Proxy Re-encryption enabled Secure and Anonymous IoT Data Sharing Platform based on Blockchain

Ahsan Manzoor[a,d], An Braeken[b], Salil S. Kanhere[c], Mika Ylianttila[d], Madhsanka Liyanage[d,e]

[a]*Rovio Entertainment Company, Espoo, Finland. E-mail: ahsan.manzoor@rovio.com*
[b]*Industrial Engineering INDI, Vrije Universiteit Brussel VUB, Brussels, Belgium. Email: an.braeken@vub.be*
[c]*School of Computer Science and Engineering, University of New South Wales, Sydney, Australia. Email: salil.kanhere@unsw.edu.au*
[d]*Centre for Wireless Communications, University of Oulu, Finland. E-mail: {firstname.lastname}@oulu.fi.*
[e]*School of Computer Science, University College Dublin, Ireland. Email : madhusanka@ucd.ie*

Abstract

Data is central to the Internet of Things (IoT) ecosystem. With billions of devices connected, most of the current IoT systems are using centralized cloud-based data sharing systems, which will be difficult to scale up to meet the demands of future IoT systems. The involvement of such a third-party service provider requires also trust from both the sensor owner and sensor data user. Moreover, fees need to be paid for their services.

To tackle both the scalability and trust issues and to automatize the payments, this paper presents a blockchain-based marketplace for sharing of the IoT data. We also use a proxy re-encryption scheme for transferring the data securely and anonymously, from data producer to the consumer. The system stores the IoT data in cloud storage after encryption. To share the collected IoT data, the system establishes runtime dynamic smart contracts between the sensor and data consumer without the involvement of a trusted third-party. It also uses a very efficient proxy re-encryption scheme which allows that the data is only visible by the owner and the person present in the smart contract. This novel combination of smart contracts with proxy re-encryption provides an efficient, fast and secure platform for storing, trading and managing sensor data. The proposed system is implemented using off-the-shelf IoT sensors and computer devices. We also analyze the performance of our hybrid system by using the permission-less Ethereum blockchain and compare it to the IBM Hyperledger Fabric, a permissioned blockchain. .

Keywords: Proxy Re-Encryption, Blockchain, Smart Contracts, IoT Data Sharing, Security, Ethereum, Hyperledger Fabric

1. Introduction

The Internet of Things (IoT) is an emerging technology which has great technical, social, and economic significance. Current predictions for the impact of IoT are very impressive. It is anticipating that 100 billion connected IoT devices will be used by 2025[1]. It will also have a global economic impact of more than $11 trillion[2].

Data is central to the IoT paradigm. IoT data is collected to serve many different types of applications such as smart home, smart city, wearable, healthcare, smart grid, autonomous vehicles, smart farms, industries and manufacturing, and retail sector[3]. Therefore, numerous heterogeneous sensors exist to measure a variety of parameters. The collected data from these IoT sensors can be useful for different stakeholders. For instance, air quality measurements are of interest to governmental organizations, application developers and inhabitants of the relevant spaces. However, many challenges arise when organizing this data sharing as these IoT devices, which are typically resource-constrained, require efficient mechanisms to guarantee the data integrity and to enable proper processing and security[4]. Due to the large number of IoT devices, scalable deployment, and maintenance costs[3] should also be taken into account.

Currently, almost all the sensor systems use centralized cloud-based solutions to share sensor data with different stakeholders. IoT manufactures often use these third-party cloud service providers for storage, access control or even implementing their business intelligence services [5]. In that case, both data producers and consumers have to trust the third-party service provider and also need to pay some fee for their services. In addition, it is needed to establish an agreement between the data producers and consumers about the pricing and the amount of data shared. Moreover, these agreements can be even established without the consent of the IoT sensor [6]. Most of these agreements are static and takes significant time and administration to be established. For instance, the sensor data consumer has to pay the correct amount or buy a subscription to access data, which requires the involvement of other trusted parties such as banks. It will result in a significant increase in time before the actual data sharing can be realized [7].

On the other hand, the trustworthiness of the sensor data is also an important issue to bear in mind. Untrusted or third-party entities can alter information according to their own interests, so the information they provide might not be completely reliable [8]. Data consumers need to be sure that the information provided by IoT devices and by other external entities, such as data producers, has not been tampered or altered in any way. Thus, the current centralized architecture model in IoT systems

does not provide any solution for enhancing trust and will also struggle to scale up to meet the demands of future IoT systems.

In order to further monetize from the sensor data, auctions are used in the marketplace. The open ascending price auction is possibly the most common form of auction in use today. Most of the online auction platforms that currently exist are based on one centralized operator. They rely on proprietary and closed software [9]. As a result of this centralization, these auctions often lack transparency and bidders have no way to ensure the legitimacy of a higher bid.

Our Contribution: To solve the issues, the decentralized and consensus-driven blockchain technology and the underlying cryptographic processes behind it can offer an intriguing alternative. Thus, we propose a novel architecture, that uses blockchain in combination with a cloud service provider for the trading of the sensor data. We also propose a proxy re-encryption scheme to ensure the confidentiality and integrity of the data. The advantage of using blockchain to sell the sensor measurements with different entities is that the corresponding financial transactions are automatically managed through the agreed smart contract, stored on the blockchain. We also present a smart contract-based open ascending bid auction for selling the IoT data on competitive prices. Consequently, compared to the current IoT platforms where the data is stored in a cloud-based infrastructure, there is no need for manual verification of the payments and the predefined requirements. Also, disputes on these aspects are completely avoided.

To the best of our knowledge, our proposal is the first to use a hybrid architecture of combining blockchain with cloud storage to solve this particular issue of sharing data. Moreover, we discuss the different aspects of the implementation of the proposed scheme. On the one hand, we propose a very efficient proxy re-encryption scheme to be used as the security mechanism and how it allows that the data is only visible by the owner and the person present in the smart contract. On the other hand, we discuss the practical points such as the use of a smart contract, the storage of data and the communication between the cloud server provider and the blockchain. In order to verify the viability of the proposal, a prototype implementation of the hybrid architecture and proxy re-encryption scheme is done on a test bed. We used off-the-shelf devices and a commercial cloud service provider for the prototype implementation. Moreover, a detailed performance analysis is provided to demonstrate the scalability and performance metrics of the approach. To validate the generalization of our solution, implementation of the scheme in this paper was done using a public Ethereum blockchain i.e. Rinkeby Test Network [10] and later on permissioned IBM Hyperledger Fabric. To further understand these blockchain platforms, we compare their respective performances and transactions costs and also discuss their limitations in the end. The conference paper published earlier proposed the basic Proxy Re-Encryption Scheme and included implementation only on private Ethereum network without performance or security analysis.

The remainder of this paper has the following structure. Section 2 provides the background information and Section 3 presents related work. The proposed scheme is explained in Section 4. The security aspects and security analysis are presented in Section 5 and Section 10.Section 6 and Section 8 present the implementation of the proposed scheme on Ethereum and Hyperledger Fabric. The performance analysis results are presented in Section 7 and Section 9. Discussion and practical limitations are presented in Section 11. Finally, Section 12 presents our conclusions.

2. Background

2.1. Blockchain

A blockchain is a continually evolving, tamper-evident, shared digital ledger[11]. It holds the records of the transactions such as the exchange of assets or data between the peers in a public or private peer-to-peer network. The ledger is shared, replicated, and synchronized among the member nodes in the network. This ledger holds the records permanently in a sequential chain of cryptographic hash-linked blocks[11].

Without the involvement of a central authority or third-party mediator, the participant nodes in the blockchain network govern and agree by consensus on the updates to the records in the ledger. These records cannot be altered or reversed unless the change is agreed by all members of the network in a subsequent transaction[12].

Consensus mechanisms in blockchains offer the benefits of a consolidated and consistent dataset with reduced errors, near-real-time reference data, and the flexibility for participants to change the descriptions of the assets they own [12]. Moreover, none of the participating members own the source of origin for information contained in the shared ledger. The blockchain leads to increased trust and integrity in the flow of transaction information among the participating nodes [12].

2.2. Ethereum

Ethereum blockchain was introduced by Vitalik Buterin [13] in late 2013 and addressed several limitations faced by the Bitcoin network. It instantly became the second most common public blockchain. The Ethereum state consists of accounts, where each account has a 20-byte address and state transitions. Block generation time on Ethereum is decreased to 13 seconds and so is the size of the block. The potential usages of Ethereum are described as token systems, financial derivatives, identity and reputation systems, file storage, insurance, cloud computing, prediction markets, etc. [13].

In Ethereum "Gas" is a fundamental unit for computation. Each transaction requires a certain amount of computation and the "Gas Limit" states the maximum number of computational steps the transaction is allowed to consume. The usual price is 1 gas per individual computational step plus the fixed additional price for reading and writing to the data area.

Ethereum offers a smart contract functionality through the Ethereum virtual machine (EVM) running on the distributed nodes. Smart contracts are translated into the EVM code and then executed by the nodes [14]. Smart contracts on Ethereum are "Turing-complete", meaning it supports a broader set of instructions, including loops. Ethereum as a platform is suitable

for the issuance of tokens. Ethereum based tokens are smart contracts that implement the ERC20 Token Standard [15]. One of the most popular programming languages for writing smart contracts is Solidity [16].

2.3. Hyperledger Fabric

Hyperledger Fabric [17] is a distributed ledger by IBM and Linux foundation. Its modular architecture delivers a high degree of confidentiality, resiliency, flexibility, and scalability. The Hyperledger project was started in 2015 and launched in mid-2017. The Hyperledger Fabric is a private and permissioned blockchain, in which identities of all the participants are known. It is designed to support the pluggable implementation of different components to support complexities that exist across ecosystems.

Fabric supports modular consensus protocols, which allows the system to tailor to particular use cases and trust models. Hyperledger Fabric can store data in multiple formats, and it is also the first blockchain system that runs distributed applications written in standard, general-purpose programming languages, without systemic dependency on a native cryptocurrency [18].

Hyperledger Fabric consists of various components such as endorsers, ordering service, and committers. Due to numerous components and phases, Fabric provides various configurable parameters such as block size, endorsement policy, channels, state database. Hence, one of the main challenges in setting up an efficient Fabric network is finding the right set of values for these parameters, depending on the application and its requirements.

