A blockchain-based distributed Trust System for the Internet of Things

Roberto Di Pietro
HBKU-CSE

Xavier Salleras
UPF

Matteo Signorini
Nokia Bell Labs

Erez Waisbard
Nokia Bell Labs

Abstract

One of the biggest challenges for the Internet of Things (IoT) is to bridge the currently fragmented trust domains. The traditional way to solve this problem, as in the well-known PKI model, has always been via a common root of trust. However, such an approach does not fit well with the extremely heterogeneous IoT ecosystem, composed by billions of devices, differing in available resources and belonging to independent administrative domains. In this work we describe a distributed trust model for the IoT and define a new cryptographic primitive, denoted as *obligation chain*. Our model leverages the existing trust domains and bridges them to create an end-to-end trust between IoT devices without relying on any common root of trust. Our obligation chain is designed as a credit-based Blockchain with a built-in reputation mechanism. Its innovative design enables a wide range of use cases and business models that are simply not possible with current Blockchain-based solutions. Furthermore, it does not experience lengthy delays for committing transactions—unlike today's Bitcoin based system. We provide a security analysis for both the obligation chain and the overall architecture. Finally, experimental tests that show the viability and quality of our solution.

1 Introduction

The Internet of Things (IoT) real value will be unleashed when billions of devices will be connected to the Internet, and able to interact with each other. However, while it is true that more and more devices are becoming connected, the grand vision of IoT is still far from being achieved since these devices do not communicate with each other; while there are several reasons for this, a major one is the lack of trust between devices, that is essential for establishing secure communication. Indeed, the trust model that works well for the Internet does not fit the scale and diversity of the IoT, where there is no common root of trust that is accepted by all parties. Instead, we see different domains in which manufacturers create a root of trust that allows devices within each single domain to communicate securely. We refer to these domains as *Islands of Trust*, where in each domain the trust is provided and regulated by an entity independent from the entities administering the trust for other domains. Industry consortia such as the Open Connectivity Foundation (OCF) [9] attempt to solve this problem by agreeing on a common root of trust. However, even these consortia do not cover the entire IoT landscape. As a result, the current IoT landscape is made of individual manufacturers and platforms that can communicate securely only if they agree on a common root of trust (e.g. through a consortium) or, if they establish direct mutual trust through bilateral agreement.

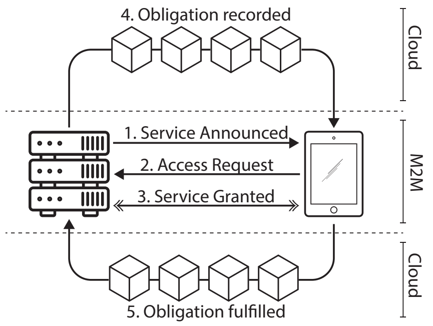


Figure 1: Our solution in brief.

In this work we are taking a new approach for bridging trust between the different domains (i.e. the islands). Instead of relying on a central or shared root of trust, we use Blockchain technology [6] to create a distributed trust mechanism. We start by introducing a new tool, named *Obligation Chain*, which is a new platform for a distributed credit-like system (this in contrast to the Bitcoin [12], which is a cash-like system). Furthermore, this credit system has a built-in reputation mechanism

[14] that allows peers to decide whether or not to accept an obligation based on the credit history of a consumer (someone that has a history of not fulfilling his obligations is unlikely to have his newly mint obligations accepted). Figure 1 shows an overview of our main scheme in which a service provide (left side) interacts with a service consumer (right side) allowing him to access some services in real time (machine to machine communications only) while postponing the customer obligation fulfillment.

The best way to communicate the benefits of our construction is probably through the following use case: a service provider offers a service, together with its terms of use. For example, let us consider a small coffee-shop that wishes to offer WiFi services for customers (for an additional fee). Currently, major operators and major coffee chains have bilateral agreements that allow this, but there is no solution that fits any small/family run coffee-shops. We would like to have a solution that allows anyone to consume these services simply by providing a public obligation for fulfilling the terms of use as specified by the service provider. Following good practices, the service provider is not expected to automatically accept an obligation from anyone, but it is expected to first assess the risk of accepting that obligation. Naturally, a large part of potential consumers might not have enough reputation to access the service. This is indeed where many reputation systems fail in practice [10] and this is why our system goes a step further to leverage existing trust. In the above example, we have leveraged the trust these users already have with their mobile operator and the fact that the mobile operator already has a well established reputation. Furthermore there is already a full fledged public key infrastructures (PKIs) in place mediating between the mobile operator and its customers. By bridging trust between the coffee shop (i.e. the service provider) and the mobile operator, we are able to provide a complete path of trust between any customer of the mobile operator and the service provider. The coffee shop just needs to publish the terms of use for accessing the WiFi service. Any large mobile operator that wishes to grant free access to the WiFi services to its customers can sign an obligation to fulfill the terms of use. The users would get the signed obligation based on their existing trust relationship with their operators. The users would then present the signed obligations to the coffee shop that would need to decide whether or not to grant access based on the public credibility of the signing operator. Our scheme is flexible and allows every service provider to conduct its own assessment based on the credit history of the obligation's signer, the service value and any other element it could deem relevant. Thus, one service provider may decide to accept an obligation from a certain consumer, while another one will not. As the entire history of obligation fulfillment is available on the immutable blockchain[1], the service provider can make its decision of whether or not to accept the obligation. If the obligation is accepted and later fulfilled then the service provider would report it on the blockchain. By doing so, it would add credibility to the reputation of the operator.

The solution described in this paper enables the above flow, providing the following distinguished features:

Complex business models . The service provider has the flexibility to offer complex business models that may depend on the actual consumption of the service that is not known in advance.

Minimal requirements from the end devices . The end devices are typically low power devices that need to establish trust quickly and efficiently. In our system the main role of the devices is to convey the obligation from one side to the other one. Most of the work is done by the back-end servers. Furthermore, as obligations are commitments to pay later, the system does not suffer from the long waiting time that is associated with using the Bitcoin system to directly pay for the service;

Secure reputation system . Our system leverages the security of the blockchain for managing the reputation of the different parties, and does not rely on any particular server —hence enhancing reliability and security.

The rest of this paper is organized as follows. In Section 2 we summarize the building blocks leveraged in this paper. In Section 3 we introduce the core elements and main concepts defined in this solution, paving the way to Section 4, where we describe our solution. In Section 5 we briefly describe the prototype we developed, whereas in Section 6 we provide an analysis of the security of our solution and compare its performances against the well-known Bitcoin system. Section 7 lists the related work, while in Section 8 we draw our conclusions.

2 Background

In this section we describe the building blocks of our solution. In detail, we first describe the concepts of trust and reputation and then we introduce the blockchain technology. Indeed, the proposed solution leverages security and reliability properties from the blockchain technology and applies them to build a trusted reputation sys-

[1]We note that although the transactions are public they do not need to contain personal information. Furthermore, it is always possible to encrypt the obligations and to provide decryption keys only to the desired third parties.

tem which can be used to define fine grained access models.

