Imperial College London

BEng Project

Imperial College London

Department of Computing

Implementing a Decentralized Trust Inference System

Author:

Christos Porios (christos.porios@imperial.ac.uk)

Supervisor:

Anandha Gopalan (axgopala@doc.ic.ac.uk)

Acknowledgements

Dionysis Zindros, whose contributions to my professional and personal growth stretch far beyond the contents of this report.

Orfeas Litos, who co-authored the Trust Is Risk paper on which a lot of my work is based.

Dr. Anandha Gopalan, who supervised this project and provided valuable feedback.

Prof. Susan Eisenbach, for her guidance, advice and understanding as my personal tutor over my time as an undergraduate student at Imperial.

Contents

1 Introduction and Motivation

Many every-day peer-to-peer online interactions and exchanges share a structure similar to the *Prisoner's Dilemma* [18], where the best overall outcome is achieved by mutual cooperation between the participants, which however appears to be irrational on an individual level. For example, when two users on an online marketplace agree to trade an item for another, each individual can choose to either send the item to her peer, or to deceive the other person and not follow through with the trade. If only one person chooses to deceive, then she manages to obtain both items, which is the ideal outcome for her. However, if by following this individually rational strategy both participants attempt to deceive each other, then the trade will not take place at all and everyone will be worse off.

A solution to this problem are *reputation systems*. On most online peer-to-peer applications, for example on eBay, this problem is solved by requiring users to submit feedback after every interaction and assign star ratings to their peers. These systems are not perfect: they can require manual intervention and are often vulnerable to Sybil attacks [28].

In decentralized peer-to-peer systems, in which the cost of acquiring new pseudonymous identities is very cheap and there is no central authoritative party which can "punish" misbehaving users, the trust problem is even harder to solve. The luxury of manual intervention is not available. How can we build a system which allows for two pseudonymous users to infer the trust they can have towards each other, in a way that will reduce or eliminate the risk of deception if they engage in a trade or financial transaction? How can we make such a system work with no central authority in a sybil-resistant way? This is the trust problem discussed in this report.

The most important example of a decentralized system which faces the trust problem is *OpenBazaar* [16], a completely decentralized market place. It easy to think of many more decentralized services which likely don't exist because of the trust problem prohibits their existence, for example a peer-to-peer loaning service. As internet users become more privacy conscious, and cryptocurrencies and decentralized systems become more popular, the trust problem becomes even more relevant. A practical, generalized solution to the problem of decentralized trust would have great benefits and could be applicable to multiple applications.

There are many proposed systems which attempt to solve this problem, each with different desiderata [31] [27] [30]. None have working, usable implementations.

In this report we present our two main contributions to the problem of decentralized trust. First, we present the first implementation of the trust system Trust Is Risk [31], namely TrustIsRisk.js, which was funded by OB1[15], the company behind OpenBazaar and developed in close communication with the authors of the original paper and the OpenBazaar developers. We also propose and formally define a different trust inference system, Trust In Friends.

2 Background

2.1 Trust

2.1.1 The problem of online trust

In today's online world there's a plethora of ways to exchange items of value with other people. The most well-known example is eBay: A peer to peer market place with more than 160 million users, where people buy and sell their belongings with complete strangers. Naturally, before engaging in any kind of exchange with a stranger, one important question comes to mind: *Can this person be trusted*? What are their motives behind the interaction? Are they really here to sell this item, or are they fraudsters? Will the received goods be as described?

These questions are by no means merely academic: A study by BCG [34] has found that 11% of online customers have paid for goods that never arrived, and that more satisfied customers buy more goods online than less satisfied customers do.

Because eBay tries to protect buyers from scams, scammers often have to come up with ingenious ways of tricking other users into paying them. For example, there's the "Expensive Picture Scam", where the seller takes a picture of an expensive item, for example a game console. The scammer enters a long description of the photographed item, and somewhere they slip in a statement like "You are bidding on the picture only and not the actual item!". The potential buyer doesn't read the entire description and buys the item, resulting in a very expensive picture of a game console.

All this happens even in the presence of a central authority (eBay), which can require ID verifications and bank account details, ban misbehaving users and continuously adapt to the ever-evolving network of scammers.

2.1.2 Star and review based reputation systems

On eBay and most other similar websites, the prime indicator of trust are past interactions. Users who participate in an exchange rate each other after the interaction. People with a history of good behavior on the website are considered less likely to behave maliciously in future transactions. However, there's a number of problems

with this system:

- **Vulnerability to sybil attacks**: A scammer can often create multiple accounts and give herself positive reviews for a very small cost.
- **Difficulty of building initial reputation**: It is difficult for a new user to build reputation initially, as people prefer to interact with users who have long and good history of interactions.
- **Exit scam**: A user who has legitimately gained a good reputation in a system can use this reputation for a big scam and then disappear from the website.

2.2 Trust Inference Systems

2.2.1 EigenTrust

Peer-to-peer file sharing systems, due to their anonymous and decentralized nature, are often used to spread self-replicating inauthentic files. EigenTrust [30] is an algorithm that clients can use to infer how much each peer can be trusted to server authentic files. It is based on the notion of transitive trust, according to which if a peer A trusts a peer B, it should also trust the peers of B.

First, each peer calculates a local trust value $s_{i,j} = \text{good}(i,j) - \text{bad}(i,j)$, where $\text{good}(i,j)$ is the number of good files i has received from j, and $\text{bad}(i,j)$ is the number of bad files i has received from j. $s_{i,j}$ is then normalized so that $\forall i. \sum_{j=1} s_{i,j} = 1$. This implies that every peer has a "trust allowance", which it can distribute among its peers.

Every peer then asks the neighbors it knows about to report other peers and their trust levels, and weights each response by its own trust towards that neighbor. Then it builds a trust vector for all the nodes in the network, which has been proven to converge to the same values for all peers in the network. That vector can be thought of as the *global trust vector*.

EigenTrust is sybil attackable, as discussed in the original paper in which it is presented. The authors suggest solving the problem by introducing a small cost for creating fake identities, possibly by asking new users to solve a captcha. However, this implies a central authority and is impossible in a decentralized setting.

2.2.2 TrustDavis

TrustDavis [27] views parties as vertices in a directed and weighted graph. An edge from A to B with weight M represents a *reference* from A to B, and such a reference is an acceptance of limited liability for the misbehavior of B. References are meant to be sold and bought, and the pricing of references is discussed in detail in the paper. Strategies that will limit the maximum possible loss during an online transaction are then proposed.

TrustDavis is sybil resilient in a decentralized setting. However, it requires that participants pay to insure themselves against the risk of losing money in their transactions with untrusted parties, which is an undesirable property.

2.2.3 Trust is Risk

Trust Is Risk [31] is a very recent development in the field, and the system which was chosen to be implemented in this project.

In Trust Is Risk, trust is expressed in monetary terms. That is, instead of expressing trust in an arbitrary scale (e.g. the eBay star rating system) which can have different meanings for different people, trust is expressed in currency. For example, Alice may express her trust of 10 USD towards Bob by taking 10 of her own dollars and giving Bob the capability to spend her money but not her permission. In conventional currencies, this could be done by depositing 10 USD in a bank account shared between Alice and Bob. The concept of provably giving others the ability to spend your own money but not your permission is very important in Trust Is Risk and forms the underpinning of every trust relationship.

The reasons why Alice might decide to trust Bob for some amount do not matter to the system: She might be friends with Bob, or they might have signed a legal contract. From Alice's viewpoint it is important that she ensures that Bob will suffer some out-of-band consequences if he betrays Alice and steals her money. If Alice is friends with Bob, then she can be confident that Bob would not steal her trust money for very the same reason that she would have lent Bob the same amount, if Bob had asked her for a small loan: Bob (hopefully) values their relationship higher than the trust amount. In this case we refer to the implications the stealing act would have on the real-world relationship of Alice and Bob as the out-of-band consequence.

Giving someone the ability to spend your money while still being able to spend the money yourself is not easy with conventional currencies, but it is trivial with Bitcoin. In our example, Alice can place the Bitcoin equivalent of 10 USD (3 mBTC at the time of writing) in a 1-of-2 multisig transaction output. That output can then be spend by using either one of Alice's or Bob's private keys and thus the money is spendable by both of them simultaneously. In the Trust Is Risk implementation described later in this report, *Direct Trust* is expressed exactly by the use of 1-of-2 Bitcoin multisig outputs.

We can view direct trust relationships as edges in a directed and weighted graph, with the amount of trust being the weight of the edge. The people participating in the relationships are represented as vertices. For example, if Alice directly trusts Bob for 10 USD, then this is represented as an edge with weight 10 from Alice to Bob. These trust relationships form a *trust graph*.

By applying the *Direct Trust* relationships transitively, we obtain the notion of Indirect Trust: If Alice directly trusts Bob and Bob directly trusts Charlie, then we say that Alice *indirectly trusts* Charlie. *Indirect trust* is simply referred to as *trust*. The monetary amount to which we say that Alice trusts Charlie is the *maximum flow* from Alice to Charlie in the trust graph. By defining indirect trust in this matter we obtain some very valuable properties which are proven in the paper:

- **Trust Invariance Theorem**: Suppose Alice wants to engage in a transaction with Charlie during which she will risk X USD. If before the transaction she indirectly trusts Charlie for Y USD with $Y > X$, she can redistribute her outgoing direct trusts to reduce her indirect trust towards Charlie to $Y - X$. Then when she engages in the transaction the money being risked remains invariant.

- **Sybil resilience**: An adversary looking to increase someone's indirect trust towards them (potentially in order to fraud them) obtains no benefit from creating and controlling collusions of multiple users in the network, unless she convinces the victim to place trust in the collusion which is no easier than convincing them to trust the adversary directly.

Therefore in a peer to peer marketplace Indirect Trust can be seen as a *spending allowance towards a vendor*. The users do not have to directly trust any vendors themselves, but by placing direct trust in entities they know to be trustworthy (e.g.

their friends), they can infer the maximum amount for which a stranger can be trusted and transact with them up to amount while keeping their risk invariant.

2.3 Bitcoin

The implementation of Trust Is Risk, TrustIsRisk.js, which is one of the main deliverables of this project is based on the Bitcoin blockchain and uses Bitcoin transactions to express direct trust between users. It is therefore worthwhile to go over how Bitcoin works, and specifically look into the parts of Bitcoin that are used extensively in TrustIsRisk.js.

