Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: December 7, 2018

J. Pouwelse, Ed.
Delft University of Technology

June 5, 2018

Trustchain protocol
draft-pouwelse-trustchain-01

Abstract

Trustchain is a protocol for a networked datastructure, designed to act as a trust ledger. This protocol acts as a decentralized alternative to platforms like eBay, Airbnb, and Uber. It is specifically designed to record transactions among strangers without central control, support high transaction volumes, be application neutral, and avoid vendor lock-in. The protocol defines recording transactions in an ordered list using an append-only datastructure, a new communication overlay, and a horizontally scalable consensus protocol based on checkpoint consensus, called CHECO. Trustchain has resistance to traditional blockchain attacks, such as the 51 percent majority attack. This is achieved by using a graph-based append-only datastructure combined with a personal blockchain for each participant with their own genesis block.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 7, 2018.

Table of Contents

1. Introduction

For the past 10 years various distributed ledgers have been deployed and used. This protocol aims to establish some form of trust using software.

Creating trust between strangers is at the core of numerous successful Internet companies. Starting 22 years ago, Craigslist

offered an unmoderated mailing list of advertisements and gossip on which buyer and seller could be trusted. eBay formalised this in 1997 and introduced a star-based rating system that enables traders to build a trustworthy profile [resnick2002trust]. The e-commerce platform was launched at a time when people were still hesitant to use their credit card on a technology called The Internet. Nowadays, people let strangers sleep in their houses using Airbnb (since 2008). We trust Uber (since 2009) with our physical security and get into cars late at night with a driver that has not undergone a criminal background check or given a government license.

The Trustchain protocol aims to create a generic approach and continues this evolution of building trust. Compared to successful central platforms, we propose a distributed open underlying infrastructure, based on blockchain inspired technology. Bitcoin created money without the need for banks [nakamoto2008bitcoin]. In the past, people were required to trust a central bank and a host of other intermediaries when making payments [kokkola2011payment]. The fundamental technology of Bitcoin, blockchain, radically reduced the need to trust financial middlemen. It bootstrapped an economy where no one can be stopped from spending their money. Despite widespread speculation and ecosystems being worth billions, blockchain in general suffers from scalability issues due to inefficient mechanisms for fraud prevention. Bitcoin is theoretically limited to seven transactions per second and Ethereum has a throughput of around 20 transactions per second [vukolic2015quest]. Despite various scalability efforts like proof-of-stake and sharding, broader adoption of blockchain stays out. Mt. Gox was at one point the largest Bitcoin exchange worldwide. While a majority of Internet users trust the company behind popular platforms, the events involving Mt. Gox highlighted how digital trust can be established and compromised[mcmillan2014inside]. In 2014, hackers stole Bitcoin, worth around $460 million at that time. This event, together with major data breaches in 2017 at high-profile companies like Uber and Equifax, exposed the weakness of centralized architectures [apostle2017uber]. These events motivated this proposed protocol around decentralised infrastructures, not owned or operated by a single authority. The generic problem of building trust between strangers resides on the edge of technology, sociology and behavioural science [yan2008trust]. The question whether someone can be trusted, depends on properties like personality, level of authority, culture and past behaviour. In this protocol, we address the trust problem from a technological perspective, using tamper-proof interactions on a scalable blockchain. This structure is built to help ease the detection of fraudulent behaviour and misrepresentation. Trust calculations are out of scope of this work, we provide the enabling mechanisms.

1.1. Purpose

This draft describes the Trustchain architecture, protocols and the used technologies, designed to model application neutral trust between interacting parties. Trustchain relies on a new communication layer on top of existing communication networks, which is designed with carrier-grade NAT infrastructure in mind, as well as the network protocol based on the CrawlRequest and TxBlock message types. A consensus protocol called CHECO (Cong et al, 2017 [cong2017blockchain]) is incorporated into Trustchain, which will be discussed but not elaborated in this draft. It is based on the blockchain paradigm where the complete network represents a ledger where agents' transactions infer an amount of trust between the involved parties, as is described by pouwelse, 2017 [pouwelse2017trustlaws].

As protocols have slowly been moving towards the business layer in the past decade, Trustchain is implemented on top of a networking overlay and as such is network agnostic. Other examples of moving networking to the business layer are: R3 Corda (Brown, 2017 [brown2017introducing]) and IOTA (Atzori, 2016 [atzori2016blockchain]). The overlay, audaciously called IPv8, provides encrypted communication between public keys. This overlay has integrated NAT puncturing to support, for instance, Android-to-Android overlay communication, does not require any central server, lacks central authorities, and can run directly on top of UDP, TCP, or other protocols. As such, IPv8 provides a set of communication methods and messages that provide the required functionality to let Trustchain function properly on both PC networks or smartphone-only networks.

1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.3. Terminology

Identity
: The actual user representation, but can not be directly used, since all information is identifier based.

Identifier
: A reference that is owned by a given identity, referring to this identity. Any identity can have multiple identifiers, whilst staying anonymous.

Agent
: A node in the Trustchain network representing an identifier for a given identity.

Message
: The basic unit of Trustchain communication. A message will have different representations on the wire depending on the transport protocol used. Messages are typically multiplexed into a datagram for transmission.

Datagram
: A sequence of messages that is offered as a unit to the underlying transport protocol (UDP, etc.).

Transaction
: An interaction between two agents containing information on both parties and what has been transacted. This is application agnostic, meaning that any given application can infer what type of information it needs based on a collection of transactions.

Signature
: A cryptographic function that used the private key to create a representing string, which can be verified by any other party using the signer's public key.

Hash
: The result of applying a cryptographic hash function, more specifically SHA-256 [FIPS180-4], to a piece of data.

2. Trustchain Stack: Engineering trust

Our principal mechanism to establish some form of trust is: if everyone keeps their secret keys secure, then no signed transaction can be spoofed on the overlay by any significant likelihood. We refer to the Trustchain protocol when discussing the mechanism to record interactions which are cryptographically signed by multiple parties. We explicitly do not support transactions signed by only a single party, which is the foundation of Bitcoin. Our foundation is a multi-signature agreement, without mono-signature support. The Trustchain stack refers to the full system which also includes the upper application layer, the network overlay and self-sovereign identity layer. Together they form a complete solution stack.