2.1 Trust and Reputation

There are many different definitions of trust and reputation [5]. Trust is generally perceived as a belief that an entity is honest and will not harm other entities. This belief is subjective and based on past experiences. On the other hand, reputation is a global perception of an entity's behavior based on the trust that other entities have established. Reputation systems collect, aggregate, and distribute feedback about entities' past behaviors [14]. As such, the reputation is a more objective belief than trust. The goal of a Trust and Reputation System (TRS) is then to guarantee that actions taken by entities in a system reflect their reputation values and cannot be manipulated by unauthorized entities. In the past, reputation systems have been validated in protecting consumers from financial frauds. Indeed, it has been shown that such systems can significantly reduce transitional losses and improve consumers' confidence while also driving sales growth for sellers [1, 4, 13].

TRSs can be generalized as composed by entities, observers, disseminators and reputation servers [10]. Entities are usually considered as end devices that do not process the reputation directly, but only use it. Observers are referred to TRS's elements which create and manage the trust leveraged by entities. Observers can generate trust information based on the historical information on the entities and can even cooperate among them. Data exchanged between observers, or sent to other elements (such as entities or reputation servers), rely on secure and trusted communications which are carried by the disseminators. However, despite their effectiveness, TRSs have been also threatened by different attacks [10]. In this paper, we focus on how to design a blockchain based TRS which allows to bridge trust between secure domains. In the proposed approach the entities will be the service consumer devices, the observer and disseminators will be the blockchain peers, and the reputation servers will be represented by the service provider back-ends.

2.2 Blockchain Technology

In the last few years, a new technology named *Blockchain* [6, 20], based on the distributed paradigm, has obtained a great success. This technology can be roughly seen as a digital ledger that sits at the core of decentralized ecosystems and keeps track of any changes by holding a new record for each transaction. Blockchains first emerged with Bitcoin [12], a distributed cryptographic currency payment protocol designed as *a purely peer-to-peer version of electronic cash* and capable to provide direct payments from the payer to the payee only.

In a more abstract way, the blockchain can be seen as an ordered and back-linked list of blocks carrying transactions which encode exchanged information between two or more participants. Each block consists of a collection of transactions and is linked to the previous block in the blockchain, thus creating a chronological order of blocks that all together build a chronological order of transactions. Peers work to build and maintain the blockchain via different consensus algorithm, thus realizing a state machine replication system which is the essence of any blockchain technology.

3 Settings and Definitions

In this section we introduce all the settings and definitions which we will later use in Section 4. For clarity sake, all the abbreviations being used in this paper have been summarized in Table 1.

The main goal of our solution is to build an access control system that leverages consumers' reputation and that is used by service providers. To this end, our basic setup is made by a *service provider* (SP) and a *service consumer* (SC)—the scenario can be enriched in complexity, but for the sake of exposition we will stay stick to the simple model just introduced. Each of the two entities has access to powerful *back-ends* and more constrained *end-devices*. All internal communications inside the SP and the SC are secured (i.e. encrypted and authenticated) leveraging the existing trust within the same Island of Trust (i.e. same domain) while external communications between the SP and the SC are secured using our solution (see Section 4 for more details).

Within SPs and SCs, we have then defined the following entities and roles:

Service Provider back-ends (SPBs) responsible for:

- Act as certification authorities (CAs) within their own Islands of Trust (i.e. their domain);
- Create and share their terms of use;
- Verify the credibility of obligation issuers;
- Post accepted obligations in the obligation chains;
- Control access to their own services;

Service Provider end-device (SPDs) responsible for:

- Interact with service consumers' devices;
- Pass service requests and obligations to the service provider back-end, thus acting as a gateway;
- Grant access to their services based on the back-end decisions;

Service Consumer back-end (SCBs) responsible for:

- Download service provider' terms of use from the Cloud and select the one that they want their devices to access;
- Issue obligations based on the desired terms of use;
- Fulfill valid obligations;

Service Consumer end-device (SCDs) responsible for:

- Interact with service provider devices;
- Sign stored obligations to access services;
- Confirm to their back-ends whether the service has been accessed or not;

Symbol	Description
SP	service provider
SPD	service provider end-device
SPB	service provider back-end server
SC	service consumer
SCD	service consumer end-device
SCB	service consumer back-end server
TX_a	set of Bitcoin transactions from SC A
tx_a	a Bitcoin transaction from SC A
ST_a	set of SC A fulfilling transactions
st_a	a fulfilling transaction from SC A
OT_a	the set of obligations created by SC A
$oblg_a$	obligation created by SC A

Table 1: Table of symbols

SC and SP has a self-generated public key, used to sign messages and as an ID, recognized by others. The reputation is associated with this ID. We stress that in our solution there is no certification authority (CA) that manages these IDs. Instead, they are self-generated within each Island of Trust and may be replaced at any time. However, as explained in the rest of this paper, both SCs and SPs are usually keen to keep the same IDs as needed to claim their reputation.

Now that all our entities have been defined, we can define two key concepts that are used by SPs and SCs to establish trusted interactions:

Terms of use (TERMS): generally defined as terms of service, the terms of use are here intended as rules the SCs must agree to abide to, in order to access services provided by SPs. TERMS are created by service providers and signed by both service providers and consumers—they do represent an agreement between them;

Obligations: generally defined as *"something by which a person/device is bound or obliged to do certain things, and which arises out of a sense of duty or results from custom, law, etc."*, our SCs leverage obligations to publicly state that all the conditions listed in one or more TERMS will be fulfilled as *per* what has been specified by SPs.

The publicly signed obligation requires the SCs to fulfill the TERMS. As usual in life, building a good reputation (which is needed to have the obligations accepted by SPs) is much harder than to ruin it. This mitigates the attack in which an SC builds a good reputation over time to sign later on some obligations it does not want to fulfill—the cost of building such a good reputation being bigger than the attack gain.

4 Our Solution

From this point on we show how TERMS and obligations can be used by SPs and SCs to set up trusted interactions between untrusted devices. To this end, we have designed a new blockchain named *obligation chain* that is linked to to another blockchain and used to build a tamper-proof reputation system. In this section we first introduce this new chain and then describe how we have leveraged it to bridge the different islands of trust (i.e. different domains).

4.1 Obligation Chains

Being based on the blockchain technology, our obligation chain is eventually agreed by the whole network. It can be seen as a distributed ledger storing obligations as of commitments signed by SCs to access SPs' services without immediately paying for them. For the sake of simplicity, we can imagine our obligation chain as an append only log database where obligations and TERMS are kept. However, unlike other solutions, our chain does not contain digital assets which have to be recognized and verified by other peers at run-time (as for bitcoins in the Bitcoin blockchain). Indeed, our obligation chain contains obligations that are *locally* accepted by SPs and then shared to the rest of the network via the same SP.

Obligations are generated by SCBs and initially contain only the TERMS and their signature. They are then downloaded to SCDs which will later exchange them, in the form of an obligation transaction, to get access to services provided by SPs. Obligation transactions contain, among the others, the following values:

- **Obligation ID**: a unique identification value;
- **TERMS**: the terms of use which have been previously published by SPBs and agreed by SCBs;

- **SPBs and SPDs Signatures**: required to provide non repudiation to the service consumer. Once signed, both SPs back-ends and end-devices commit on providing the service as described within the TERMS;

- **SCBs and SCDs Signatures**: required to provide non repudiation to the service provider. SCs signatures represent a *proof of commitment* which is directly given to the SPs. This commitment is then shared on a global scale (via the obligation blockchain), thus requiring SCs to pay to fulfill their obligations. If they do not, their reputation will be compromised.