Bitcoin is a digital cryptocurrency invented by the notorious Satoshi Nakamoto in 2008. Satoshi's true identity remains unknown to this date. Satoshi published the invention to a cryptography mailing list as a research paper [33]. It described a distributed payments system, in which transactions take place peer-to-peer without an intermediary or central authority, and are verified by a distributed blockchain ledger.

Bitcoin quickly grew in popularity and its price rose accordingly [6]. In early 2010 one Bitcoin was worth less than $0.01. In early 2011, its value surpassed the value of the dollar. On 27 November 2013, Bitcoin broke the $1000 dollar threshold for the first time. At the time of writing, Bitcoin is trading in the $2500 - $2600 range.

Despite a number of setbacks in the history of the Bitcoin blockchain[23], excitement over the potential applications of other blockchains has been increasing sharply from Governments[24][25], industry[32][19] and the media [26][29].

A huge, active and very liberal community has formed around Bitcoin, which has give rise to a sub-culture that can be described as *digital anarchy*. Bitcoin enthusiasts reject the idea that a central trusted authority is required for regulating many of the systems they interact with in their every day lives. For them, the benefits of decentralization and the superiority of such systems over centralized systems is clear, and decentralizing the monetary system was just the beginning. For example, Namecoin [13] decentralizes the domain name assignment system and Openbazaar [16] is a decentralized market place similar to eBay[1]. Numerous other less popular examples exist.

[1] http://www.ebay.co.uk/

2.3.1 Proof of work

Arguably, the most important concept in Satoshi's paper was not the distributed nature of Bitcoin nor the creation of a purely digital currency, but the idea that systems that traditionally required a central, trusted authority to operate reliably could be replaced by a distributed, purely trustless and completely decentralized network of agents, each acting in their own self-interest. The monetary system, which Bitcoin could in theory replace, is only one such example: Since the invention of currency thousands of years ago, there has always been a central authority which was responsible for issuing a currency and setting monetary policy. For example, today the Federal Reserve has this responsibility for the U.S. Dollar. There is no such authoritative party in Bitcoin. Inflation, (i.e. the number of Bitcoins in circulation) is predetermined for every point in time, and an individual owning some amount of Bitcoins can spend them however they like, without any authority or government being able to enforce their own "terms and conditions".

In technical terms, Bitcoin achieves this by a cryptographic invention called *proof-of-work*. Describing the intricacies of proof-of-work in detail is outside the scope of this report, but the basic idea is the following: The network is made up of a set of agents, the *miners*, who are responsible for producing blocks. A *block* is a *set of transactions*, and at all times each miner is working on producing a block. For a block to be valid and accepted by the rest of the network, a hard computational problem which depends on the transactions contained within the block must be solved. The difficulty of the next problem is automatically adjusted so that a single block is produced by the whole network of miners every 10 minutes on average. For the problem to be solved, a lot of computational power must be spent. However when a miner announces that they have solved the problem and produced a valid block, it is trivial for the rest of the network to verify that the transactions in the block and the solution presented are indeed correct. A newly produced valid block is considered to be the *consensus* (i.e. the most current, true state) of the system. The next block will contain a reference to the previous block, therefore forming a chain of blocks, the *blockchain*.

Each time an individual wants to perform a transaction, they only have to announce their transaction to the network. Every transaction includes a small amount of Bitcoin that is awarded to the miner who eventually mines the block, so it is in every miners' self-interest to check whether the transaction is valid and if it is, include

it in the block they are producing. From the point of view of the end user however, all this complexity is hidden and making a transaction is as simple as clicking a button and waiting 10 minutes for the next block to be mined.

The key concept in this process is that a consensus for a distributed database can be maintained by a trustless network of independent agents by producing cryptographic proof of computational work as part of every valid state of the database. We will use this functionality in TrustIsRisk.js and build an extended Bitcoin node which supports additional operations, like inspecting and modifying a trust graph. Because the basic trust primitive of our system will be a special kind of Bitcoin transaction, the Bitcoin blockchain will ensure that all nodes achieve consensus in a decentralized matter.

2.3.2 Bitcoin Transactions

Because the basic building block of the trust networks we will describe in detail later are implemented as specially crafted Bitcoin transactions[4], it is important to look into what a transaction is and introduce the concepts of inputs, outputs and scripts. We assume that the reader has a basic understanding of asymmetric cryptography concepts, like public/private keys and signatures.

A transaction is transfer of Bitcoin value from some entity to another. It is made up of *inputs* and *outputs*. A transaction input is a reference to a previous transaction output. When a transaction references a previous transaction output, we say that the transaction *spends* that output. An output can only be spent once, so a transaction that uses an already spent output as an input is considered invalid and would be rejected by the miners.

Every output has a value and an *output script*, which is a simple program that decides whether it can be spent by some input. Most commonly, the script of an output simply requires the spender to prove they own the private key corresponding to a Bitcoin address. Such an output essentially transfers the value it carries to the owner of an address and is called a *Pay-To-Public-Key-Hash* (P2PKH), because a Bitcoin address is a essentially a public key hash.

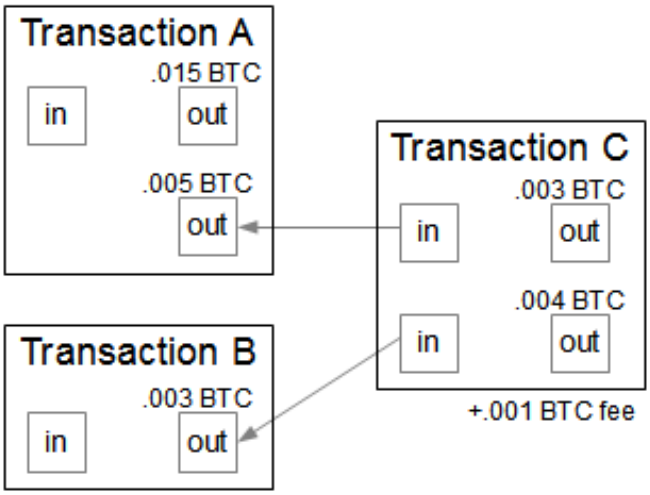
Similarly to Kirchoff's Circuit Law, a regular Bitcoin transaction's total output value may not be greater than the total value of the inputs, meaning that Bitcoins can not be created out of thin air.

The exception to this rule are coinbase transactions: Such a transaction has no inputs and exactly one exists in every block. It pays a predetermined amount of Bitcoins to the miner who mined the block.

In all other transactions, the difference between the total output value and the total input value is the *miner's fee* and is usually non-zero. It is awarded to the miner who mines the block that first includes the transaction, and serves to incentivize the miner.

We say that a user's capital or balance is the total value of the unspent transaction outputs that she can spend.

Figure 1: A Bitcoin transaction [5]



In figure 1 we see an example transaction C with two inputs referencing outputs from two different transactions. The total value of the inputs is 0.008 BTC, and that amount is split in two outputs and a small miner fee. It is worth noting that the first output from transaction A is still unspent, even though the second output is spent by C.

Bitcoin output scripts can be more sophisticated than simple P2PKH scripts, by requiring more complicated conditions to be met. The particular case which interests us are multisignature (or multisig) transactions. A multisig transaction output defines a set of N public keys and a number M, with $M \leq N$. At least N signatures with public keys from the set are required for the output to be spent. We call a transaction containing such an output a *M-of-N multisig transaction*. For example, a 2-of-3 multisig transaction defines three people who jointly own some

funds, and requires agreement between at least two of them in order to spend them.

In the next section we will use 1-of-2 multisig transactions, which essentially make some funds spendable by either one of two people.

3 TrustIsRisk.js

TrustIsRisk.js is a software library implementing the Trust Is Risk trust inference system described in the paper by Dionysis Zindros and Orfeas Litos [31]. It was developed with the OpenBazaar[16] usage scenario in mind, although it is a generalized platform that could be used in any system requiring decentralized trust.

Within OpenBazaar, our library could be used to display the user's *spending allowance* towards a vendor, which would be the indirect trust from the user to the vendor. The user will then know that if they can buy goods up to their allowance from the vendor without exposing themselves to more risk than they are already undertaking by having directly trusted their friends beforehand.

3.1 Organizations involved

During the development of TrustIsRisk.js I worked in close collaboration with the authors of the original Trust Is Risk paper, namely Dionysis Zindros and Orfeas Litos and the company behind OpenBazaar, OB1, who kindly agreed to sponsor the project.

During the development process, I would meet with the authors and representatives from OB1 weekly to discuss progress and major design decisions.

3.2 Basic concepts and terms

TrustIsRisk.js is a JavaScript library that provides an API for maintaining and operating on the Trust Graph described in the Trust Is Risk paper. It is implemented as an extended Bitcoin full node.

In TrustIsRisk.js, the pseudonymous entities in the trust network are identified by a unique Bitcoin address. This address can, but doesn't have to be used for regular trust-unrelated Bitcoin payments. For example, an OpenBazaar vendor will have a Bitcoin address which will act as her *unique identifier* in the trust network, but he may use a different address for accepting payments, or even an entirely different cryptocurrency like Monero.

Since an address is uniquely tied to an entity, we use the terms *address*, *person*,

entity and *pseudonymous entity* interchangeably throughout this section. However, we reserve the use of the term *user* to describe the programmer who uses the library, who may have control over multiple entities or none at all. We say that the library user *controls* an entity when they have the private key corresponding to that address.

3.3 Functionality

Specifically, TrustIsRisk.js provides the following functionality to the user:

- It maintains the full Trust Graph by monitoring all Bitcoin transactions being transmitted on the network and lets the user inspect the graph by:
  - Listing incoming and outgoing direct trusts from any entity.
  - Getting the direct or indirect trust between any two entities.
  - Getting an adjacency matrix of the trust network.
- It provides methods to modify the trust graph by producing valid Bitcoin transactions which:
  - Increase the direct trust from an entity the user controls to any other entity.
  - Decrease the direct trust from an entity the user controls to any other entity.
  - Decrease the incoming direct trust directed towards an entity the user controls, which we also refer to as the act of stealing incoming direct trust.

3.4 Technologies

As its name suggests, we decided to build TrustIsRisk.js in JavaScript. The main reasoning behind this decision was that, in accordance with the ethical principles of decentralization, we wanted our tool to be transparent, open and hackable by as many people as possible. Thus JavaScript was an obvious choice due to its popularity.