2.4. Smart Contracts

A smart contract is a computer protocol intended to digitally facilitate, verify, or enforce the negotiation or performance of a contract [19]. Unlike traditional contracts that rely on the reputation of the counterparties, smart contracts can be made between untrusted, anonymous people. Also, the execution of contractual terms is automatic and does not rely on any third party. The concept of smart contracts was introduced in 1990 by Wei Dai [20]. However, smart contracts were not possible in many traditional systems since the participating parties used to maintain separate databases and they did not rely on a proper trust model. However, the possibility to develop a trusted and shared database based on a blockchain has eliminated this limitation.

A blockchain-based smart contract is a self-executing code on a blockchain that automatically implements the terms of an agreement between parties [21]. Blockchain-based smart contracts could offer a number of benefits, such as fast, dynamic and real-time updates, low cost of operation, high accuracy and fewer intermediaries[19, 21]. These benefits also fuel the adaptation of a smart-contract in different applications. Thus, blockchain-based smart contracts are getting a significant interest across a wide range of industries.

Several blockchain platforms such as Bitcoin [20], Ethereum [14], Hyperledger Fabric [18], Stellar [22], iOlite[23] and Lisk [24] are available to develop smart contracts. However, Ethereum [25, 21] is the most popular platform due to its stable and safe operation.

2.5. Cryptographic Operations and Notations

The cryptographic scheme is based on Elliptic Curve Cryptography (ECC) [26], allowing to offer lightweight public key cryptographic solutions. For instance, corresponding with an 80-bit security parameter, a field size of 160 bits for ECC is sufficient, while RSA based solutions require 1024 bits. ECC is based on the algebraic structure of elliptic curves (ECs) over finite fields. We denote the curve in the finite field F_p by $E_{p(a,b)}$, defined by the equation $y^2 = x^3 + ax + b$ with a and b two constants in F_p and $\Delta = 4a^3 + 27b^2 \neq 0$. We denote by P the base point generator of $E_{p(a,b)}$ of prime order q. All points on $E_{p(a,b)}$, together with the infinite point form an additive group G.

The product $R = rP = (R_x, R_y)$ with $r \in F_q$ and $R_x, R_y \in F_p$ results in a point of the EC and represents an EC multiplication. The scheme relies on two computational hard problems.

- The Elliptic Curve Discrete Logarithm Problem (ECDLP). This problem states that given two EC points R and Q of $E_{p(a,b)}$), it is computationally hard for any polynomial-time bounded algorithm to determine a parameter $x \in F_q^*$, such that $Q = xR$.
- The Elliptic Curve Diffie Hellman Problem (ECDHP). Given two EC points $R = xP, Q = yP$ with two unknown parameters $x, y \in F_q^*$, it is computationally hard for any polynomial-time bounded algorithm to determine the EC point xyP.

In addition, we denote a one-way cryptographic hash function (e.g., SHA2 or SHA3) with output a number in F_q^* by H. The concatenation and xor operation of two messages M_1 and M_2 is denoted by $M_1||M_2$ and $M_1 \oplus M_2$ respectively. We further assume that the EC parameters and the associated EC operations, together with the hash function are implemented in each entity participating in the scheme.

3. Related work

There exist different studies on the security and privacy of the IoT [27, 28] and the vast majority of this research work is on understanding and identifying these threats [29]. The IoT devices sense, gather and share a large amount of data, thus opening up significant security and privacy concerns. Khan and Salah [30] in their paper have reviewed different security challenges in IoT and identified insecure transferring of IoT data as a high-level security risk. Authors in [31] demonstrated the lack of basic security by hacking off-the-shelf smart home IoT devices.

Blockchain got popular as it was used for recording financial transactions (e.g. Bitcoin), where transactions are encoded in blocks and kept by all the participants. Any modification in the blockchain transactions can be easily traced and detected. Later, blockchain has been used in other applications such as IoT, healthcare, transportation, and energy networks. Ethereum is one such blockchain system used in many applications. Ethereum is a blockchain-based distributed computing

system that has its own language such as Solidity [16]. It gives developers the flexibility to write their own codes and run them on the blockchain. Huh et al. [32] proposed using Ethereum blockchain to build an IoT system that could easily manage the configuration of the IoT devices and provide a platform for the key management system. They utilized smart contracts to manage the system in a fine-grained way. They implemented and evaluated the proof of concept with a few IoT devices to prove the feasibility of the system. Banerjee et al. [33] have analyzed security risks in sharing IoT data and proposed the use of blockchain to ensure the integrity and security of the shared datasets. They discussed two blockchain based approaches and presented nine research questions regarding them. However, they did not implement any of the proposed approaches. Authors in [34] compared a cloud server with a blockchain-based model for the security of IoT and they concluded that a cloud architecture costs more and is susceptible to manipulation. One of the main reasons for blockchain's incorporation into IoT is to strengthen its security. Authors in [35] have demonstrated how blockchains can be utilized for security and privacy of the smart home devices. They integrated local storage of sensor data in the blockchain. They have additionally analyzed their proposed framework with respect to fundamental security goals like confidentiality, integrity, and availability. Hyperleder Fabric is permissioned distributed ledger technology in which identities of all the participants are known to participate in the blockchain network [17]. Authors in [36] provide an architecture for commissioning of an IoT device into a cloud ecosystem using permissioned blockchain. However, they did not provide any implementation of the architecture. Carames et al. [37] has identified the new distributed applications in the blockchain IoT combination and later compare the blockchain platforms for the technical fit. Most of the applications identified in the paper uses permission-less blockchain.

The idea of IoT sensors as a tradable asset is related to that of Sensing as a Service (Sensing aaS) model. The following two technical architectures for data marketplaces are somewhat relevant to our work. First, the MARSA platform [38] is related to our work as it provides a marketplace where owners have an incentive to trade their data. This platform uses centralized trusted authority to manage the market place that is done by blockchain in our proposal. Second, Nokia Sensing as a Service[39] is an end-to-end solution that brings a platform for real-time processing, analyzing and selling of the sensor data. It allows the sensor owner to bring new monetization opportunities through digitalization and enables micro-transactions based on smart contracts that leverage the blockchain. This platform does not provide any details for secure transferring of the data from the owner to the customer. In this classification, our work is unique, as it enables fair and secure data exchanges amongst sensor owners and publishers on one side, and sensor data consumers, on the other. Sheikh et al. [40] proposed a Byzantine based blockchain framework for the energy trading process between electric vehicles and the distribution network. They evaluated and compared the proposed framework with other PoW blockchain but didn't implement any of the proposed approaches. Authors in [41] and [42] also purpose a secure Energy Trading Scheme based on public and private blockchain respectively. Nan-Li et al. [43] propose a secure decentralized trading solution for open data trading by sharing encryption keys using a smart contract on the blockchain. They implemented and evaluated the proposed framework only on private Ethereum blockchain. Dai et al. [44] introduce a secure data trading ecosystem on blockchain where they secure the data processing and give access to the analysis result to the buyer. They implemented the ecosystem on the Ethereum blockchain and Intel Software Guard Extensions and also performed in-depth analysis.

In 1998, Blaze, Bleumer, and Strauss [45] initially introduced the concept of proxy re-encryption and constructed the first bidirectional proxy re-encryption application. Proxy re-encryption has many exciting applications such as law enforcement, email forwarding and secure network file storage [46]. Jiang and Guo [47] proposed an encrypted data-sharing scheme for secure cloud storage based on the conditional proxy re-encryption. They proved the correctness and security of the proposed system and also analyzed the space and computational costs of it. Authors in [48], [49] also propose a similar scheme but it is not dynamic, hence making it unsuitable for cloud data sharing. In [50], a very efficient solution for data storage in the cloud is proposed using a pairing free proxy re-encryption scheme. However, the scheme is not implemented in practice. Although the underlying structure of our proposed scheme is based on it, some important modification like the inclusion of metadata is included to ensure a practical usage of the scheme.

Most of the prior work partly addresses the problem of securely sharing the IoT data. It is nearly impossible to come up with device-embedded security to solve all the security threats to the IoT devices. Limited computing and power resources of IoT also make the execution of complex security algorithms harder on the device. We propose using the combination of a blockchain and a pairing free proxy re-encryption scheme to provide a trading platform and to ensure secure transfer of the sensor data to the user.

4. Proposed Architecture

In this section, we present our new architecture based on the mechanisms of blockchain and re-encryption for secure storing and sharing of the sensor data. We consider four entities in the system: IoT sensors, Data requester, cloud provider, and the blockchain, as shown in Figure 1.

4.1. Stakeholders and their roles

4.1.1. Sensors

An IoT sensor node labeled as sensor owner in Figure 1 is a computing device that connects to the blockchain network and can capture and transmit data. It interacts with

(a) Cloud storage, to save and retrieve the encrypted sensor data from the database

(b) Blockchain, to perform smart contract transactions and manage the electronic wallet.

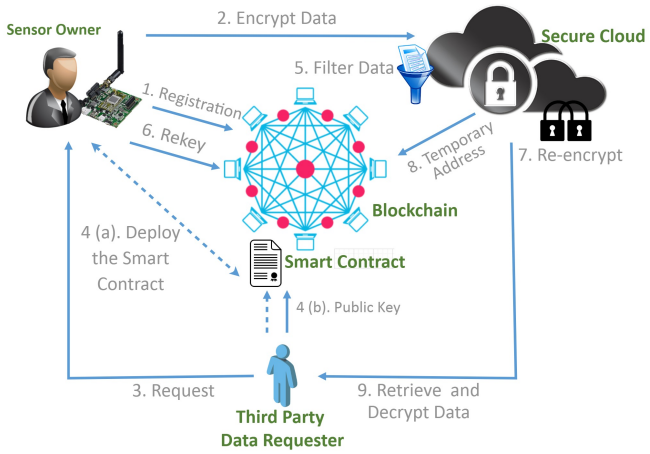

Figure 1: Proposed Architecture

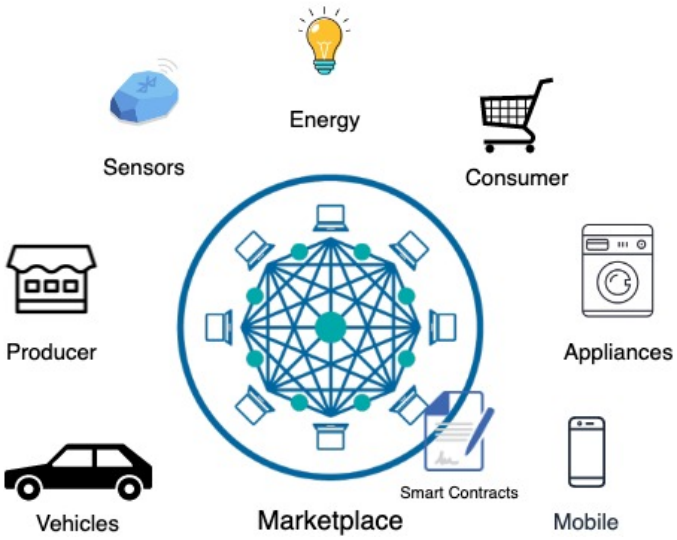

Figure 2: Blockchain Marketplace

The sensor node consists of a single or multiple low-powered IoT sensors, connected to a powerful IoT gateway node through Bluetooth low energy. The IoT gateway collects data from each source, encrypts it, and sends it to the cloud. Due to the low computational power of IoT sensors, these gateway nodes also manage all the blockchain transactions. It also runs a blockchain node and keeps a local copy of the ledger.