The concept of a Self-Sovereign Identity (SSI) (Abraham, 2017 [abrahamself]) means that agents have full control over their identity data, and provide it to those who need to validate it, without relying on a central repository of identity data of any kind. A large part of any SSI based system is rooted in the problem of

proofs and attestations: proof can be anything, such as a secret message that is re-encrypted with a shared secret. But the most challenging part is working with attestations. An Attestation refers to the process wherein a third party validates that according to their records, the claims are true, that is, a more transitive property of trust: C attests to B that A is who it claims to be, Azouvi et al, 2017 [azouvi2017secure]. As such, an attestation from the right authority could be more trustworthy than a proof, which might have been forged. However, attestations can be a burden on the agent as the information can be sensitive, hence, the information needs to be maintained so that only specific agents can access it. Our stack does not constrain the choice of SSI system, but our implementation is focused on the Boneh-Franklin [boneh2004secure] 2-DNF scheme ("Evaluating 2-DNF Formulas on Ciphertexts").

Based upon the assumption that these identities are persistent and secure, the new architecture (or Fabric) is designed to use Peer-to-Peer communication to increase the transaction throughput. This communication is based on the new networking overlay: IPv8, which handles peer discovery, making connections with them across NAT boxes and peer-to-peer cryptographically signed messaging. As IPv8 is transport and application agnostic, it can run over any transport protocol: it does not depend on IP and may run on top of NDN, XIA, and other new Internet architectures.

To ensure that the blockchain is always in a valid state, a new horizontal scaling consensus protocol is proposed: CHECO. CHECO is specifically designed to counter the vulnerabilities that a distributed, permissionless, multi-chain architecture will have to cope with (although this also creates innate vulnerabilities to other kinds of attacks). By creating an indication of the state of validity for each agent, the responsibility of verification lies with the agent itself. A malicious agent in an invalid state can easily be detected, and should be avoided for interactions.

We do not constrain or limit the applications utilising this blockchain, as each transaction block can contain information of arbitrary content and length. Currently, there is a basic decentralised market implemented. More specialised markets have to also be implemented and emulated such as an open Taxi service market and a mortgage investment market. Multiple applications can be used at the same time, next to the decentralised market, for instance: it could be used for byte accounting in an ad-hoc Manet [jethanandani2017accounting].

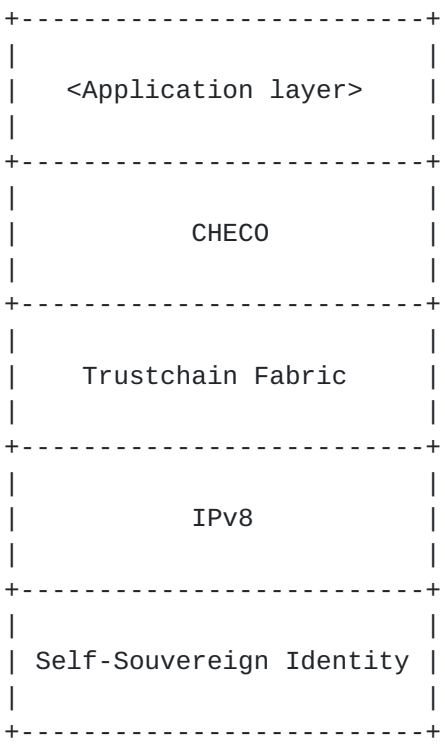


Figure 1

3. Trustchain Fabric: internal data structure

Trustchain is designed to be a non-blocking format for agents that supports simultaneous interactions with other agents. Non-blocking is a requirement rooted in the immutability of the chain and the strict ordering of the blocks. To support this, the blocks are designed to be dependent on signing by all participating agents, and will be called TxBlocks henceforth, as is described in this section, along with the macro data structure in which these blocks are used.

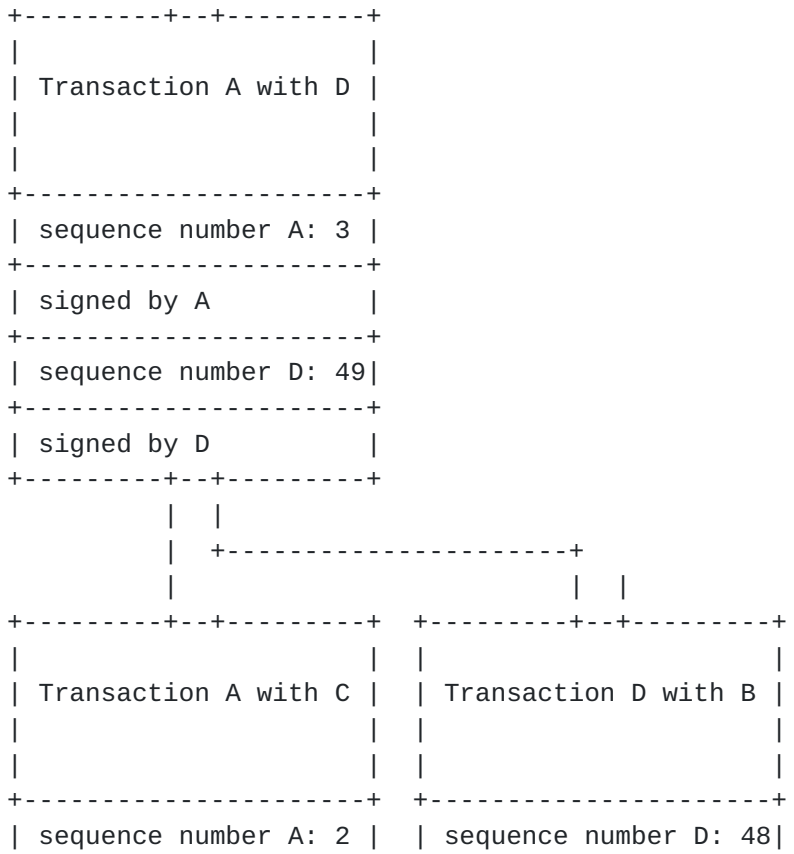
3.1. Architecture

In contrast to traditional blockchains, in Trustchain every agent in the network has its own genesis node, in essence creating a personal blockchain for each agent. Each interaction creates a new transaction block, which is based on the last block of the two (or more) concerned parties. This does not only influence the block-creation speed, but also the amount of effort needed to verify a chain. Along with some other security properties, this is one of the implicit capabilities of this protocol.

By removing the proof-of-work mechanism needed for classic blockchain implementations, Trustchain yields inherent horizontal scalability.

However the cost of scalability is that each application requires a mechanism to guard against transaction spam and abuse. Trustchain is based on the assumption that both parties agree on the transaction before signing it, making tampering inherently easy to detect. One of the aspects that supports this is the fact that Trustchain is organised as a set of temporally ordered, intertwining chains, which form a Directed Acyclic Graph (DAG). This is called a "bottom-up consensus model", giving the participants the responsibility to verify the correctness of the transaction instead of a central (sometimes randomly chosen) elected leadership.

