Blockchains and Smart Contracts for the Internet of Things

Konstantinos Christidis, *Member, IEEE,* and Michael Devetsiokiotis, *Fellow, IEEE*

***Abstract*—Motivated by the recent explosion of interest around blockchains, we examine whether they make a good fit for the IoT sector. Blockchains allow us to have a distributed peer-to-peer network where non-trusting members can interact with each other without a trusted intermediary, in a verifiable manner. We review how this mechanism works and also look into smart contracts – scripts that reside on the blockchain that allow for the automation of multi-step processes. We then move into the IoT domain, and describe how a blockchain-IoT combination (a) facilitates the sharing of services and resources leading to the creation of a marketplace of services between devices, and (b) allows us to automate in a cryptographically verifiable manner several existing, time-consuming workflows. We also point out certain issues that should be considered before the deployment of a blockchain network in an IoT setting; from transactional privacy to the expected value of the digitized assets traded on the network. Wherever applicable, we identify solutions and workarounds. Our conclusion is that the blockchain-IoT combination is powerful and can cause significant transformations across several industries, paving the way for new business models and novel, distributed applications.**

***Index Terms*—blockchain, distributed systems, Internet of Things.**

I. Introduction

BLOCKCHAINS have recently attracted the interest of stakeholders across a wide span of industries: from finance [1] and healthcare [2], [3], to utilities [4], real estate [5], [6], and the government sector [7]. The reason for this explosion of interest: With a blockchain in place, applications that could previously run only through a trusted intermediary, can now operate in a decentralized fashion, without the need for a central authority, and achieve the same functionality with the same amount of certainty. This was simply not possible before.

We say that the blockchain enables *trustless* networks, because the parties can transact even though they do not trust each other. The absence of a trusted intermediary means faster reconciliation between transacting parties. The heavy use of cryptography, a key characteristic of blockchain networks, brings authoritativeness behind all the interactions in the network. Smart contracts –self-executing scripts that reside on the blockchain– integrate these concepts and allow for proper, distributed, heavily automated workflows. This should make blockchains enticing to researchers and developers working in the Internet of Things (IoT) domain.

Of course the transition to a decentralized network may not always make sense. On top of that, even if such a transition is desirable, the application's requirements may be such that a blockchain-based network cannot fulfil them. Blockchains and smart contracts bring a slew of advantages to the table, but as we will see, they also come with a bag of disadvantages.

The goal of this work is to provide a detailed description of how blockchains and smart contracts work, to identify the pros and cons that their introduction brings to a system, and highlight the ways the blockchains and IoT can be used together. This will allow the reader to identify potentially new use cases for their IoT work, and also make educated decisions when integrating a blockchain in their project.

The paper is structured as follows. In Section II we examine what a blockchain is, how a blockchain network operates, and how smart contracts allow us to radically redefine how interactions between transacting parties on a network can be set up and automated. We end the section with a taxonomy for blockchains. In Section III, we look into how IoT and blockchains can be used together, and highlight existing IoT-on-the-blockchain applications. We note down the issues that the IoT developer/researcher would need to keep in mind when deploying a blockchain-based solution for their project in Section IV, and present our conclusions in Section V.

II. Preliminaries

A. How blockchains work

A blockchain is a distributed data structure that is replicated and shared among the members of a network. It was introduced with Bitcoin [8] to solve the double-spending problem [9]. As a result of how the nodes on the Bitcoin network (the so-called miners) append validated, mutually agreed-upon transactions to it, the Bitcoin blockchain houses the authoritative ledger of transactions that establishes who owns what [10].

However a blockchain can stand on its own just fine – no cryptocurrency needed [11]. Think of the blockchain as a log whose records are batched into timestamped blocks. Each block is identified by its cryptographic hash[1]. Each block references the hash of the block that came before it. This establishes a link between the blocks, thus creating a *chain* of blocks, or *blockchain* - see Figure 1. Any node with access to this ordered, back-linked list of blocks [12] can read it and figure out what is the world state of the data [10] that is being exchanged on the network.

We get a better understanding of how a blockchain works, when we examine how a *blockchain network* runs. This is a

K. Christidis (kchrist@ncsu.edu) and M. Devetsiokiotis (mdevets@ncsu.edu) are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC.

Manuscript received March 6, 2016. Revised manuscript received April 23, 2016.

[1] Whether the hash is generated over the block's contents or its header, as is the case in Bitcoin for instance, is a design choice.

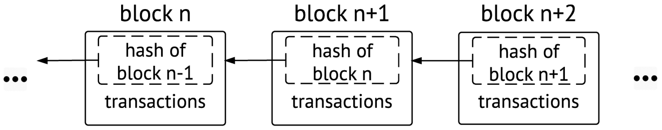


Fig. 1. Each block in the chain carries a list of a transactions and a hash to the previous block. The exception to this is the first block of the chain (not pictured), called genesis, which is common to all clients in a blockchain network and has no parent.

set of nodes (clients) that operate on the same blockchain via the copy each one holds. A node can generally act as an entry point for several different blockchain users into the network, but for simplicity, we assume that each user transacts on the network via their own node. These nodes form a peer-to-peer network where:

1) Users interact with the blockchain via a pair of private/public keys [13]. They use their private key to sign their own transactions, and they are addressable on the network via their public key[2]. The use of asymmetric cryptography brings authentication, integrity, and non-repudiation into the network. Every signed transaction is broadcasted by a user's node to its one-hop peers.
2) The neighboring peers make sure this incoming transaction is valid before relaying it any further; invalid transactions are discarded. Eventually this transaction is spread across the entire network.
3) The transactions that have been collected and validated by the network using the process above during an agreed-upon time interval, are ordered and packaged into a timestamped candidate block. This is a process called *mining*. The mining node broadcasts this block back to the network. (The choice of the mining node and the contents of the block depend on the consensus mechanism that the network employs. Refer to Section II-B for more information.)
4) The nodes verify that the suggested block (a) contains valid transactions, and (b) references via hash the correct previous block on their chain. If that is the case, they add the block to their chain, and apply the transactions it contains to update their world view. If that is not the case, the proposed block is discarded. This marks the end of a round.

Note that this is a repeating process.

When we talk about transaction validation in step #2, the natural question is: what constitutes a valid transaction? We need to remember that in a blockchain network what we have essentially is a set of non-trusting writers sharing a database with no trusted middleman [14]. In order to prevent chaos from erupting in this distributed environment, and in order to help the network reach a common global view of the world (i.e. reach *consensus*), each blockchain network needs to establish certain rules that each database transaction should conform to. These application-dependent rules are programmed into each blockchain client, which then uses them to decide whether an incoming transaction is valid, and consequently whether it should be relayed to the network or not. In the simplified "shared database" model we present here, let us assume that each row of the database is mapped to a public key (or address) that controls who can edit it. A valid transaction then is one that attempts to modify a row for which the corresponding signature is present.

When each node in the network follows the steps listed above, the shared blockchain it operates on becomes an authenticated and timestamped record of the network's activity [10]. Note how the nodes do not have to trust any other entity, giving rise to the term *trustless environment*; instead, as noted in [12], trust is achieved as an emergent property from the interactions of different participants in the system.

The above is what happens from a high-level view and in the general case. Things get more interesting when we examine how blockchains can be used for the transfer and tracking of assets (Section II-C), or to run code (Section II-D).