4.1.2. Requester

The user node is labeled as a third-party requester in Figure 1. It can be seen as a software agent acting on behalf of the user for specifying the requirements and type of sensor data to be queried. It manages cryptographic keys of the user needed for the re-encryption scheme. The user node also acts as the blockchain node and manages all the financial associated transactions related to the user e.g. transferring tokens to IoT sensor or bidding for the data.

4.1.3. Cloud Server

The cloud storage node stores the encrypted sensor data coming from the IoT gateway and it also entertains the user request by returning the records that match the third-party requester specified criteria. The cloud server also runs a blockchain node and connects to the network to do transactions and maintain a copy of the ledger.

4.1.4. Blockchain

A known and trusted application shares and synchronizes transaction data across multiple nodes. It interacts with all the entities in the system and logs those interactions in the form of transactions. Smart contracts only live in the blockchain context and are used for accessing blockchain external data. The smart contract manages the financial transaction costs and checks the corresponding requirements (e.g. Data Location) related to the data.

4.1.5. Market Place

Blockchain-based or decentralized marketplaces are peer-to-peer networks that directly connect consumers and producers without any intermediaries. IoT Data from different sources can be sold and bought on this platform. Just like the traditional marketplace, producers supply information about their products and consumers look for goods and make purchases. It offers a new business model, resulting in new opportunities for user interaction, payments in cryptocurrency and agreements based on smart contracts.

4.2. Functions of the Proposed Architecture

1) The sensors' owner activates the sensors and registers them on the blockchain, either via a smart contract function or Certificate Authority (CA), depending on the distributed ledger technology. The owner provides the price of the data along with the registration transaction. Later, he can also auction the data by using the bidding smart contract.

2) After successful registration, the sensors' owner provides the sensor with the required key material such that the measured data can be sent encrypted to the cloud storage server. The cloud server provider also checks the authentication and integrity of the data received from the sensors. If correct, the data is securely stored locally on the server.

3) A user requests access to one of the sensor(s) data via the smart contract function and initiating transfers of tokens to the sensor owner. He can also participate in the bidding process if the data is auctioned on the marketplace.

4) After receiving the request, the sensors' owner and requester come to an agreement, a smart contract is mined on the blockchain by the requester, confirming the deal. The requester also shares its public cryptographic key along with the smart contract.

5) On receiving the user request, cloud storage is notified using the smart contract events. The software running on

the cloud then filters the data according to the request and saves it on a temporary location.

6) Sensor owners are notified about the address of the newly mined smart contract using the swarm messages. It generates the re-encryption cryptographic key using the public keys from sensor owner and requester before uploading it on the blockchain.

7) The software running on the cloud receives the re-encryption cryptographic key and re-encrypts filtered data using the re-encryption key, before storing it again on a temporary location onto the cloud storage server.

8) When the data is ready, the requester is notified of the temporary location path of the data by the smart contract's emit event function through the blockchain.

9) The requester can download the data and can only decrypt it using its own private cryptographic key.

5. Security aspects

We will discuss in this section the requirements, cryptographic operations and notations, and the proposed re-encryption scheme.

5.1. Requirements

The system should satisfy the following fundamental security features.

- Confidentiality: The sensor data needs to be securely sent to the cloud server provider. Only the sensor owner, sensor and subscribed user are able to retrieve the actual measurements of the sensor. Consequently, no outsider, even not the cloud server, is able to derive useful information from the transmitted messages.
- Integrity: The content of the data from the sensors to the cloud server provider should not be altered without being notified by the cloud server provider.
- Authentication: Everybody, in particular, the cloud server provider and subscribed user, is able to check the authentication and integrity of the data.
- The system should be resistant against well-known security attacks like man-in-the-middle attacks, impersonation and replay attacks.

The challenging part is to establish these security features in the most efficient way from the part of the sensor. Furthermore, the following assumptions are made:

- The cloud server provider is responsible for the correct storage of the data. Moreover, it takes care of the communication of the addresses of the stored data to the blockchain. We consider the security model of an honest but curious server, meaning that it will perform all the required actions but is curious in deriving the data itself in order to use it for its own purposes (e.g. selling). The owner can check the correct behavior at any time by consulting the blockchain transactions.
- We consider a secure communication between the cloud server provider and blockchain using traditional security mechanisms like Secure Sockets Layer (SSL) as both entities are not considered to be constrained devices. Consequently, the focus of the description on the security mechanisms is between the IoT sensor and the cloud server provider.
- We assume a Yao-Dolev attack model in which the attacker can be both active or passive. In case of a passive attacker, the attacker is able to eavesdrop the communication, while an active attacker can also modify, delete or replay (part of the) communication. As a consequence, special attention should be given to protection against man-in-the-middle attacks, replay attacks and impersonation attacks.

5.2. System model

We propose to apply a Certificate Based Proxy Re-Encryption (CB-PRE) scheme, which constitutes of seven polynomial-time algorithms: Setup, CertifiedUserKeyGen, Encrypt, ReKeyGen, ReEncrypt, Decrypt1, and Decrypt2. We now explain each of these phases into more detail. In our proposal, we have combined the phases UserKeyGen and Certify to one phase called the CertifiedUserKeyGen phase.

The proposed scheme is very similar and based on (REF), in which a complete formal security proof has been given. However, to make the scheme practically, we had to make some slight changes like the addition of metadata. Note that this change has no impact on the security strength and the proof given in (REF) can still be applied [51]. We use metadata associated with the message and ciphertext for efficiency reasons to facilitate the look-up of information and the corresponding request from the delegate. For instance, a delegate can be interested to obtain all information posted by a particular delegator (in our context sensor) during a certain period. The metadata can consist of multiple fields. As a minimum, we propose the following fields:

- id_A: The identity of the delegator
- T_0: The timestamp of the creation of the message.

Additional fields can be for example a list of keywords, an access control list, and corresponding access rights, etc. The purpose of the different phases is summarised below:

- Setup(l): This algorithm is executed by the Certificate Authority (CA) and takes as input a security level l, and generates based on this a list of public parameters *params*. In addition, also a master secret key *msk* of the proxy is generated. The public parameters are published and *msk* is kept secret.

- CertifiedUserKeyGen($params, id_u$): This phase enables the generation of a private and public key of the sensor/user in such a way that only the involved entity is aware of the private key and no secure channel for the distribution of the key material is required. The construction of the public key requires the usage of a certificate generated by the CA.

- Encrypt($params, M, id_A, d_A, T_0$): In this phase, the delegator A is able to generate a valid ciphertext C_A for M and corresponding metadata *meta*, using $params, id_A$, the timestamp T_0, and its private key d_A. The output C contains $C_A, meta$ and auxiliary information (h_A, s_A) to check the authentication.

- ReKey($params, d_A, id_B, Cert_B, C_A, meta$): The ReKey phase generates a re-encryption key rk_{AB} for the output C of the Encrypt phase coming from id_A for the delegate with identity id_B, using its certificate $cert_B$ to compute the corresponding public key P_B.

- ReEncrypt($params, C_A, rk_{AB}$): This algorithm re-encrypts the ciphertext C_A, using the re-encryption key rk_{AB} to obtain the ciphertext C_B. Replacing C_A into C_B of C, results in C'.

- Decrypt1($params, C, d_A$): The first decryption algorithm allows the delegator to retrieve its message from the ciphertext C_A and to check the integrity.

- Decrypt2($params, C', d_A$): The second decryption algorithm allows the delegate to retrieve the encrypted message of id_A from the ciphertext C_B of C', using its private key d_B. Also, the integrity and authentication are verified.

5.3. Operation of the System

We now discuss the different operations in more detail.

- SetUp(l): Given a certain security parameter l, the following steps will be executed to derive the public parameters *params* and the master secret key *msk*.
  - First, the CA chooses an l-bit prime q. Next, an EC of order q is generated, and a corresponding generator point P is defined. Denote by G the group of EC points.
  - A random value $\alpha \in F_q^*$ is chosen and $P_\alpha = \alpha P$ is computed.
  - Four different hash functions are determined. $H_1 : G \times \{0,1\}^{32} \rightarrow F_q^*$, $H_2 : F_q^* \times \{0,1\}^{64} \rightarrow F_q^*$, $H_3 : \{0,1\}^{64} \times G \rightarrow F_q^*$, $H_4 : F_q^* \times \{0,1\}^{64} \times \rightarrow F_q^*$.
  - The public parameters are now $params = \{G, q, P, P_\alpha, H_1, H_2, H_3, H_4\}$ and the master secret key is put as $msk = \alpha$.

- CertifiedUserKeyGen($params, id_U$): This algorithm is based on the Elliptic Curve Qu Vanstone (ECQV) certificate mechanism [52] and consists of the following three phases:
  - First, the involved entity id_U generates a random value $r_U \in F_q^*$ and computes $R_U = r_U G$. Next the tuple (id_U, R_U) is sent to the CA.
  - Upon arrival, the CA checks the identity of id_U. Next, it also chooses a random value $r_t \in F_q^*$ and computes $R_t = r_t P$. Then the certificate $Cert_U = R_U + R_t$ is derived. Finally, auxiliary information to derive the private key for the involved entity is computed by $r_a = H_1(Cert_U || id_U) r_t + \alpha$. The tuple $(r_a, Cert_U)$ is sent back.
  - The involved entity computes first its private key $d_U = H_1(Cert_U || id_U) r_U + r_a$. Its public key equals to $P_U = d_U P$. If $P_U = H_1(Cert_U || id_U) Cert_U + P_\alpha$, it accepts the key pair (d_U, P_U).

- Encrypt($params, M, id_A, d_A, T_0$): The metadata is generated for the message M, ie. $meta = (id_A || T_0)$. Next, the following computations are made.

$$\begin{aligned} r &= H_2(d_A || meta), R = rP \\ C_A &= M \oplus H_3(meta || rP_A) \\ h_A &= H_4(C_A || meta) \\ s_A &= r - h_A d_A \end{aligned}$$

  The output C of this algorithm equals to $C = (C_A, meta, h_A, s_A)$.