Obligation transactions are leveraged to keep track (on the chain) of the agreements that are established (off the chain) between SCs and SPs. As such, as both SCs and SPs built their reputation on the chain, they both need tools to prevent fraudulent interactions. On the one hand, the proof of commitment described above is the tool used by SPs against malicious SCs. On the other hand, the *proof of fulfillment* is the tool we have designed for honest SCs which have to protect themselves against malicious SPs.

The proof of fulfillment has been realized by linking our obligation chains with standard blockchains. For the sake of simplicity, we will consider a toy example in which services provided by SPs need some kind of payment in order to be accessed. To this end, throughout the rest of the paper, we will consider the linked blockchain to be the Bitcoin one. However, it is important to remark that the second blockchain, the one we use to link our obligation chain, can be any type of blockchain.

4.1.1 Obligation Chain usage

We will now introduce all the operations that will be used by SPs to decide whether or not to take the risk of giving access to their services (see Section 4.2). For the sake of simplicity, we have summarized all symbols used in Table 1.

- **reputation_bootstrap**(C): given a service consumer C, the SP builds a local obligation database containing all the $oblg_c$. The SP then scans the Bitcoin blockchain looking for all the st_c that links to the obligations in the local database thus deleting them from the database. The remaining $oblg_c$ within the database are the non fulfilled obligations and are compared against the fulfilled one to evaluate a local reputation score for C. It is important to highlight that this function is executed only once and that the reputation score assigned to C is local to the SP and can change if the same function is executed by other SPs;

- **reputation_build**(C,b): given a service consumer C and a block index b, the SP updates C's reputation score executing the same process of *reputation_bootstrap*(C) but this time from the b^{th} block in the obligation blockchain;

- **reputation_add**(C,$oblg_c$): for a given service consumer C and a new obligation $oblg_c$ signed by C, if SP receives such obligation, he updates the local C's database and the C's local reputation score;

- **reputation_fulfill**(C,st_c): for a given service consumer C and a new Bitcoin transaction st_c signed by C, if such a transaction is used to pay an obligation, the SP that has previously received that obligation from C removes it from the local database and updates C's local reputation score;

- **reputation_update**(C,b): for a given service consumer C and a new obligation chain block b, a SP executes the function *reputation_add*(C,$oblg_c$) for all the obligation transactions in that have been signed by C;

As detailed in Section 4.2, the above operations can be executed by SPs in two different ways:

- **Asynchronous**: in this approach, SPs cannot stay always updated with both the obligation and the Bitcoin blockchains. As such, they connect to them and download new blocks only when they need to update service consumers' reputation scores. As an example, **rep_bootstrap**(C) needs to be executed for a given service consumer C on the first time it approaches a service provider while **rep_build**(D,b) needs to be executed for a given service consumer D that is known by SP but for which SP does not have an updated information;

- **Synchronous or Cached**: in this approach, SPs always receive the latest obligation and Bitcoin blocks in the chains. As such, there is no need to build the local reputation scores for their service consumers from scratch. Hence, given a service consumer C only **rep_update**(C,b) is executed every time a new block b is received.

Our measurements have been conducted in an *asynchronous* environment as it represents the worst case scenario for the service consumer. In fact, in such a scenario, assuming that both the obligation and Bitcoin blockchains are large in size, a service consumer C will have to wait for the service provider while it computes C's local reputation score. This score is indeed computed on the number of obligation transactions which have been fulfilled with Bitcoin transactions and thus, it requires the service provider to browse the whole obligation and Bitcoin chains (see Section 6 for the experimental results).

In the next section we will describe how the aforementioned operations are used by SPs to control accesses to their services only based on SCs' reputation.

4.2 Islands of Trust

Now that we have described the obligation chain and how it is leveraged to store SCs' obligations, we can introduce the concept of *bridging the trust* between different islands of trust, i.e. different secure domains. In our solution, we assume the following:

- Each Island of Trust has a full local PKI and CA. Such a PKI is not recognized or trusted by other islands and it is only used for the internal device management. However, PKI information (such as the public keys) are used as publicly available IDs. Reputations are built on top of such IDs;

- Each device has a certificate that was issued by its local CA and that is used to secure the communications between end-devices and back-ends within the same island. Each device belonging to either the SPB or the SCB can be uniquely identified by them thanks to unique IDs.

The protocol that we have designed for the trust bridging is a three-way handshaking protocol composed by a *setup*, a *spend*, and a *fulfilling* phases. During the *setup*, SCBs generate (and sign) obligations based on the TERMS published by SPBs. These obligations are passed to SCDs to be used when interacting with SPDs. In the *spend* phase, end-devices coming from both SPs and SCs interact with each other and exchange their own version of TERMS. If they match, the obligations created in the *setup* phase are passed to the SPB. If the SPB decides to accept the obligations based on the local SC's reputation score, the SC receives the access to the service and its obligations are sent in broadcast to the rest of the network to be included in the obligation blockchain. In the final step, SCs make a connection between the Bitcoin and the obligation chain by paying for them. This final step causes all the local reputation scores within all the SPs to be updated which also increases the likelihood of SC's new obligations to be accepted in the future.

Figure 2 depicts our three-way handshaking protocol and highlights each individual operation executed by both SC and SP entities (both end-devices and back-ends). The complete flow is as follows (see Figure 3 for the sequence diagram):

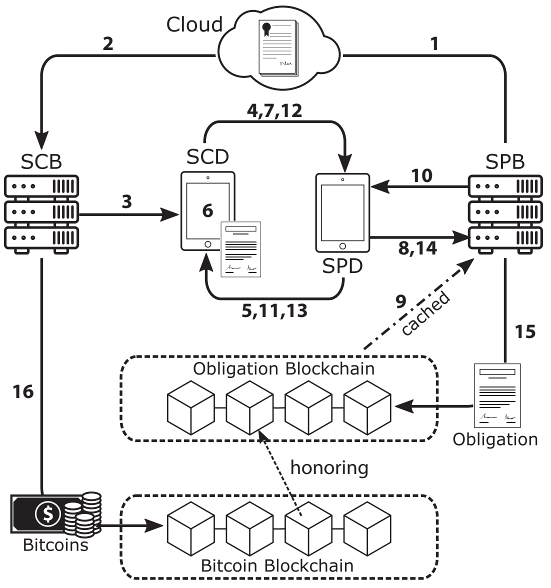


Figure 2: Protocol Overview.

1. The SPB publishes the TERMS[2] with all the required details;

2. The SCB downloads SP TERMS and creates obligations. The obligations contain all the details written within the TERMS and are signed by the SCB;

3. The above obligations are downloaded within the SCDs that intend to use them;

4. The SCD sends to the SPD a service request;

5. The SPD sends the TERMS to the SCD;

6. The SCD compares the TERMS to the ones in its obligation to ensure they match;

7. If the TERMS match the SCD passes the obligation to the SPD, otherwise it ABORT;

8. The SPD sends the obligation to its SPB;

9. The SPB looks at the obligation chain and payment chain to assess the credibility of the obligation issuer. Depending on the approach being used (which can be either *synchronous* or *asynchronous* as described above) *rep_bootstrap()*, *rep_build()* and *rep_add()* are executed accordingly. The result is an updated knowledge of SC's trustworthiness. This result is local to the SP that is computing it and is based

[2]We do not specify exactly how these TERMS are published. We envision them to be posted on the service provider's web site, but any other solution would do. We also note that the TERM may change over time and that the SCB is the solely responsible for keeping track of changes.

