Knirsch *et al. EURASIP Journal on Information Security* (2019) 2019:2
https://doi.org/10.1186/s13635-019-0085-3

EURASIP Journal on Information Security

RESEARCH Open Access

Implementing a blockchain from scratch: why, how, and what we learned

Fabian Knirsch, Andreas Unterweger* and Dominik Engel

Abstract

Blockchains are proposed for many application domains apart from financial transactions. While there are generic blockchains that can be molded for specific use cases, they often lack a lightweight and easy-to-customize implementation. In this paper, we introduce the core concepts of blockchain technology and investigate a real-world use case from the energy domain, where customers trade portions of their photovoltaic power plant via a blockchain. This does not only involve blockchain technology, but also requires user interaction. Therefore, a fully custom, private, and permissioned blockchain is implemented from scratch. We evaluate and motivate the need for blockchain technology within this use case, as well as the desired properties of the system. We then describe the implementation and the insights from our implementation in detail, serving as a guide for others and to show potential opportunities and pitfalls when implementing a blockchain from scratch.

Keywords: Blockchain, Private chain, Implementation, Practical insights

1 Introduction

After being originally proposed as a public, decentralized and trust-less ledger for digital currencies, blockchain technology has gained widespread adoption in many fields [1–3]. Recently, utilities and network operators in the energy domain have begun focusing on using the novel possibilities provided by blockchain technology for realizing such decentralized, trust-less applications that do not rely on a single trusted third party. This includes protocols for tariff matching [4, 5] and energy trading [6, 7]. Utility providers and network operators aim at exploring the properties and benefits, as well as the shortcomings of this technology for their respective use cases[1].

In this paper, a novel and previously unsolved real-world use case of two major Austrian utility companies is investigated: National legislature, which also applies to other European countries, requires that for a shared ownership of small photovoltaic power plants, such as those commonly found in multi-party rental apartments, customers can exchange portions of their energy production with neighbors. This allows customers to save money by shifting portions to other consumers when they need less energy themselves, e.g., when they are traveling.

For this use case, we evaluate the need for blockchain technology and the applicability of different types of consensus algorithms and permissions/visibilities. A number of existing implementations, such as MultiChain [8], OpenChain[2], Ethereum [9], and a Bitcoin fork [1], were initially taken into consideration. However, due to scalability issues on the desired hardware and for achieving a lightweight and simple solution, a blockchain has been implemented from scratch. Furthermore, this use case does not only involve blockchain technology, but also requires user interaction and customer acceptance, as well as the processing of privacy-sensitive data, all of which adds an additional layer of complexity.

This paper describes the real-world use case, the design, and the final implementation as well as the outcomes in detail. It aims at serving as a guide for others by showing potential opportunities and pitfalls when implementing a blockchain for a particular field of application other than financial transactions.

The rest of the paper is structured as follows: In Section 2, we give an introduction to blockchain technology. In Section 3, we describe the underlying use case and its legal requirements in detail, including related work on the subject. We further motivate the use of

*Correspondence: andreas.unterweger@en-trust.at
Salzburg University of Applied Sciences, Center for Secure Energy Informatics, Urstein Süd 1, 5412 Puch bei Hallein, Austria

Springer Open

© The Author(s). 2019 **Open Access** This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

blockchain technology for the proposed application. In Section 4, we discuss existing blockchain implementations and their drawbacks in our use case and continue to motivate and describe our custom implementation in detail in Section 5. Finally, in Section 6, we discuss the lessons learned from our implementation before concluding the paper.

2 Blockchain technology overview

This section provides an overview of blockchain technology. The scope of this section is to introduce and present a generic view of the technology and to go beyond its well-known use for financial transactions. First, blockchain technology is introduced as a decentralized, trustless, and immutable database. It is further discussed how a consensus is achieved within such a distributed network of nodes, and the most common consensus algorithm is briefly discussed. Finally, the drawbacks of blockchain technology are shown.

2.1 Blockchain technology

Blockchain technology can be described as a trustless and fully decentralized peer-to-peer data storage that is spread over all participants that are often referred to as *nodes*. The blockchain is designed to hold immutable information once data is committed to the chain, and it is therefore a decentralized, distributed, and immutable database in which data is logically structured as a sequence of smaller chunks (blocks). Each block $B_{i>0}$ is immutably connected to a single preceding block B_{i-1} through a cryptographic hash function $H(B_{i-1})$. Changes to B_{i-1} would yield an invalid hash in B_i and all following blocks. The very first block B_0 is called the *genesis block* and is the only block without a predecessor. In order to assure the integrity of a block and the data contained in it, respectively, the block is usually digitally signed.

For some applications, it is more useful to view a blockchain as a state machine [9]. Each block contains a new state with the very last block representing the current state. Given the list of blocks and the data in this block, there is a unique and immutable order of transitions that lead to the current state.

The main features of blockchain technology can be summarized as follows:

- *Decentralization:* Instead of relying on a single trusted entity, trust is spread across multiple or all participants, depending on the agreed-upon consensus algorithm [10]. This does not only mean that multiple copies of a data item are stored on all nodes, but also that the integrity of the data is governed by many decentralized parties.
- *Immutability*: Once data is committed to the blockchain and a sufficient number of participants have agreed on this state, the information is stored permanently and immutably. Changing the information contained in a particular block would require to also change all the following blocks up to the last block, which is considered to be infeasible [1, 11].
- *Scalability*: The block rate, comprised of the throughput and propagation time of information, depends on the consensus algorithm and the number of participants. This can be a limiting factor for applications that require high throughput [10]. Since all nodes hold a copy of the blockchain, scalability issues also arise in terms of the total amount of data that can be stored. Furthermore, in order to check the integrity of the blockchain, a new node needs to download a copy and validate the integrity of the entire chain. Note that more recent proposals for BFT-based consensus algorithms improved on this, e.g., [12].
- *Limited privacy*: All data in the blockchain is publicly visible to all participants. Private or permissioned blockchains limit the range of disclosure. However, they do not cryptographically protect the data. In order to achieve privacy, additional layers, such as zero-knowledge proofs [13] or a commitment scheme are required [14].

In the originally proposed Bitcoin protocol from [1], the blockchain is used to keep track of *coins*, i.e., a public list of financial transactions and how many coins are owned by each participant. For this purpose, each transaction contains sender and receiver information, as well as the number of coins to be transferred. A number of such transactions – once confirmed by the peers – become a new block. Such a block also includes the hash of the previous block and is appended to the chain. The transactions are therefore permanently linked to the series of previous transactions.

This list of chained blocks is public, kept by all members in the network, and can be verified by all participants by checking the integrity of the new block and the correct calculation of the hash. Participants in the network are identified by a private-public key pair, which is often referred to as the *ID* or *address*.

A blockchain can be generalized to store arbitrary data. In its simplest form, a block B_i consists of the following data:

- *Hash of previous block* h_i: A cryptographic hash of the previous block, i.e., $h_i := H(B_{i-1})$, as described above.
- *Payload* p_i: Arbitrary data that is stored in this block. In many practical applications, this data has to follow a predefined pattern (e.g., transactions in Bitcoin or operations in Ethereum).

- *Signature* s_i: A digital signature of the block data, i.e., $s_i = \sigma_{\text{sk}}(h_i|p_i)$, signed with the secret key sk of the creator of the block. This signature can be verified by the public key pk.

In public blockchains, all participants can create and append new blocks. Once a new block is created and successfully linked to the chain, it is broadcasted to the network. If other participants receive such a new block and consider it to be valid (i.e., by verifying the signature, checking the hash, and checking the validity of the payload), they extend their local copy of the chain with the newly created block and eventually broadcast the block to other participants. If a block is invalid, it is discarded and does not become part of the chain.

Blockchains therefore boil down to the question of how to achieve consensus in a distributed network with potentially faulty participants. This is referred to as *Byzantine Fault Tolerance* (BFT), originally introduced in [15], together with optimal algorithms for a variable number of adversaries, up to one third of the participants. This has been further investigated for asynchronous networks, such as blockchains, by many others, e.g., [11, 16]. It must be noted, however, that BFT algorithms for asynchronous networks are only practical up to about 1000 participants [17] due to the incurred overhead of the cryptographic algorithms.