- ReKey($params, d_A, id_B, Cert_B, C_A, meta$): First $r = H_2(d_A || meta)$ is derived from C. Then, the public key of id_B is computed as $P_B = H_1(Cert_B || id_B) Cert_B + P_\alpha$. This leads to the definition of the ReKey as

$$rk_{AB} = H_3(meta || rP_A) \oplus H_3(meta || rP_B)$$

  The output is the key rk_{AB}.

- ReEncrypt($params, C_A, rk_{AB}$): The re-encryption phase changes the ciphertext C_A to C_B by

$$C_B = rk_{AB} \oplus C_A$$

  Note that C_B also corresponds to $M \oplus H_3(meta || rP_B)$, which will be used in the decrypting phase of the delegate. The output C' is now the tuple, containing $C_B, meta, ID_B, h_A, s_A$.

- Decrypt1($params, C, d_A$): Here the delegator wants to decrypt the ciphertext to derive the original message and to check its authenticity. Therefore, the following computations are required:

$$\begin{aligned} r &= H_2(d_A || meta) \\ M &= C_A \oplus H_3(meta || rP_A) \\ h_A &= H_4(C_A || meta) \\ \text{Check: } s_A &= r - h_A d_A \end{aligned}$$

- Decrypt2($params, C', d_B$): In this phase, the delegate B derives the message M from C' by the following operations.

$$\begin{aligned} R &= s_A P + h_A P_A \\ M &= C_B \oplus H_3(meta \| d_B R) \\ \text{Check: } h_A &= H_4(C_A \| meta) \end{aligned}$$

5.4. Connection with the blockchain

We now discuss the relationship between the phases of the proxy re-encryption scheme and the blockchain using the steps of the proposed architecture in Figure 1.

- In step 2, the sensor owner generates keys for the encryption scheme using Elliptical Curve Cryptography (ECC), using the curve P-256. Key holders can use their private key to encrypt and sign a piece of data and save them on the cloud.

- In step 4, the data requester generates its own key pair and share it with the data producer. The Data producer is responsible for the deployment and management of the smart contract on which the public key P_B is uploaded by the data requester.

$$public\_key < -H(P_B)$$

- In step 6, the re-encryption key rk_{ab} is derived by the IoT sensor using the public key P_B. It is then shared with the cloud using the same smart contract. The re-encryption key is hashed and signed using the ethereum private key of the sensor, before being mined by the blockchain. The following operations are made.

$$\begin{aligned} rk_{AB} &= H_3(meta \| rP_A) \oplus H_3(meta \| rP_B) \\ reKey &< - H(rk_{AB}) \end{aligned}$$

- In Step 7, the cloud server then generates a valid ciphertext C_B using re-encryption key rk_{ab} and uploads it on the cloud storage.

$$C_B = rk_{AB} \oplus C_A$$

- In step 9, the encrypted data is shared with delegate B, who decrypts it with its private key d_b and verifies the signature.

6. Implementation on Ethereum

This section provides the implementation details of the Blockchain based proxy re-encryption scheme.

We demonstrate the feasibility of the system design with the prototype implementation containing a permissionless Ethereum blockchain, IoT sensors and a cloud server for storage of the data. Figure 3 illustrates the setup of the system with

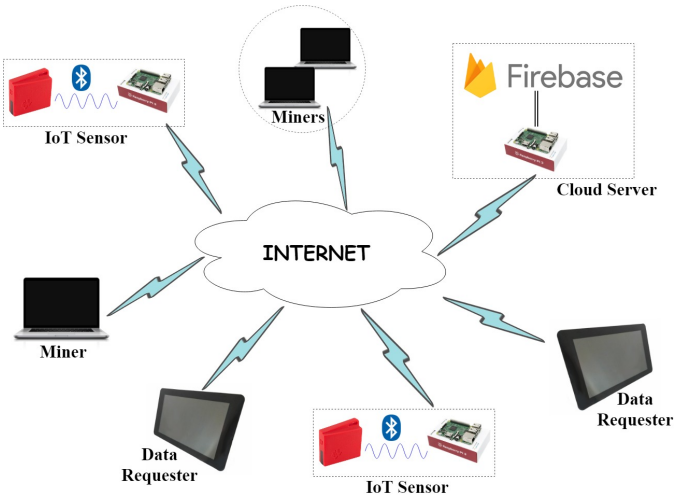

Figure 3: Overview of the Architecture Implemented

Table 1: Devices and their capabilities

	Miner	Full node	Iot Sensor
Device	Dell Latitude	Raspberry Pi 3	TI Sensortag CC2650
# of nodes	3	5	2
RAM	4GB	1GB	-
OS	Ubuntu 18.04	Raspbian Jessie	BLE Stack 2.2.2
Geth	v1.8.0	v1.8.0	-
Connectivity	Internet	Internet	BLE

three IoT sensors, three mining computers, five ethereum full nodes, two regular users and one cloud storage server. We summarize the device types and their capabilities in Table 1.

We configured and connected all the computer devices to the internet. We used the auto-discovery protocol of Geth to connect the Ethereum miners and the full nodes, and in the end we configured Google firebase cloud for storage.

6.1. Miners

The proposed system consists of three miners that generate a block of transactions on average every 13 seconds. These miners are running on a virtual machine with the same hardware capabilities. All the mining devices were configured to use one Ethereum wallet that collects the mining reward. Newly mined tokens are then added to the overall balance that can be bought by users to pay for services, hence compensating for the miner electricity consumption. These miners are running on Geth v1.80 [53] with four mining threads each.

6.2. Smart Contracts

We developed two smart contracts[1] on truffle[54] and compiled them with Solidity 0.4.24 [16]. The first smart contract

[1] https://github.com/ahsan100/smart-contract

consists of the functions to register the sensor, request data, and financial functions. The second smart contract is dynamically created in the runtime when the user requests for the data. The address of the second smart contract is shared with the concerned entities via the swarm messaging. To increase the security we used function modifiers to automatically check the identity of the person calling the smart contract functions and allowing only the concerned entities to call them.

6.3. IoT Sensors

Each sensor TI Sensortag CC2650 connects to a Raspberry Pi 3 Model B (RSP) through Bluetooth Low Energy, as shown in Figure 3. This RSP manages the sensor and the Ethereum account to perform transactions on the blockchain on behalf of sensors. A sensor application is developed in Python 2.7.12 that connects to the sensor, performs the cryptography functions described in the proxy re-encryption scheme on the sensor data and uploads that data to the cloud storage server. This application synchronizes with the blockchain using the Python- JSON-RPC (JavaScript Object Notation - Remote procedure Calls) library. As the blockchain notifies the application about the data request, it queries the Google Firebase and downloads the filtered data. It gets the re-encryption key from the smart contract and performs proxy re-encryption of the filtered data.

The registration of the sensor device on the system is performed by calling the function "methodRegister" of the smart contract. A fixed price of the sensor data and MAC address is sent along the function for registration. The MAC address of each sensor acts as its identity and is used for re-encryption. Once registered, the sensor starts uploading the encrypted data to the cloud server. This RSP stores the private key locally and provides the cloud with a re-encryption key via the blockchain. It is assumed that the BLE connection between sensor and RSP is completely secure.

6.4. User Application

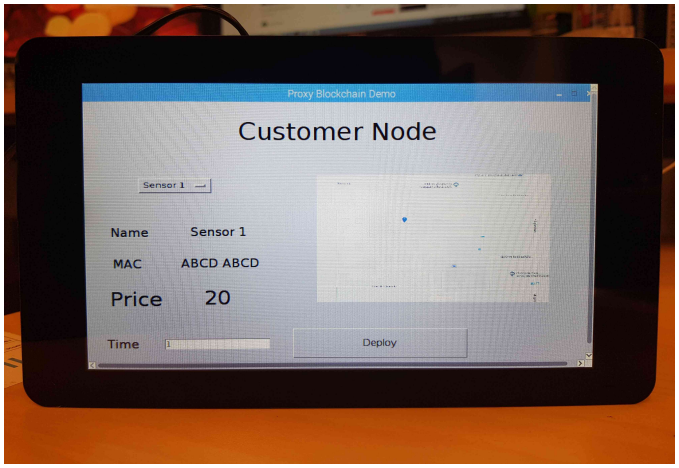

Figure 4: User Interface

A customized application is designed as the user interface in Python 2.7.12, running on a Raspberry Pi 3 attached to a touchscreen, as shown in Fig. 4. This application uses JSON-RPC to

Algorithm 1: Sensor Registration and User Balance Contract

```
mapping (bytes32 => device) sensorList
mapping (address => uint256) balances
Init: uint256 total_sensor
constructor (InitialSupply):
    balanceOf[Owner] = InitialSupply

Event Transfer(Sender, Receiver, amount);
struct {
    string name
      string sensorID
      address creator
      uint256 price
} devices
Function methodRegister(name, MAC, price):
    newKey = stringToBytes32(name)
    total_sensor += 1
    sensorList[newKey].name = name
    sensorList[newKey].sensorID = MAC
    sensorList[newKey].creator = Requester
    sensorList[newKey].price = price

Function sendCoin(Requester, Amount):
    if balanceOf[Requester] > Amount then
        balanceOf[Requester] += Amount
        balanceOf[Owner] -= Amount
        return TRUE
    else
        return FALSE
    end

Function getBalance(Requester):
    return balanceOf[Requester]

Function getPrice(name):
    Key = stringToBytes32(name)
    return sensorList[Key].price

Function requestData(name, time):
    Key = stringToBytes32(name)
    receiver = sensorList[Key].sensorID
    price = sensorList[Key].price x time
    sendcoin(receiver, money)
    return True

Function getSensors():
    return total_sensor
```

get the sensors' information from the blockchain. After selecting the required sensor, the user enters details for specifying the data requirements. It interacts with the blockchain via a personal ethereum account. As the user calls the "requestData" function of the smart contract, we notify the sensor device and the cloud sever using the Event function in the smart contract. The "sendCoin" function is used to buy and transfer the tokens, required to pay for the sensor data.

We deploy a new smart contract on the blockchain in runtime based on the user-selected options for the requested data (e.g. Sensor selection, Price). The user also sends the public key as the constructor value when deploying the smart contract.

This application keeps track of the Ethereum wallet along with ECC [26] secret key of the data requester. The cloud server notifies the application when it updates the smart contract with the temporary address of the filtered sensor data. The application downloads the data from the cloud server, checks for the signature and integrity, and decrypts the requested data.