B. Reaching consensus on the network

The nodes need to agree on the transactions and the order in which these are listed on the newly-mined block. Otherwise, the individual copies of the blockchain will diverge and we will end up with forks; the nodes will have a different view of the world state and the network will no longer be able to maintain a unique authoritative chronology [14] (i.e. blockchain) unless this fork is resolved.

A distributed consensus mechanism is therefore needed in every blockchain network in order to achieve that. As we wrote in Section II-A, the type of consensus mechanism used depends on the type of the blockchain network and the attack vector that the network operator adopts. (The "it depends" answer will unfortunately be a recurring theme throughout this work. It may make things less straightforward to explain, but it also speaks to the versatility of the blockchain to adapt in a number of situations.)

In an ideal scenario, all validating nodes would vote on the order of transactions for the next block, and we would go with what the majority decided. In an open network that anyone can join though, this would be catastrophic because of the Sybil attack [15]: a single entity could join with multiple identities, get multiple votes, and thus influence the network to favor this entity's interests. In other words, a minority could seize control of the network.

Bitcoin works around this problem by making mining computationally "expensive"; impersonating multiple entities on the network will not help, as the computing resources of any single entity are limited. Specifically, any node can have their assembled block be the next mined block on the network, if they can find the right random number (nonce) in that block's header that will make the SHA-256^2 [16], [17] hash of the header to have the amount of leading zeroes [3] that the network expects [12]. Any node that can solve this puzzle, has generated the so-called *proof-of-work* (PoW) and gets to shape

[2] Depending on the implementation, the address can be the public key itself or (usually) a hash of it.

[3] Note that the more leading zeroes are required, the more difficult the solution of this puzzle is. We call this an increase of *the target*. The network adjusts the level of difficulty every 2,016 blocks to account for changes in the network's CPU power, and to make sure that blocks are generated at a steady rate.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2016.2566339, IEEE Access

the chain's next block [8]. Since a one-way cryptographic hash functions is involved, any other node can easily verify that the given answer satisfies the requirement (and adopt this block for its blockchain if that's the case), but cannot do the opposite; i.e. guess from the result requirement what the input should be.

Note that a fork may still happen on the network, when two competing nodes mine blocks almost simultaneously. Such forks are usually resolved automatically by the next block; the proof-of-work mechanism dictates that the nodes should adopt the fork that carries the greatest amount of work, and it is unlikely that the two competing forks will generate the next block simultaneously. Whichever fork grows longer first will be adopted by the nodes as the correct one. This allows the network to reach consensus on the proper order of events again.

Other hashing algorithms can be used for PoW besides SHA-256, such as Blake-256 [18] and scrypt [19] (used in Litecoin [20]). There are also mechanisms that combine several of these algorithms together, such as Myriad [21].

Proof-of-stake (PoS) is an alternative to proof-of-work that requires far fewer CPU computations for mining. In PoS, the chances of a node mining the next block are proportional to that node's balance[4]. PoS schemes have their own strengths and weaknesses [22], and actual implementations are proving to be quite complex [23].

A detailed description and evaluation of *proof-of-X* mechanisms can be found in [24].

In private networks however, where the participants are whitelisted, costly consensus mechanisms such as proof-of-work are not needed; the risk of a Sybil attack is not there [25]. This practically removes the *need* for an economic incentive for mining, and gives us a wider range of consensus protocols to pick from.

Practical Byzantine Fault Tolerance (PBFT) [26] is such an algorithm. It provides a solution to the Byzantine Generals Problem [27] that works in asynchronous enviroments like the Internet. (Bitcoin, via the mechanism described above, also provides a practical solution to the same problem [24].) It involves a three-phase protocol and the notion of a "primary" (leader) node that acts as the block miner; the leader can be changed by the rest of the network via a so-called "view-change" voting mechanism, if it crashes or if it exhibits arbitrary behavior (Byzantine faults). PBFT works on the assumption that less than one third of the nodes are faulty (f), which is why say that it requires at least[5] $3f+1$ nodes.

Tangaroa, a Byzantine Fault Tolerant (BFT) variant of the popular Raft algorithm [30], is used as a consensus mechanism in Juno [31]. *Tendermint* [32] provides BFT tolerance and is similar to the PBFT algorithm; however it provides a tighter guarantee with regards to the results returned to the client when more than one third of the nodes are faulty, and allows for a dynamically changing set of set of validators, and leaders that can be rotated in a round-robin manner, among other optimizations [33].

Ripple's [34] consensus algorithm [35] uses "collectively-trusted subnetworks" called "Unique Node Lists" (UNL) to deal with the high latency that usually characterizes BFT-tolerant systems. A node needs to query only its own UNL, instead of the whole network, in order to reach consensus. It can tolerate less than one fifth of its nodes being faulty ($5f+1$ resilience).

In the *mining diversity* [36] scheme (used in MultiChain [37]), whitelisted miners add blocks to the chain in a round-robin manner, with some degree of leniency to allow for malfunctioning nodes [14]. A network parameter called "mining diversity" is used to calculate the number of blocks that a miner should wait for before attempting to mine again (otherwise its suggested block will be rejected). A low value of the mining diversity parameter means that fewer miners need to collude in order to take over the network; if the number of colluding miners is equal to or bigger than the number of blocks each miner should wait before attempting to mine again, then there is a probability that this will happen. Conversely, a higher value of the mining diversity parameter ensures that more permitted miners are included in the rotation, thus making the network take-over by a minority more difficult.

Sieve [38], a mechanism used in the HyperLedger Fabric project, augments the PBFT algorithm [26] by adding speculative execution and verification phases, inspired by the execute-verify architecture presented in [39]. This allows the network to detect and filter out possible non-deterministic requests, and also achieve consensus on the output state of the suggested transactions (in addition to consensus on their input order).

[40] contrasts proof-of-work and BFT consensus protocols, and offers an excellent overview of the state of the art with a focus on scalability.

Note that regardless of the consensus mechanism used, the miners in a blockchain network "have far less power than the owner of a traditional centralized database since they cannot fake transactions" [14].

C. Transfering digital assets on a blockchain

In order to show how an asset transfer works, it is best to consider a simplified example from the banking world. Imagine a bank's (centralized) database that tracks the aggregate balances of each customer. We are basically looking at a table with three columns: "asset type", "owner" ("counterparty" [41]), and "quantity" ("amount"). For example, a row in that table with "USD", "Alice", "10" identifies Alice as having $10 deposited in that bank. Bob has an account in the same bank with $0 in it. When Alice transfers $2 to Bob's account, the "quantity" of the USD/Alice (asset type/owner) row gets updated to $8, and that of USD/Bob now reads $2. An asset ($2 USD), or rather the *digital representation* of this asset, was transferred between two entities via a transformation of the appropriate rows in the database.

[4] Note that both PoW and PoS effectively require a blockchain that supports cryptocurrency; the former in order to incentivize the miners to run the power-hungly (and thus, costly) hash calculations needed to mine a block, and the latter in order get the miners to have their (crypto-)money at stake when mining.

[5] If more than $3f+1$ nodes are used, then the quorum thresholds listed in [26] may lead to forks. Instead, the bounds indicated in [28] should be used. This is for instance the implementation adopted in the HyperLedger Fabric project [29].