In [1], a practical solution for achieving consensus in asynchronous networks of millions of participants and under the presence of Sybil attacks, where one attacker is in control of multiple nodes, is presented. Bitcoin requires synchrony among nodes to achieve consensus and uses the *Proof-of-Work* (PoW) algorithm. In order to consider a new block to be valid, the participant who initially provides this block has to prove that a significant amount of work has been spent for creating this block. The node therefore varies the input of a cryptographic hash function in order to get an output that has a certain pattern, e.g., a certain number of leading zeros. This becomes a computationally expensive problem by exploiting the preimage resistance of cryptographic hash functions [18]. Given such a hash function and its output $h = H(m)$, it is practically infeasible to find m for a given h. In order to incentivize nodes to verify transactions and append new blocks (and thus spend time computing time and energy), the effort is rewarded in the form of newly created coins, referred to as *mining reward*. The process of creating new blocks is therefore also referred to as *mining*.

In practice, branches may occur, where one or more nodes create new blocks at the same time. Commonly, the branch that contains more work, i.e., the longer branch, is considered to be the valid one. PoW therefore prevents malicious nodes from forging data or—in the case of crypto currencies—from spending the same coin twice, also referred to as the *double-spending attack*. In order to create a valid branch, a malicious subset of nodes must control at least 50% of the computing power in the network [1]. However, it is shown in [11] that there is a theoretical threshold of only 33% for specific attacks. The 50% threshold is improved by [19]. Therefore, a blockchain does not require a single trusted party, but instead is trustless if at least half of the computing power used for creating and verifying blocks is spent by honest peers.

Furthermore, peers and transactions are pseudonymous in the sense that the sender and the receiver are only identified by their addresses and that a new pair of keys (and therefore a new address) can be created for every transaction. Further advances in blockchain technology are described in Section 4.

2.2 Drawbacks

Despite the advantages of decentralization, trustlessness, and immutability, there are two major issues with current blockchain technology—scalability and power consumption [10]. Scalability refers to the time needed for propagating, processing, and validating transactions. The higher the number of nodes is, the more limiting network bandwidth, overall storage space, and power consumption become.

The current power consumption (as of May 2018) of the Bitcoin network is approximately 70 TWh per year[3]. This is mainly caused by the approximately 35 exahashes per second (3.5×10^{19} H/s) which need to be computed for the PoW. Thus, for energy-sensitive use cases, using Bitcoin in its current state is not a sustainable approach.

3 Use case description

In this section, we describe the use case for which we implemented our blockchain. We first describe the legal requirements which make a (technical) implementation of the use case necessary in the first place and provide a high-level description of the use case with all actors. Finally, we outline the reasons why a blockchain-based implementation is sensible and discuss related work.

3.1 Legal requirements

Solar energy plants allow generating electricity at customer premises. This additional energy can be consumed by the customer directly and thus reduces the household's total energy consumption. It also avoids feeding power into the public grid. If the amount of electricity generated exceeds the amount of electricity consumed, the excess energy has to be stored in (typically expensive) batteries or fed into the public grid, the reimbursements for which are often very small [20].

This problem exacerbates for multi-party homes with shared solar energy plants, where the excess energy

of one party could be "transferred" to another instead of feeding into the public grid [21], thereby reducing the energy costs for the receiving party. However, various legal aspects and technicalities have made such setups practically impossible [22] until recently [21, 23], despite the large number of multi-party apartment complexes and potential roof areas in urban regions [23].

In 2017, legislation to enable energy "transfers" between parties of a shared photovolatic power plant has been introduced in both Austria [24] and Germany [25]. In particular, § 16a ElWOG 2010 [26] as of 2017 [24, 27] as well as § 21, 21b and 21c EEG 2017 [25, 28, 29] require that the energy provider considers "transferred" portions of electricity generation for each customer at 15-min granularity. More precisely, the participants agree on a distribution key for the energy generated by the shared solar power plant for each 15-min time slot of each day, which in turn reduces each participant's energy consumption accordingly.

3.2 Conceptual overview

For the implemented use case, two categories of actors exist—utility providers and customers. A group of customers is assigned to a shared photovoltaic power plant. Usually, this power plant is installed at a multi-party apartment building and each customer receives a certain percentage of the generated power. Customers further receive power from the grid and are therefore assigned to a utility provider.

For this work, it is assumed that participants who invested in a shared solar energy plant agree on a default distribution for their photovoltaic power, e.g., customer A, 20%; customer B, 20%; and customer C, 60%. For each 15-min interval, customers can then transfer their contributions independently and as desired, e.g., customer A transfers the full 20% to customer B for the next 16 time slots (i.e., 4 h).

If customers do not shift their portions, the default distribution applies. After each day or 96 time slots[4], customers are billed based on their energy consumption from the grid, minus the power generation relative to the portions they were holding in each time slot. Note that customers cannot transfer more portions than they hold, but they can transfer portions they received. For instance, if, for one time slot, customer A transfers 20% to customer B (now holding 40%), customer B can— for this time slot—transfer this portion entirely to customer C (then holding 100%).

In summary, customers agree on a distribution of shares for each 15-min time slot. This information is forwarded to the utility provider that individually bills each customer based on their energy demand and supply for each time slot.

3.3 Use of blockchain technology

The reasons for building this use case atop a blockchain are manifold. First, a global state needs to be stored that all parties agree on. Second, customers do not fully trust each other and malicious customers may attempt to double-spend their portions in order to profit. Third, there is no single trusted third party due to the number of stakeholders involved. Therefore, a private and permissioned blockchain is implemented. In [30], the use case requirements for such a blockchain are defined as follows:

- *Need for storing a state*: For the desired use case, a global state representing the portions for each customer at each time slot needs to be stored.
- *Dealing with multiple writers*: Customers can transfer portions at any time, simultaneously.
- *Always-online trusted third party cannot be used*: Since each customer in the EU can choose their energy provider individually [31], no utility provider can be considered trusted by all participants in the general case. Furthermore, customers in different locations do not share the same network operator. Thus, no trusted third party can be appointed.
- *All writers are known*: Initial portions of photovoltaic power are assigned to customers. All customers trading in the network are therefore known and need to be verified for billing.
- *Not all writers are trusted*: Malicious users are assumed to maximize their shares, i.e., try to double-spend portions.
- *Public verifiability is not required*: The distribution of shares only needs to be verified by other participants within the network and the energy providers. The latter need to observe valid states, i.e., no more than 100% of the portions are assigned in total.

In our private and permissioned blockchain, a consortium of energy providers and network operators assigns permissions to new customers. These permissions allow customers to interact, i.e., read from and write to, the blockchain, similar to [32]. A revocation process ensures that inactive users no longer participate in the blockchain.

3.4 Related work

Due to the novelty of the legislature regulating the realization of our use case, related work is sparse. Apart from literature on energy trading in general, as outlined below, we are not aware of any publications which specifically describe blockchain-based implementations to record transferred photovoltaic energy portions as in our use case at the time of writing (May 2018).

Blockchain-based solutions to trade energy between households have been proposed. In [33], a Bitcoin-based application is described which allows for energy trading, but comes at the cost of requiring micropayments

for transactions as well as the general limitations of the Bitcoin network such as transaction speed. Even though they simulate solar energy production and support partial transfer of energy, the transaction costs of their protocol are impractically high for a use case such as ours.

In [34, 35], a custom cryptocurrency for local energy trading is proposed which changes its monetary value based on demand and supply. It allows "reserving" a certain amount of power for consumption at a future point in time at the current market value. Our use case does not require this feature since the prices are set by the energy provider and individual customers can only transfer portions, but not negotiate prices. Furthermore, the proposal of a custom cryptocurrency by itself is not sufficient to implement trading in practice, as opposed to our real-world implementation. Similarly, the concept presented in [22] proposes a private blockchain to handle local energy trading, but rather focuses on trading aspects and the market setup instead of presenting insights from an implementation.