on an arbitrary trust score evaluation which is beyond the scope of this work. The SPB also verifies the signature w.r.t. the public key of the issuer;

10. If the signature is valid, the TERMS match, and the SPB decides to trust the SC based on his obligations history and thus his reputation, then the SPB sends an OK to the SPD. Otherwise, it sends ABORT. Along with the OK, the SPB also generates and sends back to the SPD a new payment address. As we are considering the toy example of a service provisioning that require Bitcoin payments, this new address will be a new Bitcoin addresses. This address needs to be sent back to the SCD and its back-end as it will be later used to fulfill the obligation;

11. The SPD either conveys the reply to the SCD, if it has received an OK from its SPB, or it ABORT the protocol;

12. The SCD signs the obligation;

13. The SCD passes the signed obligation to the SPD;

14. The SPD verifies the SCD signature. If the signature is valid, then it gives access to the service otherwise ABORT;

15. The SPD signs the obligation just received from the SCD;

16. The SPD sends its signature back to the SCD (to serve as a receipt of the transaction);

17. The SPD conveys the signed obligation to the SPB;

18. The SPB broadcast the obligation to the other peers in the obligation chain network in order for the obligation to be added in the obligation chain;

19. The SCB periodically monitors the obligation chain looking for its obligations which are then fulfilled by issuing payment transactions to the exact Bitcoin addresses that have been created by the SPB at step 9. This allows the SP to monitor those addresses and to detect new bitcoin incomes. The result is for the SP to execute *rep_fulfill()* and to remove pending obligation from its internal database, thus also updating the SC's reputation.

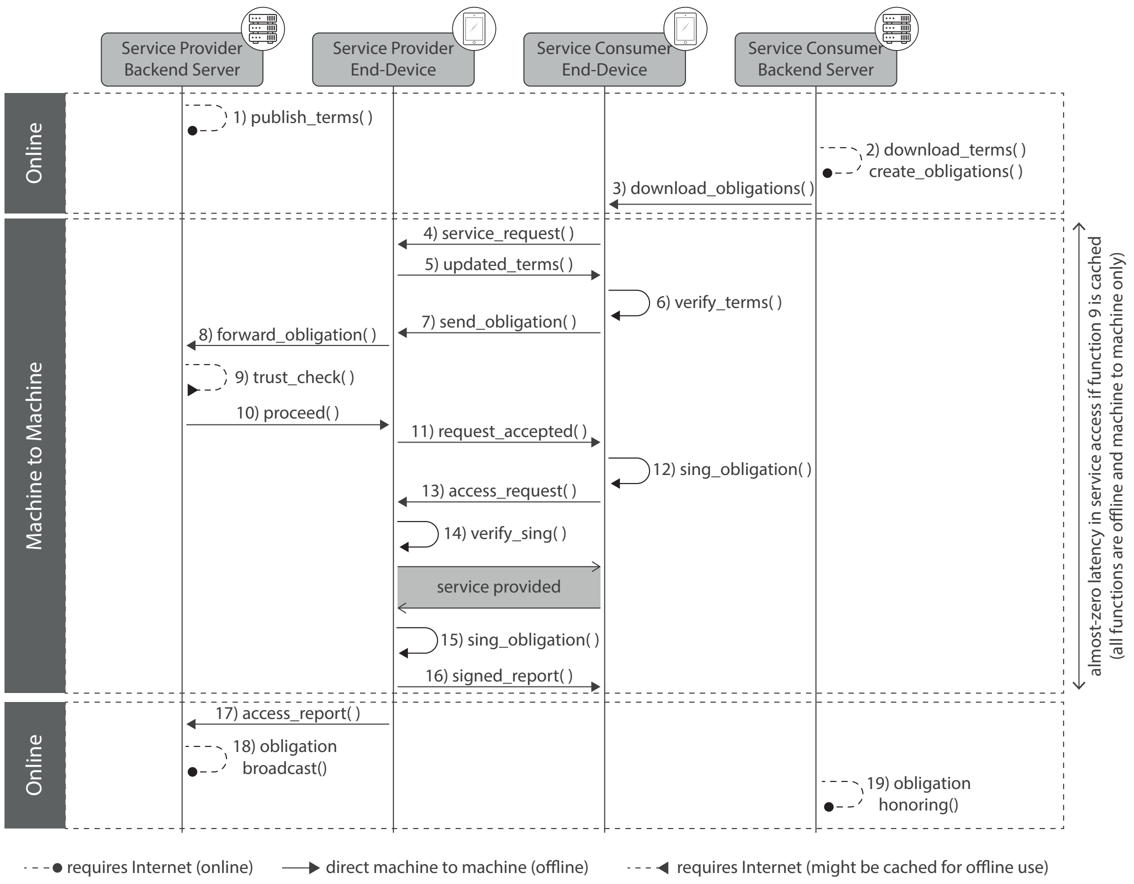
It is again important to highlight that, whilst all the aforementioned steps are executed at run-time, the step number 8 can also be executed off-line in the *asynchronous* mode (see Section 4.1 for more details). In this mode, SPs can update SCs' reputations by executing *rep_update()* each time they receive a new block head for the obligation chain. This makes the whole process faster as SPs are already aware of SCs' reputations even before they receive new SCs' requests. During our experimental tests we have used the asynchronous mode as it represents the worst case scenario—the SP has to rebuild the consumer's reputation every time. However, even in the worst case scenario our solutions proved to be faster than standard blockchain based payment systems such as Bitcoin.

One of the main advantages of this solution is that service providers and consumers do not have to explicitly establish a contractual agreement. SPs publish the possible business models as part of their TERMS while SCs pick the ones that best suit their needs. Then the decision to accept the obligations is left to the SPs.

Another key advantage of the proposed solution is the establishment of self-enforced and tamper-resistant reputations. Indeed, as we are assuming that SCs and SPs do not know each other in advance, they can judge each other trustworthiness only based on the reputations stored within our obligation chain. In this work the reputation is meant as the average of obligations fulfilled on time. This information is accessible to anyone having access to both the obligation blockchain and the Bitcoin blockchain as it is only required to check how many obligations have been fulfilled by using the new Bitcoin addresses created in the step 9 of our protocol. Hence, peers with a high frequency of interactions with others will create many obligations and will need to fulfill them as established with the SP. If they do so, they will have a good reputation, otherwise not.

4.3 Privacy

So far we argued about the benefits of having all transactions publicly recorded as part of a distributed reputation platform. However, this also raises some privacy concerns. In traditional payment systems transactions and commitments are only known to the parties that are directly involved (noting that the credit company is included in this definition). So the question is, can we have all the advantages described above, while preserving the privacy we are used to in traditional systems? The answer to this question is yes and in this section we sketch the method to do so. We use encryption to provide confidentiality of the obligations between the parties. We encrypt the content of the TERMS, thus hiding all information from anyone who does not have the decryption keys. Naturally, the service provider and the service consumer would both have these keys and would be able to see all the TERMS and signatures. However, keeping the key solely between the service provider and service consumer would not enable other service providers to assess the reputation of a consumer. Thus, when a consumer approaches a new service provider, he would need to disclose the decryption keys. We argue that this is also the



Service Provider Backend Server
Service Provider End-Device
Service Consumer End-Device
Service Consumer Backend Server
Online
1) publish_terms( )
2) download_terms( ) create_obligations( )
3) download_obligations( )
Machine to Machine
4) service_request( )
5) updated_terms( )
6) verify_terms( )
7) send_obligation( )
8) forward_obligation( )
9) trust_check( )
10) proceed( )
11) request_accepted( )
12) sing_obligation( )
13) access_request( )
14) verify_sing( )
service provided
15) sing_obligation( )
16) signed_report( )
Online
17) access_report( )
18) obligation broadcast()
19) obligation honoring()
almost-zero latency in service access if function 9 is cached (all functions are offline and machine to machine only)
requires Internet (online)
direct machine to machine (offline)
requires Internet (might be cached for offline use)

Figure 3: Protocol sequence diagram.

case today when users need to present their bank records and credit history when asking for a loan from someone who do not know them. This enables the new service provider to see the good credit history of the consumer. However, it is expected that consumers would attempt to hide bad credit history, but we solve this as follows. When an obligation is not fulfilled the service provider publishes the obligation that was not fulfilled in an unencrypted form, includes the original decryption key in it, and links it to the previous encrypted one. Now anyone can see the two obligations and see that one is indeed in an encrypted form of the other and all non fulfilled obligations are now public.