Trustchain depends on signatures from all participants in a transaction, creating an n-to-n node. This system is extendable, as mentioned before, by extending the transaction description to provide specific properties. Each transaction is stored in a block, in agreement-block format, each block has parts that are signed and submitted by all participating parties, where the initial request is called the block-proposal and a completely signed and validated block is called an agreement-block (where a pair indicates the cooperation, not the limitation to two participating parties). These are signed and sequenced so that each sequence number is unique in its accessory chain, as can be seen below in the general structure diagram.



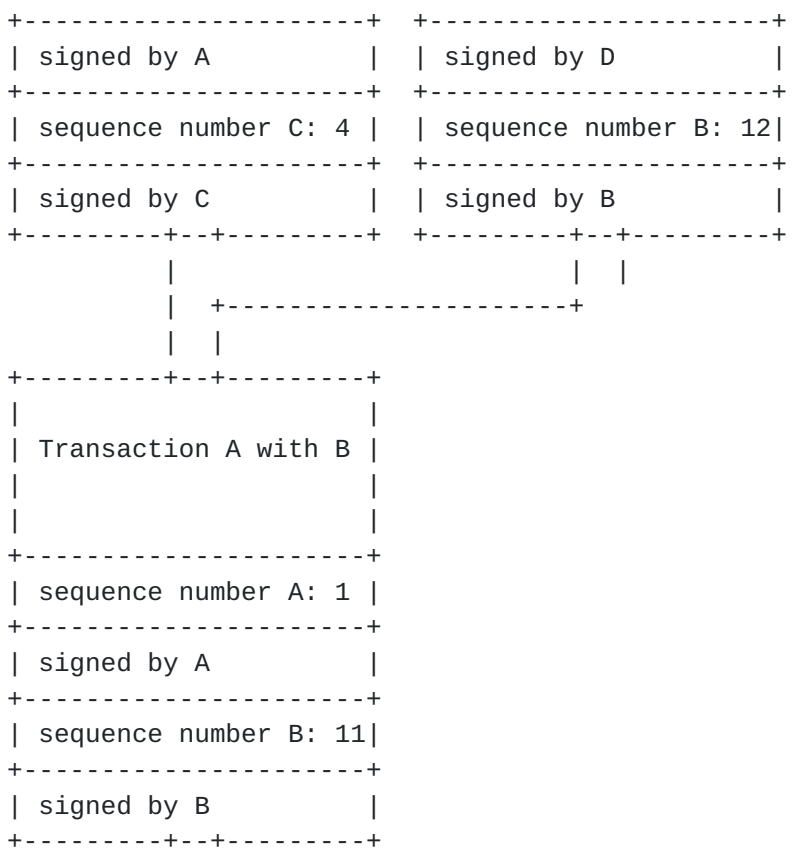


Figure 2

3.2. TxBlock specification

Using the proposal-block and agreement-block formats means singing the blocks on the current views of the respective parties: the requester and the responder(s). Each party signs and fills the block with the information that it has at that specific point in time. The requester fills the structure with its own previous hash and its own part of the transaction data, signs it and sends it to the responder(s), which in turn construct the other sections of the block, if it agrees with the content before sending it back. This nullifies any ordering and asynchronicity issues, since the requester constructs the block with the information that it has, and keeps it in memory while it waits on the responder to send the finished block back.

Number	Description	Type	Size (bytes)
1	Requester public key	Char array	74
2	Requester sequence number	Unsigned int	4
3	Responder public key	Char array	74
4	Responder sequence number	Unsigned int	4
5	Requester previous hash	Char array	32
6	Signature	Char array	64
7	Transaction block size (n)	Unsigned int	4
8	Transaction block	Char array	n
	Total:		256 + n

Table 1: TxBlock fields description

3.3. Asynchronicity

Because there is the need to communicate between the requester and responder(s), there will be a delay which may be significant. To have a high level of asynchronicity and enable multiple peers interacting simultaneously, extending the chain should be possible while waiting for a response. In order to do this, the block refers to the previous block using the hash of the requester's part, since this is the only stable reference at that point. The other hash reference (the "previous hash responder") can then either be the "hash requester" or "hash responder" part of the head-block of the responder chain. Which one is used depends on whether the responder was the requester or responder in its previous interaction. This mechanic is also used for the "previous hash requester" field, but this reference is known when the block is created. In effect, this results to theoretically unlimited horizontal scalability: the more actors are active on the chain, the more throughput can be achieved. Though this is in fact limited by the memory speed, or database slowdown when the chain grows.

One of the drawbacks of this mechanic is when the responder does not sign and respond, whether because it will/can not. In such cases, there will be an orphan block. While this is not a vulnerability in itself, it might be the starting point of a certain type of attack (the other "normal" types of attacks used for blockchains can be mitigated, at least to a certain level, as is described in resistances (Section 5).). The adversary might let someone initiate a transaction, i.e. a block creation, after which it will create an orphan. Doing this multiple times in a short time span will force the requester to use a considerable amount of processing power and memory, all the while injecting orphan blocks into its chain. As mentioned before, this is not a vulnerability in itself, but might be

a launchpad for a more elaborate attack. Although, one coping method could be to split larger, more vulnerable transactions up into multiple smaller transactions. This way, the consequences stay the same for the malicious actor, but the losses are smaller.

3.4. CHECO: Consensus protocol and block format

The final part of the Trustchain Fabric is CHECO, a horizontally scaling consensus protocol specifically designed for multi-chain implementation and completely application agnostic. CHECO is based on three separate protocols: A consensus protocol, a transaction protocol and a validation protocol, as well as an extension of the architecture by introducing a new type of block next to the TxBlock: CpBlock. Every round a set of so-called facilitators is selected at random, which collect the CP and TX blocks, to feed to the validation protocol, after which, the results are broadcast before a new round starts.

CHECO is designed to create an internal state ledger for each chain, without having to rely on separate methods or instances. This is achieved by introducing a new block: the Checkpoint block (CpBlock), which contains a hash pointer to the previous block, along with a hash of the consensus result, round and sequence number and, lastly, a signature. The consensus result is defined as a tuple containing the validity states of the blocks agreed on by the facilitators of that round and the round number. If your chain is deemed valid, a CpBlock is injected, thus validating the state of the chain. While the content of the injected CpBlock differs from the TxBlock, these blocks do not interfere with the transaction protocol, since they fit in the architecture without modifying it.