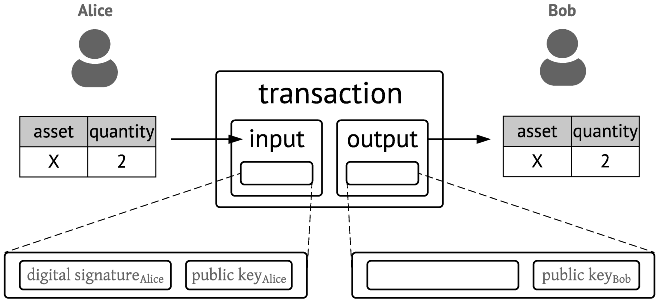


Fig. 2. A transaction that transfers a tokenized asset (X) from Alice to Bob. Alice signed her input, and created an output locked against Bob's public key, so that only Bob can spend it.

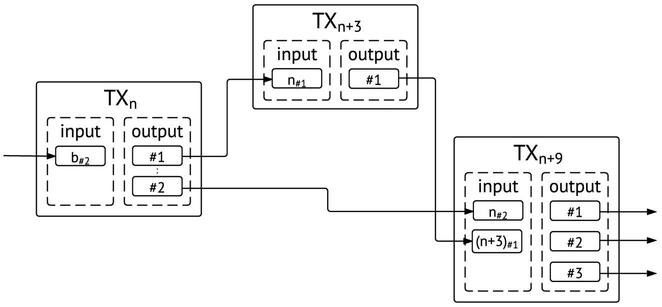


Fig. 3. Transaction n spends the second UTXO that transaction b (not pictured) created ($b\#2$), and generates two new outputs ($n\#1$, and $n\#2$), spent by transactions $n+3$ and $n+9$ respectively. A similar process applies to every transaction in the network. Transactions are therefore linked to each other and allow for easy provenance tracking.

This transfer of *digital tokenized assets* can be achieved easily and in a cryptographically verifiable manner using a blockchain network that employs the Bitcoin transactional model[6]. Consider again the model of a database that is shared by non-trusting writers in a trustless environment, as in Section II-A. Each row carries the same fields as in the banking example above, with the difference that the "owner" field now holds the public key of the user that is allowed to edit the row. Assume that the database shows that Alice owns 10 units of asset X. (We will get to how this truth was established, i.e. how these assets were generated shortly.) That is, a row in that database, carries Alice's public key in the "owner" column, and the values "X" and "10" in the "asset type" and "quantity" columns respectively. Assume that Alice knows Bob's public key. How does Alice transfer 2 units of X to Bob? She signs a transaction that modifies her row, decreasing the value of X by 2, and creates a new row, whose "owner" is set to Bob's public key, and whose "asset type" and "value" fields are set to "X" and "2" respectively.

Alice transferred 2 units of X to Bob by creating a new row with that information and assigning it to him; see Figure 2. (In fact, Alice's transaction also deleted her own row, created a new row assigned to one of her public keys, and moved the 8 remaining units of X she holds there. That is done in order to control concurrency –see Section II-E– and prevent conflicts between concurrent write operations in the system; rows are not modified, instead they are deleted and new rows are created with the updated values [11].)

Bob's new balance of asset "X" can be calculated by aggregating all the rows in the database that correspond to his public keys, and whose "asset type" is set to "X". Same goes for Alice.

Some of the validation checks that we would encode into the nodes of a blockchain network that is set up for such asset transfers would be:

- Does the proposed transaction address an existing row?
- Is it properly signed so as to delete that row (or rows)?
- Has this row been addressed (used) by a previous validated transaction? An asset cannot be spent twice.
- Does it transfer the right amounts to new rows? For example, if the row the transaction reads "10 units of X", an attempted transfer of "2 units of X" (to Bob) and "9 units of X" (back to Alice) should fail. Same goes for an attempted transfer of, say, "10 units of Y". The sum of inputs should equal the sum of outputs, i.e. a transfer should not increase the total quantity of an asset type.

Note that a transaction can address several existing rows instead of just one, i.e. transfer assets scattered over the database, as long as it is properly signed to access them. These existing, not-yet-deleted rows are called unspent transaction outputs (UTXO) in Bitcoin; they were created by earlier transactions in the system. The UTXO that a transaction consumes are called transaction inputs; the UTXO that a transaction creates are called transaction outputs [12].

A transaction then basically deletes a set of rows (UTXO) and creates a set of new rows (UTXO) in the database (see Figure 3 for an example).

One outstanding question from the description above is: how do we generate assets and introduce them in the chain? Before we get to the state of Alice having 10 units of X, these 10 units of X have to come from somewhere. The answer is that it depends on the network and its purpose. Generally, a properly-authorized node uses a special type of transaction to introduce an asset (or new units of the asset) into the network.

For example, assume a private blockchain network between Alice, Bob, and Carol. Carol sets this up using MultiChain [37], a blockchain platform that assigns permissions (can connect to the network, can transact to the network, can issue on the network) to public keys. Carol configures the network so that her public key can issue assets on the network. She invites Alice and Bob to join; both of them are OK with Carol's ability to issue assets on the chain. All of them have a pair of private and public keys. Carol submits a signed transaction that generates 10 units of X. The nodes on the network consider this transaction valid, since her public key is permissioned appropriately. Carol then transfers these newly-generated units of X to Alice, which brings us to Alice having 10 units of X.

In the case of Bitcoin, new bitcoins are introduced into the network with every mined block: The mining node includes a so-called coinbase transaction [42] in the block of transactions

[6] As we will see later on, it can also be implemented with a smart contract, but with a few key differences, especially when it comes to performance; see Section II-E.

it broadcasts to the network. This coinbase transaction has no inputs and rewards the mining node with a pre-determined (by the network) amount of bitcoins[7].

The key thing to keep in mind is this: if you have a set of users (a) who want to trade digital tokens, and (b) have agreed on how these tokens are generated, then a blockchain network is an ideal tool to use both for exchanging these tokens, and tracking who has what. No middleman is needed to facilitate the exchanges cause every node on the network runs the the necessary checks and reaches consensus on the accepted result. Asset tracking comes out-of-the-box since every node has access to the agreed set of cryptographically verifiable transactions on the blockchain.

D. How smart contracts work

Nick Szabo introduced this concept in 1994 and defined a smart contract as ”a computerized transaction protocol that executes the terms of a contract” [44]. Szabo suggested translating contractual clauses (collateral, bonding, etc.) into code, and embedding them into property (hardware, or software) that can self-enforce them [45], so as to minimize the need for trusted intermediaries between transacting parties, and the occurence of malicious or accidental exceptions.

Within the blockchain context, smart contracts are scripts stored on the blockchain. (They can be thought of as roughly analogous to stored procedures in relational database management systems [46].) Since they reside on the chain, they have a unique address. We trigger a smart contract by addressing a transaction to it. It then executes independently and automatically in a prescribed manner on every node in the network, according to the data that was included in the triggering transaction. (This implies that every node in a smart contract-enabled blockchain is running a virtual machine (VM), and that the blockchain network acts as a distributed VM.)

Smart contracts allow us to have general purpose computations occur on the chain. Where they excel however, is when they are tasked with managing data-driven interactions [47] between entities on the network. Let us unpack this statement with an example. Consider a blockchain network where Alice, Bob, and Carol participate, and where digital assets of type X and Y are being traded. Bob deploys a smart contract on the network that defines: (a) a “deposit” function allowing him to deposit units of X into the contract, (b) a “trade” function that sends back 1 unit of X (from the contract’s own deposits) for every 5 units of Y it receives, and (c) a “withdraw” function that allows Bob to withdraw all the assets that the contract holds.

Note that the “deposit” and “withdraw” functions are written so that *only* Bob (via his key) can call them, because this is what Bob decided, and also what makes sense for our example; they could have been written so that they can be called successfully by any user on the network.

