blockchain.

Another key property is identity anonymization. Accounts in Bitcoin are pseudonyms and each person can have nearly any number of accounts they want. Although, once a pseudonym is uncovered, it is trivial to link all of its past, present and future transactions. In a later stage came different techniques that obfuscate linking attempts, such as CoinJoin [43].

Bitcoin, with a capital *B*, usually refers to the protocol whereas bitcoin, with a lowercase *b*, refers to the digital currency Bitcoin creates. In Bitcoin, coins are minted every time a new block is created by solving a cryptographic puzzle. Cryptographic signatures protect transactions in the Bitcoin network. Both of these are mostly why bitcoin is also referred to as a crypto-currency: it is based almost entirely on mathematical principles.

The first block was created in January 3rd, 2009 by Satoshi Nakamoto [52]. The first real purchase with bitcoin was in 2010, 10,000 bitcoins for two pizzas [39]. Back then, 10,000 bitcoins were worth about $42. Today 10,000 bitcoins are worth $2,234,100. See entry [33] in the Bibliography for the complete history.

2.1.3 Bitcoin, beyond digital currency

Bitcoin has been around for five years. In that time, what appears to have happened is that we now have a working system for distributed trust. It may not be perfect, but it has worked at least as well as alternatives that get hacked quite a lot [32] [30] [5]. This has raised interest in both the financial impact of a currency that is international, secure, and novel. The blockchain, which is the guts of Bitcoin, is now also being explored for ideas not necessarily related to money, but to consider the implications of a secure, time-stamped, unalterable public ledger. This can change the notion of contracts and recording of data. MIT has just announced an initiative to explore all of these dimensions [24]. To some this is an opportunity as big as the Internet.

To list only a few innovations built on top of a blockchain architecture: Storj, a decentralized peer-to-peer cloud storage network [73]. IBM's Adept, an Internet of things architecture [31]. Onename, a distributed and secured identity platform [55]. ChangeTip, micro-payment platform for online tipping [7]. Coinbase, an online bank for Bitcoin, platform for merchants and the first regulated Bitcoin exchange in the U.S. [11].

It is hard to tell what all of the implications mean, and there is a lot of early investment clouding the picture, but as the MIT effort will attempt to show, there is room for experimentation and research to develop those key ideas [76]. In the Media Lab fashion, the domain spans social and technical work: people keep Bitcoin running and will define much of its use, but technology will determine which uses are possible and reliable.

Our work began by considering centralized systems for things other than money that might work better in a distributed fashion. Figure 2-2 presents a very high-level view of the difference between centralized and decentralized systems. The banking system is a good example of a completely centralized system. All of our transactions go through their network and are stored in their ledgers. This creates a burden for the banking system for the possibility of data breaches and therefore requires invest-

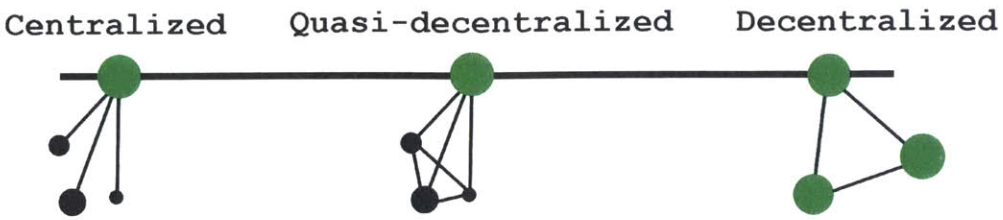

Figure 2-2: From centralized to decentralized systems, which is better?

ment in information security and highly trained personnel. Also, we have high costs of switching between the different alternatives which makes us more bound to their services. A step forward on the curve of digital decentralization is for example Uber [72]. We consider Uber to be "quasi-decentralized" because there is still an orchestrating server that maintains the network. However, individual nodes can interact with one another directly and choose with almost no costs to use an alternative service. At the very end of the curve, an example of a decentralized system is Bitcoin for the reasons mentioned in previous sections.

We began with identity. Polychronis tried this in his thesis [75]. He moved identity from being centralized to being user-distributed, where the identity migrated from each user to owner-controlled banks. He was ahead of his time and lacked a platform for it to gain critical mass in the real world. The initial idea of myself, Zyskind and Nathan was to distribute it using an extended protocol to that of Bitcoin [40]. Our project was named Ethos. Our intentions were to create a new blockchain that supports an additional data-layer to distribute user sensitive data across the nodes in the network. Because we were on a tight schedule, we chose to fork an existing blockchain, Ethereum, and handle the distribution of sensitive data by adding a layer of a Distributed Hash Table [19] [66]. To test our system we built a web dashboard and two mobile applications. The mobile applications gathered user sensitive data, and the web dashboard allowed users to view the gathered data and revoke access to it. Ethos also won first place at the MIT Bitcoin startup competition. There are lots of cases where this can work: health care, Facebook, AirBnB, and other use cases where our sensitive personal data is managed by other services that either occlude

its usage or fail to secure it [54]. There have also been lots of efforts in this area: Onename, OpenPDS, Abine and more [55] [15] [1].

In the end, We developed a simple, yet useful application that is a potential test bed for the development of more of these. We call it "Certified Mail". A detailed description of this system is given in Section 4.1.

2.2 Related work

The field of decentralized applications and platforms is quite new due to its reliance on the Bitcoin protocol, which was published only 5 years ago [51]. As of now, there hasn't been any other solution that uses blockchain technology to solve data liabilities and provide data auditing in a decentralized manner.

However, there have been a few attempts to address the privacy problem with regard to personal and sensitive data. OpenPDS is one example [15]. They understand the importance of keeping our digital identity safe but have chosen to solve it using centralized and trusted data stores. Abine is another example of an attempt to keep our data safe while browsing the Internet, smartly blocking exploits and masking our email addresses by proxying it through their trusted servers [1]. While the former were more real-world attempts to address privacy, Alternet is more a concept for an ideal world where user-privacy protection is part of the Internet infrastructure [25].

MaidSafe is another example [42]. It first appeared in 2007 and strove to solve personal data security. Their initial approach was to create a peer-to-peer network in which data is sharded, encrypted and distributed across the network with additional redundancies for backup and availability. Similar to torrent clients, the way MaidSafe incentivized its clients to share their disk space and have it available online is by increasing their bandwidth and capacity. In 2014 MaidSafe entered into the Bitcoin world and created their own blockchain and coin called Safecoin. Among other benefits, this transition gives their users another incentive to join the SAFE network. The difference between our platform and MaidSafe is twofold: our platform doesn't require the creation and maintenance of a peer-to-peer network and the data stored

in our system is kept private within what we call blind escrows (see Section 3.5.1).

Related to data privacy, there is also the work of Perlman on assured deletion of files in a specially designed file system and in third-party cloud storage services [56] [34].

Ethos, A proof of concept system that was developed by myself and two collaborators, had a similar goal albeit a very different architectural approach [40]. Ethos forked Ethereum and handled sensitive data using a Distributed Hash Table [19] [66]. To be adopted it required the creation of a complete peer-to-peer network and although encrypted, the user sensitive data was distributed across the nodes in the network and required handling similar to those of torrent files [69]. Our implementation uses Bitcoin's blockchain along with other sophisticated trusted blind escrow services to handle sensitive data, which makes it immediately available for deployment at scale without any prerequisites.

A few months after we published Ethos, a new service called Factom appeared [21]. It is an additional layer on top of the Bitcoin blockchain that provides functions and features beyond currency transactions. It aspires to solve three main drawbacks related to the Bitcoin protocol: block speed, transaction costs and the blockchain size. Compared with Invisible Ink, Factom introduces a new crypto-currency called *Factoid.* To perform specific actions, such as adding a new *entry* into Factom, you need to pay with their digital token. Also, while our data is securely stored in a distributed and trusted manner, Factom uses a similar method to that of Ethos where the data is stored in a Distributed Hash Table across their peer-to-peer network, which requires creating such a network in order to secure the stored data.

A more radical approach to protect sensitive data is a recently introduced programming language called Jeeves [74]. It builds from the ground up programs that treat privacy as first-class citizens in code. That gives more control over sensitive data and its flow across the program.

As for our implementation for certified mail, in which participants can prove their conversation history to third parties, while keeping the conversations stored in trusted locations, there has been prior work built using different technological

solutions. ReadNotify is one example of a service that allows its subscribers to track their emails. However, the data itself is stored on both the email client servers and ReadNotify's servers [58]. Bleep from BitTorrent is another example [4]. It is a peer-to-peer messaging application. Messages are stored only locally and there are no centralized servers that eavesdrop. The difference between our certified mail service is that Bleep users can't prove to third parties any of their conversations.

Chapter 3

System Design and Implementation

In the following sections we outline an existing challenge and our suggested solution and describe elaborately how the Invisible Ink system works.

3.1 The challenge

People lose control over their personal data as soon as they share it with third parties online. This introduces three main challenges: information security, lack of control and transparency. Information can be insecurely stored and therefore might leak. People can't recall the information they have shared and have no transparency over how their data is being used.

3.2 Our solution

A user-trusted distributed system that securely stores personal data, manages permissions with third parties and audits data interactions to provide a complete transparency and proof-able event trail.

3.2.1 User perspective

The user enjoys the additional privacy it receives by managing their own sensitive data. They benefit from the protection against single point-of failures that are usually

caused by centralized systems. By storing an audit trail inside the blockchain, users can rest assured they will have proofs for data exchange and interaction in case they need them. For example, let's assume you opened an account in a fictional service X. Part of the registration process requires you to give your social security number. After a while, you decide to stop using this service and therefore issue a new transaction that indicates your intentions. Not long after, you see in the news that service X was hacked and released a list of all leaked social security numbers. You notice that your social security number is within that list and therefore file a lawsuit against X indicating that you have a proof they shouldn't have your information after a certain point in time, marked within your irreversible transaction.

3.2.2 Service perspective

Services benefit mainly from the way sensitive data is managed and stored. First, they don't need to manage a large amount of user data. For instance, how to store the data, to index, to distribute and more. Second, they are less liable to potential data breaches. Because no sensitive data is stored on their machines, they are less prone to attacks. Finally, similar to users, they benefit from the proof-able nature of all data-related transactions. In case of a dispute, they can look up specific transactions and prove to third parties contracts made with users.

Similar to tokenization in data security, authentication with escrows would require meeting high security standards in order to protect user sensitive data [68].

3.3 Overview

You share your information with third parties and lose control and ownership immediately. What happens when you decide to stop using a service? What happens to all the data you previously shared with the service?

Right now your data is forever lost and kept inside the databases you have no control over.

Invisible Ink shifts this paradigm and gives people what rightfully belongs to them.

User's data will be stored in user-trusted locations, and permission management and data-interactions will be logged inside the Bitcoin blockchain.

For example, let's assume you live in Cambridge Massachusetts and your medical insurance is from Anthem Blue Cross. You are a professor at MIT and from time to time, like many, you go to see the doctor. After a couple of years doing what you love the most, you receive an offer that looks like your dream job: going back to Israel, and managing the Computer Science department at the Technion Institute of Technology. Your wife and family couldn't be more happy and are all excited for moving back to Israel. In Israel, you get a new medical insurance. Not long after you're back, you read in the news about Anthem being hacked and that your medical record leaked out. You fear now for your new position and status, because some of that medical record shows of an unfortunate illness you have that might make people look at you differently. What can you do?