Other blockchain-based approaches include [6, 7, 36]. While [36] focuses on machine-to-machine trading of energy in the context of the Chemical industry, [6] and [7] trade actual monetary units similar to [33]. In contrast, in our work, the blockchain is only used to represent portions, i.e., percentages of ownership for photovoltaic energy production for each time slot. Billing data is therefore never released publicly, but processed within the utility's premises—the individual customers receive their receipt along with the regular power bill.

Further related work on local energy trading without blockchains is presented, e.g., in [37, 38]. They assume trading processes via agents, but require a stock-market-like exchange. Such a centralized trusted party (at neighborhood- or provider-level) is not required in our use case due to the security guarantees of blockchains.

4 Available implementations

For implementing the private and permissioned blockchain, a number of available implementations were considered. The following sections describe the relevant implementations and our reasons for not choosing them. Table 1 summarizes the properties of these implementations, as well as of ours at the time of selecting basis implementations (September 2017).

4.1 Bitcoin

Bitcoin [1] and its client *Bitcoin Core*[5] are designed to handle financial transactions. Even though it is possible to include small amounts of meta data in each block, the limited available space as well as the low throughput and high delay of the Bitcoin network make it unsuitable for storing any significant amount of meta data [4].

A private Bitcoin network would be more suitable for our case—the meta data size limits as well as the throughput could be increased. This would also allow zero-cost transactions. However, the consensus algorithm of Bitcoin only deals with double-spending of Bitcoins, not portions of photovoltaic power. Modifying this algorithm would be possible, but the design of Bitcoin does not fit the use case well. For example, new Bitcoins are generated with new blocks, whereas, for our use case, balances for each time slot must be available regardless of the block rate.

Thus, we deemed the modifications to Bitcoin and its most popular implementation to be infeasible. The effort to modify the existing source could would very likely exceed the effort of implementing a blockchain from scratch. The same is true for ByzCoin [39], which uses a BFT-based consensus algorithm, but is otherwise comparable in design.

4.2 Ethereum

Ethereum [9] allows implementing *smart contracts* in various programming languages, e.g., Solidity[6]. Such smart contracts are capable of representing practically arbitrarily complex transactions. These are not limited to financial transactions as in Bitcoin, but allow for representing states and state changes as required by our use case.

Despite Ethereum's flexibility, previous research found that even smart contracts of modest complexity are relatively expensive [14]. Furthermore, frequent changes in the cost structure of executed operations as well as changes in the exchange rate to Euros[7] make costs unpredictable [40, 41].

Even though the cost issue could be eliminated by using a private blockchain, the complexity of Ethereum exceeds the requirements for our use case by far. Whenever a value needs to be stored in a smart contract, a modified Merkle Patricia Trie is updated, which is relatively time-consuming, as has been shown in [42]. In addition, it has been shown that this leads to a slow speed of execution when the volume of data increases, which is detrimental for our use case, especially from the users' perspective.

4.3 MultiChain

MultiChain [8] is a private and permissioned blockchain implementation. Like Bitcoin, it is primarily focused on financial transactions and thus the consensus algorithm would have to be modified significantly. Although this is likely easier than it is for Bitcoin, MultiChain's currently supported platforms are a major limitation for our use case.

At the time of writing (May 2018), MultiChain is only supported on 64-bit systems and has several related software dependencies[8]. Furthermore, the current documentation indicates that only x86-64 CPU architectures are supported explicitly, which excludes low-power and low-

Table 1 Comparison of available implementations and their properties *at the time of* selecting basis implementations (*September 2017*) and our work for comparison

Name	Consensus	Permissioned	Limitation
Bitcoin	PoW	–	Extent of modifications infeasible
Ethereum	PoW	–	Complexity exceeds requirements
MultiChain	PoW	√	Limited to high power consumption platforms
OpenChain	PoA	√	PoA algorithm not suitable for use case
Hyperledger Sawtooth	Dynamic	√	Not mature at time of evaluation
Hyperledger Fabric	Dynamic	√	Known security flaws
HAWK	PoW	–	Implementation not available
Corda	PoA	√	PoA algorithm not suitable for use case
Tendermint	BFT	√	Not available at time of evaluation
Stellar	BFT	–	Not mature at time of evaluation
EOS	BFT	√	Not mature at time of evaluation
NEO	BFT	–	Complexity exceeds requirements
OmniLedger	BFT	–	Not available at time of evaluation
ByzCoin	BFT	–	Extent of modifications infeasible
This work	PoW*	√	Requires tamper-proof hardware†

*See Section 6. †This is a requirement for electricity meters in the energy domain; see Section 5

costs hardware such as the Raspberry Pi, which is based on ARM.

Even though it is possible to emulate x86-64 instructions on a Raspberry Pi, the possibilities are limited and the overhead is significant. One of the most prominent options for emulation, *QEMU*[9], offers x86 support, but only limited x86-64 support[10]. Using an x86-64 platform is not an option due to the price—Up Squared[11], for example, which is one of the most similar to the Raspberry Pi, costs about six times as much[12] with a CPU power consumption that is more than ten times as high[13,14].

As power consumption and price are main criteria in our use case, we use Raspberry Pis without the overhead of simulating x86-64 architecture on it. Thus, we cannot use MultiChain to base our implementation on.

4.4 OpenChain

OpenChain[15] is a private blockchain designed to be efficient in terms of energy consumption, network communication, and block rate. It is therefore built on a client-server model rather than a peer-to-peer network and uses proof of authority instead of proof of work as a consensus algorithm. Since the objective of this work is to build a decentralized and trust-less model, a centralized approach such as implemented by the proof of authority consensus algorithm in OpenChain cannot be used. Nominating a designated authority that validates transactions would contradict the desired security and trust properties. The same holds for all blockchains which use Proof of Authority (PoA) to achieve consensus, e.g., Corda [32].

4.5 Other implementations

Hyperledger[16] allows operating private blockchains. However, at the time of selecting implementations (September 2017), Hyperledger Sawtooth[17] had not yet reached version 1.0 and uses a storage structure similar to the one of Ethereum. As described in the Section 4.2, this structure is not suitable for our use case due to processing time. Hyperledger Fabric has not been chosen since it is insecure in case single nodes act maliciously [43].

HAWK [44] is a privacy-preserving blockchain. However, at the time of writing (May 2018), no implementation is available[18].

In addition, there are several blockchain implementations that use BFT as a consensus algorithm, e.g., Tendermint[19], Stellar [45], EOS[20] and OmniLedger [46]. However, at the time of selecting implementations (September 2017), the aforementioned solutions were either not available (Tendermint, OmniLedger) or had not yet released a stable version (Stellar and EOS). In contrast, the BFT-based NEO blockchain[21] can be considered stable enough, but its complexity, which allows to execute smart contracts similar to Ethereum, exhibits the same issues as the latter (see above).

Due to the specific requirements of our use case and in order to get insights into the full cycle of blockchain development and implementation, we decided to build our own custom blockchain tailored to the particular legal requirements and needs for sharing percentages of solar power plants.

5 Implementation

In this section, we describe our implementation. First, we give an overview of our system. Second, we go into detail for each of the three core components—the nodes, the clearing server, and the smartphone app. Finally, we discuss our network setup which guarantees that only qualified participants can actually interact with the rest of the network.

The implemented solution is currently in use in two locations operated by different energy providers and network operators—the Austrian Smart Grid Model region Köstendorf in Salzburg[22] and Böheimkirchen in Lower Austria[23].

5.1 Overview

Figure 1 shows our system and its main components. Each participant has a node installed in their home which runs the blockchain node software as well as a portable app which allows sending commands to the node, in particular to send and receive portions of solar energy. All nodes are interconnected via the Internet, albeit within a virtual private network (VPN), so that they can communicate with one another despite not using public IP addresses. At each utility's premises, there is a clearing server installed. It is connected to a node listening on the blockchain and reading the portions of the utility's customers in order to calculate the net energy consumption of each participant.