To make our code short and as readable as possible, we used the latest version of JavaScript, specifically ECMAScript 7 and made extensive use of both new languages

features, like block scoping, promises and async/await functions and the good old functional programming primitives that JavaScript provides, like map and fold.

We also decided to statically typecheck our code by using the open-source tool developed by Facebook, Flow [8]. Flow is a tool that quickly and unobtrusively checks JavaScript code for type errors and features advanced type-inference. Aside from saving development time, having to write fewer tests and automatically and easily guarding against some of the most common JavaScript bugs, the explicit type-annotations of the API also act as one kind of always up-to-date documentation.

Because TrustIsRisk.js is an extended Bitcoin node, we first had to find a way to provide the functionality a Bitcoin node provides. Naturally, writing a production-quality Bitcoin node from scratch was out of the question, so we had to find a way to interface with an existing node. After researching the available options, we had a choice between the two most promising alternatives:

- Using bitcore.io [7] to interface with the original Bitcoin Core implementation.
- Using bcoin [2], an alternative Bitcoin implementation written in JavaScript.

We first attempted to use bitcore.io. Because of the way bitcore is built, we had to compile our flow-annotated ES7 code to Node.js v4 compatible JavaScript by using Babel [9], a JavaScript compiler. To make the development process easier and faster, we would then start a Docker [1] container in which the compiled TrustIsRisk.js code would be loaded and executed as a bitcore module. We soon however ran into problems where bitcore was not behaving as one would expect from the documentation. These problems, in combination with the necessary complexity of the development environment, and the fact that bitcore's most recent commit was many months ago lead us to abandon its usage and look for an alternative approach.

The second choice, bcoin, does not use the "official" Bitcoin Core implementation and implements a full Bitcoin node in JavaScript instead. It is very actively under development, is used in production by many other projects and has a very active community built around it. Most importantly, its code is very well written and made us confident that if we were to run into any technical issues, we could patch bcoin to fix them ourselves. Unlike bitcore, using bcore in a way such that we could extend its functionality was trivial: We simply include it as dependency and use its API.

We therefore decided to base TrustIsRisk.js on bcoin and distribute the library as an NPM [14] module. We also used Mocha [12] for testing, Github [10] for source code hosting and Travis [22] for continuous integration, as well as many other smaller node modules.

3.5 Trust Is Risk Transaction Format

As we discussed in 2.2.3, the basic principle of Direct Trust in Trust Is Risk is the idea of giving others the capability, but not permission to spend your money in a way that is verifiable by others. In TrustIsRisk.js this is expressed in 1-of-2 multisig transaction outputs. In this section we will describe in the detail how Bitcoin transactions are parsed as Direct Trust changes that affect the trust graph, and how a Direct Trust modification can be expressed as a Bitcoin Transaction.

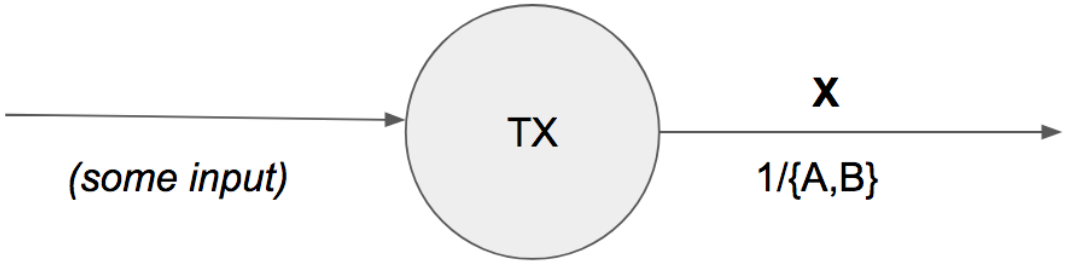
The library initially starts with an empty trust graph, containing all the possible entities as vertices but no edges connecting any of them. Then, every Bitcoin transaction that has ever occurred is processed in order. For each transaction, the library tries to match it to one of the formats described below and modifies the trust graph accordingly. If the transaction is not in one of these special formats, it has no effect on the graph and can be ignored.

The basic principle of direct trust is that when an entity, Alice, wants to increase her direct trust towards another entity, Bob, for 10 BTC, she creates a transaction with a 1-of-2 multisig output with a value of 10 BTC that is spendable by either Bob or Alice. This way Alice can access her money if she needs to just as easily as if she had kept it spendable by herself only. More importantly, this proves to the network that she trusts Bob for (at least) that amount, and since every Bitcoin transaction is public, anybody can verify that the transaction occurred. We will use the notation *1/{Alice, Bob}* to denote such a multisig output.

The first obvious problem is that when we see a transaction with a *1/{Alice, Bob}* output, there is no way to know whether this is an output expressing trust from Alice to Bob or an output expressing trust from Bob to Alice. In other words, the direction of trust is not encoded in the transaction.

To decide on the direction of trust represented by the transaction depicted in 2, one has to know whether the Bitcoins being spent originally belonged to Alice or Bob. So one might think that it suffices to check the script of the output the trust

Figure 2: Is this trust from A to B or vice versa?



transaction input references.

We can however come up with more complicated cases: What if the input to the transaction is a 1-of-2 multisig itself? What if there are multiple inputs and multiple outputs? It is clear that we need to strictly define the format of Trust Is Risk transaction so as to preserve the following properties:

1. The Direct Trust between any two entities A and B must be based on a set of **unspent 1-of-2 multisig outputs *1/{A, B}***. The total value of the outputs must equal the trust amount. In other words, if there exists an edge $A \rightarrow B$ with weight X in the Trust Graph, there must exist a set of unspent Bitcoin transaction outputs, each spendable by $1/\{A, B\}$, with a total value of X. We then say that the direct trust relationship (or graph edge) is based on those transaction outputs, which we call **trust outputs**.
2. At most one trust relationship can be based on a single trust output.

We are now ready to introduce the notion of a Trust Is Risk Transaction. A Trust Is Risk Transaction is a *valid* Bitcoin transaction which modifies the trust graph. Any other transaction is considered a regular Bitcoin transaction, does not change the trust graph and has no special meaning for Trust Is Risk. There are two types of TIR Transactions.

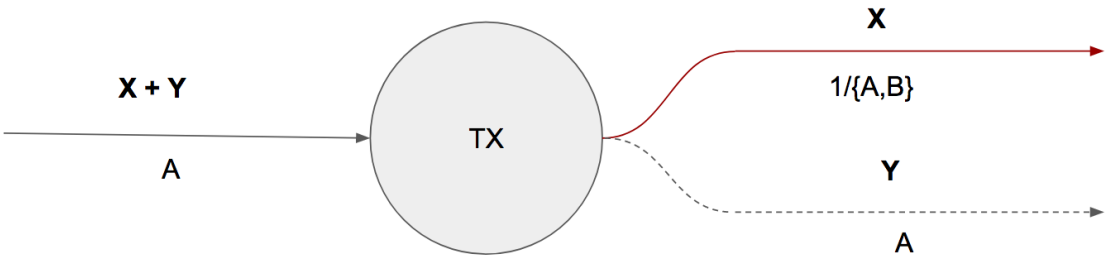
3.5.1 Trust-increasing Transaction

A Trust-increasing Transaction is a valid Bitcoin transaction that increases trust from entity **A** to entity **B** for an amount of **X**. It has the following format:

- **Exactly one input**:
  - **P2PKH**: Spends a Pay To Public Key Hash transaction output to an address **A** with a value of at least **X**.
- **One or two outputs**:
  - **The Trust Output**: A 1-of-2 multisig output for an amount of **X**. One of the two addresses must be address **A**, the other is address **B**. The output script must have the following format:

    `OP_1 <pubKeyA> <pubKeyB> OP_2 OP_CHECKMULTISIG`
  - **The Change Output** (Optional): A Pay To Public Key Hash output returning any change back to address A.

Figure 3: Trust-Increasing Transaction



3.5.2 Trust-decreasing Transaction

A trust-decreasing transaction is any transaction which spends a trust output. To define trust-decreasing transactions like this is a necessity: We can not allow a trust output to be spent without modifying the trust graph appropriately, or we would be violating property 1, which requires trust relationships to be based on **unspent** Bitcoin transactions.

Because Bitcoin output scripts can be very complicated, we define two types of trust-decreasing transactions. The first type is the type of trust-decreasing transaction our library creates and understands fully. The second type is a catch-all term for Bitcoin transactions which spend trust outputs, and therefore must be considered for

property 1 to hold, but can not be parsed as transactions that the library created. As a result, we can be as strict as we like when defining the format of the first type of transactions, but we have to be very general for the format of the second type.

Proper Trust-Decreasing Transaction
Decreases direct trust from entity **A** to entity **B** by **X**. It has the following format:

- **Exactly one input**:
  - **A trust output**: Spends a 1-of-2 multisig output that is a valid trust output with value **Y**.
- **One or more outputs**:
  - **A trust output**: A 1-of-2 multisig with its script identical to the output script being spent by the input. The value of the output is **Y** - **X**.
  - **Any other outputs** that are not 1-of-2 multisig outputs.

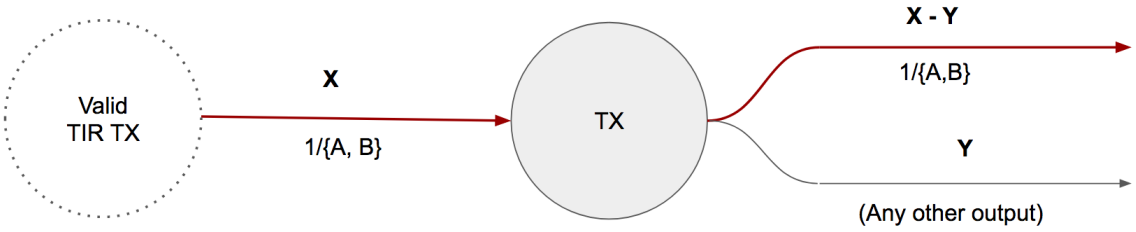
The fact that all Trust Is Risk transactions are valid Bitcoin transaction ensures that **Y** > **X**.

A trust-stealing transaction, which occurs when the receiving end of a direct trust relationship spends the underlying transaction, thereby "stealing" the funds, is simply a proper trust-decreasing transaction signed by the receiver.