Now let's assume Anthem Blue Cross have used the Invisible Ink system. When you first got your medical insurance, you also opened an account in Invisible Ink. Part of the process required you to provide a link to where your data will be stored and managed. You're not very paranoid and tend to trust the people close to you, especially the company or organization you work for. Therefore, you choose to store and manage your sensitive medical information inside MIT's blind escrow service. You understand that your data will never be stored in clear-text and that only you and Anthem Blue Cross are capable of decrypting it. During the registration process for Invisible Ink you see an option to log all the interactions done on your personal data inside the Bitcoin blockchain. This gives you the ability to track changes in your stored data, performed on your behalf by Anthem for example. To enjoy such a service you are required to pay a relatively low fee each time there are changes on your data. Because each change cost about $0.01, and the volume depends on the amount of time you go to see a doctor, you accept this feature. After your first visit to the doctor, you log in to Invisible Ink dashboard to see how and what information on you have been stored and read. You feel in control of your personal health record. Before leaving the United States, you log again to your Invisible Ink dashboard to revoke

permissions to your data from Anthem Blue Cross. After reading about Anthem being hacked, you breath freely knowing that your private data is secured. If for some unfortunate reason you find out that your data leaked out as well, you can file a lawsuit against Anthem for breaking a contract and storing an illegal copy of your sensitive information. That contract is located inside the Bitcoin blockchain and would never disappear.

By managing permissions to user-data and logging data-manipulation inside the blockchain we ensure the validity of these actions, without the need to trust anyone but yourself.

3.3.1 Create an account

Figure 3-1: Register and create a new account

To enjoy the benefits of Invisible Ink, first you need to create an account (see Figure 3-1). It is a very simple process in which you choose any `username`/`password` combination and provide common user details that you may choose to share with future online services in later stages.

Your `username` and `password` will never be sent to Invisible Ink's servers and therefore must be stored in a safe place. Because these credentials are only known to you, you are responsible for memorizing them. During the registration process, your mouse movements and keyboard strokes are used as entropy in the process of creating asymmetric elliptic curve keys [36]. These keys represent your Bitcoin initial address

and wallet.

An important part during registration is selecting your escrow (see Section 3.5.1). Selection is simply made by providing a valid url address.

3.3.2 Add funds to your account

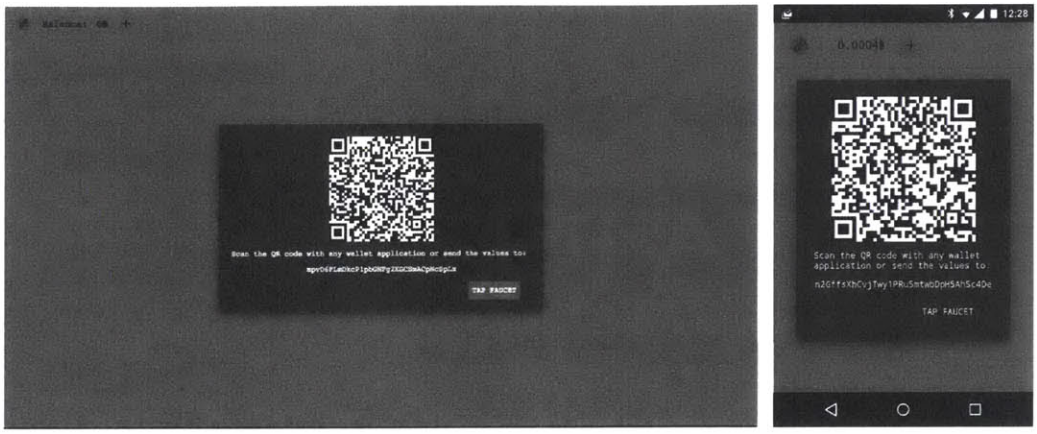

Figure 3-2: Web and mobile interface to send money to your Invisible Ink account

After you've registered, you first need to add bitcoins to your account. These bitcoins are used to pay for tracking interactions performed on your personal data. Each time a new transaction is sent to the Bitcoin network, it comes with a fee. These transactions hold on record actions to read, write, update or delete data that belongs to you. By storing these records inside the Bitcoin blockchain, you guarantee they will be durable, irreversible and organized by time. You will be able to use these public records to prove a third-party contracts made with a service you're using or have used.

When using the *testnet*, you can quickly add funds to accounts using a *testnet* bitcoin faucet (see 3-2) [65]. Otherwise, you can either scan the barcode or manually send funds to your Bitcoin address.

In future implementations, paying for personal data tracking can be incorporated inside an already incurring cost, therefore reducing the complexity further.

3.3.3 Connect to a service

Similar to existing identity authentication services, connecting with Invisible Ink is fairly straightforward [20] [27]. The heavy lifting is performed behind the scene, giving a smooth and fast user experience in the front-end. In Chapter 4.1, we will present a sample service built on top of Invisible Ink and show how users authenticate with it.

3.3.4 Data and privacy

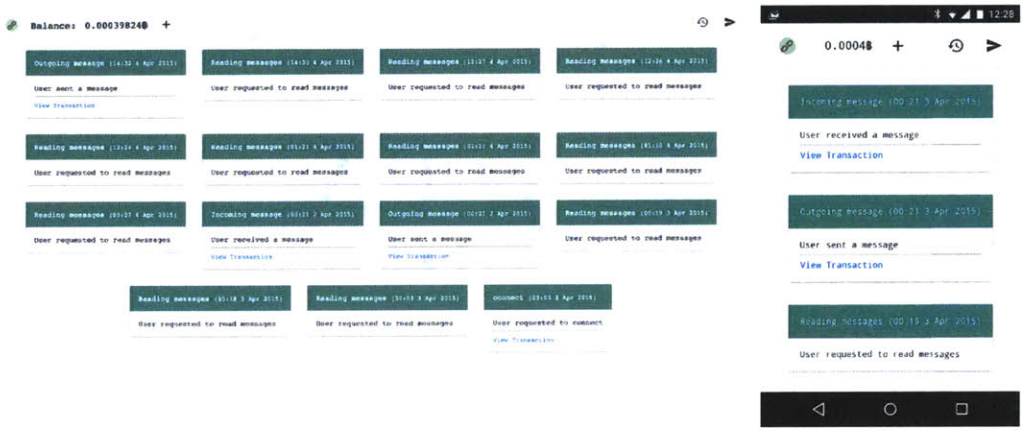

Figure 3-3: Data manipulation history

All user-sensitive data will be audited inside the blockchain (see 3.7) and stored inside your selected escrow. All the interactions done on your data will be visible only to you through your account in Invisible Ink dashboard (see Figure 3-3).

If you notice suspicious actions or decide to stop using a service, you can simply *disconnect*. *Disconnecting* creates a new Bitcoin transaction that restricts access from services within your escrow, where your data is stored.

3.4 Technical overview

3.4.1 Write data

Figure 3-4 shows the high-level flow of sensitive data between a user and a service. The user types in sensitive information, such as an address, credit card, social security

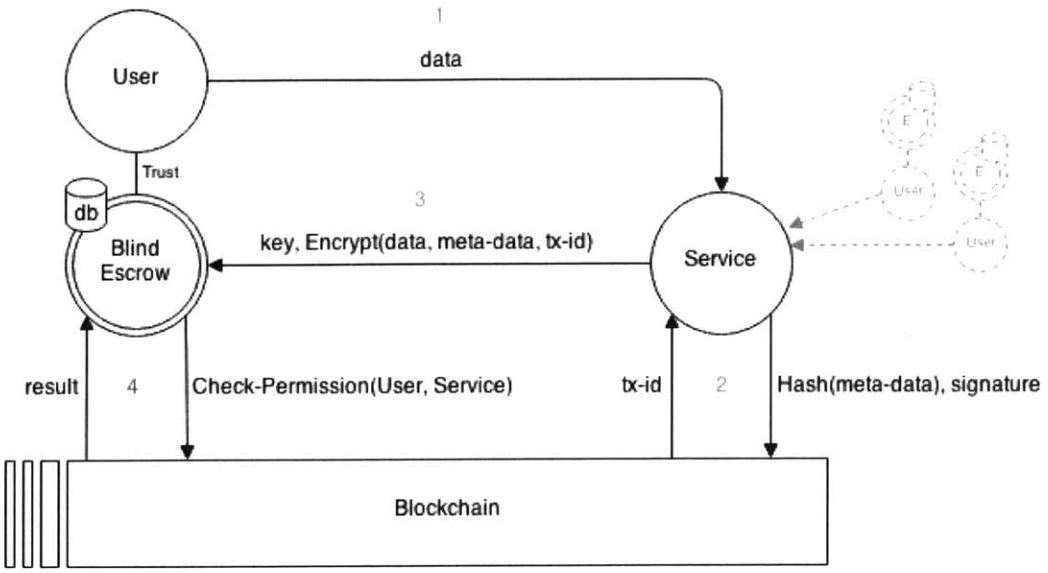

Figure 3-4: High level write data diagram

number, etc. and sends it to a third party service. The service extract the meta-data, hashes it, appends its signature for validation and broadcasts a transaction in the Bitcoin network. In return, it receives a transaction Id. The service then encrypts everything using a shared (with the user) symmetric key and sends the cipher-text to the user's trusted blind escrow server (explained in Section 3.5.1).

Sensitive information is stored in a highly distributed manner; each user can have its own trusted escrow. Escrows always check for permissions inside the blockchain before responding to requests (see Figure 3-6).

3.4.2 Read data

The information stored in escrows is available on-demand to services without requiring explicit action by the user, given that the user provided access rights and recorded them inside the blockchain. Figure 3-5 shows a high-level diagram for an on-demand read request issued by a service. First, the service is required to audit its intentions inside the blockchain, for example "reading user address". Then it sends a request to retrieve desired data from the user's escrow. The escrow checks that the service indeed has access rights and responds with the stored encrypted data. The user and the service, who share the encryption/decryption key, are the only ones who can

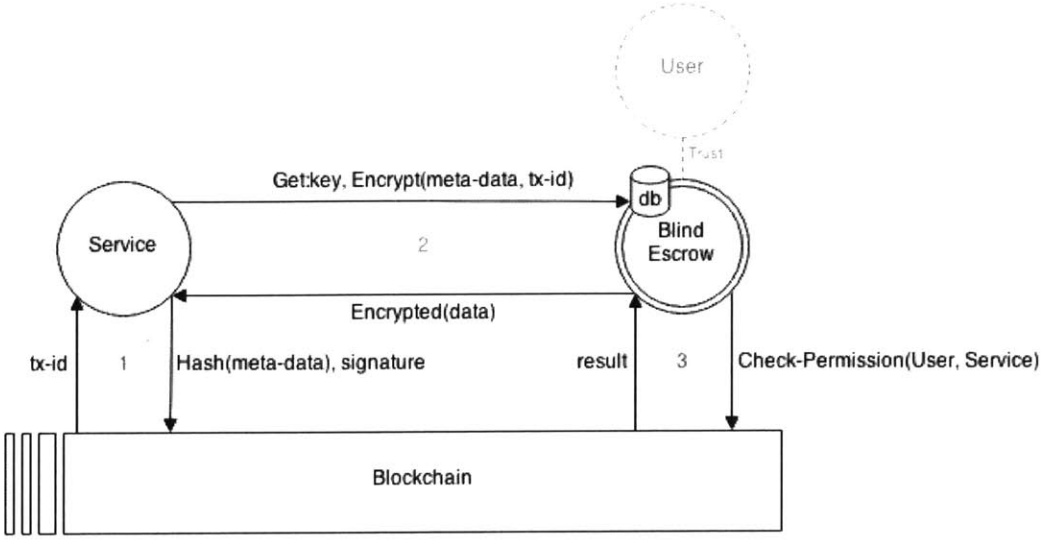

Figure 3-5: High level read data diagram

decipher that response.