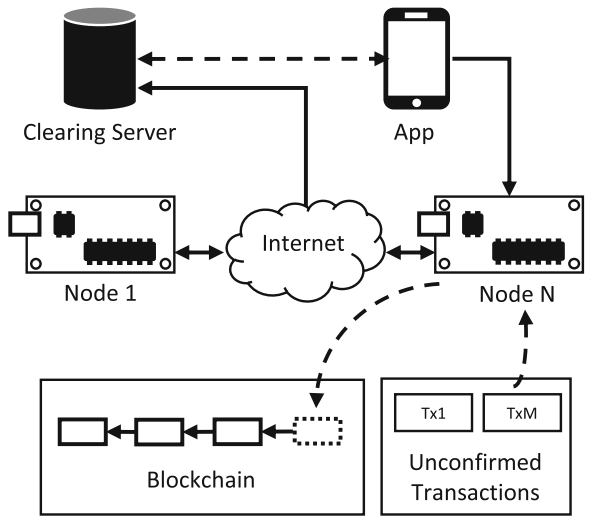

Fig. 1 Overview of the setup, including nodes, the clearing server, the app, and a conceptual view of the blockchain. Each customer possesses an app that is connected to a node (for sending new transactions and receiving confirmations) and to the clearing server (for billing). New transactions (denoted as Tx) reside in a pool of unconfirmed transactions until they are mined by a node and appended to the blockchain. The clearing server retrieves data from the blockchain and sends billing information to the customers' smartphone apps

This information is also fed back to each user through the app.

The following sections describe the design of the nodes and the network. While the interface of the blockchain is described, the details of both the clearing server and the app are not within the scope of this paper. Thus, they are only described briefly.

5.2 Nodes

The nodes are implemented in Java 8 and designed to run on Raspberry Pi 2 Model B. This hardware was chosen due to its general availability, ease of use, and little energy consumption, which closely resembles the hardware capabilities of a smart meter [47]. The device will be sealed and installed in the customer premises, as is legally required for electricity meters [48]. For the prototypical implementation of this use case, there are no specific performance requirements, but cost and proximity to legal requirements is of the essence. Given the relatively low retail prices[24] and the availability of well-tested software[25], the Raspberry Pi is a reasonable choice as a platform.

Figure 2 illustrates the fundamental architecture of our node implementation. The internal state of a node is represented by two main data structures—a tree and a state table. The tree stores a representation of the blockchain as currently seen by the node, and the state table is a tabular view of time slots and the corresponding portions for each customer. The latter is extracted from incoming valid transactions. On startup, the node reads the last persisted state from disk and initializes the internal tree data structure, which represents the blockchain, as well as the state table and connected auxiliary data structures. After that, it starts the watchdog for the server threads to listen for incoming messages from other known participants.

It also starts the mining thread which processes unconfirmed transactions and combines them to blocks which are appended to the tree data structure. The list of unconfirmed transactions as well as other internal auxiliary data structures is not depicted in Fig. 2 for clarity.

The mining thread is an infinite loop that waits for unconfirmed transactions, checks their integrity and validity, and then creates a new block. A block is considered valid if the numeric interpretation of its cryptographic hash (SHA-256) is below a predefined threshold. This is a slightly adapted version of the proof of work consensus algorithm proposed in [1]. In our use case, every node is a sealed hardware device provided by the network operator upon registration in the network. In this realization of the PoW, every node has the same hash rate. Upgrading to more powerful mining devices is not possible, which makes the 51% attack hard, since a collusion with half of the network is unlikely in practice.

When mining succeeds, the mined block is distributed to other known nodes through client threads. Similar to

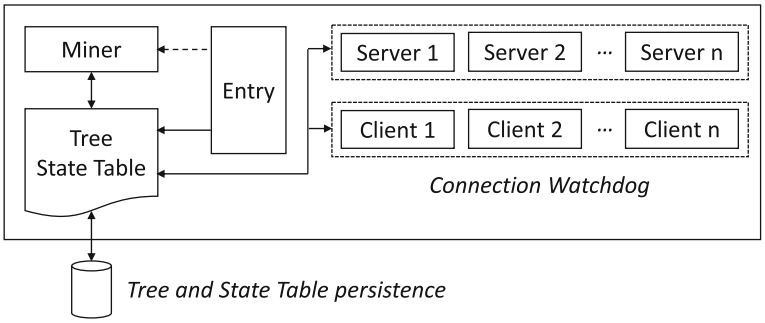

Fig. 2 Program overview: The mining thread as well as up to *n* server threads and up to *n* client threads operate on the core tree data structure representing the blockchain and the state table representing the assignments of portions. The state of the data structure as well as auxiliary representations like time-slot-accurate portion assignments are persisted to disk. Server and client threads include watchdogs to detect and potentially resume lost connections to other nodes

the server threads, there is one thread per peer, which avoids delays due to permanent or temporary unreachability of other nodes. Watchdogs make sure that connection timeouts or remote failures lead to a re-initialization of the corresponding connection. Note that there is no mining reward as each node's incentive to spend resources is to operate a trustless network without a centralized party such as the energy provider.

The central tree data structure representing the blockchain is read from and written to by all threads concurrently. This requires the use of concurrent programming paradigms [49], which are simplified by Java 8's concurrent data structures for the auxiliary lists and hash maps as well as atomic primitive types like counters[26]. Under the absence of a concurrent tree class, an append-only blockchain is implemented as a custom tree data structure. The state of the tree is persisted using Java's serialization API[27], and the state table is stored in an SQLite[28] database. Keeping a persistent tree reduces runtime memory usage by having only a small portion of the tree in the internal memory and by loading branches and blocks from the disk only if needed.

In order to achieve a consensus with all nodes, blocks and transactions need to be valid before actually being stored in the tree. Both incoming blocks from other nodes and blocks for mining undergo a conformity check before being appended and broadcasted, respectively. This is a two-stage check for (i) block integrity and (ii) transaction validity. For checking the block integrity, the signature of the block is checked and the hash of the block is recomputed. Then it is verified whether the block hash satisfies the currently active difficulty level and whether the block is appendable to the current tree. Checking the transaction validity is a more complex process that works as follows:

1 Append the block temporarily to the tree and check whether this block changes the longest chain.
2 If the longest chain has changed, find the first common parent node for the branch of the previous longest chain and the current longest chain; otherwise proceed with step 5.
3 Temporarily revert all transactions (i.e., remove them from the state table) from the previous branch up to the common ancestor (parent) node.
4 Process all transactions from the current longest branch starting from the common ancestor node.
5 If a single transaction in the newly added block is invalid (e.g., due to double spending or unknown recipients), the whole process is reverted and the new block is discarded.

After a block has been appended successfully (or discarded in case of an invalid block), the longest chain and the state table represent the current state of the system, i.e., all portions of all participants for all possible time slots. For convenience, this information can be queried by others, e.g., the smartphone app and the clearing server through the node's API.

5.3 Clearing server

Billing is one off-chain by a clearing server. One instance of this server is operated by each utility provider. In our use case, there are currently two clearing servers active. Note that, while the clearing server is an essential component for this use case, it is not required for running the blockchain itself. It guarantees that the billing process of each utility is handled off-chain and by each utility individually.

To feed the billing process, the clearing server uses the node's API to request the currently known distribution of portions. The portions held by each customer for each time slot are then used for billing. Therefore, the customers' energy consumption and energy production (as read from the smart meters) is mapped to their respective portions. The resulting value in kWh and monetary units

is displayed to the customer in the app. The details of the mapping, the meter readings, and the internal billing are out of scope for this paper.

5.4 Smartphone App

For user interaction, a smart phone app is provided to the customers. The app focuses on a high degree of usability and on being an abstraction layer between the underlying complexity of the blockchain and the high-level customer interaction. The app is connected to a node and the clearing server of the customer's utility provider. The connection to the node is used for sending new transactions to the network and for receiving information about incoming portions. The app therefore requests a list of blocks from the longest chain and filters the contained transactions for those related to the respective customer. These transactions are visualized so that the customers can see for each point in time how many portions they hold.