6.5. Cloud Storage Server

The cloud storage server consists of the RSP and the Google Firebase. RSP acts as ethereum full node and connects to the blockchain, while Google Firebase is used for the storage of the data. The authentication and integrity of the data are performed on the RSP and encrypted sensor data along with the meta-data is uploaded to the Google Firebase in JSON format. This cloud also performs proxy re-encryption and updates the smart contract variable for data address sharing.

6.6. Proxy Re-Encryption

An application is implemented in Python 3.6.5 to perform the proxy re-encryption on the RSP. As the blockchain notifies the application about the data request, it queries the Google Firebase and downloads the filtered data. It gets the re-encryption key from the smart contract and performs proxy re-encryption of the filtered data.

6.7. Data Auction

A smart contract was written for the auctioning of the data as shown in Algorithm 2. Any data producer can publish its data for public auction and any consumer can participate in it. Consumers can withdraw from the bidding process at any time and producers can also cancel the bidding. In the end, all the bidders are informed about the result as its an open auction and the highest bidder get the data after transferring money to the producer.

Algorithm 2: Contract for bidding

```
Init: uint startBlock
Init: uint endBlock
Init: bool canceled
Init: uint highestBid
Init: address highestBidder
mapping (address => uint256) Bidders
constructor (_startBlock, _endBlock):
    startBlock= _startBlock
    endBlock= _endBlock

Function placeBid():
    if msg.value == 0 then
        revert()
    else
        uint newBid = msg.value
    end
    if newBid <= highestBid then
        revert()
    else
        fundsByBidder[msg.sender] = newBid
        highestBid = newBid
        highestBidder = msg.sender
    end
    return True

Function cancelBidding():
    canceled = true
    return True

Function withdraw():
    if canceled then
        msg.sender.transfer(fundsByBidder[msg.sender])
    else
        if msg.sender == owner then
            msg.sender.transfer(highestBid)
        else
            msg.sender.transfer(fundsByBidder[msg.sender]
        end
    end
    return True
```

7. Performance Analysis for Ethereum

In this section, we describe the experiments to evaluate the proof of concept implementation. Experiments were designed to study the performance of the framework. We have performed multiple experiments, starting from the time taken to register a sensor to performing data requests and blockchain transactions. We tested the impact of proxy re-encryption on the overall system and performed some scalability tests.

7.1. Execution Cost

Gas is a unit denoting the price of computation on the Ethereum, paid in Ether by users to miners in order to utilize the computational power of the network. Table 2 shows the execution cost expressed in Gas price, for the different smart contract functions on Ethereum blockchain. The most expensive function is the deployment of the Algorithm 1 contract, it consumes around 753,324 gas.

Table 2: Execution cost of smart contract

Operation	Gas Used	Price in $
Algorithm 1 Deploy Cost	753,324	$0.357
methodRegister	126,073	$0.060
sendCoin	51,101	$0.024
Algorithm 2 Deploy Cost	718,218	$0.340
cancelBidding	22,147	$0.011
placeBid	66,135	$0.031
withdrawBid	79,837	$0.038

At the time of writing, one Ether costs $155 [55], however, the price is still very volatile. As a result, transaction costs may frequently vary. The main drawback when setting a low Gas price is the increase of time required before a transaction is validated. Assuming the average Gas price to be 3 Gwei, the time required for a transaction to be validated on public Ethereum network is 29 seconds or on average 2.2 blocks.

7.2. Sensor Registration

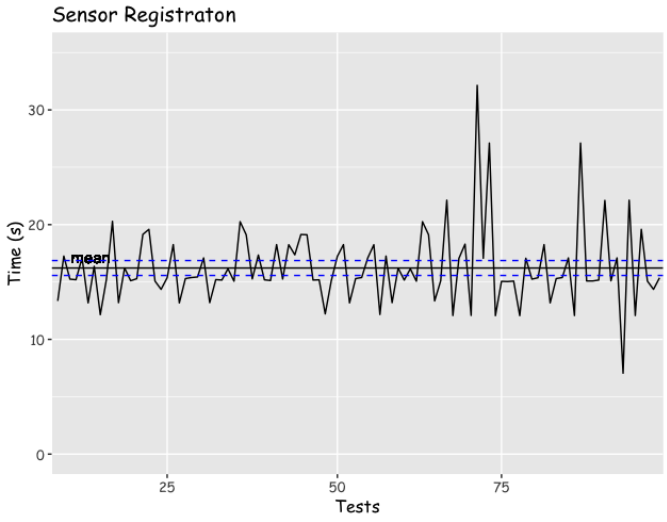

Figure 5: Sensor Registration Delay

In the first experiment, we measure the time taken to register a sensor on the blockchain. In order to register, the sensor node sends a smart contract transaction, and we measure the delay it takes to mine that transaction. Figure 5 illustrates the average registration delay and variation over 100 runs of the scenario. Experiment results reveal that it takes 16.22s on average to register a single sensor with error margin of 0.65s. Blue dotted lines show the 95% confidence level for the mean. This delay is closely linked with the average time for block generation in Rinkeby Ethereum network, i.e. 15 s. This concludes that block generation has the highest impact on the sensor registration.

7.3. Impact of Proxy Re-Encryption

In the second experiment, we measure the impact of proxy re-encryption on the proposed system. The second experiment is divided into two parts. The first part consists of a user request for the data using the blockchain and the sensor data is shared through the cloud without any encryption, while in the second part encrypted data is uploaded on the cloud and re-encrypted by the cloud.

7.3.1. Without Proxy Re-Encryption

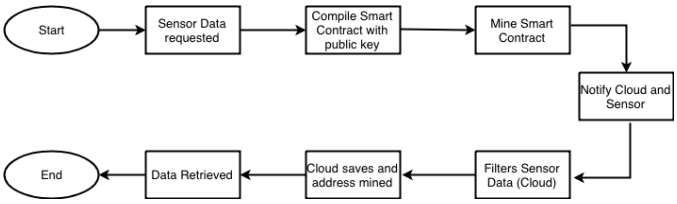

Figure 6: Flow Chart without Proxy Re-encryption

Figure 6 shows the flowchart describing the steps followed to measure the delay for the whole process without the proxy re-encryption scheme. We ran the experiment for 50 times before taking average for each phase and it takes on average 30.23 s for the end-to-end process.

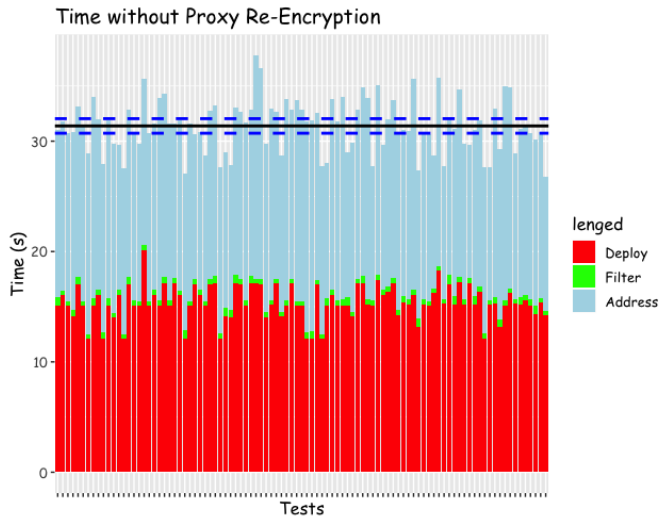

Figure 7: Impact without Proxy Re-Encryption

We start measuring the time as soon the user requests for the data. In the deploy phase, the user compiles the smart contract in runtime and sends it to the blockchain as a transaction. As the contract is deployed, the cloud starts filtering the data according to the requirement, this phase is marked as the filter on Figure 7.

The last phase includes the mining of the temporary address on the blockchain. As can be concluded from Figure 7, deploying of the contract and mining of the temporary address have the highest impact on the delay.

7.3.2. With Proxy Re-Encryption

In the second part, the sensor encrypts the data before uploading it to the cloud and later re-encrypts it for sharing the data. This experiment was similar to the first part except for the addition of the re-encryption phase that includes the generation of the re-encryption key and later mining it on the blockchain for sharing the key with the cloud server. Figure 8 shows the flowchart with the process followed for the measurements.

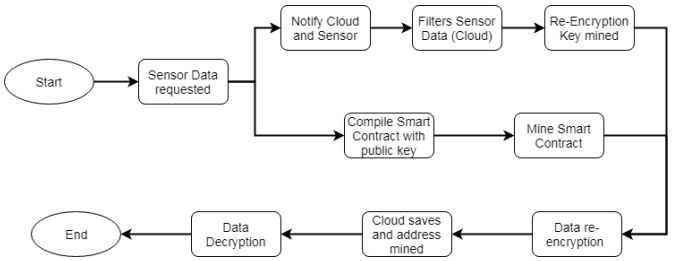

Figure 8: Flow Chart with Proxy Re-encryption

In Figure 9, the time of the different parts in the scheme is illustrated. As can be seen, it takes on average 48.01 s to share the encrypted data with the user after the initial request with a confidence interval of 2.07 s. Consequently, adding proxy re-encryption to the scheme increases the delay by 60% due to the mining of the re-encryption key. The re-encryption phase includes filtering and re-encrypting of the data and it requires less than 1 s for most of the cases.

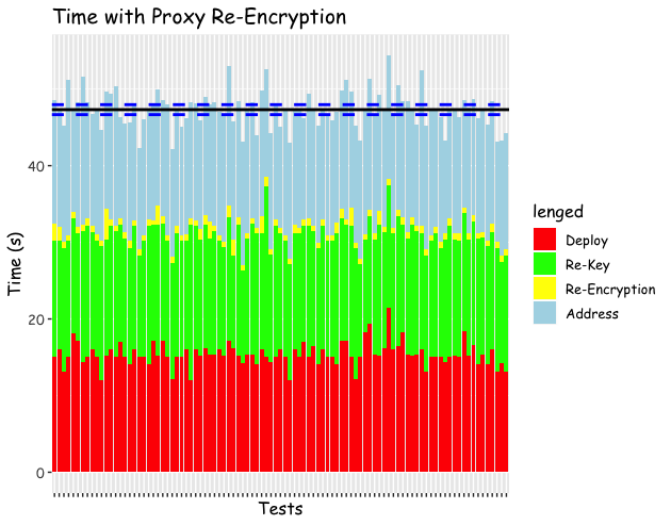

Figure 9: Impact of Proxy Re-Encryption

7.4. Scalability

In the third experiment, we measure the scalability of the architecture by performing multiple transactions from multiple requesters to one sensor. The whole process was repeated 10 times for each scenario before taking the average. In the first scenario, only 1 request was initiated by the user and time was measured from the request to the retrieval of data by the requester. In the latter scenarios, the process was repeated by increasing five requests until the overall request reached to 100.