3.4.3 Grant permissions

To make these types of communications possible, first the user needs to give the service permissions to handle its data. Figure 3-6 presents a high-level overview for giving permissions to a service. It starts with the user generating asymmetric keys P_1, P_{r1} that are used to validate transaction signatures inside the blockchain, an ID, a symmetric key to encrypt its sensitive information, an email address (or any other validation method) and the address of the escrow. The service receives the user ID, along with the shared encryption key and email address, then it generates its own asymmetric keys P_2, P_{r2}, and a unique ID and sends back to the user its ID, public key and an email with a verification code. The user updates its escrow with the received verification code and broadcasts a transaction with the permission, two public keys P_1, P_2 and verification code. These public keys will associate future related transactions by their signatures.

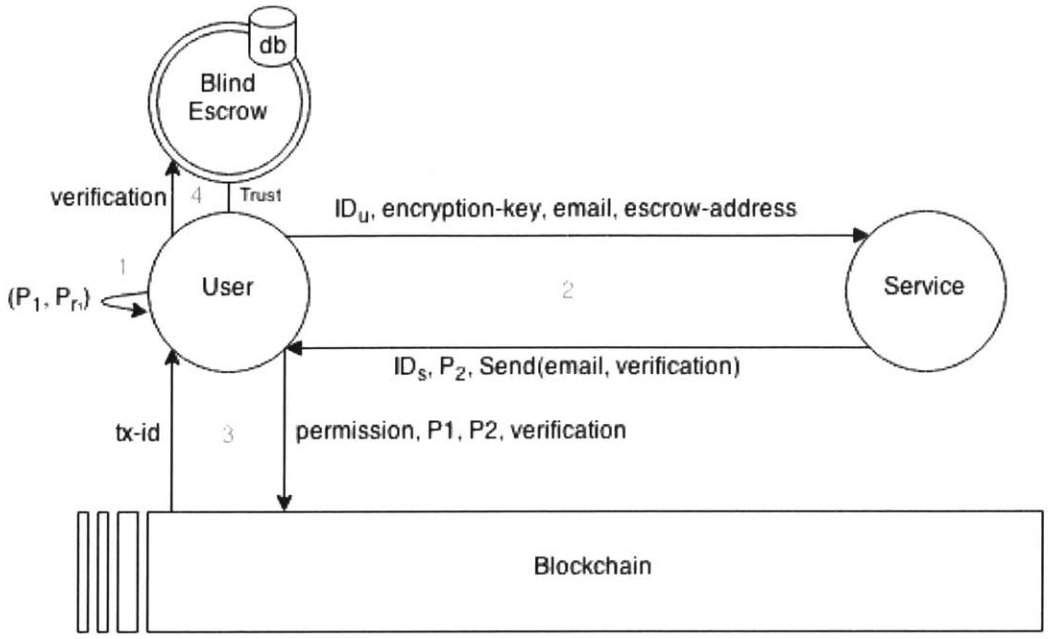

Figure 3-6: High level grant permissions diagram

3.4.4 Revoke permissions

Figure 3-7 presents a high level overview of how a user revokes permissions from a service to access their sensitive data. The key part in this diagram is the transaction stored inside the blockchain that indicates the user's intentions to revoke access to its data for that specific service. Later, the user's escrow will find this transaction and deny access to sensitive data for the corresponding service. The user also triggers a call to write an *audit-log* to keep it for its record.

3.4.5 User-Escrow key exchange

When creating an account, you also choose a blind escrow service (see Section 3.5.1). Figure 3-8 presents a high level overview of how a user initially connects to its escrow. It is a similar process to a Pretty Good Privacy (PGP) key exchange. First the user generates a random `32 byte` string that will be used to identify the user. Then it generates asymmetric keys using an elliptic prime curve of 384-bit. Along with its ID, the user sends also its public key to allow the escrow to securely communicate with it. The escrow performs a similar process and responds with an encrypted value

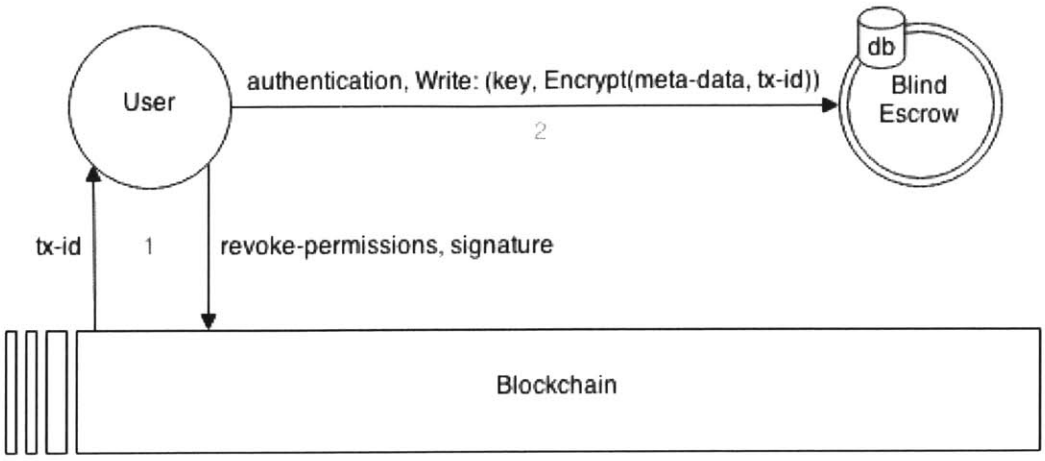

Figure 3-7: High level revoke permissions diagram

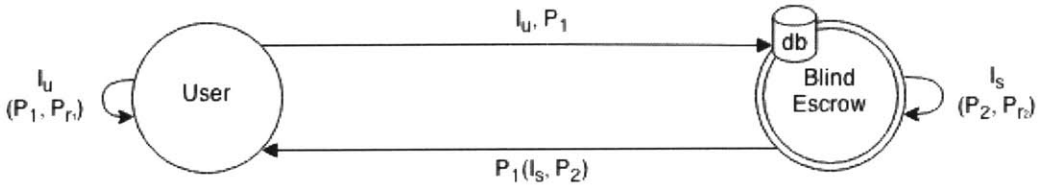

Figure 3-8: High level user-escrow key exchange diagram

of its ID and public key.

3.5 Terminology, interfaces and actors

3.5.1 Blind escrow service

Each account has an escrow service that guards its sensitive data. They act as gateways for services to your data. All requests to *read/write* are passed to escrows and first checked for recorded permissions inside the blockchain. Escrows are considered blind because they are not capable of decrypting the information they store. The data is usually encrypted by services using a key they share with the user. The escrow's server code is open source, therefore anyone can create an escrow for themselves, otherwise, they can use whatever escrow server they trust. The only requirement is that it will stay always connected such that the data it "guards" will be available on demand.

Escrows are accountable for keeping the data they store safe, but are less prone to

attacks compared with existing models, which suffer from single point-of failures and are very attractive targets for hackers [53] [47] [62]. Security-conscious individuals can run their own escrows easily with a Raspberry PI, for instance, on their home network [57].

The common interface each escrow should implement is listed in Table 3.2.

Method	Relative Path	Parameters	Response	Description
POST	/keyexchange	**userId**: user id **pubkey**: user public key	See Listing 3.1	Creates a new association between the escrow and the client issuing the request. The asymmetric keys exchanged will be used in all future communications to ensure security and authentication.
POST	/connectservice	**userId**: user id **userServiceId**: $SHA_{256}(I_u, I_s)$ **proofCipher**: authentication **proofTag**: authentication KEM **verificationCode**: service generated verification code **txid**: Bitcoin transaction ID	See Listing 3.2	Update escrow for connected service
POST	/write	**id**: user-service id **key**: index key for stored data **data**: stored data **proof**: $SHA_{256}(verification + nonce_s)$ - used to authenticate the sender	See Listing 3.3	Store data in escrow
POST	/read	**id**: user-service id **key**: index key for stored data **data**: read operation meta-data (see Listing 3.4) **proof**: $SHA_{256}(verification + nonce_s)$ - used to authenticate the sender	See Listing 3.5	Read stored data
POST	/history	**userId**: user id **userServiceId**: user-service id **proofCipher**: authentication **proofTag**: authentication KEM	See Listing 3.6	Read all stored data

Table 3.2: Escrow common interface

```
1  success: [true / false],
2  data:
3    tag: [Key Encapsulation Mechanism tag],
4    cipher: [encrypted output of: escrowId, escrowPubkey]
5  [,error: [error description]
```

Listing 3.1: Escrow keyexchange response

```
1  success: [true / false]
```

```
2    [,error: [error description]
```

Listing 3.2: Escrow connect service response

```
1    success: [true / false]
2    [,error: [error description]
```

Listing 3.3: Escrow write response

```
1    timestamp: [unix time],
2    title: [operation title. E.g. 'Reading messages'],
3    reason: [operation reason. E.g. 'User requested to read messages']
```

Listing 3.4: Escrow meta-data

```
1    success: [true / false],
2    data: [encrypted data]
3    [,error: [error description]
```

Listing 3.5: Escrow read response

```
1    success: [true / false],
2    data: [encrypted data]
3    [,error: [error description]
```

Listing 3.6: Escrow history response

3.5.2 Storage

There are two storage locations, each for a different purpose. First is the distributed escrow servers. They guard encrypted sensitive data and potentially store information for a single user. Second is the Bitcoin blockchain. There is only meta-data stored in the format of hashes to keep a small memory footprint and avoid bloating the blockchain. It's main purpose is to keep a durable, irreversible and transparent audit trail of actions (see Appendix B).

Figure 3-9: Invisible Ink dashboard

3.5.3 Blockchain

The blockchain we chose to use is Bitcoin's. Because the transactions stored inside the blockchain are fairly simple and comply with Bitcoin's protocol it is unnecessary to create a brand new blockchain. By using Bitcoin's blockchain we can enjoy its existing network and underlying coin. Other blockchains are available as well, but Bitcoin is by far the most common protocol.

3.5.4 Transaction explorer

To go through all the transactions and see how our data is being manipulated we have created a web explorer that displays everything ordered by time and services (see Figure 3-3). Our next steps will include a search tool that will allow easy navigation between different transactions and events.