Improper Trust-decreasing Transaction
Any Trust-decreasing Transaction that is not proper.

Such a transaction simply spends one or more trust outputs and nullifies their contribution in the trust graph. The TrustIsRisk.js library never creates improper transactions, however they might be created by an adversary or by accident and are thus need to be taken into account. When a transaction is parsed as an improper trust decreasing transaction, it is the equivalent of the library saying "I don't know how to parse this transaction, but it spends one or more trust outputs, so I have to not consider those trust outputs when building the trust graph".

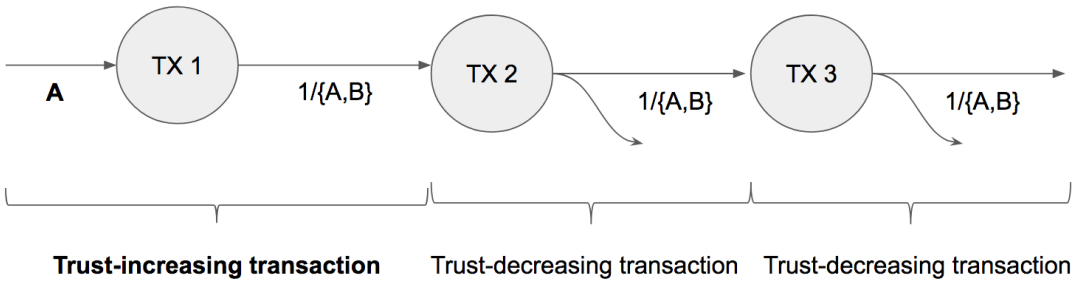
Figure 4: Proper Trust-Decreasing Transaction



3.6 Trust Chains

From the above definition of a Trust Is Risk Transaction we can derive that Trust Outputs form chains by connecting together transactions that look like in figure 5.

Figure 5: A trust chain



Every trust chain starts with a trust-increasing transaction and possibly continues with a series of trust decreasing transactions. An improper trust decreasing transaction has no trust outputs, and therefore ends a chain. A *trust chain tip* is an unspent trust output.

A trust chain is associated with a single sender and recipient. In the figure 5, the sender is A and the recipient is B.

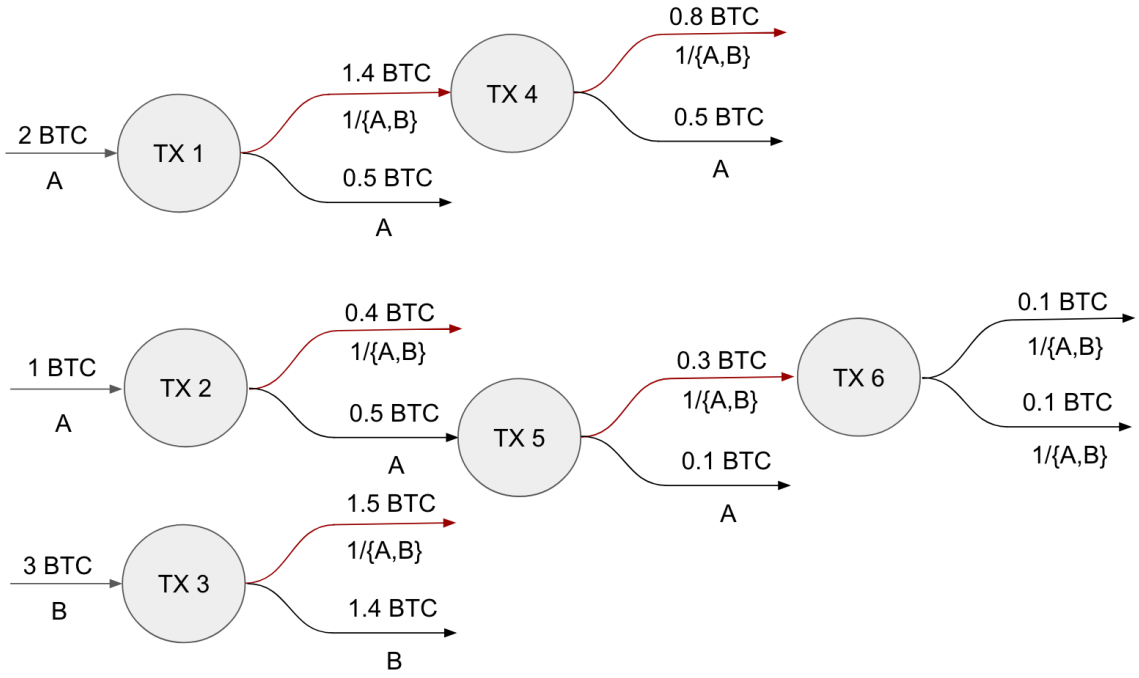
The direct trust between from an entity A to an entity B any point is the sum of the values of all the *unspent trust outputs*. It is worth noticing that the total trust from A to B may consist of the endpoints of multiple trust chains.

The two properties described in 3.5 can now be easily proven by structural induction on the trust chain.

3.7 Example

We will now discuss the larger example of figure 6, which contains transactions various types. We have two entities, Alice and Bob, denoted by the letters A and B in the diagram. Trust outputs are denoted in red lines.

Figure 6: Trust transaction examples



Initially, Alice decides to set her direct trust towards Bob to 1.4 BTC. She creates and publishes TX 1, which is a *trust-increasing transaction* as defined in section 3.5.1. As per the definition, the transaction spends a P2PKH payable to Alice. It has two outputs, one trust output of 1.4 BTC and one change output of 0.5 BTC. The remaining 0.1 BTC is the miner's fee.

Then, at some later point in time, Alice decides that she wants to increase her trust towards Bob by a further 0.4 BTC. She produces another *trust-increasing transaction*, TX 2, which spawns a new trust chain. Again, she spends a P2PKH payable to herself which is split among the trust output of 0.4 BTC, the miner's fee of 0.1 BTC and the change output to herself of 0.5 BTC. After transactions TX 1 and TX 2 have occurred and before any of the other transactions depicted in the figure have occurred, Alice directly trusts Bob for 1.8 BTC.

The first two transactions only affected Alice's direct trust towards Bob. Because trust is directional, the trust of Bob towards Alice hasn't been changed.

Later, Bob decides that he wants to increase his direct trust towards Alice by 1.5 BTC. For this purpose, he creates a *trust-increasing transaction* TX 3. This leaves the trust from Alice to Bob unaffected.

Next, Alice decides that she wants to reduce her direct trust towards Bob from 1.8 BTC to 1.2 BTC. She might have decided that Bob isn't as trustworthy as she had previously though, or maybe she wants to use some of her money which is currently held in direct trusts to buy something. To reduce her direct trust, Alice produces TX 4, which is a *proper trust-decreasing transaction* as defined in section 3.5.2. It spends her previous trust output of 1.4 BTC and splits the amount among a new trust output of 0.8 BTC, a P2PKH payable to herself of 0.5 BTC and a small miner's fee of 0.1 BTC. Since the trust output used as an input was worth 1.4 BTC, and the new trust output created by the transaction is 0.8 BTC, Alice's direct trust towards Bob has been reduced by $1.4 - 0.8 = 0.6$ BTC.

If instead Alice wanted to decrease her direct trust towards Bob by more than 1.4 BTC, she would have to create *two* proper trust decreasing transactions: Each one would spend one of her two trust outputs.

At this point in time, after all the depicted transactions except TX 5 and TX 6 have occurred, Alice's direct trust towards Bob is equal to 1.2 BTC. To calculate this number, we need to add up the values of all the unspent trust outputs from Alice to Bob. The trust output from TX 1 has been spent by TX 4, so it does not count towards the sum and the trust output from TX 3 is of the inverse direction (as indicated by its input) and thus is also discarded. The trust outputs of TX 2 and TX 4 sum to $0.4 + 0.8 = 1.2$ BTC.

Next, Alice again decides to increase her direct trust towards Bob to 1.5 BTC, and thus creates TX 5, which is a *trust-increasing transaction* of 0.3 BTC.

TX 6 spends a trust output, thus it must be a *trust-decreasing transaction*. However it can not be a proper trust-decreasing transaction because it has two multisig outputs, which is not allowed in the specification of section 3.5.2. Therefore, it is an *improper trust-decreasing transaction* and *none of its two outputs are trust outputs*. Its effect on the trust graph is the invalidation of the trust output it spends.

After all transactions have occurred, Alice directly trusts Bob for 1.2 BTC and

Bob directly trusts Alice for 1.5 BTC.

3.8 API Usage

Below we show a commented example of how the library can be used in practice.

```
var TrustIsRisk = require(trust-is-risk);


// Suppose we have a method createWallet(),
// which creates a new public/private key pair
// and finds the corresponding address.
// bcoin provides such functionality, but the syntax
// is a little more complicated:
var alice = createWallet();
var bob = createWallet();
var charlie = createWallet();


// Create an extended bcoin node, which has the additional 'trust' property,
// containing all the TIR-related functionality.
var node = new TrustIsRisk.FullNode({
  network: testnet
  // ...other options...
});


async function start() {
  // These are bcoin methods that start the full node:
  await node.open();
  await node.connect();
  node.startSync();

  // Check how much alice trusts bob directly:
  node.trust.getDirectTrust(alice.address, bob.address);
```

```
// Check how much alice trusts bob indirectly:
node.trust.getTrust(alice.address, bob.address);

// Increase trust from alice to bob by 1000 satoshis:
var mtx = node.trust.getTrustIncreasingMTX(
  alice.privKey,
  bob.pubKey,
  alice.getUnspentP2PKHOutpoint(),
  1000
);

// Convert the mutable transaction to a regular transaction and send it:
node.sendTX(mtx.toTX());

// Decrease trust from alice to bob by 100
var mtxs = getTrustDecreasingMTXs(
  alice.privKey,
  bob.pubKey,
  100
);

// Convert the mutable transactions to regular transactions and send them:
mtxs.each((mtx) => node.sendTX(mtx.toTX()));

// Steal 200 of alice's trust to bob and send the money to charlie.
// Notice how we only need bob's private key this time:
var mtxs = await getTrustDecreasingMTXs(
  alice.pubKey,
  bob.privKey,
  150,
  charlie.address, /* recipient */
  true /* steal */
);
```