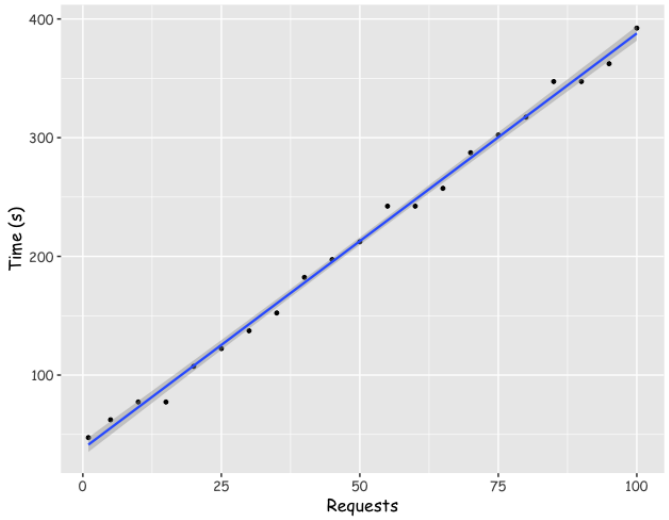

Figure 10: Scalability Test

As seen from Figure 10, the process shows a gradual increase in the delay due to the increase of transactions. This increase in the delay is caused by the scalability problem of the Ethereum blockchain. There seems to be a tradeoff between speed and reward for the generation of the new block. The number of transactions mined in a single block of Ethereum blockchain depends on multiple factors such as gas price and limit. Another factor that increases the delay is the cloud storage server. As all the requests are sent concurrently, the cloud takes time to filter, re-encrypt and save the data. This can be reduced by adding more cloud servers and distributing the load.

8. Implementation on Hyperledger Fabric

In this section, we present the architecture of the application based on the permissioned blockchain and proxy re-encryption scheme. Fig. 11 shows the Hyperledger fabric network and its components in detail.

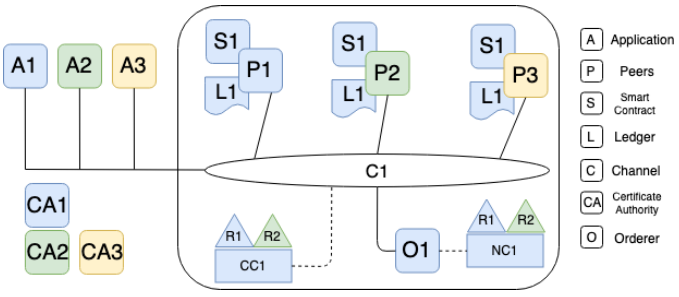

Figure 11: Hyperledger Fabric Architecture

We also prototyped the system with the permissioned Hyperledger fabric blockchain, IoT Sensors, and cloud server. Fig. 11 illustrates the setup of the Hyperledger fabric network with three organizations and one peer node each. The network also includes one orderer peer, three certificate authorities and one channel to log all the transactions. Table 3 summarizes the device types and their capabilities that were used to implement the Fabric network.

Table 3: Hyperledger Fabric Devices

	Peers Nodes	**Orderer**	**CA**
Devices	AWS EC2 t2.medium	Macbook Pro	AWS EC2 t2.small
# of nodes	3	1	3
CPU	2.3 GHz Intel Broadwell	2.9 GHz Intel Core i9	2.5 GHz Intel Xeon
RAM	4GB	32GB	1GB
OS	Ubuntu 18.04	Ubuntu 18.04	Ubuntu 18.04
Hyperledger	v1.4	v1.4	v1.2

8.1. Organizations

Hyperledger Fabric allows organizations to collaborate in the formation of the blockchain network. The proposed network consists of three organizations, one for each entity i.e. sensor owner, the third-party requester and the cloud storage. The organization is joined to a network by adding its Membership Service Provider (MSP) to the network. Organizations manage the Hyperledger Fabric network by establishing the policies that control the network

8.2. Peers

A peer node in the Hyperledger Fabric network is a fundamental element, that maintains a ledger and runs chain-code containers in order to perform read/write operations to the ledger. Each organization has one peer node acting as the gateway for performing transactions. Peers are owned and maintained by members of the organization. Peers can be created, started, stopped, reconfigured and even deleted. When a new sensor owner or third-party requester joins the network, they extend the network by adding their peers to the respective organizations. These peer nodes also expose a set of APIs that enable the applications to interact with applications.

8.3. Channels

A channel is the primary communication mechanism by which all the members of the network can communicate with each other. In the proposed system, there is only one channel and all the transactions between the members are recorded inside this channel. This channel is the marketplace for the IoT data. All the peer nodes and applications will automatically join this channel as default. Channels are very useful as they provide private communication between the members. The channels provide an efficient sharing of infrastructure while maintaining data and communication privacy.

8.4. Orderer

In Hyperledger Fabric there is a node called an orderer that does the transaction ordering. Along with the other ordering nodes, they form an ordering service. This ordering service is responsible for creating the blocks and adding them to the ledger. Any block generated by the ordering service is guaranteed to be final and correct. In our prototype, there is only one ordering node that validates the block and updates them in the ledger.

8.5. Smart contract

Smart contracts on the Hyperledger fabric were written in JavaScript using the Algorithms 1 and 2. Smart contracts are used to help generating transactions that can be subsequently distributed to every node in the network. After developing the contract, it was installed on the peer node that is connected to the channel. In order for other components of the network to interact with the smart contract, it needs to be instantiated on the channel. After instantiating the contract, it is logically hosted by the channel.

8.6. Certificate Authority

Hyperledger Fabric Certificate Authority (CA) issues Public Key Infrastructure (PKI) based certificates to the network member organizations and their users, who then use them to authenticate themselves in the messages they exchange with their environment. The blockchain network relies on these PKI standards to ensure secure communication between various participants. In our system, each organization has its own certificate authority to identify the users from the respective organizations and grant the rights over the blockchain network.

8.7. Application

We connected the client applications to the blockchain network through the channel. Just like the peers and orderer, the application will also have its identity that is associated with the organization. In our proposed architecture, we have three different applications associated with their organizations. The transaction done through these applications will be recorded on the channel and shared with other members of the network.

9. Performance analysis on Hyperledger Fabric

In this section, we describe the experiments to evaluate the proof of concept implementation on Hyperledger Fabric. We have performed multiple experiments, starting from the time taken for end-to-end process for proxy re-encryption and throughput of the network. Our Hyperledger Fabric test network consists of 3 organizations, each with one peering node as shown in the Figure 11. There is one channel where all the entities perform the transactions and also one solo ordering node for

the creation of the blocks. We performed multiple experiments to test the performance of Fabric with ”invoking” and ”querying” the custom smart contract written for proxy re-encryption application.

9.1. *Throughput*

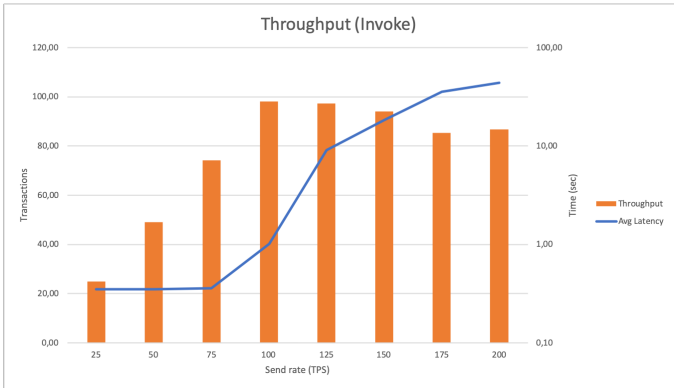

Figure 12: Throughput for invoking custom chaincode

In the first experiment, we measured the throughput of the architecture by submitting multiple transactions to the blockchain. These transactions were to invoke the smart contract function and write information to the blocks. The chaincode was written in Java using the Algorithm 1. We started the test with 25 transactions per second (TPS) for 60 seconds and increased the send rate to 200 tps.

With an increase in transaction arrival rate, the throughput for writing on blockchain increased linearly as expected until it flattened out at around 95 tps, the saturation point, as shown in Figure 12. When the arrival rate was close to or above the saturation point, the latency increased significantly (i.e., from an order of few ms to 50s)

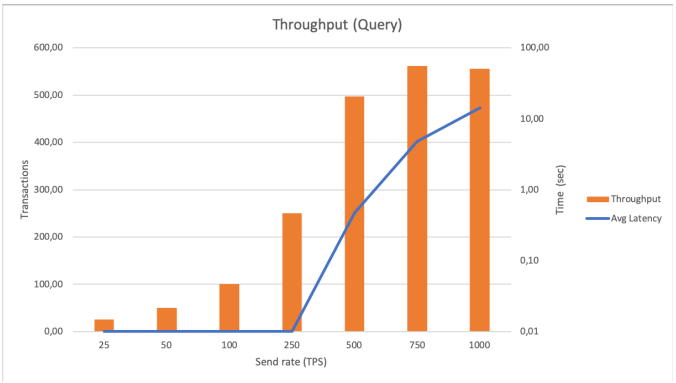

Figure 13: Throughput for querying custom chaincode

As shown in Figure 13, the throughput for querying the blockchain reached its saturation point around 550 transactions per second with latency increasing significantly with the increase in the send rate of transactions.

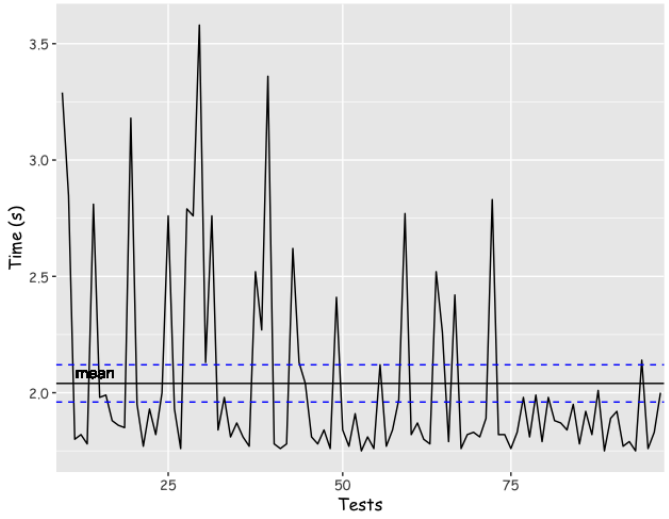

Figure 14: End-to-End delay in Hyperledger

9.2. *Impact of Proxy Re-encryption*

In the second experiment, we measured the time taken for the end-to-end process of the proxy re-encryption. We start measuring the time as soon the consumer requests for the data till the data is ready and the temporary address is shared with the consumer. We ran the experiment for 100 times before taking an average. As shown in the Figure 14 it takes on average 2.039 s for the end-to-end process with a confidence interval of 0.08 s. This time includes all the phases described in the Flowchart 8.