3.5.5 Dashboard

To provide the user with an easy-to-use interface for the information shared with services and their corresponding audit events, we created a web dashboard. Here users will be able to monitor their data, link and unlink accounts, define and manage

escrows, copy data between services and more (see Figure 3-9).

3.5.6 Server

The server doesn't keep any information in plain-text nor can it decrypt any sensitive information it stores. The keys are generated in the client side and only the client knows the credentials used to generate them. This is also dangerous in case clients forget their credentials, because the server wouldn't be able to recover them.

The server has two important tasks:

- Stores encrypted credentials for its users. Providing a fluid user experience while maintaining zero-knowledge about the data it stores. Besides logging-in to the dashboard (see Section 3.5.5) from multiple clients, users will have many different keys shared with different services. Remembering a 32 character long password is not easy.
- For each user, the server stores an encrypted list of all its linked services. The server will never be able to know details about these services as they are encrypted and decrypted only by the client.

3.5.7 Database

There are two main actors: Invisible Ink server and the escrow server. Each actor has its own database. See Table 3.4 for the Invisible Ink database schema and Table 3.6 for the escrow database schema.

Both of these are subject to changes and optimization. For example, if you choose to create your own escrow service - your only requirement is to implement its interface. How you store, manage, index, etc. the data, that's for you to decide.

Key	Type	Default	Description
id	String		User unique identifier
auth_token	String		API temporary authentication token
profile	String		Encrypted profile data
services	Array		List of encrypted service data
nonce	Number	0	Anti replay-attack variable sent in each request. Increments after each successful request

Table 3.4: Invisible Ink database schema

Key	Type	Default	Description
clients			
user_id	String		User unique identifier
user_pubkey	String		User public key
escrow_id	String		Escrow unique identifier
escrow_pubkey	String		Escrow public key
escrow_prvkey	String		Escrow private key
nonce	Number	0	Anti replay-attack variable sent in each request. Increments after each successful request
services			
id	String		Service unique identifier
data	Object		Encrypted service data that belongs to the user
verification	String		Service generated verification code
nonce	Number	0	Anti replay-attack variable sent in each request. Increments after each successful request

Table 3.6: Escrow database schema

3.6 Failed protocol

In Bitcoin there are several ways to store arbitrary data inside the blockchain. Each with its own pros and cons. In this section we describe our initial attempts to store and manage arbitrary data in a decentralized network.

3.6.1 Introduction

Our first attempt at designing Invisible Ink didn't include escrow services. Instead, the data was stored in existing third-party cloud storage, such as Dropbox and Google Drive [17] [28]. The auditing process was by storing log-files inside these services. The files were hashed periodically and the result was stored inside the blockchain. By keeping the periodic hashes inside a durable, irreversible, time-based and distributed database people could know at any point in time whether the log-file had been altered.

In most major cases, we have chosen to use the Bitcoin testnet compared with mainnet and regtest. We made this decision based on two main factors: first to play with fake bitcoins and second to enjoy existing block explorers and other bootstrapping services.

3.6.2 Transactions

Bitcoin enables a special transaction that can hold *40 bytes* of arbitrary information. We will use this transaction to keep our meta-data inside the blockchain.

The generic format we will use is:

`OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <DATA, 34 bytes>`

- `ID` will always be "`PRVCY`" (`0x5052564359`)
- `TYPE` can be:
  - `0x01: log`
  - `0x02: link`
  - `0x03: unlink`
  - `0x04: data`

3.6.3 Log transaction: attempt 1 - *testnet* only

Storing arbitrary data inside a transaction that uses multiple `OP_RETURN` scripts.

This implementation is valid only in the *testnet* because it uses a transaction with multiple `OP_RETURN` outputs which is blocked in the *mainnet*.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <SERVICE ID, 2 bytes>
           <NUM OF ENTRIES, 5 bytes> <USER HASH, 20 bytes>
[output 1] OP_RETURN <FILE HASH, 32 bytes>
```

- `SERVICE ID` helps to identify the service. An encrypted name is associated with this id inside our server.

- `NUM OF ENTRIES` represents the number of hashed log entries. This value is used to authenticate the integrity of the log-file.

- `USER HASH` is the user's pseudonym that is randomly generated for each of its services. The user uses this hash to find its link transactions. We get a secure 20 byte hash by using a $RIPEMD_{160}$ hash of the SHA_{256} *user id.* I.e. $RIPEMD_{160}(SHA_{256}(userId))$.

- `FILE HASH` is a SHA_{256} hash of the log file. The file itself will be stored in the cloud storage. Its authenticity can be proved by hashing the file and comparing it to this value.

Example

```
[output 0] OP_RETURN 5052564359 01 0001 0000000002
           51d75544b04a9471eec80d5c1b8f5e127b093582
[output 1] OP_RETURN
           f094ce936bdef34e1d63109cf3fe8dd21801e4a470309da63dbf3a49...
```

Live example

`http://blockexplorer.com/testnet/tx/`
`4bfec6ed3067265e66900a3e621e83905eef02e4d94ec217080aaffa646fe4d1`

3.6.4 Log transaction: attempt 2 - erasing *satoshis*

Storing arbitrary data inside a *Pay To Public Key Hash (P2PKH)* script.

Although this implementation is valid in the *mainnet*, it is much worse than the one above because it requires sending bitcoins to a fake address. There is a minimum limit on the amount sent to an address, and it's called *dust* [48]. Currently it is set to *5460 satoshis* ($0.0128856USD$, when $1BTC = 236USD$). Besides the extra cost, the *5460 satoshis* sent to the fake address are now lost forever because no one has the private key for the fake address.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <SERVICE ID, 2 bytes>
           <NUM OF ENTRIES, 5 bytes> <USER HASH, 20 bytes>
[output 1] <ID, 5 bytes> <TYPE, 1 byte> <FILE HASH, 32 bytes>
           OP_CHECKSIG
```

Example

```
[output 0] OP_RETURN 5052564359 01 0001 0000000002
           51d75544b04a9471eec80d5c1b8f5e127b093582
[output 1] 5052564359 01
           f094ce936bdef34e1d63109cf3fe8dd21801e4a470309da63dbf3a49...
           OP_CHECKSIG
```

Live example

`https://blockchain.info/tx/`
`6067591b805ca060214777e5ad1644bcdbd5af9393f3282332030172f0833a2f`

3.6.5 Log transaction: attempt 3 - hacking *multisig*

Storing arbitrary data inside a *multisig public key* script.

`M of N OP_CHECKMULTISIG` is standard (meaning, it will propagate and eventually be inserted into the blockchain) if `N <= 3`. Each public key can be anywhere between `33 (compressed) to 65 (raw) bytes`.

To go around the previous problem where we "burned" funds, here we use public keys as placeholders for our random data. This gives us `130 bytes` of data. The third key is a real address that we own. Later, we can use these funds for any arbitrary need. This raises one big problem, identity linkage. If we buy something with these funds someone could link our identity with this *log-transaction.* One possible solution is CoinJoin [43]. In a nutshell, it combines several keys together as inputs and creates "new keys" as outputs. This makes it hard to guess which key belongs to which output and if repeated a couple of times makes it nearly impossible to identify the rightful

owner. Although, each such cycle requires paying transaction fees.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <USER HASH, 20 bytes>
[output 1] OP_1
           [key 0] <ID, 5 bytes> <TYPE, 1 byte> <SERVICE ID, 2 bytes>
                   <NUM OF ENTRIES, 5 bytes> <USER HASH, 20 bytes>
           [key 1] <ID, 5 bytes> <TYPE, 1 byte> <FILE HASH, 32 bytes>
           [key 2] <real address>
           OP_3 OP_CHECKMULTISIG
```

Example

```
[output 0] OP_RETURN 5052564359 01
           51d75544b04a9471eec80d5c1b8f5e127b093582
[output 1] OP_1
           505256435901000100000000251d75544b04a9471eec80d5c1b8f5e
           127b093582 505256435901f094ce936bdef34e1d63109cf3fe8dd21
           801e4a470309da63dbf3a49955d9579 03ab48e4ece8f1c7ffb4b59a
           154e6ef88d3fed23dea6fcc166bb7b02f3e7232c63
           OP_3 OP_CHECKMULTISIG
```

Live example

```
http://blockexplorer.com/testnet/tx/
6b7e12f3cf813da5f2baaaa0c6223497a7e4fb3636cf94d3d451f229b01108f1
```

3.6.6 Log transaction: attempt 4 - legally hacking *multisig*

A similar attempt to the previous one with only public keys with exactly `33` or `65` `byte` length for compressed and uncompressed respectively. This attempt was later proved to be somewhat useless compared with Attempt 3.

Although in the previous attempt the transaction was marked "standard" and

reached the pool of unconfirmed transactions, it took many days until it eventually propagated and was added to a block. We tried a couple of transaction fees to encourage miners to add it to a block but non prevail immediately. We believed that the reason it didn't work was that the fake public keys weren't "standard". For a public key to be standard it must be either `33 or 65 bytes` exactly and have a prefix of `0x02/0x03` and `0x04` respectively (as far as we knew while making this attempt). That led us to this attempt.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <USER HASH, 20 bytes>
[output 1] OP_1
           [key 0] <compressed public key prefix, 1 byte>
                   <ID, 5 bytes> <TYPE, 1 byte> <SERVICE ID, 2 bytes>
                   <NUM OF ENTRIES, 4 bytes> <USER HASH, 20 bytes>
           [key 1] <uncompressed public key prefix, 1 byte>
                   <ID, 5 bytes> <TYPE, 1 byte> <FILE HASH, 32 bytes>
                   <RESERVED, 26 bytes>
           [key 2] <real address>
           OP_3 OP_CHECKMULTISIG
```

Example

```
[output 0] OP_RETURN 5052564359 01
           51d75544b04a9471eec80d5c1b8f5e127b09358
[output 1] OP_1
           505256435901000100000000251d75544b04a9471eec80d5c1b8f5e
           127b093582 04505256435901f094ce936bdef34e1d63109cf3fe8dd
           21801e4a470309da63dbf3a49955d957900000000000000000000000
           00000000000000000000000000000000 03613a80d61c79d4ba7e870413
           3f63e53435add99275bfd894bab1f700e90dc8fd
           OP_3 OP_CHECKMULTISIG
```

Live example

https://blockchain.info/tx/

42409ab67cd856ecf648e1c63eaff23bf99ad8a5e8793f31812bfa6eb30c6112

3.6.7 Log transaction: attempt 5 - conservative *multisig*

A similar attempt to the previous one with only one fake public key that stores arbitrary data.