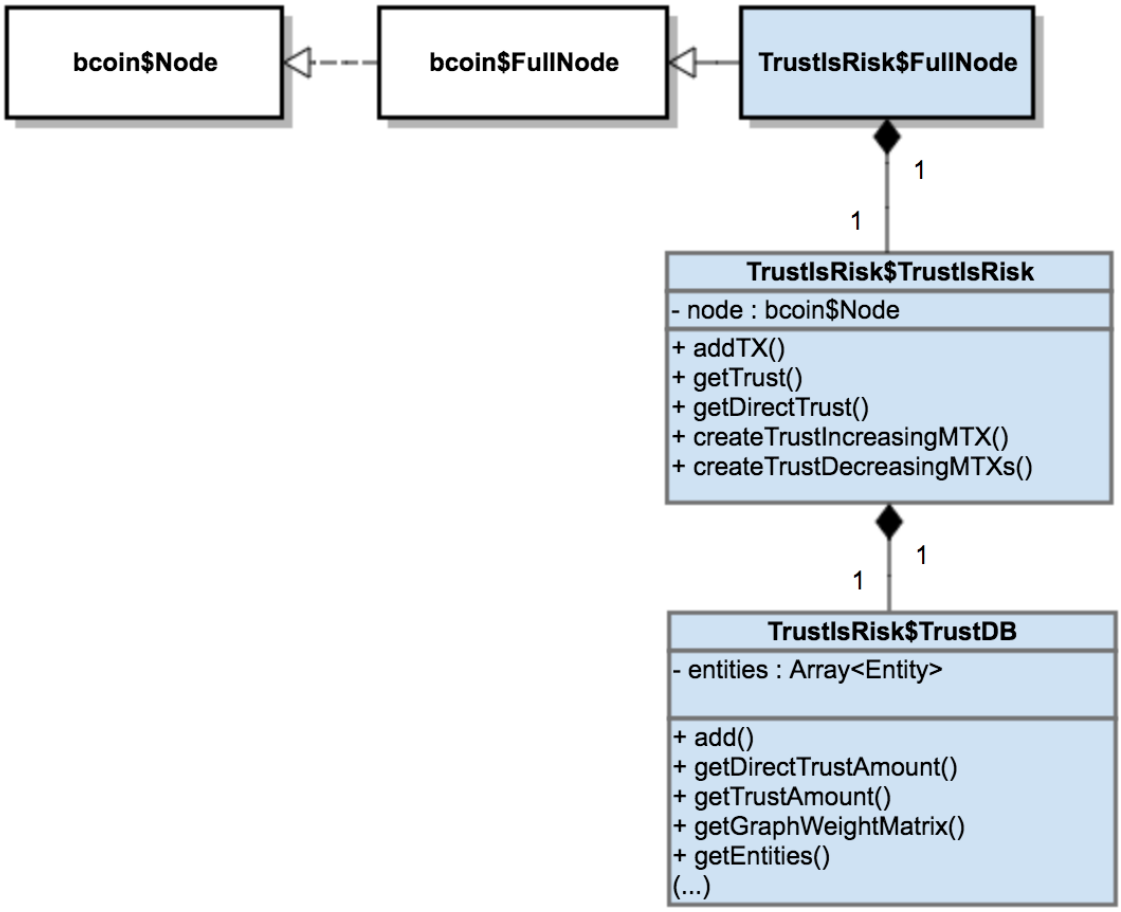
```
  // Convert the mutable transactions to regular transactions and send them:
  mtxs.each((mtx) => node.sendTX(mtx.toTX()));
}


start();
```

3.9 Design

In this section we present the design of the library and explain some of the decisions that were taken. We then document the public methods of the imporant classes using flow-style [8] type annotations and explain how they operate when needed.

Figure 7: Partial UML Diagram



3.9.1 TrustIsRisk$TrustIsRisk

This is the main controller of the Trust Is Risk implementation and represents a trust network. The user of the class can add new Bitcoin transactions to the trust network, create Trust Is Risk Transactions, and query the trust graph. It requires a bcoin node to operate, but remains as agnostic as possible as to what type of node that can be.

Most of what the class does is related to translating Bitcoin transactions to direct trusts and creating Trust Is Risk Transactions. An instance of TrustIsRisk$TrustDB is managed and used internally to store direct trusts and perform queries, which is discussed later.

- **constructor(node : bcoin$Node)**: Attaches a transaction event handler (addTX) to the node passed as an argument, to ensure that the node will notify the instance whenever a new transaction occurs.
- **addTX(tx : bcoin$TX) : boolean**: Attempts to add a Bitcoin transaction to the trust graph, by trying to parse it as a Direct Trust. If it succeeds, it adds the resulting Direct Trust in the TrustDB instance and returns true. If it fails and the transaction was a regular Bitcoin transaction, unrelated to Trust Is Risk, it does nothing to the graph and returns false.
- **getTrust(origin : Entity, dest : Entity) : Number**: Returns the indirect trust between two entities by forwarding the query to the TrustDB instance.
- **getDirectTrust(origin : Entity, dest : Entity) : number**: Returns the direct trust between two entities by forwarding the query to the TrustDB instance.
- **getTrustIncreasingMTX(origin : PrivateKey, dest : PublicKey, outpoint : bcoin$Outpoint, trustAmount : number, fee : ?number) : Promise<bcoin$MTX>**: Creates a trust-increasing mutable transaction which increases direct trust by the given `trustAmount` from the `origin` to the `dest`. The transaction will spend the output referenced in `outpoint`, which must be a P2PKH spendable by `origin` and with a high enough value, or the function will throw. A custom miner `fee` can be optionally set.
- **getTrustDecreasingMTXs(origin : (PrivateKey | PublicKey), dest : (PrivateKey | PublicKey), trustDecreaseAmount : number, payee : ?Entity, steal : ?boolean, fee : ?number) : Array<bcoin$MTX>**: Returns a list of mutable

bitcoin transactions which when applied reduce the trust from `origin` to `dest` by `trustDecreaseAmount`. The value removed from the trust relationship will be sent to `origin` by default, but this can be overridden by providing `payee`. When `steal` is not provided or is set to false, `origin` must be a private key and `dest` must be a public key, or the function will throw. When `steal` is set to true, then the function will create transaction that steals trust from `origin`, and therefore requires `origin` to be a public key and `dest` to be a private key. A custom, per-transaction miner `fee` can be optionally set.

3.9.2 TrustIsRisk$FullNode

This is a very small convenience class which extends the bcoin FullNode class by adding an additional member `trust : TrustIsRisk$TrustIsRisk` to the class. The TrustIsRisk$FullNode will first call the super class constructor and then initialize the TrustIsRisk controller by connecting it to the full node.

3.9.3 TrustIsRisk$TrustDB

This is an internal class that is not meant to be used by the user of the library. As the name suggests, it serves to store and manage trust relationships. Trust relationships can be added and then modified, and methods are provided to answer direct or indirect trust queries.

3.10 The Code

The source code for TrustIsRisk.js is hosted on Github (https://github.com/decrypto-org/TrustIsRisk.js). At the time of writing, the code resides in the branch tx-creation (https://github.com/decrypto-org/TrustIsRisk.js/tree/tx-creation), and is ongoing a code review in Pull Request 7 (https://github.com/decrypto-org/TrustIsRisk.js/pull/7). The Travis CI page for the library, showing all the tests passing, is at https://travis-ci.org/decrypto-org/TrustIsRisk.js.

In this section we will examine some of the most critical sections of the code that directly translate Trust Is Risk concepts into code. We have devoted a lot effort in making the code as readable as possible without relying on comments.

3.10.1 The DirectTrust data structure

This file implements the class DirectTrust. A DirectTrust is associated with a Bitcoin transaction output of the same amount (the trust output), except for nullifying direct trusts which are associated with a whole Bitcoin transaction and not a specific output. A DirectTrust may be spent by another a DirectTrust, thus forming the trust chain discussed in the previous sections. A DirectTrust is considered an *increase* if it starts a trust chain, or a *decrease* if it continues a trust chain by spending a previous DirectTrust. Below we present the part of the class that defines the data structure, but we omit certain convenience functions and the constructor.

```
class DirectTrust {
  origin : Key
  dest : Key
  amount : number

  // Every DT is associated with a transaction output, except for
  // non-standard trust decreasing transactions, which reduce
  // trust to zero and are related to a whole transaction and not
  // a specific output.
  txHash : string
  outputIndex : (number | null)
  script : (bcoin$Script | null)

  prev : (DirectTrust | null)
  next : (DirectTrust | null)

  isNull() {
    return this.amount === 0;
  }

  isIncrease() : boolean {
    return this.prev === null;
  }

  isDecrease() : boolean {
```

```
    return !this.isIncrease();
  }

  isSpent() : boolean {
    return this.next !== null;
  }

  isSpendable() : boolean {
    return !this.isSpent() && !this.isNull();
  }
```

3.10.2 Parsing transactions

The member function of the TrustIsRisk class presented below tries to translate a Bitcoin transaction into an increasing DirectTrust data structure, or returns null if that is not possible. It directly translates the restrictions presented in section 3.5.1.

```
parseTXAsTrustIncrease(tx : bcoin$TX) : (DirectTrust | null) {
  if (tx.inputs.length !== 1) return null;
  var input = tx.inputs[0];
  if (input.getType() !== "pubkeyhash") return null;
  if (this.db.isTrustOutput(input.prevout.hash.toString("hex"),
   input.prevout.index)) return null;
  var origin = tx.inputs[0].getAddress().toBase58();

  if (tx.outputs.length === 0 || tx.outputs.length > 2) return null;

  var trustOutputs = this.searchForDirectTrustOutputs(tx, origin);
  if (trustOutputs.length !== 1) return null;

  var changeOutputCount = tx.outputs.filter(
   (o) => this.isChangeOutput(o, origin)).length;
  if (changeOutputCount + 1 !== tx.outputs.length) return null;

  return trustOutputs[0];
}
```

Similar methods exist for parsing other types of trust relationships, but are omitted here. Please refer to the file `src/trust_is_risk.js` in the repository, and specifically to the method `getDirectTrusts`.