5 Implementation

The implementation of our solution has been realized by leveraging two blockchain technologies: *Multichain*[3] and Bitcoin Core[4]. The former is an open source platform that implements the concept of arbitrary digital assets on top of the blockchain. The latter is home to the most famous and widely adopted cryptocurrency used nowadays. More in detail, we used our own *regtest* bitcoin network for the obligation payments in order to have the transactions locally accepted without the need to pay any real fees.

As regards MultiChain, the technology being used in this paper to implement the concept of obligations, it is an off-the-shelf platform for the creation of private blockchains that proposed a different approach to Bitcoin via integrated management of user permissions that: i) ensures that the blockchain's activity is only visible to chosen participants, ii) introduces controls over which transactions are permitted, and iii) enables mining to take place securely without proof of work and its associated costs. It is however important to highlight that our solution does not depend on any specific technology and can provide the same performance with any other blockchain technology, even those based on proof-of-work (PoW). In fact, as our obligations are accepted locally by the SPs, the block validation process does not affect our performance. As such, the same results and properties shown in this paper might be achieved with Ethereum[5], IBM Hyperledger Fabric[6], or any other blockchain technology. The same applies for the blockchain being used for payments (i.e. Bitcoin) that could be replaced by Litecoin[7], RippleCoin[8] etc..

As per the implementation of the obligation chain and the management of obligations on top of it, we have used a particular feature implemented by the multichain technology, the streams. A stream provides an abstraction from the raw blockchain implementation, thus allowing to focus on general data retrieval, timestamping, and archiving. Indeed, it can be used to implement three different types of databases such as: a) NoSQL key-value databases, b) time series databases, and c) identity-driven databases. In the proof of concept developed and deployed for this work, we have implemented our obligation chain to use streams in the form of key-value databases where service providers' IDs have been used as keys while service consumers' obligations have been used as values. The result obtained with this approach is twofold. First, SCs are able to easily organize and retrieve all the obligations created over time, both fulfilled and not fulfilled, while avoiding the creation of additional databases—thus reducing the storage requirements. Second, SPs are able to easily organize the obligations in fulfilled (i.e. paid by the SC) and pending (i.e. not paid). The result is a better and easier update of SCs' reputations scores.

To deploy our proof of concept environment and to allow SCs and SPs peers in our network to build their own reputation on top of the fulfilled and not fulfilled obligations, we have used two multichain docker images labeled as *master node*[9] and *slave-node*[10] both built on top of a basic Ubuntu distribution. The master-node runs the Multichain daemon and boostraps the whole blockchain. This node is the one responsible to give access to other peers to the chain in a permissioned environment. However, as we do not need any identity management, we used the master node only to build and initially configure the chain—but we kept the latter open. Neither the SP and the SC back-ends, nor their end-devices, run within this node. The slave-node implements both end-devices and back-end servers belonging to SCs and SPs. Slave nodes might implement, by default, a JSON-RPC server[11] to receive information from other nodes. Furthermore, they build a peer-to-peer network used to keep the blockchain synchronized. In our proof of concept, we have used the JSON-RPC interfaces to implement all the communications external to the blockchain (see Figure 2). The above two images were then used to instantiate two nodes for the SP (one SPB and one SPD) as well as four different nodes for the SC (one SCB and three SCDs). The result is shown in Figure 4, where the six nodes introduced above have been depicted. As represented in the figure, the nodes communicate with each other via a JSON-RPC interface and all of them write/read within our two blockchains.

[3]https://www.multichain.com
[4]https://bitcoin.org
[5]https://www.ethereum.org/
[6]https://hyperledger.org/projects/fabric
[7]https://litecoin.com
[8]https://ripple.com/
[9]https://hub.docker.com/r/kunstmaan/master-multichain/
[10]https://hub.docker.com/r/kunstmaan/node-multichain/
[11]https://www.multichain.com/developers/json-rpc-api/

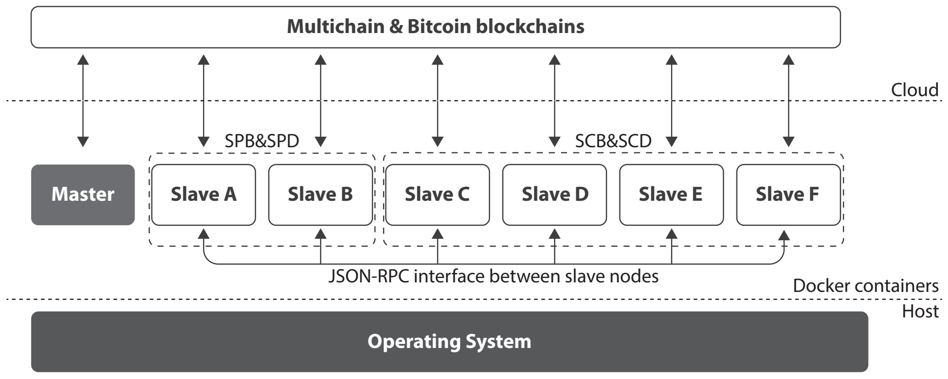


Figure 4: Implementation overview.

In Section 6 we analyze the results obtained from the above implementation. However, despite the proof of concept realized with multiple docker containers, for the experimental analysis we have changed the environment by using only one container acting as SC and SP at the same time, i.e. the masternode. The reason behind this decision is that we wanted to identify the core overhead of our solution without taking into account network delays brought by the JSON-RPC interface or by any other message passing protocol being adopted. As such, for the performance analysis, we have worked on two main directions: the generation and the verification of all the needed signatures involved by the obligation exchange, and the computation of the SC reputation. The first one mainly affects the SC as it has to be computed each time a SC wants to *spend* its obligations. Hence, a slow computation will result in a slow service provisioning and a bad user experience for the SC. The second part, i.e. the update of the SC's reputation, mainly involves the SP, as it has to be computed as soon as possible in order to be ready to serve future SC requests. Obtained results show that even in the worst case scenario, our solution performs better than the Bitcoin best case scenario (see Section 6 for more details), and that the overall sustained overhead is pretty viable.

6 Analysis

6.1 Security

Trust and reputation systems (TRS) have been suggested as an effective security mechanism for open and distributed environments. However, it has also been shown that such mechanisms can be threaten by different attacks, as shown by Fraga et al. [10].