9.3. *Performance analysis of Orderer node*

In the third experiment, we measured the performance of the network by running the orderer node on different virtual machines with different resources allocated. Table 4 show the configuration of the orderer nodes.

Table 4: Orderer Node Configuration

Parameters	Configuration 1	Configuration 2	Configuration 3
# of CPU cores	4 Cores	8 Cores	12 Cores
RAM	12 GB	12 GB	12 GB
# of Channels	1	1	1
Database	GoLevelDB	GoLevelDB	GoLevelDB
Block Size	25 TPB	25 TPB	25 TPB

We performed each test 10 times and took an average. We increased the arrival rate of the transaction until the saturation point was reached to calculate the throughput. When the number of cores allocated to the orderer node increased, the throughput increased and the latency decreased. As shown in

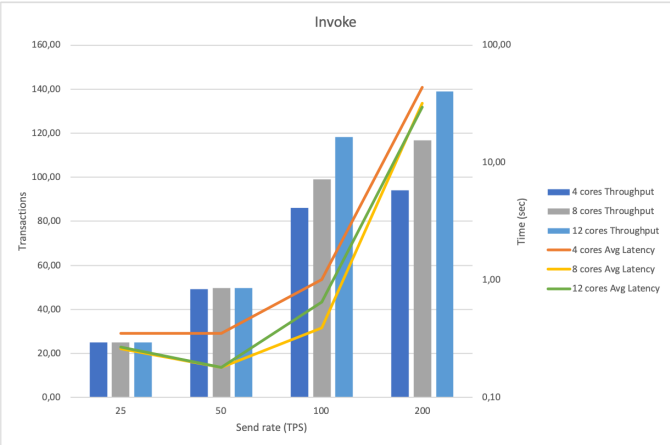

Figure 15: Throughput for invoking chaincode

Figure 15, increasing the cores to 12, increased the writing speed up to 50% and average latency also increased near the saturation point.

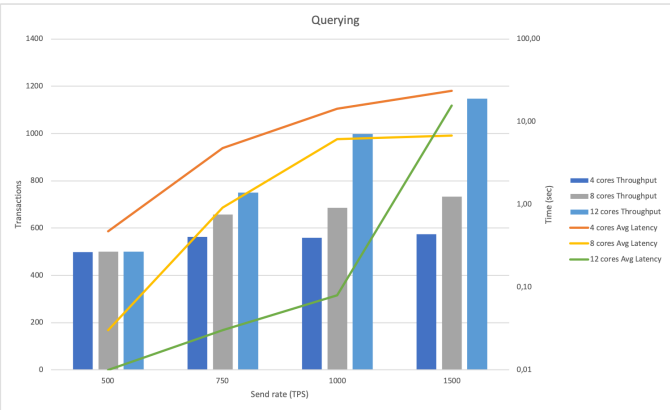

Figure 16: Throughput for querying chaincode

Figure 16 illustrates that throughput efficiency for reading data from the blockchain doubles from 550 TPS to 1150 TPS when the number of allocated cores increased. Configuration of the orderer nodes directly effects the throughput of the Fabric ledger.

10. Security Analysis

10.1. Security requirements

Let us first discuss if the required security features are obtained in the proposed scheme.

- Confidentiality: This feature is guaranteed by the fact that the data is sent in the encrypted format C_A to the server. Due to the ECDHP, without knowledge of the secret key of the sender, it is impossible to derive the original message M. After re-encryption, the message M can also due to the ECDHP only be found by the persons who are in possession of the private key of the receiver or the random value determined by the sender, which is constructed by means of the private key of the sender. Consequently, only the sender is able to derive the message before re-encryption, and both sender and receiver after re-encryption.

- Integrity: The integrity of the message C can be verified by both the sender (Decrypt1 phase) and the receiver (Decrypt2 phase) by verifying the last equality in the process.

- Authentication: Identity-based authentication is established in this scheme, thanks to the usage of the ECQV certificates. From the identity and the certificate, the corresponding unique public key can be derived, guaranteeing the link between identity and public key.

10.2. Possible attacks and their prevention

We now discuss how resistance is offered against the most well-known attacks.

- Man in the middle attacks and impersonation attacks. First of all, due to the existence of identity-based authentication in the scheme, no impersonation attacks are possible. In addition, when the re-encryption key is not correctly applied in the scheme, the integrity test in the Decrypt1 and Decrypt2 phase will fail.

- Replay attacks. These types of attacks are avoided as the timestamps are involved in the metadata, which plays an important role in deriving the corresponding session key.

11. Discussion

By now we have demonstrated the proposed design of a blockchain based proxy re-encryption scheme for secure IoT data sharing. Our proposed system benefits all the concerned entities without compromising the security. The sensor owner is motivated to join the system as they get a platform to securely sell their data guaranteeing the privacy of it. The sensor owner can make profit in a traditional way, while the data requester can easily get the required data. Miners in the system get an incentive by trading the generated token. The trading platform is able to avoid single points of failure thanks to this distributed, self-regulated blockchain for transaction processing. However, some aspects of the system design can be further improved.

11.1. Comparison of Ethereum with Hyperledger Fabric

There are still many challenges that lie ahead for the adoption of blockchain platforms. Performance is one of the biggest concerns in adopting blockchain platforms as it is necessary to provide a viable alternative to existing platforms. Even though throughput and latency of well-known blockchain platforms have previously been quantified, the scenarios when transactions are made by a large number of users have not yet been explicitly assessed and limitations of the platform have not been widely researched.

Two blockchain platforms are chosen for this paper, namely Ethereum and Hyperledger Fabric, due to their popularity and

potential in being developed to use in a wide variety of applications. Both of these platforms have a different mode of operation: permissionless and permissioned. Ethereum, a public blockchain, anybody is allowed to participate in the network and the consensus mechanism, while participation in Hyperledger Fabric is permissioned and selected participant is selected to access the network. As seen from the analysis, the mode of participation has a profound impact on how consensus is reached, and it directly affects the performance of the platform.

Another noteworthy difference is that Ethereum has a build-in cryptocurrency called Ether that is used to compensate the miners for consensus and also to pay transaction fees. Hyperledger Fabric does not require consensus through mining, so it does not have built-in currency but with fabric, it is possible to develop a digital token with chaincode.

The analysis in this paper shows that both Ethereum and Hyperledger Fabric are highly flexible and have a powerful smart contract engine that makes it a generic platform for a wide range of applications. Further, the modular architecture of Fabric allows it to be customized for a range of applications. However, Ethereum permissionless mode of operation and transparency comes at the cost of performance and scalability. Fabric solves these issues by permissioned mode and to be specific Byzantine Fault Tolerance (BFT) algorithm. The nature of Proof-of-Work mechanism of Ethereums consensus mechanism is much slower than the nature of BFT mechanism that is deployed in Hyperledger Fabric.

Table 5: Comparison between Ethereum and Hyperledger Fabric

	Ethereum	**Hyperledger Fabric**
Block Generation	15 seconds	2 seconds
Block Size	85 KB	256KB
Proxy Re-encryption	48.01 seconds	2.039 seconds
Throughput	20 TPS	95 TPS
Price Per Transaction	$0.030	$0

As seen from the result, the difference between both platforms becomes even more significant with a large number of transactions. A direct implication is that, for a given blockchain application, estimating the expected number of transactions will be very crucial in selecting suitable platforms as they can alter subsequent throughput, execution time, and latency.

11.2. Distributed cloud for scalability

In the current set-up, the storage infrastructure is organized in a very centralized way. Currently, in cloud computing, resources can be shared, and the end customer could have much lower costs and higher efficiency and uptime. However, data breaches or service interruptions in the big cloud are very common. These failures aren't a one-time occurrence and they show that the cloud computing model of centralized storage is not as secure as it could be because it has a single point of failure.

Blockchain greatest strength is its versatility. Blockchain creates a decentralized and distributed storage marketplace. Blockchain along with cloud storage makes data fully decentralized because it is stored on multiple nodes across the globe while blockchain can keep track of all the interactions. Distributed cloud storage breaks files into fragments after being encrypted and then intelligently distributes them across dozens of nodes.

11.3. Comparison with related work

Table 6 shows the features of our proposed architecture when compared with the existing solutions. It is clear from the table that our proposed architecture is a unique solution that uses blockchain in combination with cloud service providers for the securely storing and sharing of the sensor data. Moreover, a prototype implementation of the hybrid architecture with detailed performance analysis is provided to verify the viability of the proposal and demonstrate the performance metrics of the approach.

Table 6: Comparison of proposed solution with the existing works

	Ref. [32]	**Ref. [36]**	**Ref. [40]**	**Ref. [43]**	**Ref. [47]**	**Ref. [50]**	**Our Proposal**
IoT Security	✓	✓	✓	✓	-	✓	✓
Use of Smart Contract	✓	✓	✓	✓	-	-	✓
Data Trading	-	-	✓	✓	✓	-	✓
Cloud Architecture	-	✓	-	-	✓	✓	✓
Proxy Re-encryption	-	-	-	-	✓	✓	✓
Proof of Concept	✓	-	-	✓.	✓	-	✓
Public and Private Blockchain	-	-	-	-	-	-	✓

12. Conclusions

In this paper, we have proposed a blockchain based trading platform with the combination of a paring free proxy re-encryption scheme to ensure secure transfer of the sensor data to the user. We have also validated the proof of concept model on a public Ethereum testbed and private Hyperledger Fabric. We demonstrated the practicality of the system design using off-the-shelf laptops, raspberry pis and amazon managed

blockchain. Moreover, our experiments and analysis verify that combing proxy re-encryption scheme with blockchain enables a secure platform for trading and sharing of the sensor data. The use of blockchain does increase the delay but it keeps a record of all the interaction between the entities and eliminates a need for a trusted third party. Our framework provides an efficient, fast and secure platform for storing, trading and managing of sensor data.