In the previous attempt we used three addresses in which only one is valid and the first two are meta-data. Here we'll use only two output addresses, combining both meta-data outputs into a single one. This reduces the amount of total bytes we're using and reduces the risk of someone owning one of our fake addresses.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <USER HASH, 20 bytes>
[output 1] OP_1
           [key 0] <ID, 5 bytes> <TYPE, 1 byte> <SERVICE ID, 2 bytes>
                   <NUM OF ENTRIES, 5 bytes> <USER HASH, 20 bytes>
                   <FILE HASH, 32 bytes>
           [key 1] <real address>
           OP_2 OP_CHECKMULTISIG
```

Example

```
[output 0] OP_RETURN 5052564359 01
           51d75544b04a9471eec80d5c1b8f5e127b09358
[output 1] OP_1
           505256435901000100000000251d75544b04a9471eec80d5c1b8f5e
           127b093582f094ce936bdef34e1d63109cf3fe8dd21801e4a470309d
           a63dbf3a49955d9579 03613a80d61c79d4ba7e8704133f63e53435a
           dd99275bfd894bab1f700e90dc8fd
           OP_2 OP_CHECKMULTISIG
```

Live example

`https://blockchain.info/tx/`

`3cd32f8e095162596af58f9ef89833ce3926d4bd18b8faa18bceefd47c3a527c`

3.6.8 Data transaction: attempt 1 - non-encrypted

Protocol to store arbitrary data inside the blockchain. Data stored in clear-text.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <SERVICE ID, 2 bytes>
           <CATEGORY, 12 bytes> <USER HASH, 20 bytes>
[output 1] OP_1
           [key 0] <KEY LENGTH, 1 byte> <KEY, x bytes>
                   <VALUE LENGTH, 1 byte> <VALUE, y bytes>
                   [<SUFFIX, 63 - x - y bytes>]
           [key 1] <real address>
           OP_2 OP_CHECKMULTISIG
```

- `SERVICE ID` identifies the service. An encrypted name is associated with this id inside our server.
- `CATEGORY` index for the type of data stored.
- `USER HASH` is the user's pseudonym that is randomly generated for each of its services. The user uses this hash to find its link transactions. We get a secure 20 byte hash by using a $RIPEMD_{160}$ hash of the SHA_{256} *user id.* I.e. $RIPEMD_{160}(SHA_{256}(userId))$.
- `KEY LENGTH` the length of the following key.
- `KEY` variable length data-key.
- `VALUE LENGTH` the length of the following value.
- `VALUE` variable length data-value.

- SUFFIX variable length appended zeros to complete a 65 bytes length fake public key.

Example

```
[output 0] OP_RETURN 5052564359 04 0001
           000000000067656e65736973
           51d75544b04a9471eec80d5c1b8f5e127b093582
[output 1] OP_1
           06 746865736973 2b 5072697661746520426c6f636b202d20426c6
           f636b636861696e20466f722044617461205072697661637 9 000000
           000000000000000000000 f094ce936bdef34e1d63109cf3fe8dd21
           801e4a470309da63dbf3a49955d9579
           OP_2 OP_CHECKMULTISIG
[output 2] OP_1
           04 676f616c 35 5468657369732050726f706f73616c20666f72200
           d0a4d6173746572206f6620536369656e636520696e204d415320617
           4204d4954 000000000000 f094ce936bdef34e1d63109cf3fe8dd21
           801e4a470309da63dbf3a49955d9579
           OP_2 OP_CHECKMULTISIG
```

Live example

https://blockchain.info/tx/
2ecae24a049993142260861d32275d40461936679e9efa9c2504cacd4048914c

3.6.9 Data transaction: attempt 2 - encrypted

Protocol to store arbitrary data inside the blockchain. Data stored in encrypted format.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte>
```

```
            <USER-SERVICE ID, 20 bytes>
[output 1] OP_1
            [key 0] <CATEGORY OP CODE, 6 bytes> <CATEGORY HASH, 20 bytes>
                    <DATA LENGTH, 2 bytes> <SUFFIX, 5 bytes>
            [key 1] <DATA(i), x bytes> [<SUFFIX, 33/65 - x bytes>]
            [key 2] <real address>
            OP_3 OP_CHECKMULTISIG
[output 2] OP_1
            [key 0] <DATA(i+1), x bytes> [<SUFFIX, 33/65 - x bytes>]
            [key 1] <DATA(i+2), x bytes> [<SUFFIX, 33/65 - x bytes>]
            [key 2] <real address>
            OP_3 OP_CHECKMULTISIG
```

- `USER-SERVICE ID` used to index the transactions so that both the user and the service could easily find their corresponding transactions.
- `CATEGORY OP CODE` a hex representation of the word "`catgry`" (`0x636174677279`)
- `CATEGORY HASH` is the sub-index used by the user or service to search for the data that is stored in this transaction. We get a secure `20 byte` hash by using a $RIPEMD_{160}$ hash of the SHA_{256} category key. I.e. $RIPEMD_{160}(SHA_{256}(key))$.
- `DATA LENGTH` a `2 byte` integer which represents the length of the data.
- `SUFFIX` appended zeros. Reserved for future implementations.
- `DATA(i+x)` a `65 byte` maximum chunk of stored encrypted data.

Example

```
[output 0] OP_RETURN 5052564359 01
            b7192ee98abfc31db4af03dd90711d403e426ba6
[output 1] OP_1
            636174677279 2483cbf25e014c9c8ffd1cf1a309917c7ca87a18 00
            21 0000000000 da50c921d289bb05230f5b761decb7f92caba2eb96
```

8e00972b107aaed24d379e f094ce936bdef34e1d63109cf3fe8dd21
801e4a470309da63dbf3a49955d9579
OP_3 OP_CHECKMULTISIG

3.6.10 Link transaction

Using a simple version of an OP_RETURN script.

Protocol

[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte>
<USER-SERVICE ID, 20 bytes>

Example

[output 0] OP_RETURN 5052564359 02
b7192ee98abfc31db4af03dd90711d403e426ba6

3.6.11 Unlink transaction

Using a simple version of OP_RETURN.

Protocol

[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte>
<USER-SERVICE ID, 20 bytes> <SECURITY BYTES, 12 bytes>

- SECURITY BYTES protects against replay attacks (*user-service id* is randomly generated each time a user links to a service) and from malicious attempts to create this transaction. It is the last 12 bytes of the SHA_{256} of the *user id*.

Example

[output 0] OP_RETURN 5052564359 03
b7192ee98abfc31db4af03dd90711d403e426ba6
4cf6e11d89c7c4202b0acb39

3.7 The protocol

In this section we will describe Invisible Ink's protocol and provide real-world usages.

First we will present the top three choices of scripts we use to store arbitrary data inside the Bitcoin blockchain.

3.7.1 Transaction 1: OP_RETURN

Bitcoin protocol enables a special transaction that can hold up to `40 bytes` of arbitrary information without needing to send any *satoshis* (value) to anywhere.

Protocol

```
OP_RETURN <DATA, 40 bytes>
```

Live example

`https://blockchain.info/tx/`

`dd66ea35f78e8d9fac170e30a64231ff4d9b680502bd05644d69fb17fa93cec2`

3.7.2 Transaction 2: OP_RETURN + *multisig*

It is illegal to have more than one `OP_RETURN` output script in a single transaction, therefore we can use a different script type. A multi-signature transaction is a very important script format. One of its benefits is that it increases the security for bitcoin capital. Instead of requiring a single key to unlock an *Unspent Transaction Output (UTXO)*, it allows the requirement of more than one.

For example, a company can ensure that at least X out of the total Y board members are required to sign a transaction and pay for some service. It is similar to a democratic voting procedure in which to win you would have to get at least X votes. In our analogy, winning is the ability to create a valid transaction.

`M of N OP_CHECKMULTISIG` is standard, meaning that it will propagate and eventually be inserted into the blockchain if `N <= 3`. Each public key can be anywhere

between 33 (compressed) to 65 (raw) bytes. Here we use public keys as placeholders for arbitrary data. This gives us 130 bytes of data. The third key should always be a real address that we own. Later, we can use these funds for any arbitrary need. If we don't put a real address, then the funds we send are lost forever. This has three major implications: reducing the supply of bitcoins, placing a permanent memory footprint for each full-node and bloating the size of the blockchain which effectively decreases its potential to scale. Full nodes that hold the blockchain also store in RAM a table for all the UTXOs.

Using an address we own and later spend raises one big problem, identity linkage. If we buy something with these funds someone could link our identity with this type of transaction. One possible solution is CoinJoin [43]. In a nutshell, it combines several keys together as inputs and creates "new keys" as outputs. This makes it hard to guess which key belongs to which output and if repeated a couple of times makes it nearly impossible to identify the rightful owner. However, each such cycle requires paying transaction fees.

Protocol

```
[output 0] OP_RETURN <DATA, 40 bytes>
[output 1] OP_1
           [key 0] <DATA, 33/65 bytes>
           [key 1] <DATA, 33/65 bytes>
           [key 2] <real address>
           OP_3 OP_CHECKMULTISIG
```

Live example

https://blockchain.info/tx/
42409ab67cd856ecf648e1c63eaff23bf99ad8a5e8793f31812bfa6eb30c6112

3.7.3 Transaction 3: OP_RETURN + *P2SH multisig*

One of the biggest drawbacks for almost any data-related transaction is that the transaction, once accepted, is stored in RAM within all full-nodes in the network until it is spent. It may turn out to be a problem when many such transactions will appear and start filling the entire heap size for all wallets with non-currency based unspent transaction outputs. However, there shouldn't be too much of these for each wallet, because one could always spend such transactions first, leaving a maximum of a single data-transaction. Clients can improve that by making sure that these transactions are spent after a certain time-frame, thus leaving no memory footprint behind.

An alternative approach is a bit more complex but much more memory conservative. Instead of leaving a "heavy" transaction output, use a *pay-to-script-hash (p2sh)* for multi signature and to store the data, and use the redeem script later. This makes sure that a small footprint is stored in RAM for all full-nodes (because unspent transaction outputs are stored in memory), but on the other hand, to complete the data transaction you would have to wait for your first transaction to be included inside the blockchain and then submit another transaction with your data.

With respect to the blockchain size, if Satoshi's pruning approach will come into effect, then this type of transaction would still leave a permanent footprint inside the blockchain and increase its size until spent.

Protocol

```
[output 0] OP_RETURN <DATA, 40 bytes>
[output 1] OP_HASH160 <REDEEM SCRIPT HASH> OP_EQUAL
```

- REDEEM SCRIPT HASH have more flexible restriction on the number of public keys used. Here's an example of such script:

```
OP_1
        [key 0] <DATA, 33/65 bytes>
        [key 1] <DATA, 33/65 bytes>
        [key 2] <DATA, 33/65 bytes>
```

```
        [key 3] <DATA, 33/65 bytes>
        [key 4] <real address>
    OP_5 OP_CHECKMULTISIG
```

3.7.4 Selected transaction type

Our system will mainly use the second type of transaction (see Section 3.7.2) to store the meta-data. The reason for our choice is that the memory footprint will stay low due to the extensive use of compressed hashes instead of plain data. The wallets that will create these transactions will set their priority to maximum, and therefore they will always make sure they get spent first.

The following sections will elaborate on the transactions used in Invisible Ink.

3.7.5 Header transaction

All transactions will begin with a *header transaction* that uses an `OP_RETURN` script to store `40 bytes` of information.

Protocol