Bob sends a transaction to that smart contract’s address, calling its “deposit” function and moving 3 units of X to the contract. This transaction is recorded on the blockchain. Alice, who owns 12 units of Y, then sends a transaction that moves 10 units of Y to the contract’s “trade” function, and gets back 2 units of X. This transaction is also recorded on the blockchain. Bob then sends a signed transaction to the contract’s “withdraw” function. The contract checks the signature to make sure the withdrawal is initiated by the contract’s owner, and transfers all of its deposits (1 unit of X, and 10 units of Y) back to Bob.

Let us observe the following:

1) The contract has its own state and can take custody over assets on the blockchain [48]. We say that a contact has its own account on the blockchain, and the blockchain supports an *account-based model* [49]. In the example above, it can hold assets X and Y. (If we go back to the shared database model, a contract is a separate “user”/entity that can own, delete, and create rows.)
2) The contract allows us to express business logic in code; “will trade 1 unit of X for every 5 units of Y received”.
3) A properly written smart contract should describe all possible outcomes of the contract. For instance, the “trade” function above may be written so as to reject offers that bring in quantities of Y that are not multiples of 5; i.e. an offer of 12 units of Y will be rejected. Or the function may be written so as to parse the incoming offer as the biggest multiple of 5 (that will be traded without issues) and a remainder (that will be returned). An offer of 12 units of Y then, would return 2 units of X and 2 units Y to the sender.
4) The relationship that Bob wishes to establish with his counterparties is one driven by data [47]. A transaction after all is a signed *data structure* indicating a transfer of value [12]. Bob deploys a smart contract that effectively says, ”if you send this contract this data (5 units of Y), here’s how it will respond (1 unit of X)”.
5) A smart contract is triggered by messages/transactions sent to its address.
6) A smart contract is deterministic; the same input will always produce the same output. If one writes a non-deterministic contract, when it is triggered it will execute on every node on the network and may return different random results, thus preventing the network from reaching consensus on its execution result. In a properly built blockchain platform, writing non-deterministic smart contracts is either impossible (by forcing the contract developers to use a programming language that does not have any non-deterministic constructs [50]), or it is possible but an attempt to deploy such a contract on the network will be rejected [38].
7) A smart contract resides on the blockchain, and as such its code can be inspected by every network participant.
8) Since all the interactions with a contract occur via *signed* messages on the blockchain, all the network participants get a cryptographically verifiable trace of the contract’s operations.

A blockchain that supports Bitcoin-style transactions enables *asset transfers* between counterparties that do not trust each other. A blockchain that supports smart contracts however, takes this further and allows for multi-step processes

[7] This reward is halved every 210,000 blocks [43].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2016.2566339, IEEE Access

(or more generally: *interactions*) to occur between mutually distrustful counterparties. The transacting entities get to (a) inspect the code and identify its outcomes before deciding to engage with the contract, (b) have certainty of execution since the code is already deployed on a network that neither of them controls fully, and (c) have verifiability over the process since all the interactions are digitally signed. The possibility of a dispute is eliminated (when all possible outcomes are accounted for) since the participants cannot disagree over the final outcome of this verifiable process they engaged in.

Smart contracts operate as *autonomous actors*, whose behavior is completely predictable. As such they can be trusted to drive forward any on-chain logic that can be expressed as a function of on-chain data inputs, provided that the data they need to manage is within their own reach (in the example above, the contract wouldn't be able to trade assets that it did not own).

We close this section by noting that smart contracts also give rise to the concept of "decentralized autonomous organizations" (DAOs), entities on the blockchain whose behavior may be modified, if a certain process that is encoded in the contract is followed. As described in [51], the simplest example is that of a smart contract that calls another contract by address to perform its main function. This address resides on the mutable part of the contract's internal database. The contract also carries a list of members, addresses (public keys) that get to vote on its behavior. A rule can be included in the contract so that if a majority of those voters vote in a certain way, the contract will modify its behavior by calling the address that received the majority of the votes to execute its main function.

E. A blockchain taxonomy

There are several ways to categorize a blockchain network. We highlight the following, inline with the optimization/permission spectrums presented in [10]:

- *Who has access to the network*: If anyone can join, we are dealing with a *public* or *permission-less network*, whereas if we have a whitelist in place, we are dealing with a *private* or *permissioned network*. The answer to this question decides what the consensus mechanism should be II-B. Because of the Sybil attack [15], consensus in public networks is costly and an economic incentive to the miners (in the form of cryptocurrency) is usually needed. Private networks make more sense for stakeholders who wish to operate in a controlled, regulated environment, or who wish a higher throughput than what a public network can offer.
- *Who can transact or mine*: All participants may not be allowed to transact [52], deploy smart contracts, or participate in the mining process [29]. This filter usually applies only to private networks, where all the participants are identifiable.
- *Bitcoin-style transactions (UTXO model) or smart contracts (account-based model)*: As explained in Section II-C, blockchains that support the UTXO model are uniquely suited for the transfer of and tracking of digital tokenized assets, whereas blockchains that support the account-based model (Section II-D) give us the means to run *arbitrary logic* and establish verifiable multi-step processes. This support for arbitrary logic however comes at a severe cost when it comes to concurrent execution and transaction throughput [53]. Before the VM of a node processes the incoming message to a smart contract, it cannot tell how it will affect the contract's internal state, or whether it will trigger another contract in the system, so it cannot run all the transactions of a block in parallel. In the UTXO model though, every transaction explicitly identifies its inputs and outputs; if there is no dependency between the outputs and the inputs of transactions in the block (i.e. as long as a transaction does not attempt to spend the output that another transaction in the same block creates), the order of execution does not matter and all of these transactions can be processed by the block in parallel.

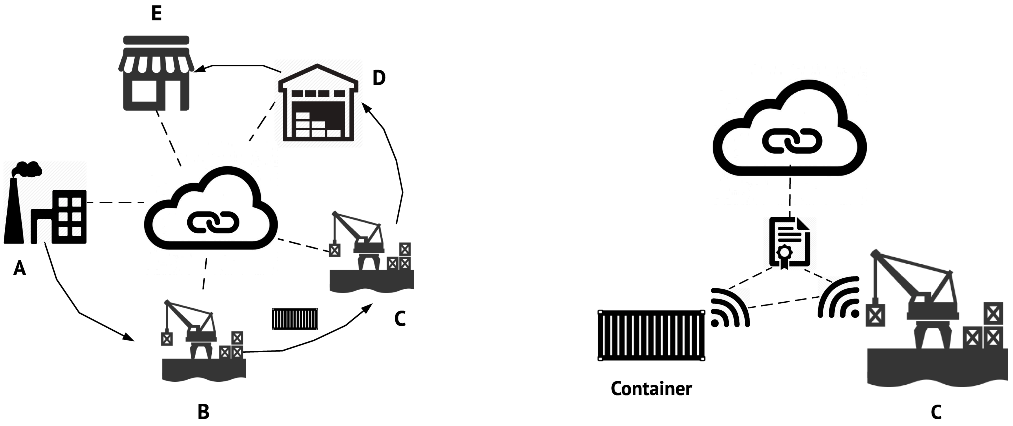
For other attempts at a categorization of blockchains and distributed ledgers, see [7] and [25]. At any rate, it is key to remember that –regardless of specific configuration– a blockchain gives us the following benefits out of the box:

- A robust, truly distributed peer-to-peer system that is tolerant of node failures.
- A network that can identify conflicts and forks and resolve them automatically so as to converge to a single, globally accepted view of events.
- Transparency, verifiability, auditability on the network's activity. We get verifiable processes, whether these concern the exchange and tracking of a digital asset, or a data-driven interaction between parties. Every transaction presents a publicly auditable proof that it was authorized to interact with the system [11]. Eliminates the possibility of disputes, makes reconciliation redundant.
- As noted in [54]: "A method for tagging different pieces of information as belonging to different participants, and enforcing this form of data ownership without a central authority."
- A system that allows non-trusting participants to interact with each other in a predictable, certain manner.