Instead of requiring a proof-of-work (as seen in more conventional blockchain implementations), CHECO is round based, creating a consensus state every so often, thus enabling a fully asynchronous and horizontally scaling protocol. These facilitators are chosen randomly each round, and will collect the CpBlocks from all other nodes since their last CpBlock. Validation is done using the Asynchronous Common Subset (ACS) algorithm based on HoneyBadgerBFT (Miller et al, 2016 [miller2016honey]), a byzantine consensus algorithm.

4. IPv8: Overlay for identity, discovery and trust

To enable this new platform to function properly, a new method to find, connect to and manage agents was needed. Additionally, new models for identity verification, network discovery and inter-peer trust were required to enable these agent methods. IPv8 is a network stack, a set of protocols and models, that separates concerns and

enables applications (such as Trustchain) to use the needed methods and protocols, without giving up interoperability and upgradeability. On top of this interface lies the actual Trustchain overlay, which uses the components and protocols of the IPv8 interface to create the specific functionalities needed for Trustchain to function.

IPv8 is built by closely abiding to the Unix Philosophy of creating small components that are easy to understand and test. This is mainly why the complete networking system used in Tribler was re-written as a generic networking interface, enabling modifications and additions without losing any functionality. Although agents might use different protocols based on their capabilities, IPv8 is a basic layer over a multitude of networking components and subsystems, creating a means to communicate with other agents regardless of networking capabilities. An open source reference implementation based on Python is available on Github, called py-ipv8. Interoperable open implementations in Java and Kotlin are still only partially functional.

4.1. Identity establishment and discovery

Identities are created, attested and distributed over the Trustchain using IPv8 as the communication interface. These are all public and self sovereign, leading to distribution when an agent creates a new identity, and are organised using a Public Key Infrastructure (PKI). This distribution is done to agents who have (or might have) an interest in obtaining the identity of this agent, since they are in the same cluster or due to other factors. Spreading information this way is called Gossiping, since agents learn of new identities, and spreading them among directly connected agents like a gossip would spread.

Discovering identities is done based on Distributed Hash Tables (DHT), somewhat similar to the Domain Name System (DNS) currently used for the web, using Random-Walk and Live-Edge-Walk: discovery protocols for DHTs, respectively based on making random DHT queries in order to learn about a large number of identities quickly and making pseudo-randomised queries about the agents with the highest trust scores in the network. The use of random-walks enables DHTs to converge much faster, whilst having a small load at the very beginning. Furthermore, by traversing the live edges, Trustchain is more spam and Sybil attack resistant.

4.2. Attestations and trust

The primary problem with identities and proofs is falsification, and, thus, these need to be verified to prevent this. However, even proofs can be forged, leading to the problem of needing trust in a

proving party, which can not be solved by having a more centralised proving party. This is where attestations are used: a witness reports that the identity in question is actually valid. This requires some level of trust between the agents. Attestations are a method to enable agents to validate interactive zero-knowledge proofs (ZKPs) within a network of agents, a so-called web-of-trust [azouvi2017secure], where the transitive property of trust is used to prevent the need for every agent to verify an identity itself as is noted in Section 2.

In this case, the probabilistic homomorphic asymmetrical encryption scheme of Boneh-Franklin [boneh2004secure] is used to validate these proofs, meaning that a form of randomness is used in the encryption, and computations on the ciphertext result in a valid plaintext. Using Boneh-Franklin leads to these ZKPs to be hardened against chosen plaintext attacks (an attacker can encrypt the suspected plaintext to see if the ciphertexts match) with the probabilistic aspect. These attestations are tied to metadata, which can be verified separately. The related identities are stored internally in a database, and the aforementioned metadata attributes are gossiped around the network using Trustchain.

4.2.1. Technical view of Attestations and Verifications

4.2.1.1. Attributes

Attributes are a generic term which can essentially refer to any verifiable piece of information. For instance, an attribute may define a peer's identity, the amount of cryptocurrency in one's digital wallet, or perhaps, something as simple as one's previous number of transactions. In the context of Trustchain, Attributes mainly refer one's identity.

4.2.1.2. Attestations

In a truly distributed peer-to-peer system, where peers regularly communicate and exchange information (hence they exchange attributes) there is a need of verifying the veracity of attributes, without relying on a well-known, well-trusted tertiary party / peer. Attestations are a means which make this task possible. They allow peers to verify the truthfulness of Attributes through a validation process, using interactive zero-knowledge proofs. A system based on Attestations is founded on the fact that, intrinsically, there exists a degree of trust between the peers of a system. To this extent, attestations are, generally speaking, a means through which peer C vouches to peer B that A is indeed telling the truth. Unlike normal centralized systems, where C is a well-known entity, in this case, C may be any trustworthy fellow peer. This enforces the idea that

trust is transitive, hence there is no need for a trusted central peer to exist, since one can rely on other peers to attest for the veracity of Attributes.

4.2.1.2.1. The Attestation process

As its name suggests, the Attestation Process is a mechanism through which a peer (the Attestee) requests the attestation of one of its Attributes from another fellow peer (the Attester). The process is initiated by the Attestee. If the process is successful, and the Attribute has indeed been attested, the Attester produces an attestation as proof, which is returned to the Attestee. This should complete the process. In the sections that follow, the messages exchanged during the aforementioned process are detailed. Additionally, a more detailed, high-level description of the peer interaction is also presented.

4.2.1.2.2. Message types

The following sections present the different types of messages that are exchanged between the Attestee and Attester peers during the attestation process, namely the Attestation Request message and the Attestation Response message. A section on the prefix field, which is employed in all messages used towards peer communication, is also presented in what follows.

4.2.1.2.2.1. Preamble field

The preamble field is present in all the messages employed in Trustchain towards peer communication. Consequently, it plays an important role in inter-peer communication. Figure 3 presents its generic structure.

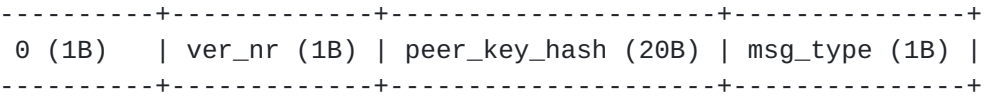


Figure 3: The structure of the Preamble / Prefix field. The size of each field is specified in parentheses in Bytes.

A description of the Preamble field's elements is presented in Table 2.

FIELD	OCTETS	DESCRIPTION
0	1	The initial byte of any message is 0 (0x00).
ver_nr	1	The current implementation version.
peer_key_hash	20	A hash of the key of the message sender's master peer.
msg_type	1	Indicates the nature of the message. For instance, when 'msg_type = 5', the message will be an ATTREQ. Similarly, when 'msg_type = 2', the message is either ATTRESP or VFRESP. Other types are also available.