```
OP_RETURN <ID, 5 bytes> <TYPE, 1 byte>
<USER-SERVICE ID, 20 bytes> <KEY, 14 bytes>
```

- `ID` will always be "`PRVCY`" (`0x5052564359`)
- `TYPE` can be:
  - `0x01`: connect
  - `0x02`: disconnect
  - `0x03`: audit
- `USER-SERVICE ID` is the pseudonym that is randomly generated each time a user and a service connect. It is used to identify and index transactions. We get a secure `20 byte` hash by using a $\text{RIPEMD}_{160}\left(SHA_{256}\left(H_{u,s}\right)\right)$.

- `KEY` is used to identify and index the type of audit stored in the transaction. It uses the last `14 bytes` of a $RIPEMD_{160}$ over a SHA_{256} hash of the *key-index*. I.e. Last14Bytes (RIPEMD$_{160}$ (SHA$_{256}$ (*key*))).

3.7.6 Audit transaction

This type of transaction is used when the following actions are performed on user data: read, write, update, revoke/grant permission, etc.

Protocol

`[output 0]` ***Header transaction*** **(see Section 3.7.5)**

```
[output 1] OP_1
           [key 0] <ULB, 1 byte> <SIGNATURE-X, 64 bytes>
           [key 1] <CLB, 1 byte> <VALUE, 20 bytes>
                   <SIGNATURE FIRST BYTE, 1 byte> <UNUSED, 11 bytes>
           [key 2] <real address>
           OP_3 OP_CHECKMULTISIG
```

- `ULB` The Uncompressed Leading Byte equal to `0x04`. In order to create an uncompressed valid public key it is required that it will lead with a `0x04` value.
- `CLB` The Compressed Leading Byte equal to `0x03`. This can be anything besides `0x04` in order to mark this public key as a compressed key.
- `SIGNATURE-X` The signature without the first byte. It is the proof that validates this transaction to be a part of the *user-service* audit trail. It is the compressed output of $ECDSA.sign\,(value, P_r)$, where P_r is a private key that belongs to either the user or the service. In order to create a valid uncompressed public key the first byte is replaced by a leading uncompressed byte equal to `0x03`. The replaced byte can be found in the second public key.
- `VALUE` is the `20 byte` hash representing the value stored. This hash is obtained by performing a $RIPEMD_{160}$ over a SHA_{256} hash of the *value*. I.e. $RIPEMD_{160}(SHA_{256}\,(value))$.

- SIGNATURE FIRST BYTE the signature's leading byte.
- UNUSED appended 0's padding to 33 bytes.

Live example

https://blocktrail.com/tBTC/tx/

527636fdd3a5d7c0339ade27143a1be1f0f579e71c73155e06801254d1a9df02

3.7.7 Connect transaction

Records the event of connecting an account to a service.

Protocol

[output 0] ***Header transaction*** **(see Section 3.7.5)**

[output 1] OP_1

[key 0] <ULB, 1 byte> <$PUBKEY_1$ HASH, 20 bytes>
<$PUBKEY_2$ HASH, 20 bytes> <VERIFICATION, 20 bytes>
<FREQUENCY TYPE, 1 byte> <FREQUENCY, 2 bytes>
<UNUSED, 1 bytes>

[key 1] <real address>

OP_2 OP_CHECKMULTISIG

- ULB The Uncompressed Leading Byte equal to 0x04. In order to create an uncompressed valid public key it is required that it will lead with a 0x04 value.
- $PUBKEY_{1|2}$ HASH is the $RIPEMD_{160}$ hash of the compressed public key used to verify future audit transaction signatures.
- VERIFICATION is used to validate the authenticity of a *connect-transaction*. The service, which issues the *validation-code*, can associate such transaction with its user. It is the $RIPEMD_{160}$ of a SHA_{256} hash of the validation code sent to the user. I.e. $\text{RIPEMD}_{160}(\text{SHA}_{256}(validation\_code))$.
- FREQUENCY TYPE defines the type of frequency used in the following parameter:

0x01: years

0x02: months

0x03: days

0x04: hours

0x05: minutes

0x06: seconds

0x10: number of transactions

- FREQUENCY defines the upper bound limit recurrence for future audit transactions. I.e. if frequency type is days and frequency is 1, then every day the service will broadcast one *audit-transaction* which contains all the operations it had done during that day if at least one operation was performed.

Live example

https://blocktrail.com/tBTC/tx/
a3a2d10e0d72cbe37283476507fa4c8e3dbe1ae5ab1472173522c8cf38e11440

3.7.8 Disconnect transaction

Records the event of disconnecting an account from a service.

Protocol

[output 0] ***Header transaction*** **(see Section 3.7.5)**

```
[output 1] OP_1
           [key 0] <ULB, 1 byte> <SIGNATURE-X, 64 bytes>
           [key 1] <CLB, 1 byte> <SIGNATURE FIRST BYTE, 1 byte>
                   <UNUSED, 31 bytes>
           [key 2] <real address>
           OP_3 OP_CHECKMULTISIG
```

- ULB The Uncompressed Leading Byte equal to 0x04. In order to create an uncompressed valid public key it is required that it will lead with a 0x04 value.

- `CLB` The Compressed Leading Byte equal to `0x03`. This can be anything besides `0x04` in order to mark this public key as a compressed key.

- `SIGNATURE-X` The signature without the first byte. It is the proof that validates this transaction to be a part of the *user-service* audit trail. It is the compressed output of $ECDSA.sign\,(0, P_r)$, where P_r is the private key that belongs to the user and its corresponding public key is stored inside the *connect-transaction.* In order to create a valid uncompressed public key the first byte is replaced by a leading uncompressed byte equal to `0x03`. The replaced byte can be found in the second public key.

- `SIGNATURE FIRST BYTE` the signature's leading byte.

- `UNUSED` appended 0's padding to `33 bytes`.

Chapter 4

System Applications

In this chapter we describe an application we have built on top of the Invisible Ink framework. We call it *Certified Mail.* It allowed us to test our platform and evaluate its potential usage in real-world applications. We released and demonstrated *Certified Mail* at the spring member's event at the Media Lab.

4.1 Certified Mail

4.1.1 The challenge

The postal service has a mechanism to ensure the sending, tracking and receiving of a mail. That is called *"Certified Mail"*. The value it brings is trivial; we can prove we sent a mail and that our mail has been received by the recipient. The postal service only operates as a proxy. It doesn't keep a copy of our mail nor does it open it and read its content. What about its digital companion? How can we track a message sent online while ensuring its privacy?

4.1.2 Our solution

We created a messaging service that allows people to send encrypted messages that can only be decrypted by the recipients. The messages are stored in a trusted blind escrow (see Section 3.5.1) and are audited inside the Bitcoin blockchain.

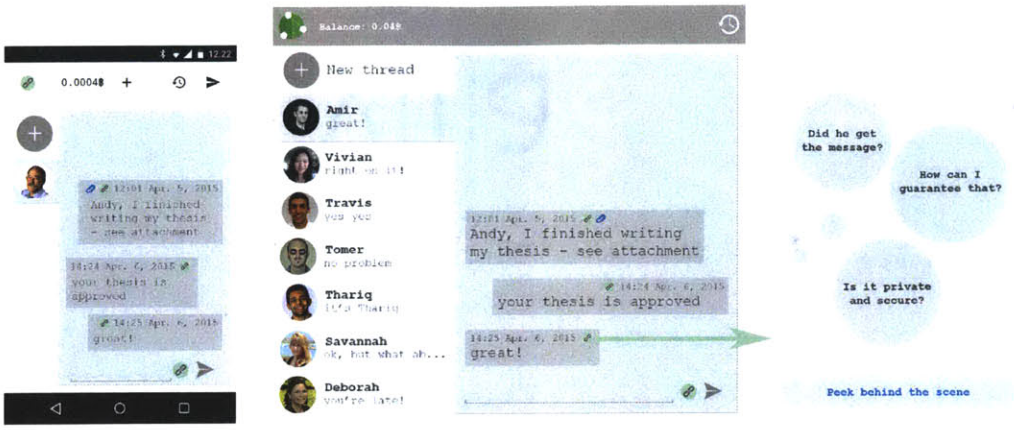

Figure 4-1: Certified Mail - proof of communication

4.1.3 Application overview

After creating an account (see Section 3.3.1) in Invisible Ink, you authenticate your profile in *Certified Mail* application. The authentication (see Figure 3-6) process is performed behind the scene and is transparent and smooth for the user.

Part of the authentication involves creating a contract between you and the service that describes your relations with the service and the permissions to your data that you provide. For example, how long can the service temporarily store sensitive information (used to provide a good user experience). Another important part of the authentication is generating asymmetric keys that will be used to encrypt messages. To send someone a message, you would use their public key for encryption. This way, *Certified Mail* will not be able to read its users conversations and only be used as a transport medium and messaging interface.

Once authenticated, you can immediately start communicating with your family, friends and colleagues. Figure 4-1 presents the interface used to send messages in the *Certified Mail* application.

In addition to any other messaging application, *Certified Mail* provides an easy way to add an *audit-transaction* (see Section 3.7.6) for each message sent. Every transaction in the Bitcoin network has a cost. This cost is usually very low and will be accepted by the network based on the miner's decision and optimization algorithm. Miners in the network are incentivized to validate and propagate transactions based on

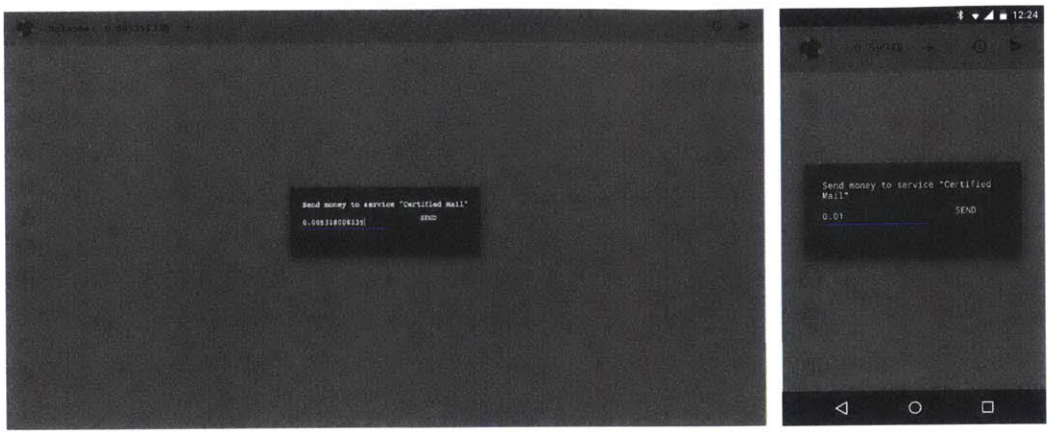

Figure 4-2: Certified Mail - send money to the service

the fees transactions leave for them. If for example, the miner has more transactions waiting than it can include in the next block due to the size limit, it will choose the top paying transactions to gain the maximum value. In *Certified Mail* we adjusted the fee to be \$0.01 based on the current value of Bitcoin, but we might have succeeded with lower values as well.

Certified Mail pays for transactions on behalf of its users for each message they send that requires auditing. Auditing a message is done by creating a Bitcoin transaction and sending it to the network so it will be stored inside the blockchain. Each account in *Certified Mail* has a Bitcoin wallet. The funds in each wallet are only used for auditing purposes. Figure 4-2 presents a web and mobile interface for sending money to *Certified Mail*.

Because *audited-messages* are "expensive" (approximately \$0.01), we included a function that can temporary disable immediate transactions. However, these *non-audited messages* can eventually become audited. *Non-audited messages* are hashed and temporarily stored inside a *Merkle tree* data structure [45]. Once an *audited-message* is sent, it includes with it the *Merkle root* that belongs to all the previously hashed *non-audited messages*, if one exists. Later, if needed, all the *non-audited messages* between two given *audited-messages* can be proved for existence by providing the entire trail of *non-audited messages* in the desired range.

Chapter 5

Conclusion

We have presented a system called Invisible Ink for managing sensitive data in a distributed, scalable and secured manner. Our system is using the Bitcoin blockchain as a trust-less distributed database that stores both the audit trail and permissions granted to user-data.

The audit trail is used as an immutable, ordered by time, record for sensitive user-data manipulation. Manipulation such as read, write, update and delete performed by a registered service.

Registered services are granted permissions to interact on sensitive user-data by the user. These permissions are stored as contracts inside the Bitcoin blockchain and themselves provide proof of user-service relationship.

The way we chose to store sensitive user-data is by using something we call a *blind escrow service.* The data stored in these services are encrypted with keys that will never be present on the escrows' physical machines. That creates a two-factor decryption step, where a malicious actor would need to control at least two machines out of the service, the user and the user's blind escrow in order to decipher sensitive data.

To evaluate our system we have built a service that uses Invisible Ink as its underlying sensitive user-data management platform. We call our application *Certified Mail.*

Certified Mail demonstrates the agility and flexibility of Invisible Ink by offering

a unique messaging service. It allows people to send contract-based messages where each message is stored in a distributed user-trusted escrow machine and logged inside the Bitcoin blockchain to provide a proof-of-communication.

5.1 Implications and opportunities

The need for data privacy will probably grow stronger as more devices and services collect data on us. We are slowly transitioning into an era of the *Internet of Things* where all electronic devices, appliances and even our clothes will be connected to the Internet and enhance our interactions with them.

This new era brings with it many challenges, one of which is securely storing sensitive and personal data. When your shower knows how many times you take a shower, your house knows when you are usually home and your refrigerator knows what you eat - you might want to consider who has access to such information.

Our platform can scale with the growing number of collected data while keeping it secured and stored in a trusted location. Moreover, when people will be in control over their personal data, when they could monitor its usages and revoke access to services at their will, we believe it might introduce a new financial model for personal data. If Facebook, for example, will want to perform experiments on its users, it will need to give them an incentive to agree to share user sensitive information.

Another interesting application might be to use the blockchain as a trusted code system distribution in which an application code is stored inside the blockchain and an external machine runs it. Read more about it in Appendix A.

Appendix A

A Trusted Code System Distribution On The Blockchain

Here we describe an application stored on the Bitcoin blockchain and a machine that runs it and prints its result.

A.1 How to create functions with parameters inside the Bitcoin blockchain

Create a transaction with two outputs:

- `P2SH`: encode your function inside the public keys and use one to store a real address
- `OP_RETURN`: store your function's signature, meaning which parameters it accepts and its return value

To run this function, create another transaction with two outputs:

- `P2SH`: include the function (all the public keys) and your real address's signature
- `OP_RETURN`: send the parameters that will be delivered to the function

to add namespaces, project references, execution stack, etc. you can replace the `OP_RETURN` script with other `P2SH` or `M of N MULTISIG` and enjoy less size restriction.

Now you can create a virtual machine that goes through the blockchain, looks for projects and their functions, follow the execution cycle and pass the parameters to the functions.

A.2 Can we make this simpler?

Yes, by encoding the code inside a similar transaction to *data-transaction* (see Section 3.6.9). Later these transactions can be used as libraries and be referenced in later transactions.

Protocol