III. Blockchains and IoT

In [55], the authors make the case for a shift towards a decentralized architecture for the ever-expanding IoT device ecosystem to be sustainable. From the manufacturer's side, the current centralized model has a high maintenance cost – consider the distribution of software updates to millions of devices for years after they have been long discontinued. From the consumer's side, there is a justified [56] lack of a trust in devices that "phone home" in the background and a need for a "security through transparency" approach. These issues can be solved with a scalable, trustless peer-to-peer model that can operate transparently and distribute data securely; the authors correctly point out that a blockchain provides an elegant solution to this problem.

Consider the following setup to get an understanding of how this could work. All the IoT devices of a manufacturer operate



(a) (b)

Fig. 4. An asset tracking example using smart contracts and IoT. In Fig. 4a (left) a container leaves the manufacturing plant (A), reaches the neighboring port (B) via railway, gets transported to the destination port (C), and then to the distributor's facilities (D), until it reaches the retailer's site (E). In Fig. 4b (right), we focus on the B-C stage. The carrier of the container performs a handshake with the dock at the destination port (C) to confirm that the container is delivered to the expected location. Once that handshake is completed, it posts to a smart contract to sign the delivery. The destination port follows along to confirm reception. If the node at C does not post to the contract within an acceptable timeframe, the shipping carrier will know and can initiate an investigation on the spot.

on the same blockchain network. The manufacturer deploys a smart contract that allows them to store the hash of the latest firmware update on the network. The devices either ship with the smart contract's address baked into their blockchain client, or they find out about it via a discovery service (see Section IV). They can then query the contract, find out about the new firmware, and request it by its hash via a distributed peer-to-peer filesystem such as IPFS [57], [58]. The first requests for this file will be served by the manufacturer's own node (also taking part into the network), but after the binary has propagated to enough nodes [59], the manufacter's node can stop serving it. Assuming the devices are configured so as to share the binary they got[8], a device that joins the network long after the manufacturer has stopped participating in it, can still retrieve the sought-after firmware update and be assured that it is the right file. This all happens automatically, without any user interaction. Compare and contrast with the centralized scenario where the device polls the manufacturer's server for an update and gets a 404 error [61].

Furthermore, a blockchain network where cryptocurrency is exchanged provides a convenient *billing layer* and paves the way for a *marketplace of services between devices*. In the example above, devices that store a copy of the binary may charge for serving it, in order to sustain their infrastructure costs (or simply to make a profit). Other examples include: Filecoin [62] which allows devices to "rent their disk space", and 21 [63] and EtherAPIs [64], which make it possible to monetize API calls – the caller needs to provide the necessary micropayment (in Bitcoin or Ethereum respectively) before requesting them. With a cryptocurrency in place, every device can have its own *bank account* on the Internet; it can then expose its resources to other devices (or users) and get compensated for their usage via microtransactions [65].

This also facilitates the sharing of services and property in general. Slock.it [66] works on smart electronic locks ("Slocks") that can be unlocked with a device that carries the appropriate token. These tokens are bought on the Ethereum blockchain [67], a public blockchain network optimized for smart contracts that uses its own cryptocurrency, called Ether. The owner of a Slock that wishes to rent their house or car sets a price for timed access to that electronic door lock. An interested party can use a mobile app to identify the slock, pay the requested amount in Ethers, then communicate with the lock via a properly signed message (using the Whisper peer-to-peer communication protocol [68]) to unlock it. Billing is simplified by having all the Slocks operating on the same blockchain.

Along the same theme, in the energy sector, the integration of IoT with blockchains allows for a peer-to-peer market where machines can buy and sell energy automatically, according to user-defined criteria. For example, TransActive Grid [69] is experimenting with the concept of a peer-to-peer market for renewable energy in a neighborhood in Brooklyn, NY [70]. Solar panels record their excess output on the blockchain, and sell it to neighboring parties via smart contracts.

The usefulness of blockchains in an IoT setting does not stop there. Consider the typical supply chain example that is used to highlight the value of a blockchain: a container that leaves the manufacturer's site (point A), gets transported via railway to the neighboring port (point B), then gets shipped to the destination port (point C), gets transported again to the distributor's facilities (point D), until it finally reaches the retailer's site (point E). This process involves several stakeholders and checks along the way, all of them depicted in Figure 4a. Each stakeholder usually maintains their own database to keep track of the asset, which they update based on inputs from the other parties along the chain. A blockchain network though that is set up to track this asset would mean that there is now one shared database to keep track of, where updates come with cryptographic verifiability,

[8] For example, via a protocol such as ipfs-cluster [60].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2016.2566339, IEEE Access

get propagated along the network automatically, and create an auditable trail of information. For example (Figure 4b), when the shipping carrier reaches the destination port, they send a signed message to a predetermined and agreed-upon smart contract to allow everyone on the chain to know that the container is now at point C. Since the transaction is signed, it acts a cryptographically verifiable receipt of the shipping company's claim that the container has reached the destination port. The receiver at the port posts to same smart contract to confirm that it is in posession of the container.

Alternatively, this whole process can be done via atomic peer-to-peer exchanges of tokens[9] if the chain follows the Bitcoin transacitonal model (Section II-C). When the blockchain is created, the manufacturer is allowed to issue a "I have the container" token; all the other stakeholders are allowed to issue a "I have received the container" token. When the manufacturer hands off the container to the freight transaportation company that will move it from point A to point B, it creates a transaction with two inputs and two outputs (see Section II-C). Input #1 points to the manufacturer's own UTXO (the "I have the container" token) and output #1 creates a new UTXO that locks this token against the transporter's public key, effectively passing ownership to it. Input #2 points to the transporter's own UTXO (the "I have received the container" token) and output #2 creates a new UTXO that transfers that token to the manufacturer. The manufacturer signs their part, then sends this incomplete (and thus, non redeemable transaction to the transporter who signs their own part, then pushes it to the blockchain. When this transaction is added to blockchain, the manufacturer has received the "I have received the container" token from the transporter, and the transporter now holds the "I have the container token". A similar atomic exchange will take place at point B between the transporter and the shipping company, until the retailer finally receives the "I have the container token" at point E. At this point there is a complete, cryptographically verifiable, timestamped trail that tracks the asset, and leaves little room for dispute between the stakeholders as to what happened.

The process described above is an upgrade over practices of the past, and a testament to the usefulness of blockchains, but it can be taken a step further and become fully automated thanks to IoT. Assume that every stakeholder carries a smart tracker with (a) a BLE radio, (b) a GSM or LTE radio so that it can connect to the Internet, (c) an installed blockchain client. A similar tracker is also mounted to the container. When the two stakeholders meet *and* the container is also present, for example at point A, the devices of the stakeholders can send signed transactions to the blockchain automatically without any user input, and the process can move to the next stage as soon as the required tokens have been exchanged. In our setup the BLE radio is needed so that the devices can tell when they are in proximity of each other, and when that happens they can transact on the blockchain via the Internet. This is just one possible configuration out of many. Filament [72], for instance, provides sensors with long-range radios called "Taps". Taps can form mesh networks, communicate with each other in a distributed and secure manner via a protocol called telehash [73], and interact with each other via smart contracts on a common blockchain. The sensors themselves do not connect to the Internet to cut down on deployment costs but can connect to gateway nodes that provide such connectivity.