In the remaining of this section we will first analyze the three macro categories in which attacks to TRS have been defined, and then we will introduce the security properties obtained by our solution and how do they mitigate the attacks described by Fraga et al. (Table 2).

The three macro categories are as follows:

- **Information Gathering Attacks**: labeled as GA_i in Table 2, they try to subvert the way in which *resources* information are gathered from *agents*, that in the Fraga et al. [10] scheme means how *observers* gather information on *entities* or other *observers*. In our scheme, these attacks are translated into attacks to the way in which SPs gather information on SCs;

- **Information Calculation Attacks**: labeled as CA_i in Table 2, they try to thwart the way in which *agents* compute and take decision on the reputation, based on the *resources* obtained in the gathering phase. This, in the scheme proposed by Fraga et al. [10], means that these attacks target the way in which *reputation servers* compute *entities*'s reputations based on the information provided by the *observers*. In our scheme, this is translated in how SPs access the obligation blockchain and take decisions on SCs' reputations;

- **Information Dissemination Attacks**: labeled as DA_i in Table 2, they try to thwart the way in which *agents* disseminate *resources*, which in the Fraga et al. [10] scheme means how disseminators share reputation information in the system. In our system these attacks are translated in attacks against the process in which SCs' information is shared among SPs;

Now that we have introduced the aforementioned macro categories, we can describe the security assumptions made in our system as well as the obtained security properties. For each of those, a reference to each specific attack listed in Table 2 is also made.

Secure Environments: our islands of trust embeds elements that are usually described in a trust and reputation system. More in detail, classical elements

		Gathering Attacks		Calculation Attacks		Dissemination Attacks
Authentication	GA1	Credential Forgery	CA1	Forgery of Credential to access TRS information	DA1	Routing credential forgery
Identification	GA2	Man in the Middle	CA2	Stolen/Cloned Identities	DA2	Man in the Middle
	GA3	Stolen/Cloned Identities			DA3	Stolen/Cloned Identity
	GA4	Sybil [7]			DA4	Sybil [7]
	GA5	Re-entry			DA5	Re-entry
Non-repudiation	GA6	Misbehavior	CA3	Authorship Rejection	DA6	Message routing rejection
	GA7	Confusion				
Authorization	GA8	Privilege Escalation	CA4	Privilege Escalation	DA7	Privilege Escalation
Availability	GA9	Dos Against entities and observers	CA5	Dos Against observers	DA8	Dos Against Disseminators
Utility/Process	GA10	Bad-Mouthing [2]	CA6	Initial Window	DA9	Loop
	GA11	Ballot-Stuffing [3]	CA7	Whitewashing [8]	DA10	Black, Gray or Worm Holes
	GA12	Incomplete, Null or Selective Information	CA8	Observer Malfunction	DA11	Misrouting
	GA13	Entity Malfunction			DA12	Disseminator Malfunction
Confidentiality	GA14	Information and Behavior eavesdropping	CA9	Information Eavesdropping	DA13	Message Eavesdropping
Integrity	GA15	Second Hand Information Manipulation [18]	CA10	Whitewashing [8]	DA14	Message Manipulation[18]
Time Integrity	GA16	Information Delay	CA11	Information Delay	DA15	Pulse Delay
					DA16	Message Re-transmission

Table 2: Trust and Reputation Systems' attacks taxonomy [10]

such as *observers* and *reputation servers* are here grouped within the SP as functionalities or back-end servers. The same also applies for classical *entities* which are here mapped to SCs' elements. The result is that, for all the attacks targeting observers, reputation servers, or entities in classical TRSs, they do not apply here as they are part of the islands, thus assumed secure. More in detail, attacks mitigated by the Islands of Trust security assumption are the following: GA1, GA3-GA8, GA13, CA2, CA4, CA5, CA8, CA9, DA3, DA7-DA12. Within this group, the most important attack for an access control scheme based on reputation is privilege escalation (GA8). In this attack, a malicious SC would create and use a big set of obligations to build a temporary good reputation which is then used for malicious purposes. Our solution mitigates this attack as new obligations written within the obligation chain do not affect SC's reputation until a payment is validated within the Bitcoin chain.

Immutable and Trusted Reputations: SCs' reputations are embedded and built upon two blockchains which are linked to each other. As such, all the information stored within them inherits the properties of being, among the others, immutable, tamper-proof, and globally verified. As such, all the attacks in classical TRSs that try to thwart either the reputation or the way in which this information is written/read do not apply here. More in detail, attacks mitigated by the blockchain security properties or the way in which blockchain transactions are sent in broadcast are the following: GA10-GA12, GA15, GA16, CA1, CA3, CA6, CA7, CA10, CA11, DA1, DA4-DA6, DA13-DA15.

Other attacks that do not belong to the above two groups and require some additional explanations are: i) man in the middle attacks, ii) denial of service, iii)communication weaknesses, and iv)replay attacks. Regarding man in the middle attacks, whilst it is theoretically possible, it is not feasible as it requires the attacker to have enough power to create two parallel blockchains (known as the 51% attack). A globally accepted one, and one re-written for the victim. Furthermore, as owning the 51% of the computing power is assumed not feasible, attackers are usually forced in *eclipsing* the victims in order to be facilitated in the creation of the fake chain. Controlling half of the network in the presence of Islands of Trust is assumed to be an overkill for attackers and compromising the Islands of Trust to unleash eclipse attacks violate our assumption on the Islands' security.

This mitigates GA2 and DA2 attacks.

As regards denial of service attacks such as GA9, this attack is translated in our scheme in *denying access to the obligation chain*. Whilst this attack is technically feasible, it can only succeed if network access to any blockchain peer is avoided—but this anomaly would be easily detectable. Beyond such a trivial attack, our scheme is protected thanks to the blockchain's immutability describe above.

Regarding GA14, i.e. information and behavior eavesdropping attacks, our solution is not focused on securing the communication between end devices, but rather between their back-ends. Indeed, our IoT devices (i.e. SPDs and SCDs) just convey messages that make any eavesdropping attack useless. Finally, message re-transmission or any kind of re-play attack (such as DA16) is mitigated in our solution by the fact that obligations are *one-time-only* and any kind of replay is easily detectable through the obligation chain.

6.2 Performance

In this section we present the result of our performance analysis. We have evaluated: i) the computational resources needed to generate and validate the obligations, and ii) the time required for SPs to compute SCs' reputations. Tests on the creation and validation of the obligations have been accomplished by using the *SigningKey*, *SECP256k1* and *VerifyingKey* python functions from the *ecdsa* library. For this test we have then realized a python script implementing the machine to machine protocol between SPD and SCD (see Figure 3). Indeed, this is the core part of our solution as it is the only one that can cause delays and overheads in accessing the service. More in detail, for the experimental tests we have implemented the functions that goes from *service_request()* to *signed_report()* in Figure 3.

The results are shown in Table 3 and Table 4. The first one shows the time needed as well as CPU and memory (RAM) consumption for all the steps involved in the protocol. As it can be seen, the whole process required less than 1.5 seconds to be completed, 33% of CPU, and a small amount of memory. Even though the CPU consumption might seem scary, it has been reported in the table that the CPU was busy processing the obligations for roughly one seconds thus proving the deployability of our protocol even on low power devices. For the sake of completeness, we have also analyzed the required resources when many obligations have to be created and verified. Table 4 shows the results for 10,000 and 100,000 obligations. Results show that, as for the case of a single obligation, the required resources are not prohibitive thus proving the feasibility of the proposed solution.