Table 2: Description of fields in the Preamble / Prefix field

4.2.1.2.2.2. The Attestation Request Message

The Attestation Request message (ATTREQ), is forwarded by the Attestee towards the Attester, in order to indicate the Attestee's desire to have one of its attributes attested by the Attester. The structure of the ATTREQ message is presented in Figure 4.

Number	Description	Type	Size (bytes)
0	Preamble / Prefix	Char Array	23
1	Public Key in Binary Format	Unsigned Short Array	n
2	Global Time (Lamport Timestamp)	Unsigned Long Long	8
3	Attestation Metadata	Char Array (Raw)	33 +

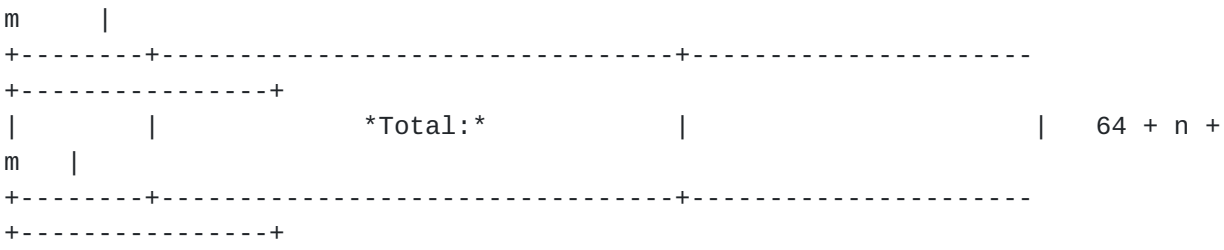


Figure 4: The structure of an ATTREQ message.

A description of the ATTREQ's fields is presented in Table 3.

FIELD	OCTETS	TYPE	DESCRIPTION
Preamble / Prefix	23	Char Array	Generic message preamble, as presented in Section 4.2.1.2.2.1, with msg_type = 5.
Public Key	n	Unsigned Short Array	The Attestee's public key in binary format. n = bin_public_key_char_length * 2, since each Unsigned Short element is 2 Bytes in size.
Global Time	8	Unsigned Long Long	A Lamport Timestamp (Scalar Clock) associated with the message.
Attestation Metadata	33 + m	Char Array (Raw)	Holds a string having the pattern: '{"attribute": "attr_name", "public_key": pub_key}' (without the ' characters). The attr_name and pub_key tokens are replaced by actual values. The minimal size of the field is 33 Bytes (if both tokens would be omitted). m is the cumulative string length of the values in the attr_name and pub_key tokens.

Table 3: Description of the fields in an ATTREQ message

4.2.1.2.2.3. The Attestation Response Message

The Attestation Response message (ATTRESP), is forwarded by the Attester towards the Attestee, in order to send the previously requested attestation to the Attestee. It should be noted that ATTRESP messages are sent only in case the attestation is successful. If the attestation fails, no such messages are sent. Furthermore, the attestation may be of arbitrary length, hence, it is impractical to send it entirely at once. Consequently, the attestation is broken up into smaller chunks (at most 800 Bytes) and each chunk is packed as part of an ATTRESP message. The structure of an ATTRESP message is presented in Figure 5

Number	Description	Type	Size (Bytes)
0	Preamble / Prefix	Char Array	23
1	Public Key in Binary Format	Unsigned Short Array	n
2	Global Time (Lamport Timestamp)	Unsigned Long Long	8
3	Attestation Blob Hash	Char Array	20
4	Sequence Number	Unsigned Short	2
5	Attestation Blob Chunk	Char Array (Raw)	800
	Total:		853 + n

Figure 5: The structure of an ATTRESP message.

A description of the ATTRESP's fields is presented in Table 4.

FIELD	OCTETS	TYPE	DESCRIPTION
Preamble / Prefix	23	Char Array	Generic message preamble, as presented in Section 4.2.1.2.2.1, with msg_type = 2.
Public Key	n	Unsigned Short Array	The Attester's public key in binary format. n = bin_public_key_char_length * 2, since each Unsigned Short element is 2 Bytes in size.
Global Time	8	Unsigned Long Long	A Lamport Timestamp (Scalar Clock) associated with the message.
Attestation Blob Hash	20	Char Array	A hash of the entire attestation blob. This can be used to check the integrity and completeness of the reconstructed blob on the Attestee's side.
Sequence Number	2	Unsigned Short	The message's sequence number in the sequence of ATTRESP messages sent for the particular attestation.
Attestation Blob Chunk	800	Char Array (Raw)	A chunk of the attestation blob.

Table 4: Description of the fields in an ATTRESP message

4.2.1.2.3. Attestee-Attester Interaction - Attestation Process

The following summary refers to the interaction between Attestee and Attester during the attestation process. The timeline diagram in Figure 6 shows the timing relationships in such an interaction. Table 5 gives a brief reminder of the messages employed in the attestation process.

The Attestee forwards an ATTREQ message to the Attester, with the aim of requesting attestation for an Attribute, as indicated in the ATTREQ message.

After receiving the ATTREQ, and successfully completing the attestation process, the Attester responds with a series of ATTRESP messages. The message sequence may be of arbitrary length. At a high-level, the sequence of ATTRESP message returns the new attestation, hence, each message will contain a chunk of the attestation blob. Additionally, each ATTRESP message also contains a hash of the entire attestation blob, a message sequence number, and additional message fields. If the attestation is unsuccessful, no ATTRESP messages are returned.

The Attestee receives the sequence of ATTREQ messages from the Attester, and puts them together. The Attestee will know it has received the entire attestation blob, when the hash of the attestation blob, present in the ATTREQ messages, matches a locally computed hash of the reconstructed attestation blob.