IV. Deployment considerations

In this section we identify several issues that may come up when IoT makers experiment further with blockchains, and have their IoT devices participate in a blockchain network. We comment on possible ways to overcome these issues, or highlight the ongoing work that is being done to address them, where applicable.

Compared to a properly configured centralized database, a blockchain solution will generally underperform, resulting in *lower transaction processing throughput* and higher latencies. We refer the reader to [40], which focuses on the scalability aspect of consensus mechanisms, but also touches upon the issue of performance. This problem is particularly pronounced in public networks, where proof-of-work mechanisms are deployed (see Section II-E and II-B for more), although new proposals, such as Bitcoin-NG [74], show promising results. In general, this performance hit is the penalty paid for trustless decentralization and resiliency. In a blockchain, each node performs the same task leaving no room for parallel task execution, i.e. we do not have sharding. This situation is even more pronounced in blockchains that do smart contracts because of the concurrency issues noted in Section II-E. Work towards shardability is being done for at least one major blockchain platform [67], though a working and tested implementation is still ways off; see Ethereum Improvement Proposal 105 [75].

Maintaining *privacy* on the blockchain is a complicated issue. Recall that each participating device is identified by their public key (or its hash). A participant does not need to know everybody else's key; they just need the key of their transacting counterparty[10]. However, all the transactions in a blockchain happen in the open. By analyzing this data, an interested party can identify patterns and create connections between addresses, and in the end make informed inferences about the actual identities behind them [76]–[79]. A couple of ways to mitigate –but not completely eliminate– this issue, *if* privacy is important for the considered application:

1) Have your device use a new key for every transaction, or use a different key per transacting counterparty, to make pattern identification difficult. Refer to the BIP0032 standard on "hierarchical deterministic wallets" on Bitcoin for instance [80], that allows the derivation of an infinite number of public keys in a manageable and *safe* manner. An issue with the "new key for every transaction" method is that this new key has to be communicated to the interested counterparty for every transaction; a potentially cumbersome and time-consuming process.

[9] See Greenspan's excellent primer on "delivery-versus-payment" and how it is facilitated by blockchains [71].

[10] Do note however that, in a private network, whoever maintains the access list should know every member's identity, otherwise vetting is not possible.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2016.2566339, IEEE Access

2) In the case of private blockchains, it is advisable to not use the same blockchain for all transactions if another participant may get a competitive advantage by tracking your device's activity. Minimize the exposure of your device, by setting up blockchains only with those entities it needs to collaborate with, and only use them for these processes you want to collaborate on. This admittedly increases the coordination cost compared to a single blockchain for everything, but it is a necessary trade-off for increased privacy. [24] provides a look into methods of blockchain analysis and into ways to prevent these analysis techniques from succeeding.

On the same note, transactional privacy (i.e. *confidentiality*) is also hard to attain, since the content of every transaction is exposed to every node on the network, so that it can be validated. As noted in [36], homomorphic encryption might be one way to tackle this; the Elements Alpha [81] experimental chain allows for confidential transactions using additively homomorphic commitments [82]. Zero-knowledge proofs, a cryptographic primitive that allows one party to prove to another party the validity of a statement without revealing its content, might be another; refer to the Hawk model [83] for more info. These methods however are resource intensive so their applicability on resource-constrainted IoT devices might be limited. As in the case of maintining privacy, a blockchain that is set up to serve a very specific process and is discarded after that use might be an acceptable workaround.

Another issue to consider when deploying (or participating) in a blockchain network is deciding on (or examining) the miner set. Recall (Section II-B) that while a miner cannot fake a transaction or rewrite history, it *can* prevent a new, valid transaction from being added to the blockchain, effectively censoring it. The tolerance of a consensus mechanism against Byzantine nodes is limited; if the number of miners that conspire violates that threshold, the risk of transaction censorship is severe. The nodes of the mining set need to be selected wisely so that the chances of collusion between them are minimized. In a private network, legal contracts should be signed so that collusions are penalized appropriately.

Legal enforceability of smart contracts is limited. Work is being done [84]) to make the technical rules of smart contracts legally enforceable and binding to all parties. Until then, what happens if, despite the verifiability of the whole process, a transacting entity disputes the outcome of a smart contract operation? A way to increase the chances of legal enforceability is to include a reference to the actual real-world contract in the smart contract, and vice versa. This is a process called "dual integration" [85] and it works as follows: (a) deploy the smart contract in question, record its address on the blockchain, and include that address in the real contract (b) hash the corresponding real-world contract, record its hash digest, store the real contract in a safe space (can be centralized, or decentralized [86]), (c) send a transaction to the smart contract that includes the real contract's hash in its metadata; the contract then stores that piece of information in its own, internal database[11]. In case of a legal dispute, you can point to the hash stored in the smart contract, then present the real-world contract (that is uniquely identified by that hash) and prove the link between the actions on the blockchain and the expected outcome in the physical world. Refer to CommonAccord [87] and Legal Markdown [85]; both tools intend to make the creation of legal "real-world" and corresponding smart contracts possible, via the use of templating systems.

Tangential to this is the issue of the *expected value of tokenized assets*. Blockchains are used to trade these tokens because they are associated with some value. However, if your device assumes ownership of a token on the chain, and you wish to redeem that token in the real world (e.g. receive cash), what assurances do you have that this will happen? In a blockchain that does not support smart contracts, dual integration is not option. Maybe the answer lies in a similar approach, that hashes a real-world contract and embeds this hash as metadata on the token that is being traded (i.e. notarization by hash [88]. At any rate, the participants need to examine beforehand who stands behind the exchanged assets, and assess the assurances they have about their value.

Complete autonomy is a double-edged sword. Before deploying a smart contract on the chain, one should inspect its logic carefully; they may also want to include fail-safe mechanisms in the code to prevent dead-ends. As we noted in Section II-D, we can have smart contracts (or DAOs) whose overall behavior may change based on user input. Or there may be a function that allows a priviledged user (identifiable by their key) to destroy the deployed contract and remove it from the blockchain's distributed VM. However, if none of these provisions are taken, we are dealing with a system that can never be modified. This by itself may not be a bad thing. If however some function on that contract is written incorrectly, any interactions with it cannot be undone. As a simple example, consider a smart contract that is supposed to act as a deposit box on the chain. You can deposit funds (units of a cryptocurrency) to it, and you can also withdraw funds from it. Whoever deployed it on the blockchain did not include any fail-safe measures, such as a "selfdestruct" function that would allow one to remove the contract and collect its funds [89]. If the contract's "withdraw" function is written incorrectly (by mistake), any funds deposited in it are irrevocably gone and cannot be recovered.

Finally, a blockchain network may also need the following mechanisms to complement its functionality – these need to be decentralized so as not to distort the network's character:

- A DNS service that holds pointers to resources. Blockstack [90], [91], for example, provides such a service on the Bitcoin network. A user sends an appropriately-encoded transaction on the Bitcoin blockchain to create or modify a record on the Blockstack service. Blockchain's nodes filter the blockchain for sequences of data that correspond to valid Blockstack transactions, and use them to modify their name database accordingly.