```
[output 0] OP_RETURN <ID, 5 bytes> <TYPE, 1 byte> <NAMESPACE, 20 bytes>
[output 1] OP_1
           [[key 0] <IOC, 6 bytes> <IOM, 3 bytes>
                    <IMPORT, 32 bytes> <SUFFIX, 24 bytes>]
           [key 1] <CODE LENGTH, 1 byte> <CODE, x bytes>
                    [<SUFFIX, 64 - x bytes>]
           [key 2] <real address>
           OP_3 OP_CHECKMULTISIG
[...]
```

- `TYPE` equals to `0xCD`.
- `IOC` Import OpCode is a `6 byte` hex-encoded version of the word "`import`" (`0x696d706f7274`) which is used to recognize an import address.
- `IOM` Import Output Mask is a `3 byte` bit-mask to flag which outputs to import from the transaction.

Example

[transaction id:

ff85fffa85bdea8cc6cd31965b14fca77ae4ef7bc47bdb2d7ea15cfd1b0f751a]

```
[output 0] OP_RETURN 5052564359 CD 00000000000000000000000000000000
           07574696c73
[output 1] OP_1
           28 66756e6374696f6e207072696e74286d7367297b616c657274282
           2676f743a20222b6d7367293b7d 00000000000000000000000000000000
           0000000000000000000 f094ce936bdef34e1d63109cf3fe8dd2180
           1e4a470309da63dbf3a49955d9579
           OP_2 OP_CHECKMULTISIG
```

[transaction id:
f258f5899a662a1053411ff6e5bf3d61b66e03bf15cd9563f32f205995afaf85]

```
[output 0] OP_RETURN 5052564359 CD 00000000000000000000000000000000
           00074657374
[output 1] OP_1
           696d706f7274 000001 ff85fffa85bdea8cc6cd31965b14fca77ae4
           ef7bc47bdb2d7ea15cfd1b0f751a 00000000000000000000000000000000
           0000000000000000000 f094ce936bdef34e1d63109cf3fe8dd218
           01e4a470309da63dbf3a49955d9579
           OP_2 OP_CHECKMULTISIG
[output 2] OP_1
           0f 7072696e74282268656c6c6f22293b 00000000000000000000
           0000000000000000000000000000000000000000000000000000000000
           0000000000000000000 f094ce936bdef34e1d63109cf3fe8dd2180
           1e4a470309da63dbf3a49955d9579
           OP_2 OP_CHECKMULTISIG
```

A virtual machine that will try to run the second transaction will first import output 1 from the first transaction, load the code and then run the second *code-transaction.*

Here's how it will work:

Usage: ***run*** *<tx id>*

`run f258f5899a662a1053411ff6e5bf3d61b66e03bf15cd9563f32f205995afaf85`

1. read transaction id `7258f5899a662a10534110f6e5b73d61b66e03bf15cd95637327205995acaf85`
2. read output type `OP_RETURN`
   (a) verify type "`CD`"
   (b) note namespace: `00000000000000000000000000000000074657374` ("`test`")
3. go over all other outputs in order:
   (a) identify first transaction as import and load output 1 of transaction id `ff85fffa85bdea8cc6cd31965b14fca77ae4ef7bc47bdb2d7ea15cfd1b0f751a`
   (b) loaded: `66756e6374696f6e207072696e74286d7367297b616c6572742822676f743a20222b6d7367293b7d (function print(msg){alert("got: "+msg);})`
   (c) run the code in output 2: `7072696e74282268656c6c6f22293b (print("hello");)`

when running this code in a browser, a pop up will open with the message:

"`got: hello`"

Appendix B

Bitcoin Auditing Transaction Example


```
{
    "status": "success",
    "data": {
        "tx": {
            "hex": "01000000019aeeb4086e518cc3d836fd848bd6f917...",
            "txid": "84999b8d3606da1f0acb18df1e4e94ed8810b31c0f...",
            "version": 1,
            "locktime": 0,
            "vin": [
                {
                    "txid": "5e317cbe178f29bb5cd40098ce68b84f...",
                    "vout": 2,
                    "scriptSig": {
                        "asm": "304402203a61c51d1b79f510a1604...",
                        "hex": "47304402203a61c51d1b79f510ab6..."
                    },
                    "sequence": 4294967295
                }
            ],
            "vout": [
                {
```

```
            "value": 0,
            "n": 0,
            "scriptPubKey": {
                "asm": "OP_RETURN 5052564359038b87e64...",
                "hex": "6a285052564359038b87e6d11def7...",
                "type": "nulldata"
            }
        },
        {
            "value": 0.0000546,
            "n": 1,
            "scriptPubKey": {
                "asm": "1 0412891a41d30cfc98be27b06b62a5-
                       9cabe70d2aefd4d3cb236a503674a02-
                       de7ce61e97fd8df5c6a62373ffb2678-
                       bcbd170873ff19226dd74169ab7af82-
                       de065ae 03ff6bb2e808dd23bf56fac-
                       eae4e41fbf5105749291c0000000000-
                       000000000000 03e9fecc155cdb8ba6-
                       6e409b1c08bfada8286ac3fb38d9b5d
                       4558bd6f32685711a 3
                       OP_CHECKMULTISIG",
                "hex": "51410412891a41d30cfc98be270d2...",
                "reqSigs": 1,
                "type": "multisig",
                "addresses": [
                    "mjDJLM1zVpxLfUSYnUHwxeJkiPM3pL9S1b",
                    "mhE1HkLACgHM2xxtHuvtqgBzML3QuMLnUL",
                    "mmWevFwY4e9AEiCRkxFKcY9HFV6uVcriJT"
                ]
            }
        },
        {
            "value": 0.00059748,
            "n": 2,
            "scriptPubKey": {
```