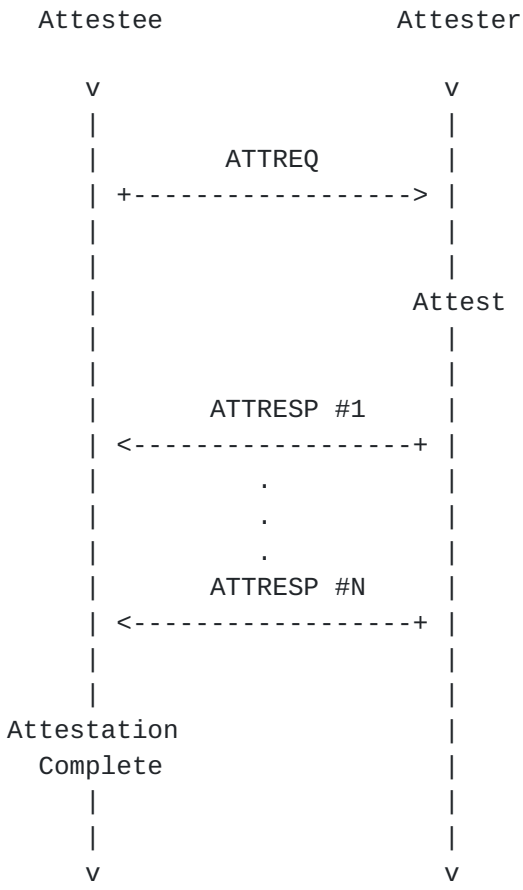


Figure 6: Timeline diagram of the messages exchanged between Attestee and Attester during the attestation process.

MESSAGE	USE
ATTREQ	Attestee message to Attester requiring the attestation of an indicated Attribute.
ATTRESP	Attester message to Attestee provided that attestation was successful. Each ATTRESP carries a part of the successful attestation blob.

Table 5: Attestation process message summary

4.2.1.3. Attestation Verification

Whilst Attestations are employed towards ensuring that peers are truthful in regards to some of their Attributes, the Attestation Verification request is employed towards ensuring that the Attestations themselves are valid. The employed mechanism does not stray far away from that which is employed in the case of Attestations. In fact, they are structurally similar: during the process of an Attestation Verification a peer C vouches to peer B that A is indeed telling the truth regarding a supplied Attestation. Once more, this enforces the idea that trust is transitive, hence there is no need for a trusted central peer to exist, since one can rely on other peers to attest for the veracity of an Attestation.

4.2.1.3.1. Attestation Verification Process

The Verification process defines an interaction between two peers, the Requester peer, and the Verifier peer, wherein the Requester demands that the Verifier verifies one of its attestations (that is, an attestation of the Verifier). Intuitively, the Requester peer initiates the process. If the verification is successful, the Verifier should return the verified attestation. This should complete the process. In the sections that follow, the messages exchanged during the aforementioned process are detailed. Additionally, a more detailed, high-level description of the peer interaction is also presented.

4.2.1.3.1.1. Attestation Verification message types

The following sections present the different types of messages that are exchanged between the Requester and Verifier peers during the verification process, namely, the Verification Request message and the Verification Response message. Incidentally, the Verification Response message is identical to the Attestation Response message, which was presented in Section 4.2.1.2.2.3.

4.2.1.3.1.1.1. The Verification Request Message

The Verification Request message (VFREQ), is forwarded by the Requester towards the Verifier, so as to demand that the Verifier verifies one of its attestations, and then sends it back to the Requester. The structure of the VFREQ message is presented in Figure 7.

Number	Description	Type	Size (Bytes)
0	Preamble / Prefix	Char Array	23
1	Public Key in Binary Format	Unsigned Short Array	n
2	Global Time (Lamport Timestamp)	Unsigned Long Long	8
3	Hash of the Verified Attestation	Char Array	20
	Total:		51 + n

Figure 7: The structure of an VFREQ message.

A description of the VFREQ's fields is presented in Table 6.

FIELD	OCTETS	TYPE	DESCRIPTION
Preamble / Prefix	23	Char Array	Generic message preamble, as presented in Section 4.2.1.2.2.1, with msg_type = 1.
Public Key	n	Unsigned Short Array	The Requester's public key in binary format. n = bin_public_key_char_length * 2, since each Unsigned Short element is 2 Bytes in size.
Global Time	8	Unsigned Long Long	A Lamport Timestamp (Scalar Clock) associated with the message.
Verified Attestation Hash	20	Char Array	A hash of the attestation which needs to be verified.

Table 6: Description of the fields in an VFREQ message

4.2.1.3.1.1.2. The Verification Response Message

The Verification Response message (VFRESP) is identical in structure to the Attestation Response message (ATTRESP), since both return an attestation. For the VFRESP's structure please see section Section 4.2.1.2.2.3. The two messages differ only slightly in terms of semantics, since one is returned as a response to a successful attestation request, thus returning a new attestation, while the other (the verification) is returned upon a successful verification request, thus returning the recently verified attestation. A sequence of VFRESP messages should only be returned when the verification was successful, i.e. the attestation could be verified. As was the case in the ATTRESP messages, the verified attestation may be of arbitrary length, hence, it is impractical to send it entirely at once. Consequently, the attestation is broken up into smaller chunks (at most 800 Bytes) and each chunk is packed as part of a VFRESP message.

Since the semantics of some of the fields in the VFRESP messages slightly differ than those in the ATTRESP messages, a description of the VFRESP's fields is presented in Table 7.

FIELD	OCTETS	TYPE	DESCRIPTION
Preamble / Prefix	23	Char Array	Generic message preamble, as presented in Section 4.2.1.2.2.1, with msg_type = 2.
Public Key	n	Unsigned Short Array	The Verifier's public key in binary format. n = bin_public_key_char_length * 2, since each Unsigned Short element is 2 Bytes in size.
Global Time	8	Unsigned Long Long	A Lamport Timestamp (Scalar Clock) associated with the message.
Attestation Blob Hash	20	Char Array	A hash of the entire attestation blob. This can be used to check the integrity and completeness of the reconstructed blob on the Requester's side.
Sequence Number	2	Unsigned Short	The message's sequence number in the sequence of VFRESP messages sent for the verification.
Attestation Blob Chunk	800	Char Array (Raw)	A chunk of the attestation blob.

Table 7: Description of the fields in an VFRESP message

4.2.1.3.2. Requester-Verifier Interaction - Verification Process

The following summary refers to the interaction between Requester and Verifier during the verification process. The timeline diagram in Figure 8 shows the timing relationships in such an interaction. Table 8 gives a brief reminder of the messages employed in the verification process.

The Requester forwards a VFREQ message to the Verifier, with the aim of requesting the verification of an Attribute's attestation, as indicated by the attestation hash in the VFREQ message.

After receiving the VFREQ, and successfully completing the verification process, the Verifier responds with a series of VFRESP messages. The message sequence may be of arbitrary length. At a high-level, the sequence of VFRESP message returns the recently verified attestation, hence, each message will contain a chunk of the attestation blob. Additionally, each VFRESP message also contains a hash of the entire attestation blob, a message sequence number, and additional message fields. If the verification is unsuccessful, no VFRESP messages are returned.

The Requester receives the sequence VFRESP messages from the Verifier, and puts them together. The Requester will know it has received the entire attestation blob, when the hash of the attestation blob present in the VFRESP messages matches a locally computed hash of the reconstructed attestation blob.