[11] This requires a smart contract with a properly-coded function that allows an identifiable (via public key) user to store some data in the contract.

- Secure communication and file exchange. As we noted above, messages in the blockchain are read by every network participant. Whenever a private communication channel is needed, a protocol such as telehash [73], [92] or Whisper [68] should be used instead. The network's file-sharing needs may be addressed by a content-addressed P2P file system such as IPFS [57], [86].

V. Conclusions

As we have demonstrated, the combination of blockchains and IoT can be pretty powerful. Blockchains give us resilient, truly distributed peer-to-peer systems and the ability to interact with peers in a trustless, auditable manner. Smart contracts allow us to automate complex multi-step processes. The devices in the IoT ecosystem are the points of contact with the physical world. When all of them are combined we get to automate time-consuming workflows in new and unique ways, achieving cryptographic verifiability, as well as significant cost and time savings in the process.

We believe that the continued integration of blockchains in the IoT domain will cause significant transformations across several industries, bringing about new business models and having us reconsider how existing systems and processes are implemented.

References

[1] J. Kelly and A. Williams. (2016) Forty big banks test blockchain-based bond trading system. [Online]. Available: http://www.nytimes.com/reuters/2016/03/02/business/02reuters-banking-blockchain-bonds.html

[2] I. Kar. (2016) Estonian citizens will soon have the worlds most hack-proof health-care records. [Online]. Available: http://qz.com/628889/this-eastern-european-country-is-moving-its-health-records-to-the-blockchain/

[3] W. Suberg. (2015) Factoms latest partnership takes on us healthcare. [Online]. Available: http://cointelegraph.com/news/factoms-latest-partnership-takes-on-us-healthcare

[4] S. Lacey. (2016) The energy blockchain: How bitcoin could be a catalyst for the distributed grid. [Online]. Available: http://www.greentechmedia.com/articles/read/the-energy-blockchain-could-bitcoin-be-a-catalyst-for-the-distributed-grid

[5] D. Oparah. (2016) 3 ways that the blockchain will change the real estate market. [Online]. Available: http://techcrunch.com/2016/02/06/3-ways-that-blockchain-will-change-the-real-estate-market/

[6] A. Mizrahi, "A blockchain-based property ownership recording system," 2015. [Online]. Available: http://chromaway.com/papers/A-blockchain-based-property-registry.pdf

[7] M. Walport, "Distributed ledger technology: beyond block chain," UK Government Office for Science, Tech. Rep., 19 Jan. 2016. [Online]. Available: https://www.gov.uk/government/publications/distributed-ledger-technology-blackett-review

[8] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[9] Double-spending - bitcoin wiki. [Online]. Available: https://en.bitcoin.it/wiki/Double-spending

[10] Eris Industries documentation - blockchains. [Online]. Available: https://docs.erisindustries.com/explainers/blockchains/

[11] G. Greenspan. (2015) Ending the bitcoin vs blockchain debate. [Online]. Available: http://www.multichain.com/blog/2015/07/bitcoin-vs-blockchain-debate/

[12] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Crypto-Currencies*, 1st ed. O'Reilly Media, Inc., 2014.

[13] (2005) Understanding public key cryptography. [Online]. Available: https://technet.microsoft.com/en-us/library/aa998077(v=exchg.65).aspx

[14] G. Greenspan. (2015) Avoiding the pointless blockchain project. [Online]. Available: http://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/

[15] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 7 Mar. 2002, pp. 251–260. [Online]. Available: http://link.springer.com/chapter/10.1007/3-540-45748-8_24

[16] "Announcing the secure hash standard," 1 Aug. 2002. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf

[17] Hashcash- bitcoin wiki. [Online]. Available: https://en.bitcoin.it/wiki/Hashcash

[18] J.-P. Aumasson, L. Henzen, W. Meier, and R. C. W. Phan, "SHA-3 proposal BLAKE," 16 Dec. 2010. [Online]. Available: https://131002.net/blake/blake.pdf

[19] C. Percival. Tarsnap - the scrypt key derivation function and encryption utility. [Online]. Available: http://www.tarsnap.com/scrypt.html

[20] Block hashing algorithm - litecoin wiki. [Online]. Available: https://litecoin.info/Block_hashing_algorithm

[21] Myriad. a coin for everyone. [Online]. Available: http://myriadcoin.org/home

[22] V. Buterin. (2014) On stake. [Online]. Available: https://blog.ethereum.org/2014/07/05/stake/

[23] V. Buterin. (2014) Slasher ghost, and other developments in proof of stake. [Online]. Available: https://blog.ethereum.org/2014/10/03/slasher-ghost-developments-proof-stake/

[24] F. Tschorsch and B. Scheuermann, "Bitcoin and beyond: A technical survey on decentralized digital currencies," *IACR Cryptology ePrint Archive*, vol. 2015, p. 464, 2015.

[25] T. Swanson, "Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems," R3, Tech. Rep., 6 Apr. 2015. [Online]. Available: http://www.ofnumbers.com/2015/04/06/consensus-as-a-service-a-brief-report-on-the-emergence-of-permissioned-distributed-ledger-systems/

[26] M. Castro, B. Liskov, and Others, "Practical byzantine fault tolerance," in *OSDI*, vol. 99, 1999, pp. 173–186.

[27] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982. [Online]. Available: http://doi.acm.org/10.1145/357172.357176

[28] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[29] hyperledger/fabric: Blockchain fabric incubator code. [Online]. Available: https://github.com/hyperledger/fabric

[30] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIX ATC'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 305–320. [Online]. Available: http://dl.acm.org/citation.cfm?id=2643634.2643666

[31] Juno: Smart contracts running on a bft hardened raft. [Online]. Available: https://github.com/buckie/juno

[32] Blockchain app development simplified tendermint. [Online]. Available: http://tendermint.com/

[33] (2015) Tendermint vs PBFT tendermint. [Online]. Available: http://tendermint.com/posts/tendermint-vs-pbft/

[34] Ripple. [Online]. Available: https://ripple.com/

[35] D. Schwartz, N. Youngs, and A. Britto, "The ripple protocol consensus algorithm," Ripple Labs Inc, Tech. Rep., 2014. [Online]. Available: https://ripple.com/files/ripple_consensus_whitepaper.pdf

[36] G. Greenspan, "MultiChain private blockchain – white paper," 2015. [Online]. Available: http://www.multichain.com/white-paper/

[37] Multichain - open source private blockchain platform. [Online]. Available: http://www.multichain.com/

[38] C. Cachin, S. Schubert, and M. Vukolić, "Non-determinism in byzantine fault-tolerant replication," *arXiv preprint arXiv:1603.07351*, 2016.

[39] M. Kapritsos, Y. Wang, V. Quema, A. Clement, L. Alvisi, and M. Dahlin, "All about eve: Execute-verify replication for multi-core servers," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, pp. 237–250.

[40] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. bft replication," in *Proc. IFIP WG 11.4 Workshop on Open Research Problems in Network Security (iNetSec 2015)*, 2015. [Online]. Available: http://www.vukolic.com/iNetSec_2015.pdf

[41] R. Brown. (2015) How to explain the value of replicated, shared ledgers from first principles. [Online]. Available: http://gendal.me/2015/04/27/how-to-explain-the-value-of-replicated-shared-ledgers-from-first-principles/

[42] P. Franco, *Understanding Bitcoin: Cryptography, engineering and economics*. John Wiley & Sons, 2014.