3.10.3 Creating transactions

Below we presented the `TrustIsRisk$TrustIsRisk` member function that translates a DirectTrust data structure into a single `getTrustDecreasingMTX`, which when transmitted decreases the trust carried by the direct trust:

```
getTrustDecreasingMTX(directTrust : DirectTrust,
    decreaseAmount : number, payee : ?Entity,
    signingKeyRing : bcoin$KeyRing, fee : number) {
  if (!payee) payee = directTrust.getOriginEntity();

  var mtx = new MTX({
    inputs: [
      Input.fromOutpoint(new Outpoint(directTrust.txHash,
        directTrust.outputIndex))
    ],
    outputs: [new Output({
      script: bcoin.script.fromPubkeyhash(
          Address.fromBase58(payee).hash),
      value: decreaseAmount - fee
    })]
  });

  var remainingTrustAmount = directTrust.amount - decreaseAmount;
  if (remainingTrustAmount > 0) {
    mtx.addOutput(new Output({
      script: bcoin.script.fromMultisig(1, 2,
          [directTrust.origin, directTrust.dest]),
      value: remainingTrustAmount
    }));
```

```
  }

  var success = mtx.scriptVector(
      ((directTrust.script : any) : bcoin$Script),
      mtx.inputs[0].script, KeyRing.fromPublic(directTrust.origin));
  assert(success);

  success = mtx.signInput(0, new Coin({
    script: directTrust.script,
    value: directTrust.amount
  }), signingKeyRing);
  assert(success);

  return mtx;
}
```

A similar method exists for creating trust-increasing transactions.

3.11 Testing

To verify the correctness of the implementation we developed a suite of unit tests and end-to-end tests.

The unit tests check that the public methods of the main classes are working correctly by testing various common cases and some less common edge cases.

The end to end tests create a private, local blockchain with a single node, and set up a TrustIsRisk.js full node. Then they produce a few blocks to create coinbase coins to be used in transactions. Afterwards, the library can be used to translate pre-defined graphs into raw Bitcoin transactions and publish those transactions to the blockchain. Finally the tests verify that the implementation correctly calculates direct and indirect trusts.

3.12 Demo

To demonstrate the functionality of the TrustIsRisk.js library, we built a graphical demo web application that lets the user modify, view and perform queries on Trust

Figure 8: A screenshot from Travis CI showing the passing test suite

```
FullNode
  ✓ should call trust.addTX() on every transaction (1925ms)
  with the nobodyLikesFrank.json example
    ✓ computes trusts correctly
    ✓ after decreasing some trusts computes trusts correctly (755ms)

TrustIsRisk
  .getDirectTrust()
    ✓ returns zero for two arbitary parties that do not trust each other
    ✓ returns Infinity for one's direct trust to themselves
  .addTX()
    with a non-TIR transaction
      ✓ does not change trust
    with a trust increasing transaction
      ✓ correctly increases trust
      ✓ which has more than one input does not change trust
      ✓ which has a change output correctly increases trust
      ✓ which has two change outputs does not change trust
      ✓ which has a second output that is not a change output does not change trust
      ✓ which has been processed before throws
    with a trust decreasing transaction
      ✓ correctly decreases trust
      ✓ which has a second input decreases trust to zero
      ✓ which has more than one trust outputs decreases trust to zero
    .getTrustIncreasingMTX()
      ✓ creates valid trust-increasing transactions
    .getTrustDecreasingMTX()
      ✓ creates correct trust decreasing transactions
      ✓ creates correct trust stealing transactions
      ✓ throws when trying to decrease self-trust
      ✓ throws when there is not enough trust
    .getTrust()
      ✓ returns zero for two arbitary parties that do not trust each other
      ✓ returns Infinity for one's trust to themselves
      after applying the Nobody Likes Frank graph example
        ✓ correctly computes trusts
        ✓ correctly computes trusts when bob trusts frank


24 passing (13s)
```

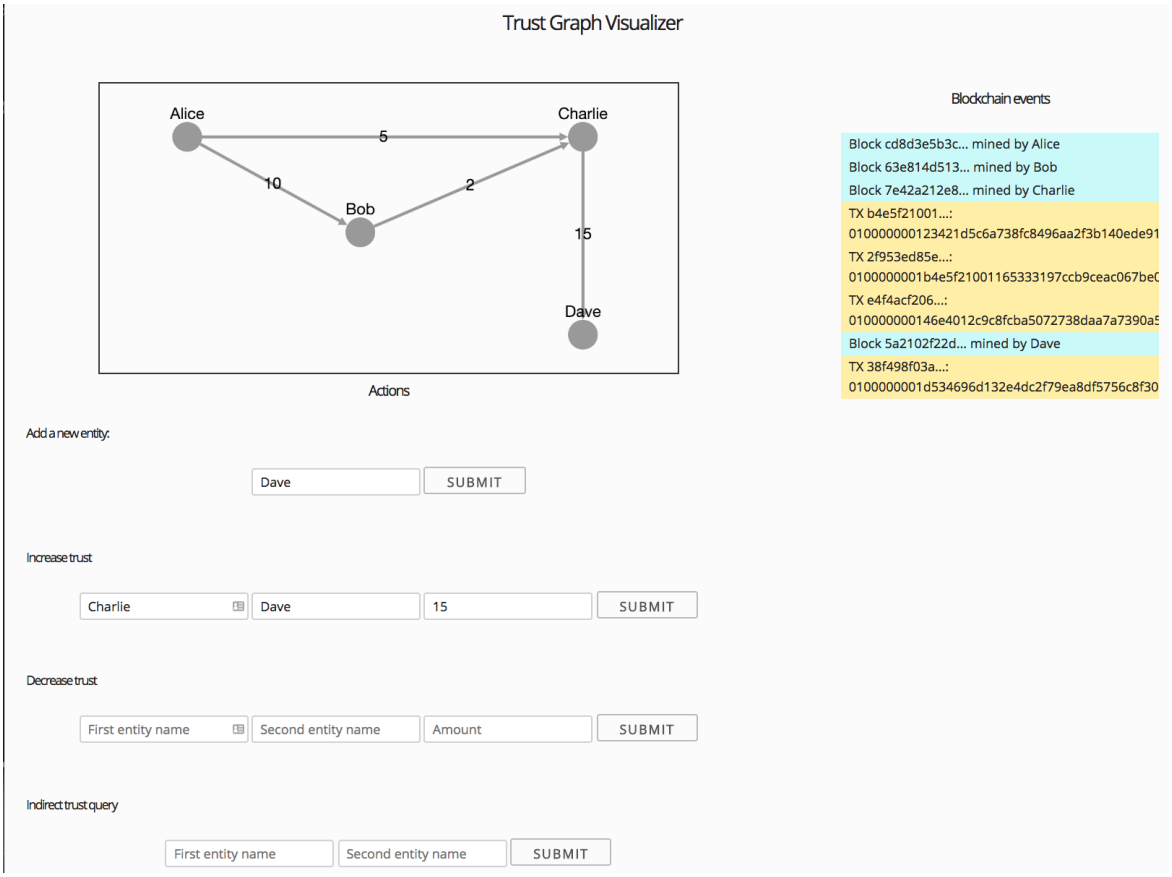
Is Risk trust graph. A screenshot of the demo application is presented in figure 9.

The frontend of the demo is a web page which displays the current state of the trust graph, recent blockchain events and raw transaction and a list of small forms which let the user execute certain actions. On the backend, the app creates a Trust Is Risk full node in regtest mode, meaning that a private, local blockchain with a single node is created. TrustIsRisk.js is then used normally on that blockchain to access the graph, perform trust queries and create trust-modifying transactions.

3.13 Evaluation and future work

TrustIsRisk.js is the first-ever implementation of a general decentralized trust inference library that is provably resilient to sybil attacks. We believe that this fact on its

Figure 9: A screenshot from the demo application



own makes it much better suited to decentralized applications like OpenBazaar than the traditional review and star rating based system that is currently being used.

The main disadvantages of the concept of Trust Is Risk are two. First, the user needs to be well-connected in the trust graph in order to gain value from its usage. Initially, there will be few people in the network and finding members who the user already knows and trusts will be difficult. Secondly, the user exposes themselves to the risk of having their money stolen by the people they directly trust. This however should be a very small and negligible risk if the trust placed on the platform is accurately based on real world relationships.

The implementation of Trust Is Risk, TrustIsRisk.js, necessarily shares these shortcomings. Additionally, the implementation could be improved by working on the following problems:

- **Performance**: Because we currently require a full Bitcoin node and have to process every single Bitcoin transaction that has ever happened, the initial set-

up takes a few days of processing power on an average machine. The solution to this problem is supporting Simple Payment Verification nodes [21], that operate without seeing every transaction in the network.

- **Privacy**: Currently, the entire trust graph is public, meaning that anyone can see how much any two pseudonymous entities trust each other. This may be an important shortcoming in some scenarios where it is important to hide one's outgoing and incoming direct trusts.
- **Transaction costs**: Because of the scalability issues Bitcoin has been experiencing over the past months [3], the transaction costs involved in creating trust-modifying relationships would be relatively high. It is difficult to quantify this cost, because transaction costs change on an hour-to-hour basis, and also because transaction fees are relative to the degree to which higher processing times are tolerable to the sender. We see the following possible solutions to this problem:
  - **Wait**: Bitcoin's scalability issues are likely to be solved in the next few months after SegWit [20] and Lightning [11] are adopted.
  - **Optimize transaction format**: We can re-define the Trust Is Risk transaction format so as to require as few transactions as possible. For example, we can allow trust-increasing transactions that extend an
  - **Use another cryptocurrency**: Trust Is Risk can be implemented in most other cryptocurrencies. As a last resort, we can move away from Bitcoin and use Ethereum.

On the 3rd of June 2017, the authors of the Trust Is Risk paper, OpenBazaar developers, OB1 representatives and I met for the deliverables presentation of OB1's funding of our research, one of which was my work on TrustIsRisk.js. We received positive comments and after the meeting the above problems were discussed in detail. It was decided that the next step towards integrating TrustIsRisk.js in OpenBazaar is to fix the performance issue by supporting SPV nodes.

4 Trust In Friends

Trust In Friends (for the lack of a better name) is my own attempt at designing a Trust Inference System. The idea is inspired by both Trust is Risk [31] and TrustDavis [27]. The most important concept in Trust In Friends is that users insure their friends against the misbehavior of their friends in a *transitive* way. This section formally expresses the ideas behind Trust In Friends.

4.1 Motivation

Like many other Trust Systems, Trust In Friends attempts to build a model of existing, out-of-band trust relationships that exist in the real world in a way that trust can be applied transitively. The intuition behind this is that *people trusted by people I trust can be trusted*. The resulting structure is again modeled as a graph.