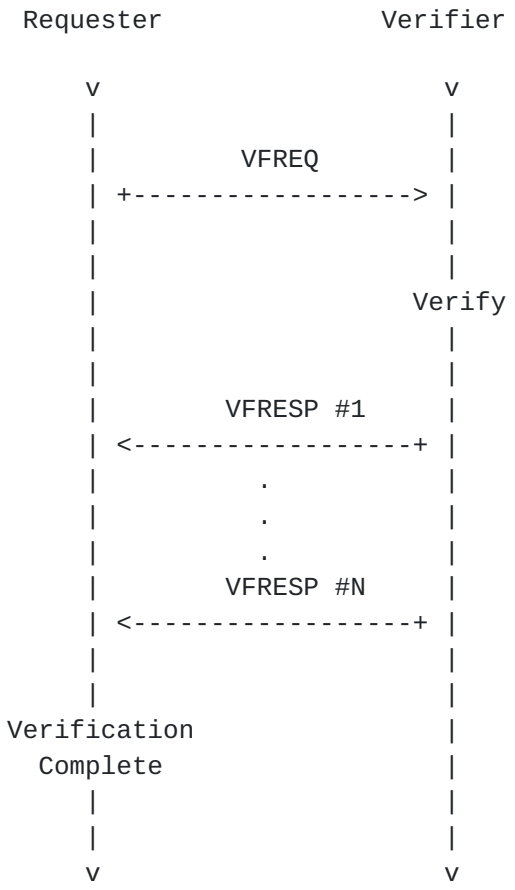


Figure 8: Timeline diagram of the messages exchanged between Requester and Verifier during the verification process.

MESSAGE	USE
VFREQ	Requester to Verifier requiring the verification of an attestation, as indicated in the VFREQ message.
VFRESP	Verifier to Requester provided that verification was successful. Each VFRESP carries a part of the recently verified attestation.

Table 8: Verification process message summary

4.3. Peer-to-peer cryptographically signed messaging

Most messages that are sent between peers are encrypted, the keys are retrieved from either the PKI, and verified based on the key attestations, or from the internal database after verification has already been done. Encryption is done using the ECDSA encryption scheme [johnson2001elliptic], which is an Elliptic Curve based cryptography system. This enables IPv8 to send messages effectively to a multitude of agents when (at the least) their identities are known, preferably also attested for. Having such a low bar to encrypt messages discourages the use of unencrypted communication channels, making interactions secure almost as early as the handshake, since the first message sent to any agent can be already encrypted with its respective key (which is retrieved from the database, other agents or the initial bootstrapping list).

4.4. NAT traversal

Network Address Translation (NAT) is omnipresent in the modern Internet, mostly due to networks being separated and the limited amount of global IP addresses available. Most consumer devices are behind a number of layers of NAT, but data center nodes can be behind NAT for security or virtualisation reasons. Containerised deployments are making things worse, as every peer based communication scheme must have a way to traverse NATs, otherwise operations will be affected. Even nodes meant to run with real IP addresses must implement NAT traversal techniques, as they may need to establish connections to peers behind NAT. Message puncturing based on UDP is key to this overlay. It conducts a random network walk to preserve connectivity under churn. Participants help each other to puncture the NAT infrastructure. Each participant will periodically introduce and connect some of its neighbors. When their random neighbors do not yet know each other, a new participant is discovered. Carefully timed concurrent UDP messages are used to traverse carrier-grade NAT infrastructures. Implementation,

deployment and measurements of smartphone users has shown that it is possible to build a healthy overlay without servers, even if nearly 100 percent of users are behind a NAT (Android-to-android overlay [TUDelft2018trustchain]).

5. Attack resistances

For blockchain implementations, attack resistance is an important requirement, especially with horizontal scalability. Therefore, Trustchain will have to cope with the same difficulties and attacks that other blockchain implementations have to, but due to the novel structure it introduces, these threats can be countered. With this novel structure, validation, uncertainty, and tamper proofing can be handled in a more intuitive manner while throughput does not need to suffer.

5.1. Sybil attacks

One of the most difficult attacks to repel, for a blockchain implementation, is the Sybil attack, where many agents are injected into the chain (and authenticity cannot easily be verified) to subvert a large portion of the system's voting power or trust (see section 3.3). Usually peer verification is used to protect against this kind of attacks (for instance: proof-of-work), usually resulting in slow systems. But when the influence of the attacker is large enough, even these methods will not be able to stop such an attack.

Trustchain deals with this problem by having an inherently different structure, where each peer has its own origin. On top of that, transaction injection can only be done with two valid signatures, meaning a Sybil attacker can only create trust with itself. This results in a network of disconnected agents that have no relation outside of their own cluster, which can easily be identified. Even when the Sybils acquire some degree of trust from outside of their cluster, by using accounting mechanisms, the profit from such an attack can only be weakly beneficial with bounded profit (using Netflow, not discussed in this paper) (Otte et al, 2017 [otte2017trustchain]).

5.2. Double spending attack

Using control over the blockchain to create a fork and creating two different transaction branches is called double spending. This kind of attack can be applied with relative ease to single chain implementations of the blockchain by injecting two conflicting transactions at the same time. Trustchain deals with this kind of attack by having the chain verified with each CHECO round, during which the hidden transaction can be easily found. By broadcasting

both blocks as a proof-of-fraud the malicious agent will have decreased trust and can be blacklisted or refused service.

5.3. Replay attack

Using the transaction signature of the counterpart, a malicious agent can try to replay a transaction on the blockchain, which results in increased trust or may be used to gain credits. CHECO and the novel structure make it trivial to find the conflicting blocks when verifying the counterparty's chain. The two blocks with the same outgoing pointer together make the proof-of-fraud, which then can then be used to decrease the trust in the malicious party and can be blacklisted or refused service.

5.4. Whitewashing attack

Abusing the permissionless structure of Trustchain to create additional identities at any given point can negate the effect of having trust, so this kind of attack differs from a Sybil attack. When an agent suffers from reputation loss, it can simply discard its current identity and take on a new one. Since refusing service to agents with little trust will affect usability and willingness to join the network, an adequate solution can be prioritising strategies. One method for implementing this is described in the paper discussing Netflow (not discussed in this paper) (Otte et al, 2017 [otte2017trustchain]).

5.5. Spam attack

Since the TxBlocks have dynamic size, spam (in the sense of useless data in the transaction field) can be used to clog an agent's network or database with excessively large messages, slowing down its operations or bringing it to a complete halt due to memory/network being full. This kind of attack can be coped with by having a throttle per connection to keep some bandwidth available and a limit on the size of the message. If large messages are to be expected, a file based buffer will enable large message transfer without exceeding the memory capacity.