5.5 Network setup

Nodes communicate with each other, the app and the clearing server over TCP/IP and XML messages on the application layer. Using a text-based protocol allows for easy debugging and an illustration of messages which is suitable also for non-technical people. This is an important aspect of this work, where the abilities of blockchain technology are observed in a practical setting.

The node's API defines four main types of messages: (i) Block, (ii) Transaction, (iii) Utility Transaction, and (iv) Status. Block messages can either propagate a newly mined block or be the response of a block request from another node. Transaction messages are sent by the smartphone app and initiate a new shift of portions. Utility transaction messages are sent by dedicated clients, e.g., the clearing servers of each utility that have the permission to setup and create new photovoltaic power plants and assign customers and default distributions to these power plants. Status messages are used for displaying the node status to customers as well as for debugging purposes. Internally, the XML for blocks and transactions is serialized from and deserialized to Java objects.

All communication links between nodes as well as between the node and the app are secured with TLS v1.2 using a hybrid encryption scheme based on Elliptic Curve Diffie-Hellman key exchange and AES-256 with CBC [50]. In order to reduce the overhead for communication, there is no authentication on the TLS layer. However, this does not weaken the security since all messages (most importantly blocks and transactions) are signed by their respective sender. In a permissioned blockchain, such as ours, the public keys of all nodes are known to one another and messages with invalid signatures are discarded.

The communication link between the clearing server and the app uses out-of-the-box HTTPS with authentication and encryption. All components of our system—the nodes, the clearing servers, and the apps–communicate over a virtual private network (VPN). Since the nodes are placed on customer premises, where unchanging public IP addresses are not guaranteed and network address translation (NAT) is common, the VPN simplifies communication on the network layer.

6 Outcomes

In this section, we outline what we have learned from implementing our own blockchain as described in the previous section. We focus on the main take-aways which are noteworthy due to their unexpectedness and/or their value for others who plan on implementing their own blockchain.

6.1 Scalability can be improved with BFT

After the real-world testing of the implemented blockchain had started in June 2018, we learned that there is a more viable alternative to the implemented PoW algorithm for a small number of users. Since software in the energy domain that runs in the customers' premises is required to be certified and usually operates on sealed hardware, an in-place upgrade would have been neither feasible nor practical.

Due to the private and permissioned nature of the blockchain, in the implementation of the nodes (see Section 5), mining is used to prevent Sybil attacks, which would give the attacker an unfair advantage. In addition to the permissioned nature, in our setup, all devices have the same computing power and need to be tamper-proof. Thus, such an attack is not feasible. Consequently and due to the small number of nodes, PoW is not needed and could be replaced by more lightweight approaches, such as BFT consensus algorithms.

BFT consensus algorithms are a better choice for networks with about 1000 nodes or less [17, 39]. In our test case, there are less than 1000 users and thus less than 1000 nodes. However, since our implementation is designed as a prototype for a more general solution, houses with significantly more than 1000 parties must be considered. For example, the *Autobahnüberbauung Schlangenbader Straße* in Berlin, Germany, has more than 2000 apartments [29], where BFT would likely be infeasible.

After releasing our implementation for real-world testing in June 2018, there appeared new works that resolve the aforementioned limitations of BFT and provide better scalability through sharding, e.g., [46].

6.2 TLS handshakes introduce significant delays

In our first prototype, we used unencrypted connections to test the basic communication functionality of our net-

work. When we switched to encrypted connections with TLS [51], we noticed significant delays despite the small amount of (test) data to be encrypted. The issue practically disappeared when the connections were kept open. This indicated that the initial connection between two nodes, i.e., the TLS handshake, was taking longer than expected.

We decided to measure this for our setup, i.e., two Raspberry Pis 2 Model B with Raspbian establishing Java-based `SSLSocket` connections with TLS v1.2. We use a local Gigabit Ethernet network with a Netgear GS108 gigabit switch between the nodes to keep the propagation delay low. To measure the handshake delay, we open a connection from one node to another and close it immediately after successfully connecting. This is performed for both regular sockets and TLS sockets. We use the built-in `System.nanoTime()` command to measure the execution time on the sending node (i.e., the one that opens a connection) and repeat the experiment 1000 times.

Figure 3 shows the connection times in milliseconds for regular sockets and TLS-based sockets. It is clear that the handshake, in particular the key exchange, requires about as much time as the unencrypted connection does in total. This makes the delay of a TLS-based socket double that of a regular socket. In our use case, this is still acceptable, but needs to be taken into consideration during debugging when configuring timeouts.

6.3 AES-256 is disabled in Java 8 by default

Due to export restrictions[30], Java 8 imposes a limit on the cryptographic key sizes by default. For example, AES is limited to 128 bits[31], meaning that AES-256 is not available. To work around this limitation, strong cryptography has to be enabled explicitly.

The official solution proposed by Oracle is to install the *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8* by overwriting two policy *jar* files[32]. For a system-wide Java Runtime Environment (JRE) installation like in our case, this requires *root* privileges which make deployment more difficult and introduce an error source for JRE updates and system resets—when the policy files are not overwritten, requesting AES-256 encryption will fail.

A more flexible, albeit unofficial solution is to remove the restrictions on strong cryptography via reflection[33]. This does not require *root* privileges and can be performed during startup of the node implementation. However, JRE updates might break the functionality as the internals might change between JRE versions without prior notice[34].

Circumventing cryptographic restrictions to achieve an adequate level of security is undesirable. However, from a practical point of view, it is more convenient than to re-implement the nodes in another runtime environment or programming language. Nonetheless, we would like to point out that Java might not be the best choice for implementing blockchains with strong cryptography. Assuring

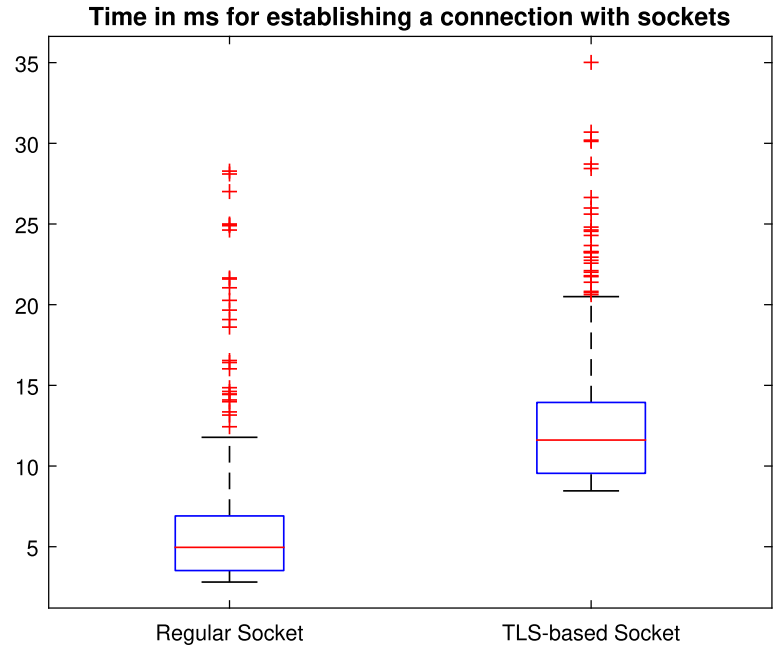

Fig. 3 Connection times in milliseconds (ms) for regular sockets and TLS-based sockets

the latter on all devices in the network is hard and error-prone, since fallbacks to AES-128 or other algorithms might significantly weaken the security of node-to-node communication.

6.4 One communication thread is not enough

When we initially designed the peer-to-peer communication for our node implementation, we dedicated one thread to handle all the messages to be sent to the node's peers. This means that one thread with a message queue would process the messages to be sent one by one and other threads, e.g., the miner, would place the messages to be sent in this queue. In a small and stable network, this works fine. However, when connections are lost or the number of nodes is larger, communication can slow down significantly or even come to a halt.