Trust in Friends has a very important property that other systems lack: A participant can only lose money as a result of over-estimating the amount of money they can trust their friends and family with. We will prove this property later in this chapter.

Additionally, Trust In Friends provides clear incentives for the participants to engage actively in the system and form multiple relationship connections, resulting in a well-connected graph. It also aims to build a system where the individually best policy for every participant from a game theoretic point of view is to not deceive or fraud others.

If Trust In Friends were to be used in a decentralized, peer-to-peer money borrowing application it would mean that lenders can lend money to pseudonymous parties that are complete strangers, with the same risk as and confidence as if they were lending money to their friends. Borrowers would be able to borrow money from complete strangers by leveraging the trust their friends and family have in them, without any other kind of verification. But similarly to other trust inference systems, Trust In Friends is a generalized trust platform and could be used in multiple applications.

Trust In Friends could be implemented as an Ethereum contract, which uses a service like Oracalize [17] to verify frauds.

4.2 Formal description

We describe a decentralized trust network system in which pseudonymous participants form mutual Direct Trust relationships which involve insuring each other against the possibility of financial loss caused by the dishonesty of parties in their network, up to some amount. The system is not concerned with the justification behind a Direct Trust relationship, but some real-world examples are that users may choose to trust their mothers, friends and co-workers.

4.2.1 Proofs of Fraud

Below we make the assumption that fraud can be conclusively proved without disputes, and that these proves can be verified by an automated system. This is an unrealistic assumption for some (but not all) potential applications. This problem is discussed in more detail in 4.4.

If a party X commits fraud to deceive a person A, essentially stealing from her an amount of L, A will obtain a virtual token that we call *Proof of Fraud* and denote as $\text{PoF}^{L}_{A\to X}$. In order to recover part of her loss, she will then be able to sell this token with certain parties in the network.

Proofs of fraud are necessary to ensure that A can not make multiple claims for the same fraud instance. PoFs are similar IOUs in the sense that they represent an obligation of X to pay an amount L to the owner of the token, but unlike IOUs they also involve the *original beneficiary*. In other words, a $\text{PoF}^{L}_{A\to X}$ may be worth a different amount of money than $\text{PoF}^{L}_{B\to X}$, even though the amount and the fraudster are the same.

Proofs of fraud can be denominated into multiple tokens of smaller amounts. That is, any user can always exchange $\text{PoF}^{L}_{A\to X}$ for two tokens $\text{PoF}^{L_1}_{A\to X}$ and $\text{PoF}^{L_2}_{A\to X}$ as long as $L_1 + L_2 = L$.

Proofs of fraud are also tied to a particular fraud instance, which in the rest of this report is implicit and not denoted in the PoF notation, as we only consider one fraud instance at a time.

4.2.2 The trust graph

We represent the resulting network as a graph $G = (V, E)$ where the set V is the set of all the parties involved in the network. Edges are weighted by the weight function $w : E \to R^+$.

We define the MaxFlow(A, B), the Maximum Flow function as it is usually defined on undirected graphs. G is the implied graph parameter in the next few sections, unless another graph is subscripted, for example MaxFlow$_{G'}(A, B)$. When $(A, B) \in E$ and $C \in V$ we also define MaxNeighbourFlow(A, B, C) to be the maximum flow from A to C but by only considering paths that pass directly through B. More formally:

$$\text{MaxNeighbourFlow}(A,B,C) = \left\{ \begin{array}{ll} max(w(A,B), MaxFlow_{G\backslash(B,A)}(B,C)), & \text{if}(A,B) \in E \\ \text{undefined}, & \text{otherwise} \end{array} \right\}$$

Above, we use $G\backslash(B, A)$ to mean the graph G without edge (B, A). Please note that defining MaxNeighbourFlow is necessary because in undirected graphs only, it is possible that MaxNeighbourFlow$(A, B, C) \neq max(w((A, B), \text{MaxFlow}(B, C))$.

We also define the *d-th neighborhood of A*, $N_d(A) = \{X | \text{ShortestPath}(A, X) = d\}$, where ShortestPath$(A, B)$ is the shortest path in the unweighted version of G. In other words, $N_d(A)$ is the set of nodes which can be reached by traversing exactly *and no less* than d edges.

4.2.3 The λ parameter

The system has a global parameter λ where $0 < \lambda \leq 1$ which has been agreed on beforehand. The purpose of this parameter is to express the fact that parties further away from us in the trust graph can not be trusted as much as parties that are closer, and also to provide an incentive for users to participate and undertake risk in the system. A higher λ expresses more trust towards parties that are far away in the trust graph and provides greater incentive for participants to undertake risk in the system. Thus, choosing the right λ is a a matter of finding the "sweet-spot" in the trade-off between greater participation incentive and a smaller degree of trust transitivity.

4.2.4 Trust

An edge between two nodes represents a Direct Trust relationship between the corresponding parties. So for an edge $e = (A, B) \in E$ we say that *A directly trusts B to an amount of* $w(e)$. In such a relationship, B promises to always accept to buy from A a $\text{PoF}^{L}_{A \to X}$ with $X \in N_d(B)$ and $L \leq \text{MaxNeighbourFlow}(A, B, X)$ for an amount of $L \cdot \lambda^{d+1}$. This also holds vice-versa, because the graph is undirected and $(A, B) \in E \iff (B, A) \in E$. For a direct trust relationship $e = (A, B)$ to exist, both parties need to agree.

The intuition behind this definition is that in a trust relationship between A and B, if a party X steals an amount of L from A, and that party is d "hops" away, B will be obligated to pay A some amount of money to cover a part of her loss.

The Indirect Trust, or simply Trust, from a node A to a node B, denoted by $T(A, B)$ is simply the maximum flow from A to B: $T(A, B) = \text{MaxFlow}(A, B)$.

4.2.5 Fraud indicates wrong Direct Trust Relationships

We define a rational player as a player that will only take an action in order to maximize their utility with the following restriction: A rational player A will only engage in a transaction with player B if the amount at a risk M is at most $M \leq T(A, B)$. In other words, rational players only transact with others up to the amount of indirect trust towards them.

Rational players may form direct trust relationships with other players as a result of an out-of-band agreement. We call these relationships *wrong* if the out-of-band cost of breaking the Direct Trust agreement is less than the agreed amount. For example, consider Alice who is friends with Bob. Alice may think that B values their friendship at more than 50$, and therefore agree to form a Direct Trust relationship with Bob in Trust In Friends for 50$. If this is not the case, and Bob would happily not follow through the agreement at the cost of breaking their friendship, then we say that the Direct Trust relationship is wrong, and that *Alice shouldn't have trusted Bob*.

We can now prove the most important property in Trust In Friends:

Theorem 1 (Trust In Friends Theorem)
In a Trust In Friends network where players behave rationally, fraud can only occur in the presence of incorrect Direct Trust relationships in the network.

Proof by contradiction: Assume all Direct Trust relationships are correct. Player X commits fraud and steals L from A. From our definition of a rational player above, we know that $L \leq \text{MaxFlow}(A, X)$. As a result of the fraud, A now obtains $\text{PoF}^{L}_{A \to X}$. Since A is rational, she will attempt to exchange her token with her neighbors, possibly by splitting it into smaller tokens. All her neighbors will agree to the exchange as they have promised, because they value their out-of-band relationship with A more than the amount they have to pay. For the same reasons, PoFs will propagate through the network, and in the end, X will have paid a total amount of L to her neighbors in exchange for all the PoFs in circulation. The PoFs are now worthless, and X has lost an amount of L. Therefore, X's decision to steal L from A was not profitable, and X is not rational, a contradiction.

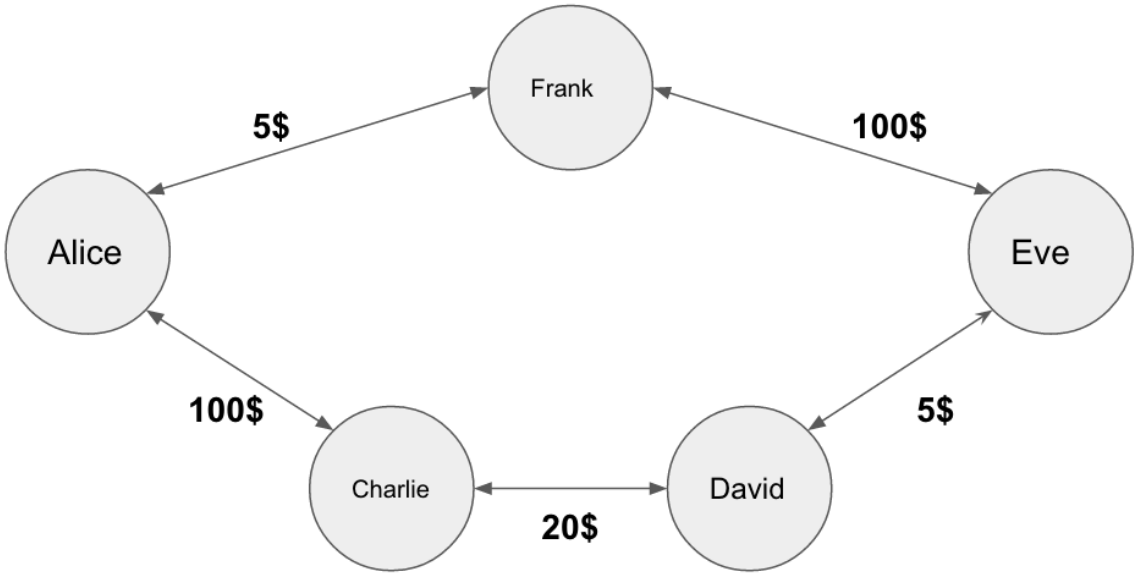
A result of this theorem is that in a Trust In Friends network with rational players and correct direct trust relationships, no player can loose any money. Incorrect direct trust relationships can result in losses to players other than those who formed them, but those losses will be largely be refunded, depending on the network topology and the λ parameter.

4.2.6 Sybil Resilience

Trust In Friends is not susceptible to Sybil attacks. The proof and intuition are almost identical to those in Trust Is Risk, and are based on the fact that indirect trust is defined as the maximum flow between two entities in the trust graph.