Finally, we have computed the delay in the obligation acceptance to check if, as expected, the proposed solution provides a virtual zero-latency user experience. Indeed, one of the major key points of our solution is that we are able to leverage Bitcoin's security and stability while providing a *zero latency* system. This is due to the fact that SPs' access control is executed by SPDs (steps 3-6 and 10-12 in Figure 2) and does not involve any blockchain operation. Interactions with the blockchain are only carried by the back-ends (SPBs and SCBs) in steps 14 and 15 which happen once the service as already been granted or denied.

To prove what claimed above we have tested the obligation acceptance delay in the worst case scenario (the one in which SPs have to always rebuild SCs' reputation from scratch) and compared it to the Bitcoin best case scenario (i.e. the smallest block acceptance delay witnessed in the last year). Figure 5 shows such a best case scenario highlighting the fastest accepted Bitcoin block appeared in the blockchain the 9 October 2017 (6.9 minutes acceptance time).

To implement our worst case scenario, we have then automated the handshake protocol between a SC and a SP with an increasing number of obligations which goes from 1 to 1460. We have then selected 1460 as it matches with the number of Bitcoin blocks created between October 2016 and September 2017.

The validation has been implemented by leveraging the *bitcoinJSON.read().find()* function provided within the *pybitcointools*[12] Bitcoin library that allows to check the balance of a given Bitcoin address. As such, we have created 1460 obligations and 1460 new Bitcoin addresses for their fullfillement. It has to be noted that, using Bitcoin addresses that have never received any bitcoin, does not weaken our results but make them stronger as checking non-used addresses requires to go back in time and to check the whole Bitcoin blockchain up to the genesis block.

The results are in Figure 6 and clearly show that checking SCs' reputations is faster than waiting for the validation of the fastest Bitcoin block. As the reputation bootstrap delay is linear in the number of obligations, it is clear from the graph that the delay is going to be bigger for more than 1460. However, it has to be noted that 6.9 minutes is the time for the fastest Bitcoin block for being added to the blockchain, but not to for being considered a trusted block. This indeed requires a time that, in Bitcoin, goes up to 1 hour. For the sake of completeness, it has to be noted that those blockchain-based solutions that use *channels* among peers can be as fast as the solution introduced in this paper. However, unlike our solution, the messages being exchanged within those

[12]https://github.com/vbuterin/pybitcointools

channels are not controlled by the blockchain, i.e. not received by other peers and thus, those solutions cannot be compared to the one proposed here. Indeed, obligations being transacted through channels would not be received by all SPs, thus preventing the update of SC reputation.

We can then conclude that our acceptance delay is always shorter than Bitcoin even in the case in which all the 1460 new obligations are verified at the same time (we refer to the asynchronous approach described in Section 3). The same results are obtained for the best case scenario (described as cached approach in Section 3) in which SCs' reputations are updated as soon as a new obligation blocks appear in the obligation chain. In this case, the acceptance time requires less than one second.

7 Related Work

Recently, in the academia, there has been the trend on redesigning rating systems (i.e. TRSs) in the new blockchain era. As examples, Shcaub et al. [16] and Soska-Christin [17] both have designed users' privacy-aware solutions based on the blockchain technology. However, the aforementioned approaches were not focused on TRS specific attacks but rather on general blockchain vulnerabilities. In this paper we are more focused on reputation attacks such as *rating frauds* in which a malicious user tries to seek inappropriate profits from the system. Such attacks occur both in content-driven and non-computational TRS systems but can be mitigated by using the blockchain technology as a source of immutable and non-repudiable rating information.

In the last few years, different proposals have been published to that end. Dennis et al. [15] proposed a solution in which the human rating factor (usually associated to *feedback*) has been removed. The result is a binary rating system in which either the service has been provided or not. Such an approach, however, weaken the rating system as it removes the *service quality* aspect. Indeed, the final goal of TRSs is to help SCs in understanding sellers. As such, if we only keep track of whether the product/service has been delivered, we will lose other factors and information which are as important as the delivery. In our solution we create commitments on the service quality which are signed by both SPs and SCs. As such, even though we use binary feedback (an obligation can be either fulfilled or not fulfilled) we maintain services' quality information within the obligation.

Other works tried to mitigate rating fraud by raising rate costs. This approach has been largely used in the past for standard TRSs. As examples, Douceur et al. prevented sybil attacks by binding accounts to unique IPs [7] while Yu et al. [19] increased the difficulty in controlling multiple accounts. Another example of raising costs or complexity has been designed by Yosang and Ismail [11]. In this solution, raters are encouraged to provide honest rates by sharing incomes with them. However, such kind of solutions are not effective if the perceived benefit from the attacks is greater than its cost. In traditional Cloud services (such as *Amazon.com* or *Expedia.com*), this problem is solved by using *verified transaction* labels but requires trust to be centralized.

The immutable and eventually agreed database held by the blockchain technology can be used to mitigate the aforementioned rating attacks. As an example, Shcaub et al. [16] proposed an interesting approach against bad mouthing attacks. In their solution, they kept the user feedback while allowing only peers who actually received tokens from seller to rate them. Compared to this approach, our solution solves the same problem but with a different perspective. Indeed, whilst Schaub et al. try to protect SCs from malicious SPs, we defend the latter from frauds. The main reason why we focus on this open challenge is that our solution is based on credits. As such, SPs accept *payment obligations* and take the decision on whether to trust the SC. Unlike what has been proposed by Schaub et al. in which SCs can take their time before making transactions with SPs, we need to take immediate (i.e. zero latency) decisions. In our scenario, SPs do not know at what time SCs will contact them and do not even know their identities. Still, they want to accept obligations while being sure to not be cozen.

TRSs can also be threaten by *ballot stuffing* sybil attacks in which the service provider colludes with SCs to gain reputations. Usually, these attacks leverage small fraudulent purchases. As such, a traditional way to mitigate them is to remove the *verified purchase* label on top of discounted transactions. As shown by Schaub et al., the blockchain technology can be used against ballot stuffing attacks as well. However, in their solution, SPs are limited in the number of tokens (feedback) that can be used. Even though this approach has proved to be effective as it creates a trade-off between ratings and profit, we think that it limits the overall business model. In our solution we do not assume any limitation on the number of obligations that can be created by SCs and accepted by SPs.

8 Conclusion

In this paper we have shown a solution that enables trust establishment among devices belonging to different domains. In particular, our solution is suited for the IoT context, given its unique features of being fully decentralized, and requiring both security and negligible overhead on end devices. These features are achieved in our proposal by leveraging the idiosyncratic properties of blockchain technologies, combined with a new archi-

	Timing	**CPU consumption**	**RAM consumption**
Signing and verifying SCs & SPs messages	0m1.122s	33.6% (1 core)	0.016 Gb
Payment Addresses creation	0m0.042s	0.7% (1 core)	0.016 Gb
Transaction creation	0m0.009s	1.0% (1 core)	0.016 Gb
Total/Maximum	0m1.173s	35.3% (1 core)	0.016 Gb

Table 3: SPD's service granting processes performance

	Timing	**CPU consumption**	**RAM consumption**
Whole protocol with 1 obligation	0m1.173s	35.3% (1 core)	0.016 Gb
Whole protocol with 10,000 obligations	0m1.226s	53.5% (1 core)	0.016 Gb
Whole protocol with 100,000 obligations	0m1.334s	54.2% (1 core)	0.016 Gb