When a connection is lost, the timeout can take several seconds to be noticed by the transport layer. In the meantime, the communication thread waits for a reply or a timeout and is not able to process other messages from its queue. This leads to the remaining nodes not receiving further messages until the timeout is noticed. Communication is slowed down significantly if timeouts occur often, e.g., when a node has an unstable network connection or is offline.

Similarly, when the number of nodes increases in a stable network, the message queue fills up because each message transmission is synchronous and the time required to send messages exceeds the time it takes to mine a new block.

Note that a single-threaded messaging implementation is also a security liability. An attacker could deliberately cause connection timeouts and pre-mine blocks which deliberately exclude certain transaction. Having the longest chain and sustaining it by further deliberate timeouts can postpone valid unconfirmed transactions indefinitely.

To avoid these issues, we implemented one communication thread per peer, each with its own message queue, as outlined in the previous section. Since timeouts on one connection do not influence the other threads, the latter can still make progress independently. However, having one thread per peer increases runtime overhead in terms of memory consumption and scheduling. In our use case, the number of peers is small enough so that this is not an issue. In larger networks, however, Raspberry Pis might not be sufficient to handle the large number of communication threads.

Bitcoin relays transactions to all of its peers, the list of which is changing through a discovery process and is limited to 8 peers per node [52]. This number is similar to the total number of clients in our network. Ethereum follows an approach similar to Kademlia[35] for peer discovery and each node has a configurable maximum number of peers.

6.5 Resynchronization is hard

Sometimes nodes get out of synchronization due to temporary network failures or due to a huge number of concurrent messages. Incoming blocks are then not appendable to the chain since one or more blocks between the last appended block and the incoming blocks are missing. In this case, a backup algorithm is used that eventually triggers a resynchronization with neighboring nodes.

The resynchronization algorithm for our blockchain implementation works as follows: Once a node receives a block that cannot be appended due to missing blocks in between, it queries the sender of that block for the missing blocks. Additionally, the hash of the last known block of the unsynchronized node is submitted in order to allow the receiver to check whether it can serve the request, i.e., sender and receiver are on the same longest branch. If the requested node does not reply (or cannot reply), the block is discarded and the node proceeds with normal operation. Similarly, if the requested node replies with invalid or unappendable blocks, synchronization with this node is aborted to prevent the node from being locked by a malicious actor.

All these actions are performed synchronously and in a blocking function. This assures that the node is always in a defined state while synchronizing and that threads are not blocked by incoming transactions or not actively requested incoming blocks. Note that our design of separate client and server threads (see above) allows for concurrent operations and communication with all remaining nodes.

Initially, we designed an algorithm that picks a random node and requests the missing blocks for synchronization. This was handled completely asynchronously, i.e., after sending the request, the other node sent blocks and they were treated just as if they were normal incoming blocks broadcasted by this node. If the list of unappendable blocks was still growing, an exponential backup was triggered that increased the window of requested blocks. This allowed to reuse already included features for resynchronization and did not pose much overhead to the requesting node.

While this simple algorithm is sufficient for small networks and low block rates, it fails if blocks are generated at a higher frequency. In this case, nodes get out of synchronization more likely and multiple nodes that are not synchronized themselves try to find a new consensus with each other. This additionally floods the network with blocks and causes even more unsychronized states.

The source code of our implemented blockchain will be made available for research purposes.

6.6 Idle chains are a security liability

Blockchains like Bitcoin and Ethereum see a constant stream of new transactions to fill up their blocks[36],

with an adaptive mining difficulty to keep the time between blocks roughly constant [1, 9]. In our use case, however, it is expected that there are periods of decreased or even no activity. On the one hand, all users are in the same time zone, meaning that periods of nightly inactivity can be expected. On the other hand, during winter or when energy generation is likely to be insignificant, there are no portions transferred as there would be no financial gain for either party. Thus, it is possible that there may be no unconfirmed transactions for several hours or even days within the network.

This could be used by an attacker to pre-mine a longer chain than the rest of the network during the idle time and subsequently deliberately exclude certain transactions from the longest chain. This issue has been addressed in related work on proof-of-work security from the pre-blockchain-era [53] as well as in [54] for a protocol similar to Bitcoin and in [55] for Bitcoin itself. Although Bitcoin and Ethereum assume that there is always a significantly high number of transactions to keep the network active, they both allow for mining empty blocks [54][37].

Our proposed solution is to mine (empty) blocks even if there are no transactions. There are sources, e.g., the Wiki of libbitcoin[38], which mention the existence of empty blocks, but not explicitly as a way to circumvent the security issues of an idle blockchain. In a use case like ours, where transaction frequency can be too low for a difficulty adjustment to be effective, mining empty blocks might be the only viable solution to prevent pre-mining attacks.

6.7 New insights would lead to a different approach

Given the outcomes mentioned above and the rapid development of blockchain technology, a re-implementation at the time of writing (May 2018) would lead to a different choice for the consensus algorithm. Recently published work, which was not available at the time of selecting basis implementations, e.g., [46], shows that scalability for BFT algorithms can be significantly improved and allows for settings with more than 1000 nodes. This achieved by sharded blockchain, where not every node has to store all blocks. A re-evaluation of consensus algorithms and available implementations would thus likely lead to a BFT-based approach instead of PoW.

In addition, the choice for BFT is justified by the fact that PoW is used to prevent Sybil attacks. In a setup as the one in our use case, where participants have no incentive to act dishonest or sealed hardware does not allow to change the algorithm or upgrade the computing power, a computationally less expensive consensus algorithm is sufficient. This would reduce power consumption and scale well nonetheless.

7 Conclusion

We motivated and presented our blockchain implementation for fulfilling the recent legal requirements for the transfer of energy from shared photovoltaic power plants. We described our architecture as well as selected details of our implemented system. In our analysis, we elaborated on six take-aways blockchain implementers should be aware of th following: (i) scalability can be improved with Byzantinum Fault Tolerance algorithms when the number of users is small; (ii) TLS handshakes for secure communication links introduce additional delays of approx. 6.6 milliseconds; (iii) AES-256 is disabled in Java 8 by default due to export regulations, which requires to remove the restrictions via reflection; (iv) one communication thread is not enough for asynchronous peer-to-peer communication; (v) resynchronization of the node's states is hard, especially for small networks and high block rates; (vi) idle chains are a security liability, which can be mitigated by mining empty blocks; and (vii) new insights would lead to a different implementation. In summary, implementing a custom, private permissioned blockchain from scratch is hard, but viable for use cases like ours.

Endnotes

[1] http://energyweb.org/news/ewf-february-5-2018/

[2] https://www.openchain.org/

[3] https://digiconomist.net/bitcoin-energy-consumption

[4] Note that, due to daylight saving time, there are 2 days in each year with 92 and 100 time slots, respectively.

[5] https://bitcoin.org/en/download

[6] http://solidity.readthedocs.io/en/develop/

[7] The same is true for US Dollars and other currencies.