Let's assume that in the trust graph $G = (V, E)$ an adversary $E \in V$ wants to perform a Sybil attack by creating a set C of corrupted entities (vertices) that E controls. For the attack to succeed, the indirect trust $T(A, E)$ needs to be increased for some node $A \in V$. But for the maximum flow $\text{MaxFlow}(A, E)$ to increase, at least some node in $V \setminus (E \cup C)$ needs to increase their trust towards a node in $E \cup C$, which is as difficult as convincing that node to trust E directly. An adversary therefore gains no advantage from controlling corrupted parties, and Sybil attacks are pointless.

4.3 Example



Consider the trust graph depicted above, and let $\lambda = 0.9$. If Alice is rational, she will only risk up to $maxflow(\text{Alice}, \text{Eve}) = 10\$$ in transactions with Eve. Suppose Eve commits fraud and steals Alice's money. Alice now owns a $PoF^{10}_{\text{Alice}\to\text{Eve}}$ token. She can split the token into two $PoF^{5}_{\text{Alice}\to\text{Eve}}$, and try to sell one to Charlie and one to Frank, who have promised to buy them as part of their direct trust relationships with Alice. Please note that even though Charlie and Alice trust each other for 100\$, Charlie has only agreed to buy tokens with a value up to $\text{MaxNeighbourFlow}(A, B, X) = 5\$$.

Assuming Frank will not betray Alice (which would indicate a wrong Direct Trust agreement), he will buy her token for $L \cdot \lambda^{d+1} = 5 \cdot 0.9^2 = 4.05$ where d is the distance between Frank and the fraudster, Eve. Similarly, Charlie will buy his token for $L \cdot \lambda^{d+1} = 5 \cdot 0.9^3 = 3.64$. Alice has recovered $5.06 + 3.64 = 8.7\$$, and only lost $1.3\$$. She could have lost less money if the λ parameter was higher, or if she was closer in the graph to Eve.

Now the same process repeats with Charlie, who tries to sell the token to David and with Frank, who tries to sell the token to Eve. Every intermediary makes a small profit from forwarding the PoF and bringing it closer to the fraudster. Since, however, it very likely that Eve will refuse to pay (otherwise, as a rational player, she wouldn't have committed fraud in the first place), all players know that a wrong

direct trust relationship must exist somewhere between Alice and Eve. Somewhere along the way, someone will break their promise to buy the PoF from their neighbor. Therefore, when offered to buy the PoF they promised they would buy, players are given a choice. They can either:

1. Buy the PoF and undertake the risk of selling it to the next person. In this case, they avoid breaking their trust relationship and suffering out-of-band consequences, and are rewarded for the risk by the difference between the buying and selling price.

2. Refuse to buy the PoF, in which case they break their Direct Trust relationship with the seller and suffer the out-of-band consequences. This indicates a wrong agreement, i.e. that the seller shouldn't have trusted the buyer in the first place.

4.4 The problem of Indisputable Fraud Proofs

Trust In Friends requires indisputable fraud proofs in its current version. While this is not always a realistic assumption, there are cases where fraud can be proven, for example when loaning money or buying digital content online.

In the money lending case, one can prove that a borrower agreed to borrow money and never returned it by providing a cryptographically signed agreement which includes the Bitcoin addresses of the lender and the borrower. In this case, the system could automatically check the validity of a fraud claim before issuing a PoF.

In most other cases, proving fraud appears to be very complicated or even a matter of opinion: How does one prove that the t-shirt they ordered from eBay was of inferior quality than what was described by the seller, in such a way that an automated system can verify the claim? I see two solutions to the problem:

- It is likely that an automated solution which is imperfect but performs well enough would suffice. In the eBay t-shirt example, it might be enough to check that a parcel of the right size and weight was sent at the right time to the correct address by using the courier company's API.

- We could require every intermediary to confirm that a Proof of Fraud is valid manually, through human interaction. Naturally this would require a time

commitment from the intermediaries, who would in turn require a higher λ value to be better compensated for their effort. In the eBay t-shirt example, the intermediaries would receive a picture of the received t-shirt and judge whether fraud was indeed committed independently.

Systems like TrustDavis successfully address this problem, by including the risk of a fraud dispute in their calculations. This general idea might be applicable to Trust In Friends.

4.5 Evaluation and future work

Trust In Friends is currently a theoretical concept that has not been tested in practice. Similarly to Trust Is Risk, it is Sybil Resilient but also has the additional property described in section 4.2.5, according to which fraud can only be a rational action in the presence of over-estimated direct trusts in the network.

Future work should be directed mainly towards building a working and functional implementation of Trust In Friends as an Ethereum contract. Additionally, the problem of indisputable fraud proofs discussed in section 4.4 should be explored in more detail with a broader range of possible applications in mind.

5 Conclusion

Decentralized systems in general are a relatively new area of research, and the problem of trust between pseudonymous entities is one that affects many applications and prohibits others from existing. We expect that the trust problem, as it is defined and discussed in this report, will become more relevant in the future, as more users abandon centralized services and switch to their decentralized alternatives. Finding ways to infer trust towards online strangers, and to prevent fraud in general, is an extremely interesting problem, and finding a solution that does not rely on a central authority is necessity rather than a luxury in many scenarios.

There was relatively little previous work that was done on the subject that was directly relevant, and no functional implementation of a potential solution. As a result, it is difficult to compare our work to any existing systems.

While working on the project, I worked on two different fronts and produced two deliverables:

- **TrustIsRisk.js**: The first-ever implementation of a decentralized trust library that is resilient to sybil attacks without relying on a central authority. This implementation was developed with a real world scenario application in mind, that is already being affected by the problem we are trying to solve: OpenBazaar, the largest decentralized online marketplace. We are hoping that our work will be integrated in OpenBazaar in the near future.
- **Trust In Friends**: A new theoretical concept for a trust inference system in which users insure others against the dishonesty of their friends in a transitive way. Trust In Friends is also sybil resilient, and has the additional provable property that fraud can only ever be rational in the presence of incorrect direct trust relationships, which we define as over-estimations of how much an acquaintance can trusted. We believe that the general direction of using a reputation/trust system to make fraud irrational from a game theoretic perspective is very promising.

There are a lot of areas for improvement and further research in both of these two fronts. Future work on TrustIsRisk.js should concentrate on making it faster and more practical with the end goal of integrating it in OpenBazaar. Future work

on Trust In Friends could be done on implementing the concept on Ethereum and testing it in practice, as well as researching the theoretical concept more thoroughly.

References

[1] **Babel: A Javascript compiler**. https://babeljs.io/

[2] **BCoin: A Fullnode Bitcoin Implementation for Miners, Wallets, and Exchanges**. https://bcoin.io/

[3] **Bitcoin Scalability**. https://en.bitcoin.it/wiki/Scalability/

[4] **Bitcoin wiki: Transaction**. https://en.bitcoin.it/wiki/Transaction

[5] **Bitcoins the hard way: Using the raw Bitcoin protocol**. http://www.righto.com/2014/02/bitcoins-hard-way-using-raw-bitcoin.html/

[6] **BitcoinWisdom.com: Bitcoin Price History**. http://bitcoinwisdom.com

[7] **Docker: A better way to build apps**. https://www.docker.com/

[8] **Flow: A static type checker for Javascript**. https://flow.org/

[9] **Flow: A static type checker for Javascript**. https://flow.org/

[10] **Github: The world's leading software development platform**. https://github.org/

[11] **Lightning Network**. https://lightning.network/

[12] **Mocha: The fun, simple, flexible JavaScript test framework.** https://mochajs.org/

[13] **Namecoin**. https://namecoin.org/

[14] **NPM: The package manager for JavaScript and the worlds largest software registry.** https://www.npmjs.com/

[15] **OB1: Making online trade free for everyone, everywhere.** https://ob1.io/

[16] **OpenBazaar: A free market for all**. https://openbazaar.org/

[17] **Oracalize: data carrier for decentralized apps**. http://www.oraclize.it/

[18] **Prisoner's Dilemma**. https://en.wikipedia.org/wiki/Prisoner%27s_dilemma/

[19] **R3 CEV Homepage**. http://www.r3cev.com/

[20] **SegWit**. https://segwit.org/

[21] **SPV, Simplified Payment Verification**. https://bitcoin.org/en/glossary/simplified-payment-verification/

[22] **Travis CI: Test and deploy your code with confidence**. https://travis-ci.org/

[23] **Mt.Gox - Bitcoin Exchange**. https://web.archive.org/web/20110919162635/https://mtgox.com/press_release_\20110630.html. Version: September 2011

[24] **Australia Post Plan for Blockchain Voting by 2017**. https://www.cryptocoinsnews.com/australia-post-plan-blockchain-voting-2017/. Version: August 2016

[25] **China's Social Security to Use Blockchain Tech**. https://www.cryptocoinsnews.com/chinas-social-security-use-blockchain-tech/. Version: September 2016

[26] How blockchain tech could change the way we do business. In: **BBC News** (2016), Januar. http://www.bbc.co.uk/news/business-35370304

[27] DEFIGUEIREDO, Dd B. ; BARR, Earl T.: Trustdavis: A non-exploitable online reputation system. In: **E-Commerce Technology, 2005. CEC 2005. Seventh IEEE International Conference on** IEEE, 2005, S. 274–283

[28] DOUCEUR, John R.: The sybil attack. In: **International Workshop on Peer-to-Peer Systems** Springer, 2002, S. 251–260

[29] HERN, Alex: Blockchain: the answer to life, the universe and everything? In: **The Guardian** (2016). https://www.theguardian.com/world/2016/jul/07/blockchain-answer-life-universe-everything-bitcoin-technology

[30] KAMVAR, Sepandar D. ; SCHLOSSER, Mario T. ; GARCIA-MOLINA, Hector: The eigentrust algorithm for reputation management in p2p networks. In: **Proceedings of the 12th international conference on World Wide Web** ACM, 2003, S. 640–651

[31] LITOS, Zindros Stefanos: Trust Is Risk: A Decentralized Financial Trust Platform. In: **International Conference on Financial Cryptography and Data Security.**, 2017

[32] LOMAS, Natasha: **Everledger Is Using Blockchain To Combat Fraud, Starting With Diamonds**. `http://social.techcrunch.com/2015/06/29/everledger/`

[33] NAKAMOTO, Satoshi: Bitcoin: A peer-to-peer electronic cash system. (2008)

[34] WILLIAMS, Moly: A Living Dream. In: **Wall Street Journal** (2001). `https://www.wsj.com/articles/SB10009189862256424 80`