In the future, we plan to address the issues and challenges identified. We plan to extend the proposed system with an implementation on a different blockchain platform e.g. R3 corda. We also plan to extend our architecture by adding a distributed cloud storage to make the system more scalable. In addition, we will try to increase the security of the cloud by adding a smart contract based Access Control List (ACL) and auditing system of the cloud server managed by the blockchain.

Acknowledgment

This research is funded by Academy of Finland under SECUREConnect, 6Genesis Flagship (grant 318927) projects and by European Union under RESPONSE 5G (Grant No: 789658), COST Action CA15127 (RECODIS) and CA16226 (SHELDON) projects.

References

[1] R. Taylor, D. Baron, D. Schmidt, The world in 2025-predictions for the next ten years, in: Microsystems, Packaging, Assembly and Circuits Technology Conference (IMPACT), 2015 10th International, IEEE, 2015, pp. 192–195.

[2] N. Suryadevara, S. Mukhopadhyay, Internet of things: A review and future perspective, Reliance (2018).

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: A survey on enabling technologies, protocols, and applications, IEEE Communications Surveys & Tutorials 17 (2015) 2347–2376.

[4] M. Liyanage, A. Braeken, P. Kumar, M. Ylianttila, IoT Security: Advances in Authentication, John Wiley & Sons, 2020.

[5] L. Hou, S. Zhao, X. Xiong, K. Zheng, P. Chatzimisios, M. S. Hossain, W. Xiang, Internet of things cloud: architecture and implementation, IEEE Communications Magazine 54 (2016) 32–39.

[6] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future generation computer systems 29 (2013) 1645–1660.

[7] E. Karafili, E. C. Lupu, Enabling data sharing in contextual environments: Policy representation and analysis, in: Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies, ACM, 2017, pp. 231–238.

[8] A. Reyna, C. Martín, J. Chen, E. Soler, M. Díaz, On blockchain and its integration with iot. challenges and opportunities, Future Generation Computer Systems 88 (2018) 173–190.

[9] "domraider whitepaper", Accessed: 29.04.2019. URL: https://s3-eu-west-1.amazonaws.com/domraider/domraider/DomRaider+ICO+Whitepaper+EN.pdf.

[10] "rinkeby testnet", Accessed: 05.09.2020. URL: https://www.rinkeby.io/.

[11] A. Panarello, N. Tapas, G. Merlino, F. Longo, A. Puliafito, Blockchain and iot integration: A systematic survey, Sensors 18 (2018) 2575.

[12] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang, An overview of blockchain technology: Architecture, consensus, and future trends, in: Big Data (BigData Congress), 2017 IEEE International Congress on, IEEE, 2017, pp. 557–564.

[13] V. Buterin, et al., A next-generation smart contract and decentralized application platform, white paper (2014).

[14] G. Wood, Ethereum: A secure decentralised generalised transaction ledger. byzantium version, Ethereum Project Yellow Paper (2018).

[15] "erc20 token standard", Accessed: 12.04.2019. URL: https://theethereum.wiki/w/index.php/ERC20_Token_Standard.

[16] Solidity, the contract-oriented programming language, Accessed: 27.09.2018. URL: https://github.com/ethereum/solidity.

[17] C. Cachin, Architecture of the hyperledger blockchain fabric, in: Workshop on distributed cryptocurrencies and consensus ledgers, volume 310, 2016.

[18] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, ACM, 2018, p. 30.

[19] H. Watanabe, S. Fujimura, A. Nakadaira, Y. Miyazaki, A. Akutsu, J. Kishigami, Blockchain contract: Securing a blockchain applied to smart contracts, in: Consumer Electronics (ICCE), 2016 IEEE International Conference on, IEEE, 2016, pp. 467–468.

[20] S. Ølnes, Beyond bitcoin enabling smart government using blockchain technology, in: International Conference on Electronic Government and the Information Systems Perspective, Springer, 2016, pp. 253–264.

[21] J.-C. Cheng, N.-Y. Lee, C. Chi, Y.-H. Chen, Blockchain and smart contract for digital certificate, in: 2018 IEEE International Conference on Applied System Invention (ICASI), IEEE, 2018, pp. 1046–1051.

[22] M. Lokhava, G. Losa, D. Mazières, G. Hoare, N. Barry, E. Gafni, J. Jove, R. Malinowsky, J. McCaleb, Fast and secure global payments with stellar, in: Proceedings of the 27th ACM Symposium on Operating Systems Principles, 2019, pp. 80–96.

[23] "iolite", Accessed: 05.09.2020. URL: https://icodrops.com/iolite/.

[24] "lisk", Accessed: 05.09.2020. URL: https://lisk.io/.

[25] M. Bartoletti, L. Pompianu, An empirical analysis of smart contracts: platforms, applications, and design patterns, in: International Conference on Financial Cryptography and Data Security, Springer, 2017, pp. 494–509.

[26] N. Koblitz, Elliptic curve cryptosystems, Mathematics of computation 48 (1987) 203–209.

[27] J. S. Kumar, D. R. Patel, A survey on internet of things: Security and privacy issues, International Journal of Computer Applications 90 (2014).

[28] Y. Yang, L. Wu, G. Yin, L. Li, H. Zhao, A survey on security and privacy issues in internet-of-things, IEEE Internet of Things Journal 4 (2017) 1250–1258.

[29] H. Suo, J. Wan, C. Zou, J. Liu, Security in the internet of things: a review, in: Computer Science and Electronics Engineering (ICCSEE), 2012 international conference on, volume 3, IEEE, 2012, pp. 648–651.

[30] M. A. Khan, K. Salah, Iot security: Review, blockchain solutions, and open challenges, Future Generation Computer Systems 82 (2018) 395–411.

[31] S. Notra, M. Siddiqi, H. H. Gharakheili, V. Sivaraman, R. Boreli, An experimental study of security and privacy risks with emerging household appliances, in: Communications and Network Security (CNS), 2014 IEEE Conference on, IEEE, 2014, pp. 79–84.

[32] S. Huh, S. Cho, S. Kim, Managing iot devices using blockchain platform, in: Advanced Communication Technology (ICACT), 2017 19th International Conference on, IEEE, 2017, pp. 464–467.

[33] M. Banerjee, J. Lee, K.-K. R. Choo, A blockchain future for internet of things security: a position paper, Digital Communications and Networks 4 (2018) 149–160.

[34] N. Kshetri, Can blockchain strengthen the internet of things?, IT Professional 19 (2017) 68–72.

[35] A. Dorri, S. S. Kanhere, R. Jurdak, P. Gauravaram, Blockchain for iot security and privacy: The case study of a smart home, in: Pervasive Computing and Communications Workshops (PerCom Workshops), 2017 IEEE International Conference on, IEEE, 2017, pp. 618–623.

[36] T. Hardjono, N. Smith, Cloud-based commissioning of constrained devices using permissioned blockchains, in: Proceedings of the 2nd ACM international workshop on IoT privacy, trust, and security, ACM, 2016, pp. 29–36.

[37] T. M. Fernández-Caramés, P. Fraga-Lamas, A review on the use of

blockchain for the internet of things, IEEE Access 6 (2018) 32979–33001.
[38] T.-D. Cao, T.-V. Pham, Q.-H. Vu, H.-L. Truong, D.-H. Le, S. Dustdar, Marsa: A marketplace for realtime human sensing data, ACM Transactions on Internet Technology (TOIT) 16 (2016) 16.
[39] "nokia sensing as a service", Accessed: 29.04.2019. URL: https://onestore.nokia.com/asset/201997.
[40] A. Sheikh, V. Kamuni, A. Urooj, S. Wagh, N. Singh, D. Patel, Secured energy trading using byzantine-based blockchain consensus, IEEE Access 8 (2019) 8554–8571.
[41] Z. Guan, X. Lu, N. Wang, J. Wu, X. Du, M. Guizani, Towards secure and efficient energy trading in iiot-enabled energy internet: A blockchain approach, Future Generation Computer Systems 110 (2020) 686–695.
[42] Y. Li, B. Hu, An iterative two-layer optimization charging and discharging trading scheme for electric vehicle using consortium blockchain, IEEE Transactions on Smart Grid 11 (2019) 2627–2637.
[43] Y.-N. Li, X. Feng, J. Xie, H. Feng, Z. Guan, Q. Wu, A decentralized and secure blockchain platform for open fair data trading, Concurrency and Computation: Practice and Experience 32 (2020) e5578.
[44] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, H. Jin, Sdte: A secure blockchain-based data trading ecosystem, IEEE Transactions on Information Forensics and Security 15 (2019) 725–737.
[45] M. Blaze, G. Bleumer, M. Strauss, Divertible protocols and atomic proxy cryptography, in: International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 1998, pp. 127–144.
[46] G. Ateniese, K. Fu, M. Green, S. Hohenberger, Improved proxy re-encryption schemes with applications to secure distributed storage, ACM Transactions on Information and System Security (TISSEC) 9 (2006) 1–30.
[47] L. Jiang, D. Guo, Dynamic encrypted data sharing scheme based on conditional proxy broadcast re-encryption for cloud storage, IEEE Access 5 (2017) 13336–13345.
[48] C.-K. Chu, J. Weng, S. S. Chow, J. Zhou, R. H. Deng, Conditional proxy broadcast re-encryption, in: Australasian Conference on Information Security and Privacy, Springer, 2009, pp. 327–342.
[49] M. Sun, C. Ge, L. Fang, J. Wang, A proxy broadcast re-encryption for cloud data sharing, Multimedia Tools and Applications 77 (2018) 10455–10469.
[50] P. Shabisha, A. Braeken, A. Touhafi, K. Steenhaut, Elliptic curve qu-vanstone based signcryption schemes with proxy re-encryption for secure cloud data storage, in: International Conference of Cloud Computing Technologies and Applications, Springer, 2017, pp. 1–18.
[51] S. Patonico, P. Shabisha, A. Braeken, A. Touhafi, K. Steenhaut, Elliptic curve-based proxy re-signcryption scheme for secure data storage on the cloud, Concurrency and Computation: Practice and Experience (????) e5657.
[52] M. Campagna, Sec 4: Elliptic curve qu-vanstone implicit certificate scheme (ecqv), institution content-type= institution> Certicom Res</institution>., Mississauga, ON, Canada, Tech. Rep (2013).
[53] Official go implementation of the ethereum protocol, Accessed: 27.09.2018. URL: https://github.com/ethereum/go-ethereum.
[54] Truffle, Accessed: 27.09.2018. URL: https://truffleframework.com/docs/truffle/overview.
[55] "eth price", Accessed: 29.04.2019. URL: https://ethgasstation.info/calculatorTxV.php.