[8] https://github.com/MultiChain/multichain

[9] https://www.qemu.org/

[10] https://qemu.weilnetz.de/doc/qemu-doc.html#x86

[11] http://www.up-board.org/upsquared/

[12] http://www.up-board.org/wp-content/uploads/2017/11/UP-Square-DatasheetV0.5.pdf

[13] https://ark.intel.com/products/95598/Intel-Celeron-Processor-N3350-2M-Cache-up-to-2_4-GHz

[14] https://developer.arm.com/products/processors/cortex-a/cortex-a7

[15] https://www.openchain.org/

[16] https://www.hyperledger.org/

[17] https://github.com/hyperledger/sawtooth-core

[18] http://oblivm.com/hawk/download.html

[19] https://tendermint.com/

[20] https://eos.io/

[21] http://docs.neo.org/en-us/whitepaper.html
[22] http://www.smartgridssalzburg.at/content/website_smartgrids/de_at/modellregion-salzburg.html
[23] https://futurezone.at/b2b/verbund-und-salzburg-ag-starten-blockchain-pilotprojekte/297.880.235
[24] https://www.raspberrypi.org/blog/raspberry-pi-2-on-sale/
[25] https://www.raspberrypi.org/blog/raspberry-pi-3-on-sale/
[26] https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/package-summary.html
[27] https://docs.oracle.com/javase/8/docs/api/java/io/Serializable.html
[28] https://www.sqlite.org/
[29] https://ugk-berlin.de/projects/schlangenbader-strasse-ueberbauung-der-stadtautobahn/
[30] https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html#AppC
[31] https://docs.oracle.com/javase/8/docs/technotes/guides/security/SunProviders.html#importlimits
[32] http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html
[33] https://stackoverflow.com/questions/1179672/how-to-avoid-installing-unlimited-strength-jce-policy-files-when-deploying-an/44056166#44056166
[34] https://stackoverflow.com/a/38625821/4568958
[35] https://github.com/ethereum/devp2p/blob/master/rlpx.md
[36] https://blockchain.info/
[37] https://github.com/ethereum/EIPs/issues/186
[38] https://github.com/libbitcoin/libbitcoin/wiki/Empty-Block-Fallacy

Acknowledgements
The financial support by the Federal State of Salzburg is gratefully acknowledged. Funding by the Austrian Research Promotion Agency (FFG) under project number 865082 (ProChain) is gratefully acknowledged. Additional funding as well as input and feedback by our company partners Salzburg AG (Georg Baumgartner), Salzburg Netz GmbH (Christoph Groiß) and Verbund AG (Werner Eder, Michael Schramel, Peter Poier, and Clemens Theuermann-Bernhardt) is gratefully acknowledged. The authors would like to thank the reviewers for their valuable comments which helped to significantly improve this paper.

Funding
Funding information is provided in the "Acknowledgements" section.

Availability of data and materials
Not applicable.

Authors' contributions
This paper was written by FK (45%), AU (45%), and DE (10%). The detailed contributions are as follows: The idea for the paper and the blockchain concept were developed by AU (40%), FK (40%), and DE (20%). The Abstract was written by AU (50%) and FK (50%). Section 1 was written by AU (50%) and FK (50%). Section 2 was written by AU (20%) and FK (80%). Section 3 was written by AU (50%) and FK (50%). Section 4 was written by AU (60%) and FK (40%). Section 5 was written by AU (40%) and FK (60%). Section 6 was written by AU (60%) and FK (40%). The conclusion was written by AU (50%) and FK (50%). The legal background research was done by AU (100%). Editorial work was done by DE (100%). All authors read and approved the final manuscript.

Authors' information
The contact information of the first author is available on the very first page of this document.

Competing interests
The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 7 June 2018 Accepted: 13 February 2019
Published online: 11 March 2019

References

1. S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System. Technical report (2008). https://bitcoin.org/bitcoin.pdf
2. F. Tschorsch, B. Scheuermann, Bitcoin and beyond: a technical survey on decentralized digital currencies. IEEE Commun. Surv. Tutor. **18**(3), 2084–2123 (2016)
3. K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the Internet of Things. IEEE Access. **4**, 2292–2303 (2016)
4. F. Knirsch, A. Unterweger, G. Eibl, D. Engel, in *Sustainable Cloud and Energy Services: Principles and Practices. Chap. 4*, ed. by W. Rivera. Privacy-Preserving Smart Grid Tariff Decisions with Blockchain-Based Smart Contracts (Springer, Cham, 2017), pp. 85–116
5. F. Knirsch, A. Unterweger, D. Engel, Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions. J. Comput. Sci. Res. Dev. (CSRD). **33**(1), 71–79 (2018)
6. E. Munsing, J. Mather, S. Moura, in *2017 IEEE Conference on Control Technology and Applications (CCTA)*. Blockchains for decentralized optimization of energy resources in microgrid networks (IEEE, Mauna Lani, 2017), pp. 2164–2171
7. E. Mengelkamp, B. Notheisen, C. Beer, D. Dauer, C. Weinhardt, A blockchain-based smart grid: towards sustainable local energy markets. Comput. Sci. Res. Dev. **33**(1), 207–214 (2018)
8. G. Greenspan, MultiChain Private Blockchain - White Paper. Technical report, Coin Sciences Ltd (2015). http://www.multichain.com/download/MultiChain-White-Paper.pdf
9. G. Wood, Ethereum: A Secure Decentralised Generalised Transaction Ledger. Technical report, Ethereum (2017). arXiv:1011.1669v3. https://ethereum.github.io/yellowpaper/paper.pdf
10. K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, R. Wattenhofer, in *International Conference on Financial Cryptography and Data Security*. On Scaling Decentralized Blockchains (Springer, Christ Church, 2016), pp. 106–125
11. I. Eyal, E. G. Sirer, in *Financial Cryptography and Data Security, 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*, ed. by N. Christin, R. Safavi-Naini. Majority Is Not Enough: Bitcoin Mining Is Vulnerable (Springer, Berlin Heidelberg, 2014), pp. 436–454. arXiv:1311.0243
12. K. Nikitin, E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, J. Cappos, B. Ford, in *26th USENIX Security Symposium (USENIX Security 17)*. CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds (USENIX Association, Vancouver, 2017), pp. 1271–1287
13. E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, M. Virza, in *Proceedings – IEEE Symposium on Security and Privacy*. Zerocash: Decentralized Anonymous Payments from Bitcoin (IEEE, San Jose, 2014), pp. 459–474
14. A. Unterweger, F. Knirsch, C. Leixnering, D. Engel, in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. Lessons Learned from Implementing a Privacy-Preserving Smart Contract in Ethereum (IEEE, Paris, 2018)
15. L. Lamport, R. Shostak, M. Pease, The Byzantine Generals Problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). arXiv:1011.1669v3
16. R. Canetti, T. Rabin, in *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*. Fast Asynchronous Byzantine