Another type of spam can be identified as a collection of useless messages, clogging the network and database with large amounts of empty messages, which is possible since transactions do not require an agent to pay transaction costs (as BitCoin does). Though this is easily countered by not accepting those messages, leading to the malicious agent having a lot of orphan blocks.

5.6. DDoS

When massive quantities of useless or empty messages are sent over a network, it might get congested, leading to dropped operations, network congestion, unreachability of agents, or, in the most extreme cases, even to system failure due to overload. Due to the distributed, peer based communications architecture, this is not feasible without flooding the network of the malicious agent itself, as it has to send messages to each target individually.

6. Acknowledgements

We very much thank the European Union for providing us the required funding for this work. Through EU FP6 and FP7 funding instruments we have been developing and deploying our own distributed ledger fabric since August 2007. An estimated 3.4 million Euro has been granted through these specific projects and leading directly to this work ([P2P-Fusion], [P2P-Next],[QLectives]).

We thank master student Stijn for his help with writing of this draft.

7. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

8. Security Considerations

From a security perspective, the usage of novel structures such as Trustchain might lead to new kinds of attacks. We consider this risk of less importance for a private and consortium network, where all participants are known to the operator and authentication mechanisms are used to restrict access to the network.

For the public blockchain networks, the usage of Trustchain might lead to new kinds of attacks. For instance, an attacker might be able to pollute the chain with refusal to sign attacks to decrease trust. The scope of such attacks and security violations needs to be investigated and is not part of this draft.

9. References

[abrahamself]
Abraham, A., "Self-Sovereign Identity", 2017, <https://www.egiz.gv.at/files/download/Self-Sovereign-Identity-Whitepaper.pdf>.

[apostle2017uber]
Apostle, J., "The Uber data breach has implications for us all.", 2014, <https://www.ft.com/content/e2bf6caa-d2cb-11e7-a303-9060cb1e5f44>.

[atzori2016blockchain]
Atzori, M., "Blockchain-based architectures for the internet of things: a survey", 2016, <https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2846810>.

[azouvi2017secure]
Azouvi, S., Al-Bassam, M., and S. Meiklejohn, "Who am i? Secure identity registration on distributed ledgers", 2017, <https://link.springer.com/chapter/10.1007/978-3-319-67816-0_21>.

[boneh2004secure]
Boneh, D. and X. Boyen, "Secure identity based encryption without random oracles", 2004, <http://link.springer.com/content/pdf/10.1007/b99099.pdf#page=454>.

[brown2017introducing]
Brown, R., "Introducing R3 Corda: A Distributed Ledger for Financial Services", 2016, <http://www.r3cev.com/blog/2016/4/4/introducing-r3-corda-a-distributed-ledger-designed-for-financial-services>.

[cong2017blockchain]
Cong, K., Ren, Z., and J. pouwelse, "A Blockchain Consensus Protocol With Horizontal Scalability", 2017, <https://repository.tudelft.nl/islandora/object/uuid:86b2d4d8-642e-4d0f-8fc7-d7a2e331e0e9>.

[FIPS180-4]
Gallagher, P., "Secure hash standard (shs)", 2008, <http://www.cs.haifa.ac.il/~orrd/IntroToCrypto/online/fips180-3_final.pdf>.

[jethanandani2017accounting]
Jethanandani, M., "Accounting in NETCONF and RESTCONF", 2017, <https://www.ietf.org/id/draft-mahesh-netconf-accounting-03.txt>.

[johnson2001elliptic]
Johnson, D., Menezes, A., and S. Vanstone, "The elliptic curve digital signature algorithm (ECDSA)", 2001, <http://www.springerlink.com/index/0L7A1W9W38XL6W6R.pdf>.

[kokkola2011payment]
Kokkola, T., "The payment system: Payments, securities and derivatives, and the role of the Eurosystem.", 2011, <https://www.ecb.europa.eu/pub/pdf/other/paymentsystem201009en.pdf>.

[mcmillan2014inside]
McMillan, R., "The inside story of Mt. Gox, Bitcoin's $460 million disaster.", 2014, <https://www.wired.com/2014/03/bitcoin-exchange/>.

[miller2016honey]
Miller, A., Xia, Y., Croman, K., Shi, E., and D. Song, "The honey badger of BFT protocols", 2016, <http://dl.acm.org/citation.cfm?id=2978399>.

[nakamoto2008bitcoin]
Nakamoto, S., "Bitcoin: A peer-to-peer electronic cash system", 2008, <http://www.academia.edu/download/32413652/BitCoin_P2P_electronic_cash_system.pdf>.

[otte2017trustchain]
Otte, P., de Vos, M., and J. Pouwelse, "Trustchain: A Sybil-resistant scalable blockchain", 2017, <https://www.sciencedirect.com/science/article/pii/S0167739X17318988>.

[P2P-Fusion]
"P2P-Fusion project", 2018, <http://cordis.europa.eu/project/rcn/105290_en.html>.

[P2P-Next]
"Next Generation Peer-to-Peer Content Delivery Platform", 2018, <http://cordis.europa.eu/project/rcn/85326_en.html>.

[pouwelse2017trustlaws]
Pouwelse, J. and M. de Vos, "Laws for creating trust in the blockchain age", 2017, <https://github.com/blockchain-lab/shared_vision_towards_programmable_economy>.

[QLectives]
"QLectives project", 2018, <http://cordis.europa.eu/project/rcn/89031_en.html>.

[resnick2002trust]
Resnick, P. and R. Zeckhauser, "Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system", 2002, <http://www.emeraldinsight.com/doi/pdf/10.1016/S0278-0984(02)11030-3>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[TUDelft2018trustchain]
Pouwelse, J., "Trustchain - creating trust with software", 2018, <https://play.google.com/store/apps/details?id=nl.tudelft.cs4160.trustchain_android>.

[vukolic2015quest]
Vukolic, M., "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication.", 2011, <http://link.springer.com/chapter/10.1007/978-3-319-39028-4_9>.

[yan2008trust]
Holtmanns, S. and Z. Yan, "Trust modeling and management: from social trust to digital trust.", 2008, <http://lib.tkk.fi/Diss/2007/isbn9789512291205/article1.pdf>.

Author's Address

Dr. J.A. Pouwelse (editor)
Delft University of Technology
Delft
Netherlands

Phone: +31 15 2782539
Email: j.a.pouwelse@tudelft.nl