```
58                          "asm": "OP_DUP OP_HASH160 41
                               c240084bd6fb24ea92eeedc1c94d8f9fa8b86f
                               OP_EQUALVERIFY OP_CHECKSIG",
59                          "hex": "76a91441c240084bd6fb24eaf9fa8b...",
60                          "reqSigs": 1,
61                          "type": "pubkeyhash",
62                          "addresses": [
63                              "mmWevFwY4e9AEiCRkxFKcY9HFV6uVcriJT"
64                          ]
65                      }
66                  }
67              ],
68              "blockhash": "0000000000001729
                   cb8ce22c28c36da278e1efebe5ecbb613a31406a1018988b",
69              "confirmations": 817,
70              "time": 1429123233,
71              "blocktime": 1429123233
72          }
73      },
74      "code": 200,
75      "message": ""
76  }
```

Listing B.1: Bitcoin auditing transaction example. Some long strings were cut for brevity

Appendix C

Third-Party Libraries and Technologies

We used the following libraries and technologies to build our platform:

- **Common Libraries**
  - *bitcoinjs-lib:* https://github.com/bitcoinjs/bitcoinjs-lib
  - *cb-helloblock:* https://github.com/dcousens/cb-helloblock
  - *sjcl:* https://github.com/bitwiseshiftleft/sjcl
  - *royal fork (Bitcoin faucet):* http://faucet.royalforkblog.com
- **Web**
  - *Polymer:* https://www.polymer-project.org/
  - *Browserify:* http://browserify.org/
  - *Grunt:* http://gruntjs.com/
  - *Yo:* http://yeoman.io/
  - **Libraries**
    * *QR code:* http://davidshimjs.github.io/qrcodejs/
    * *sockjs:* https://github.com/sockjs/sockjs-client

    * *scrypt:* https://github.com/tonyg/js-scrypt
    * *buffer:* https://github.com/feross/buffer
    * *paper:* https://github.com/paperjs/paper.js

- **Servers**
  - *MongoDB:* http://www.mongodb.org/
  - **NodeJS:** http://nodejs.org/
    * *express:* http://expressjs.com/
    * *mongoose:* http://mongoosejs.com/
    * *node-uuid:* https://github.com/broofa/node-uuid
    * *body-parser:* https://github.com/expressjs/body-parser
    * *nconf:* https://github.com/indexzero/nconf
    * *random-js:* https://github.com/ckknight/random-js
    * *sockjs:* https://github.com/sockjs/sockjs-node
    * *nodemailer:* https://github.com/andris9/Nodemailer
  - **Python:** https://www.python.org/
    * *M2Crypto*
    * *cyclone*
    * *twisted*
    * *txmongo*
    * *requests*

Bibliography

[1] Abine. Protect your privacy with DoNotTrackMe from Abine. *http://www.abine.com/donottrackme.html*, 2014.

[2] Manuel Araoz. Proof of Existence. *http://proofofexistence.com/*, 2013.

[3] Adam Back. HashCash. *http://www.cypherspace.org/hashcash/*, 1997.

[4] BitTorrent. BitTorrent Labs - BitTorrent Bleep. 2014.

[5] Kukil Bora. Dogecoin, A Digital Currency Similar To Bitcoin, Has Been Hacked, Costing Its Owners Thousands Of Dollars. *http://www.ibtimes.com/dogecoin-digital-currency-similar-bitcoin-has-been-hacked-costing-its-owners-thousands-dollars*, 2013.

[6] Arthur Budovsky. Liberty Reserve. *http://en.wikipedia.org/wiki/Liberty_Reserve*, 2001.

[7] ChangeTip. A Love Button for the Internet. *https://www.changetip.com/*, 2013.

[8] David Chaum. DigiCash. *http://en.wikipedia.org/wiki/DigiCash*, 1990.

[9] David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors. *Advances in Cryptology*. Springer US, Boston, MA, 1983.

[10] Tim Clark. Digicash files Chapter 11 - CNET News. *http://news.cnet.com/2100-1001-217527.html*, 1998.

[11] Coinbase. Bitcoin wallet, for merchants and an exchange. *https://www.coinbase.com/*, 2012.

[12] CounterParty. A platform for free and open financial tools on the Bitcoin network. *http://counterparty.io/*, 2014.

[13] Cypherpunks. Mailing List. *https://www.cypherpunks.to/list/*.

[14] Wei Dai. B-Money. *http://www.weidai.com/bmoney.txt*, 1998.

[15] Yves-Alexandre de Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. openPDS: protecting the privacy of metadata through SafeAnswers. *PloS one*, 9(7):e98790, January 2014.

[16] John R. Douceur. The Sybil Attack. pages 251–260, March 2002.

[17] Dropbox Inc. Dropbox. *https://www.dropbox.com/.*

[18] Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors. *Sybil Attack*, volume 2429 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, October 2002.

[19] Ethereum. A platform for decentralized applications. *https://www.ethereum.org/*, 2014.

[20] Facebook. Facebook Connect. *https://www.facebook.com/notes/facebook/facebook-across-the-web/41735647130*, 2008.

[21] Factom. A Scalable Data Layer for the Blockchain. 2014.

[22] John Fanning, Shawn Fanning, and Sean Parker. Napster. *http://en.wikipedia.org/wiki/Napster*, 1999.

[23] Hal Finney. RPOW - Reusable Proofs of Work. *http://cryptome.org/rpow.htm*, 2004.

[24] Brian Forde. Launching a Digital Currency Initiative âĂŤ Medium. *https://medium.com/@medialab/launching-a-digital-currency-initiative-238fc678aba2*, 2015.

[25] Sarah Gold. The Alternet. *http://www.alternet.cc/*, 2014.

[26] Gold & Silver Reserve Inc. E-Gold. *http://en.wikipedia.org/wiki/E-gold*, 1996.

[27] Google. Google+ Connect. *https://developers.google.com/+/features/sign-in.*

[28] Google. Google Drive. *https://www.google.com/drive/.*

[29] Google Trends. Account Registration. *http://www.google.com/trends/ explore#q=account registration*, 2015.

[30] Alex Gorale. Ripple tfPartialPayment Causes Gox-Style Hack on Justcoin Exchange. *https://www.cryptocoinsnews.com/ripple-tfpartialpayment-causes-gox-style-hack-justcoin-exchange/*, 2014.

[31] Stacey Higginbotham. Check out IBMâĂŹs proposal for an internet of things architecture using BitcoinâĂŹs block chain tech | Gigaom. *https://gigaom.com/2014/09/09/check-out-ibms-proposal-for-an-internet-of-things-architecture-using-bitcoins-block-chain-tech/*, 2014.

[32] Stan Higgins. 8 Million Vericoin Hack Prompts Hard Fork to Recover Funds. *http://www.coindesk.com/bitcoin-protected-vericoin-stolen-mintpal-wallet-breach/*, 2014.

[33] HistoryOfBitcoin. Bitcoin History: The Complete History of Bitcoin [Timeline]. *http://historyofbitcoin.org/*, 2013.

[34] Sushil Jajodia and Jianying Zhou, editors. *FADE: Secure Overlay Cloud Storage with File Assured Deletion*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[35] Ari Juels, editor. *Financial Cryptography*, volume 3110 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[36] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation - Elliptic curve cryptosystems*, 48(177):203–209, January 1987.

[37] Jae Kwon. Tendermint. *http://tendermint.com/docs/tendermint.pdf*, 2014.

[38] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[39] Laszlo. Pizza for bitcoins? *https://bitcointalk.org/index.php?topic=137.msg1195#msg1195*, 2010.

[40] Amir Lazarovich, Guy Zyskind, and Oz Nathan. MIT Bitcoin Project - Ethos. *http://www.mitbitcoinproject.org/winners*, 2014.

[41] Karlin Lillington. PayPal Puts Dough in Your Palm. *http://archive.wired.com/science/discoveries/news/1999/07/20958*, 1999.

[42] MaidSafe. The New Decentralized Internet. *http://maidsafe.net/*.

[43] Gregory Maxwell. CoinJoin: Bitcoin privacy for the real world. *https://bitcointalk.org/index.php?topic=279249.0*, 2013.

[44] David Mazières. The Stellar Consensus Protocol. *https://www.stellar.org/papers/stellar-consensus-protocol.pdf*, 2015.

[45] Ralph Merkle. Method of providing digital signatures, 1982.

[46] Silvio Micali and Ronald L. Rivest. Micropayments Revisited. pages 149–163, February 2002.

[47] MSN. Data store 'attractive target for hackers'. *http://www.msn.com/en-au/news/other/data-store-attractive-target-for-hackers/ar-AA8Hq15*, 2015.

[48] Satoshi Nakamoto. Dust. *https://github.com/bitcoin/bitcoin/blob/b78d1cdf82fb12cc0c8eb9049074b359b9589b7c/src/core.h#L153*.

[49] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *https://bitcoin.org/bitcoin.pdf*, 2008.

[50] Satoshi Nakamoto. Bitcoin P2P e-cash paper. *https://www.mail-archive.com/cryptography@metzdowd.com/msg09959.html*, 2008.

[51] Satoshi Nakamoto. Bitcoin open source implementation of P2P currency. *http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source*, 2009.

[52] Satoshi Nakamoto. Block 0 - The First Block Mined In Bitcoin - Bitcoin Block Explorer. *http://blockexplorer.com/block/000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f*, 2009.

[53] Ellen Nakashima. Chinese hackers who breached Google gained access to sensitive data, U.S. officials say. *http://www.washingtonpost.com/world/national-security/chinese-hackers-who-breached-google-gained-access-to-sensitive-data-us-officials-say/2013/05/20/51330428-be34-11e2-89c9-3be8095fe767_story_1.html*, 2013.

[54] Alyssa Newcomb. Anthem Hack May Have Impacted Millions of Non-Customers as Well - ABC News. *http://abcnews.go.com/Technology/anthem-hack-impacted-millions-customers/story?id=29212840*, 2015.

[55] Onename. Decentralized identity system built on the blockchain. *https://onename.com/*, 2013.

[56] R. Perlman. File System Design with Assured Delete. In *Third IEEE International Security in Storage Workshop (SISW'05)*, pages 83–88. IEEE, 2005.

[57] Raspberry Pi Foundation. Raspberry Pi. *https://www.raspberrypi.org/*, 2012.

[58] ReadNotify. Certified email with delivery receipts, silent tracking, proof-of-opening history, security and timestamps. *https://ssl1.readnotify.com/readnotify/*.

[59] Jon Russell. Coinbase Is Opening The First Regulated Bitcoin Exchange In The U.S. *http://techcrunch.com/2015/01/25/coinbase-us-bitcoin-exchange/*, 2015.

[60] Nick Szabo. Bit gold. *http://unenumerated.blogspot.com/2005/12/bit-gold.html*, 2005.

[61] Nick Szabo. Bitcoin, what took ye so long? *http://unenumerated.blogspot.com/2011/05/bitcoin-what-took-ye-so-long.html*, 2011.

[62] James Titcomb. Bank database 'presents target for hackers and hostile powers'. *http://www.telegraph.co.uk/finance/newsbysector/banksandfinance/11080609/Bank-database-presents-target-for-hackers-and-hostile-powers.html*, 2014.

[63] Nils Toedtmann, H Joerg Baach, and Ryden Mathew. OpenCoin - open source electronic cash. *https://github.com/OpenCoin/opencoin-historic/blob/master/standards/protocol.txt*, 2008.

[64] Unknown. How DigiCash Blew Everything. *http://cryptome.org/jya/digicrash.htm*, 1999.

[65] Bitcoin Wiki. Testnet - Bitcoin. *https://en.bitcoin.it/wiki/Testnet*, 2011.

[66] Wikipedia. Distributed Hash Table. *http://en.wikipedia.org/wiki/ Distributed_ hash_ table.*

[67] Wikipedia. Nicholas Negroponte. *http://en.wikipedia.org/wiki/ Nicholas_ Negroponte.*

[68] Wikipedia. Tokenization. *http://en.wikipedia.org/wiki/ Tokenization_ %28data_ security%29.*

[69] Wikipedia. BitTorrent. *http://en.wikipedia.org/wiki/BitTorrent*, 2015.

[70] Wikipedia. Double Spending. *https://en.bitcoin.it/wiki/Double-spending*, 2015.

[71] Wikipedia. Internet. *http://en.wikipedia.org/wiki/Internet*, 2015.

[72] Wikipedia. Uber. *http://en.wikipedia.org/wiki/Uber_ (company)*, 2015.

[73] Shawn Wilkinson. Stoj - A Peer-to-Peer Cloud Storage Network. *http://storj.io/storj.pdf*, 2014.

[74] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. A language for automatically enforcing privacy policies. *ACM SIGPLAN Notices*, 47(1):85, January 2012.

[75] Polychronis Panagiotis Ypodimatopoulos. Cerebro : forming parallel internets and enabling ultra-local economies, 2008.

[76] David Zeiler. VC Investing in Bitcoin Rises to the Fastest Pace Yet. *http://moneymorning.com/2015/04/17/vc-investing-in-bitcoin-rises-to-the-fastest-pace-yet/*, 2015.