Agreement with Optimal Resilience 1 Introduction (ACM, San Diego, 1993), pp. 42–51
17. M. Vukolić, in *Open Problems in Network Security*. The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication (Springer, Cham, 2016), pp. 112–125
18. J. Katz, Y. Lindell, *Introduction to Modern Cryptography, 1st edn*. (Chapman and Hall/CRC, New York, 2007), pp. 1–498
19. I. Abraham, D. Malkhi, K. Nayak, L. Ren, A. Spiegelman, in *The 21st International Conference on Principles of Distributed Systems*. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus, (Lisbon, 2017), pp. 1–17. http://arxiv.org/abs/1612.02916. Accessed 24 Jan 2019
20. J. A. Lesser, X. Su, Design of an economically efficient feed-in tariff structure for renewable energy development. Energy Pol. **36**(3), 981–990 (2008)
21. L. Krampe, M. Wünsch, Schlussbericht Mieterstrom – Rechtliche Einordnung, Organisationsformen, Potenziale und Wirtschaftlichkeit von Mieterstrommodellen (MSM). Technical report (2017). https://www.bmwi.de/Redaktion/DE/Publikationen/Studien/schlussbericht-mieterstrom.pdf?__blob=publicationFile&v=10
22. E. Mengelkamp, J. Gärttner, K. Rock, S. Kessler, L. Orsini, C. Weinhardt, Designing microgrid energy markets: A case study: The Brooklyn Microgrid. Appl. Energy. **210**, 870–880 (2018)
23. Bundeskanzleramt, Kleine Ökostromnovelle –Erläuterungen (2017). https://www.ris.bka.gv.at/Dokumente/RegV/REGV_COO_2026_100_2_1346954/COO_2026_100_2_1347360.pdf. Accessed 24 Jan 2019
24. Nationalrat, Änderung des Ökostromgesetzes 2012, des Elektrizitätswirtschafts- und -organisationsgesetzes 2010, des Gaswirtschaftsgesetzes 2011, des KWK-Punkte-Gesetzes und des Energie-Control-Gesetzes sowie Bundesgesetz, mit dem zusätzliche Mittel aus von der .. BGBl. I Nr. 108/2017 (NR: GP XXV RV 1519 AB 1527 S. 190. BR: 9831 AB 9873 S. 870.) (2017). https://www.ris.bka.gv.at/eli/bgbl/I/2017/108. Accessed 24 Jan 2019
25. Bundestag, Gesetz zur Förderung von Mieterstrom und zur Änderung weiterer Vorschriften des Erneuerbare-Energien-Gesetzes. Bundesgesetzblatt Teil I. **2017**(49), 2532–2539 (2017)
26. Nationalrat, Bundesgesetz, mit dem die Organisation auf dem Gebiet der Elektrizitätswirtschaft neu geregelt wird (Elektrizitätswirtschafts- und -organisationsgesetz 2010 – ElWOG 2010). BGBl. I Nr. 110/2010 (NR: GP XXIV RV 994 AB 997 S. 86. BR: 8420 AB 8421 S. 791.) (2010). https://www.ris.bka.gv.at/eli/bgbl/I/2010/110. Accessed 24 Jan 2019
27. Nationalrat, Bundesrecht konsolidiert: Gesamte Rechtsvorschrift für Elektrizitätswirtschafts- und -organisationsgesetz 2010, Fassung vom 14.02.2018 (2018). https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=Bundesnormen&Gesetzesnummer=20007045. Accessed 24 Jan 2019
28. Bundestag, Gesetz zur grundlegenden Reform des Erneuerbare-Energien-Gesetzes und zur Änderung weiterer Bestimmungen des Energiewirtschaftsrechts. Bundesgesetzblatt Teil I. **2014**(33), 1066–1132 (2014)
29. Bundestag, Erneuerbare-Energien-Gesetz vom 21. Juli 2014 (BGBl. I S. 1066), das zuletzt durch Artikel 1 des Gesetzes vom 17. Juli 2017 (BGBl. I S. 2532) geändert worden ist (2017). https://www.gesetze-im-internet.de/eeg_2014/BJNR106610014.html. Accessed 24 Jan 2019
30. K. Wüst, A. Gervais, Do you need a Blockchain. https://eprint.iacr.org/2017/375.pdf
31. European Parliament and Council of the European Union, Directive 2009/72/EC of the European Parliament and of the Council of 13 July 2009 concerning common rules for the internal market in electricity and repealing Directive 2003/54/EC (2009). http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32009L0072. Accessed 24 Jan 2019
32. M. Hearn, Corda: A distributed ledger, Version 0.5. Technical report. Corda. Accessed September 2018 (2016). https://www.corda.net/content/corda-technical-whitepaper.pdf
33. N. Zhumabekuly Aitzhan, D. Svetinovic, Security and Privacy in Decentralized Energy Trading through Multi-signatures, Blockchain and Anonymous Messaging Streams. IEEE Trans. Dependable Secure Comput. **15**(5), 840–852 (2016)
34. M. Mihaylov, S. Jurado, N. Avellana, K. Van Moffaert, I. M. De Abril, A. Nowé, in *11th International Conference on the European Energy Market, EEM*. NRGcoin: Virtual currency for trading of renewable energy in smart grids (IEEE, Krakow, 2014)
35. M. Mihaylov, S. Jurado, K. V. Moffaert, A. Now, in *Proceedings of the 3rd International Conference on Smart Grids and Green IT Systems*. NRG-X-Change - A Novel Mechanism for Trading of Renewable Energy in Smart Grids (SciTePress, Barcelona, 2014), pp. 101–106. arXiv:1011.1669v3
36. J. J. Sikorski, J. Haughton, M. Kraft, Blockchain technology in the chemical industry: Machine-to-machine electricity market. Appl. Energy. **195**, 234–246 (2017)
37. D. Ilic, P. G. Da Silva, S. Karnouskos, M. Griesemer, in *IEEE International Conference on Digital Ecosystems and Technologies*. An energy market for trading electricity in smart grid neighbourhoods (IEEE, Atlanta, 2012), pp. 1–6
38. Y. Wang, W. Saad, Z. Han, H. V. Poor, T. Başar, A game-theoretic approach to energy trading in the smart grid. IEEE Trans. Smart Grid. **5**(3), 1439–1450 (2014). http://arxiv.org/abs/1310.1814
39. E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, B. Ford, in *25th USENIX Security Symposium (USENIX 2016)*. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing (USENIX Association, Austin, 2016), pp. 279–296
40. J. Bucko, D. Paťová, M. Vejačka, in *Central European Conference In Finance And Economics*. Security and Trust in Cryptocurrencies (Technical University of Košice, Herlany, 2015), pp. 14–24
41. A. Unterweger, F. Knirsch, C. Leixnering, D. Engel, Update: Lessons Learned from Implementing a Privacy-Preserving Smart Contract in Ethereum. Technical report, Center for Secure Energy Informatics, Salzburg University of Applied Sciences, Austria (nov 2017). http://www.en-trust.at/papers/Unterweger18a-t.pdf
42. F. Knirsch, A. Unterweger, K. Karlsson, D. Engel, S. B. Wicker, Evaluation of a Blockchain-based Proof-of-Possession Implementation. Technical report, Center for Secure Energy Informatics, Salzburg University of Applied Sciences, Austria (2018). http://www.en-trust.at/papers/Knirsch18a-t.pdf
43. R. Han, V. Gramoli, X. Xu, in *9th IFIP Conference on New Technologies, Mobility & Security (NTMS 2018)*. Evaluating Blockchains for IoT (IEEE/IFIP, Paris, 2018), pp. 0–4
44. A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, in *2016 IEEE Symposium on Security and Privacy (SP)*. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts (IEEE, San Jose, 2016), pp. 839–858
45. D. Mazières, The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus. Technical report, Stellar Development Foundation (2016). https://www.stellar.org/papers/stellar-consensus-protocol.pdf
46. E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, B. Ford, OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. IEEE Symp. Secur. Priv., 583–598 (2018)
47. D. Engel, G. Eibl, Wavelet-Based Multiresolution Smart Meter Privacy. IEEE Trans. Smart Grid. **8**(4), 1710–1721 (2017)
48. D. Laupichler, S. Vollmer, H. Bast, M. Intemann, Das BSI-Schutzprofil: Anforderungen an den Datenschutz und die Datensicherheit für Smart Metering Systeme. Datenschutz und Datensicherheit - DuD. **35**(8), 542–546 (2011)
49. M. Herlihy, N. Shavit, *The Art of Multiprocessor Programming, rev. 1st edn*. (Morgan Kaufmann, Waltham, 2012)
50. E. Barker, W. Barker, W. Burr, W. Polk, M. Smid, C. S. Division, NIST 800-57: Computer Security. NIST (2012). National Institute of Standards & Technology. https://csrc.nist.gov/publications/detail/sp/800-57-part-1/revised/archive/2007-03-01
51. T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (2008). https://tools.ietf.org/html/rfc5246. Accessed 24 Jan 2019
52. C. Decker, R. Wattenhofer, *Information propagation in the Bitcoin network Information Propagation in the Bitcoin Network* (IEEE P2P 2013 Proceedings, Trento, 2013), pp. 1–10
53. M. Jakobsson, A. Juels, in *Secure Information Networks: Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99) September 20–21, 1999, Leuven, Belgium*, ed. by B. Preneel. Proofs of Work and Bread Pudding Protocols (Extended Abstract) (Springer, Dordrecht, 1999), pp. 258–272
54. I. Eyal, A. E. Gencer, E. G. Sirer, R. van Renesse, in *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation. NSDI'16*. Bitcoin-NG: A Scalable Blockchain Protocol (USENIX Association, Santa Clara, 2016), pp. 45–59. http://arxiv.org/abs/1510.02037
55. M. Carlsten, H. Kalodner, S. M. Weinberg, A. Narayanan, in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*. On the Instability of Bitcoin Without the Block Reward (ACM, Vienna, 2016), pp. 154–167