Table 4: SPD's service granting different database register scenarios total performance

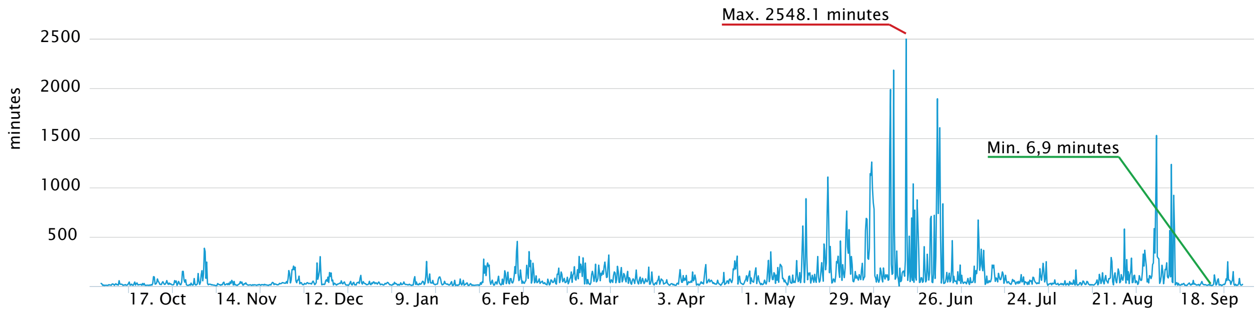


Figure 5: Bitcoin blocks average confirmation time - October 2016 to September 2017

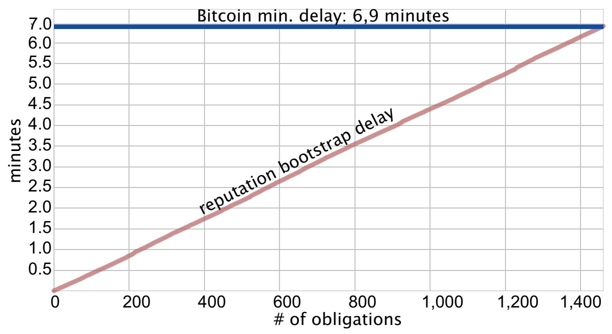


Figure 6: Number of simultaneous obligations accepted faster than one single Bitcoin block

tecture design that allows avoiding the pitfall inherited by this technology, while unleashing its advantages. In particular, as the performance of our implementation shows, all the above features are achieved in a very efficient way, and significantly faster when compared to standard Bitcoin based solution —hence enabling those use cases that otherwise could not work when facing long delays.

It is worth noting that our solution supports rich and flexible business models and use cases. Indeed, by allowing a seamless level of trust and cooperation among actors belonging to different domains, it removes entry-barriers to service providers thus introducing a disruptive degree of innovation. Finally, the obligation chain enables a wide range of credit based use cases.

We believe that the flexibility of our solution in supporting different business models, its degree of innovation, combined to its efficiency and overall deployability has a clear potential for opening further research threads in the highlighted directions.

References

[1] Ba, S., and Pavlou, P. A. Evidence of the effect of trust building technology in electronic markets: Price premiums and buyer behavior. *MIS Quarterly 26*, 3 (sep 2002), 243.

[2] BANKOVIĆ, Z., VALLEJO, J. C., FRAGA, D., AND MOYA, J. M. Detecting bad-mouthing attacks on reputation systems using self-organizing maps. In *Computational Intelligence in Security for Information Systems*. Springer Nature, Berlin, Heidelberg, 2011, pp. 9–16.

[3] BHATTACHARJEE, R., AND GOEL, A. Avoiding ballot stuffing in ebay-like reputation systems. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems* (New York, NY, USA, 2005), P2PECON '05, ACM, pp. 133–137.

[4] BOLTON, G. E., KATOK, E., AND OCKENFELS, A. How effective are electronic reputation mechanisms? an experimental investigation. *Management Science 50*, 11 (nov 2004), 1587–1602.

[5] BOUKERCH, A., XU, L., AND EL-KHATIB, K. Trust-based security for wireless ad hoc and sensor networks. *Computer Communications 30*, 11-12 (sep 2007), 2413–2427.

[6] DECKER, C., AND WATTENHOFER, R. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings* (Trento, Italy, sep 2013), Institute of Electrical and Electronics Engineers (IEEE), pp. 1–10.

[7] DOUCEUR, J. R. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems* (London, UK, UK, 2002), IPTPS '01, Springer-Verlag, pp. 251–260.

[8] FELDMAN, M., PAPADIMITRIOU, C., CHUANG, J., AND STOICA, I. Free-riding and whitewashing in peer-to-peer systems. *IEEE Journal on Selected Areas in Communications 24*, 5 (may 2006), 1010–1019.

[9] FOUNDATION, O. C. Online: https://openconnectivity.org/, 2017.

[10] FRAGA, D., BANKOVIC, Z., AND MOYA, J. A taxonomy of trust and reputation system attacks. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (Liverpool, UK, jun 2012), Institute of Electrical and Electronics Engineers (IEEE), pp. 41–50.

[11] JSANG, A., AND ISMAIL, R. The beta reputation system. In *In Proceedings of the 15th Bled Conference on Electronic Commerce* (Bled, Slovenia, 2002), Electronic Commerce Center.

[12] MARTINS, S., AND YANG, Y. Introduction to bitcoins: A pseudo-anonymous electronic currency system. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research* (Riverton, NJ, USA, 2011), CASCON '11, IBM Corp., pp. 349–350.

[13] PARK, D.-H., LEE, J., AND HAN, I. The effect of on-line consumer reviews on consumer purchasing intention: The moderating role of involvement. *International Journal of Electronic Commerce 11*, 4 (jul 2007), 125–148.

[14] RESNICK, P., KUWABARA, K., ZECKHAUSER, R., AND FRIEDMAN, E. Reputation systems. *Communications of the ACM 43*, 12 (dec 2000), 45–48.

[15] RICHARD, D., AND GARETH, O. Rep on the Roll: A Peer to Peer Reputation System Based on a Rolling Blockchain. *International Journal of Digital Society (IJDS) 7* (3 2016), 1123–1134.

[16] SCHAUB, A., BAZIN, R., HASAN, O., AND BRUNIE, L. A trustless privacy-preserving reputation system. In *ICT Systems Security and Privacy Protection*. Springer Nature, Cham, 2016, pp. 398–411.

[17] SOSKA, K., AND CHRISTIN, N. Measuring the longitudinal evolution of the online anonymous marketplace ecosystem. In *Proceedings of the 24th USENIX Conference on Security Symposium* (Berkeley, CA, USA, 2015), SEC'15, USENIX Association, pp. 33–48.

[18] XIONG, L., LIU, L., AND AHAMAD, M. Countering feedback sparsity and manipulation in reputation systems. In *2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2007)* (New York, NY, USA, nov 2007), Institute of Electrical and Electronics Engineers (IEEE), pp. 203–212.

[19] YU, H., KAMINSKY, M., GIBBONS, P. B., AND FLAXMAN, A. Sybilguard: Defending against sybil attacks via social networks. In *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2006), SIGCOMM '06, ACM, pp. 267–278.

[20] ZYSKIND, G., NATHAN, O., AND PENTLAND, A. S. Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops* (San Jose, CA, USA, may 2015), Institute of Electrical and Electronics Engineers (IEEE), pp. 180–184.