[43] Controlled supply - bitcoin wiki. [Online]. Available: https://en.bitcoin.it/wiki/Controlled_supply

[44] N. Szabo. (1994) Smart contracts. [Online]. Available: http://szabo.best.vwh.net/smart.contracts.html

[45] N. Szabo. (1997) The idea of smart contracts. [Online]. Available: http://szabo.best.vwh.net/smart_contracts_idea.html

[46] MySQL reference manual - using stored routines (procedures and functions). [Online]. Available: http://dev.mysql.com/doc/refman/5.7/en/stored-routines.html

[47] Eris Industries documentation - smart contracts. [Online]. Available: https://docs.erisindustries.com/explainers/smart_contracts/

[48] R. Brown. (2015) A simple model for smart contracts. [Online]. Available: http://gendal.me/2015/02/10/a-simple-model-for-smart-contracts/

[49] V. Buterin. (2016) Thoughts on UTXOs. [Online]. Available: https://medium.com/@ConsenSys/thoughts-on-utxo-by-vitalik-buterin-2bb782c67e53

[50] Solidity documentation. [Online]. Available: http://solidity.readthedocs.org/en/latest/

[51] White paper - ethereum/wiki. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper

[52] Permissions management - multichain. [Online]. Available: http://www.multichain.com/developers/permissions-management/

[53] G. Greenspan. (2015) Smart contracts: The good, the bad and the lazy. [Online]. Available: http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/

[54] G. Greenspan. (2015) Private blockchains are more than "just" shared databases. [Online]. Available: http://www.multichain.com/blog/2015/10/private-blockchains-shared-databases/

[55] P. Brody and V. Pureswaran, "Device democracy: Saving the future of the internet of things," IBM Institute for Business Value, Tech. Rep., Sep. 2014. [Online]. Available: http://www-935.ibm.com/services/us/gbs/thoughtleadership/internetofthings/

[56] J. Angwin. (2015) Own a Vizio Smart TV? its watching you. [Online]. Available: https://www.propublica.org/article/own-a-vizio-smart-tv-its-watching-you

[57] J. Benet. IPFS - content addressed, versioned, P2P file system (DRAFT 3). [Online]. Available: https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[58] IPFS - content addressed, versioned, p2p file system. [Online]. Available: https://github.com/ipfs/papers/raw/master/ipfs-cap2pfs/ipfs-p2p-file-system.pdf

[59] J. Benet. (2015) Replication on IPFS – or, the backing-up content model. [Online]. Available: https://github.com/ipfs/faq/issues/47

[60] J. Benet. (2015) ipfs-cluster - tool to coordinate between nodes. [Online]. Available: https://github.com/ipfs/notes/issues/58

[61] HTTP/1.1: Status code definitions. [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[62] Filecoin - a cryptocurrency operated file storage network. [Online]. Available: http://filecoin.io/

[63] J. Granata, A. Fathalian, M. Goldstein, H. Haims, S. Carignan, M. Storus, and B. S. Srinivasan. (2016) The first micropayments marketplace. [Online]. Available: https://medium.com/@21/the-first-micropayments-marketplace-38c321127d12

[64] EtherAPIs: decentralized, anonymous, trustless APIs. [Online]. Available: https://etherapis.io/

[65] S. Yassami, N. Drego, I. Sergeev, T. Julian, D. Harding, and B. S. Srinivasan. (2016) True micropayments with Bitcoin. [Online]. Available: https://medium.com/@21/true-micropayments-with-bitcoin-e64fec23ffd8

[66] Slock.it - Blockchain + IoT. [Online]. Available: https://slock.it/faq.md

[67] Ethereum Frontier. [Online]. Available: https://www.ethereum.org/

[68] Whisper overview - ethereum/wiki wiki. [Online]. Available: https://github.com/ethereum/wiki/wiki/Whisper-Overview

[69] Transactive Grid. [Online]. Available: http://transactivegrid.net/

[70] A. Rutkin. (2016) Blockchain-based microgrid gives power to consumers in new york. [Online]. Available: https://www.newscientist.com/article/2079334-blockchain-based-microgrid-gives-power-to-consumers-in-new-york/

[71] G. Greenspan. (2015) Delivery versus payment on a blockchain. [Online]. Available: http://www.multichain.com/blog/2015/09/delivery-versus-payment-blockchain/

[72] Filament - large-scale wireless networks. [Online]. Available: http://filament.com/

[73] telehash - encrypted mesh protocol. [Online]. Available: http://telehash.org/

[74] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, "Bitcoin-NG: A scalable blockchain protocol," 7 Oct. 2015. [Online]. Available: http://arxiv.org/abs/1510.02037

[75] EIP 105 (Serenity): binary sharding plus contract calling semantics. [Online]. Available: https://github.com/ethereum/EIPs/issues/53

[76] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, "A fistful of bitcoins: characterizing payments among men with no names," in *Proceedings of the 2013 conference on Internet measurement conference*. ACM, 23 Oct. 2013, pp. 127–140. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2504730.2504747

[77] D. Ron and A. Shamir, "Quantitative analysis of the full bitcoin transaction graph," in *Financial Cryptography and Data Security*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1 Apr. 2013, pp. 6–24. [Online]. Available: http://link.springer.com/chapter/10.1007/978-3-642-39884-1_2

[78] T. Robinson. (2015) Bitcoin is not anonymous. [Online]. Available: http://www.respublica.org.uk/disraeli-room-post/2015/03/24/bitcoin-is-not-anonymous/

[79] Coinalytics - blockchain intelligence. [Online]. Available: http://coinalytics.co/

[80] BIP0032: Hierarchical deterministic wallets. [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

[81] Elementsproject/elements: Feature experiments to advance the art of bitcoin. [Online]. Available: https://github.com/ElementsProject/elements

[82] G. Maxwell. Confidential transactions. [Online]. Available: https://www.elementsproject.org/elements/confidential-transactions/

[83] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," Cryptology ePrint Archive, Report 2015/675, 2015. http://eprint. iacr. org, Tech. Rep., 2015. [Online]. Available: http://eprint.iacr.org/2015/675.pdf

[84] P. de Filippi. (2015) Legal framework for crypto-ledger transactions. [Online]. Available: http://p2pfoundation.net/Legal_Framework_For_Crypto-Ledger_Transactions

[85] Eris Industries - eris:legal. [Online]. Available: https://erisindustries.com/components/erislegal/

[86] IPFS is a new peer-to-peer hypermedia protocol. [Online]. Available: https://ipfs.io/

[87] Commonacord - bringing the world to agreement. [Online]. Available: http://www.commonaccord.org/

[88] Proof of existence. [Online]. Available: https://www.proofofexistence.com/

[89] Introduction to smart contracts solidity 0.2.0 documentation. [Online]. Available: http://solidity.readthedocs.org/en/latest/introduction-to-smart-contracts.html

[90] M. Ali, J. Nelson, R. Shea, and M. J. Freedman. (2016, Feb.) Blockstack: Design and implementation of a global naming system with blockchains. [Online]. Available: https://blockstack.org/blockstack.pdf

[91] Blockstack - how blockstack works. [Online]. Available: https://blockstack.org/docs/how-blockstack-works

[92] J. Thijssen. (2013) Telehash: an encrypted p2p network for your apps. [Online]. Available: https://adayinthelifeof.nl/2013/11/12/telehash-an-encrypted-p2p-network-for-your-apps/