decode

Survey of Technologies for ABC, Entitlements and Blockchains

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no. 732546

decode

Project no. 732546

DECODE

DEcentralised Citizens Owned Data Ecosystem

D3.1 Survey of Technologies for ABC, Entitlements and Blockchains

Version Number: V1.0

Lead beneficiary: University College London

Due Date: June 2017

Author(s): Mustafa Al-Bassam (University College London), Shehar Bano (University College London), George Danezis (University College London), Mark deVilliers (Thingful), Alberto Sonnino (University College London)

Editors and reviewers: Jaromil Rojo (Dyne), Jaap-Henk Hoepman (Radbound University)

Dissemination level:		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Approved by: Francesca Bria (Chief Technology and Digital Innovation Officer, Barcelona City Hall)

Date: 29/06/2017

This report is currently awaiting approval from the EC and cannot be not considered to be a final version.

Table of Contents

DECODE provides a distributed architecture that stores data about cities and people in a secure and privacy-friendly way. This involves designing a distributed ledger (or blockchain) that is secure and scalable, and fulfills requirements for availability, integrity and privacy. It is therefore important to understand how transparency and privacy can be balanced in distributed ledgers, and how to design scalable blockchain solutions. A related goal is to enable users to define rules that specify how their data should be used. Key challenges here are secure generation and retrieval of credentials that are stored in the blockchain, how to combine attribute-based encryption and attribute-based credentials, and the ability to anonymously conduct transactions. In this document, we present a survey of data entitlements (Section 1), blockchain techniques to provide appropriate options for DECODE usecases (Section 2), and privacy technologies (such as Attributed Based Credentials—shortened to ABC) relevant to blockchains (Section 3).

1. Data Entitlements

> "One of the promises of the Internet of Things (IoT) is that everything should talk to everything else. These talkative "things" include sensors, consumer appliances, home automation systems, and even connected vehicles. The frameworks through which such interconnectivity is arranged, controlled, and mediated—that is, how these things "entitle" each other to connect—is going to be a fundamental part of IoT. Managing this entitlement, which defines who can access our device data, and under what conditions it can be found and used by others, will be one of the major challenges for consumers and businesses." [27]

Data entitlements could be thought of as an evolution of a traditional authorization scheme specialized for the securing of both personal, business and IoT data. Giving the data owner full control of the access and discovery of their data creates a system of empowerment whose currency is privacy. Privacy is a fundamental right. The relationship of privacy and data entitlement is subject to many nuances. Users might allow a party to search and access their data, but only under the understanding that they will not be personally identified or only under specific circumstances. For example, a driver of a motor vehicle with an onboard camera might want to entitle the emergency services access to his data when there is a road traffic accident.

However, the driver might not want to be identified as being in a certain geographical area or that he was exceeding the speed limit at the time. Data, when available, can directly or indirectly compromise privacy in a way that would surprise a user.

1.1. Background and Concepts

When discussing data entitlements, it first necessary to understand the concept of authentication, its relationship to authorization and their combined application as the basis for access control. Wikipedia defines authentication and authorization as follows [49]:

> "In contrast with identification, which refers to the act of stating or otherwise indicating a claim purportedly attesting to a person or thing's identity, authentication is the process of actually confirming that identity. It might involve confirming the identity of a person by validating their identity documents, verifying the authenticity of a website with a digital certificate, determining the age of an artifact by carbon dating, or ensuring that a product is what its packaging and labeling claim to be. In other words, authentication often involves verifying the validity of at least one form of identification."

> "The process of authorization is distinct from that of authentication. Whereas authentication is the process of verifying that 'you are who you say you are', authorization is the process of verifying that 'you are permitted to do what you are trying to do'. This does not mean authorization presupposes authentication; an anonymous agent could be authorized to a limited action set."

Once a computer system has authenticated an entity, the process of authorization is usually enabled by enforcing a series of policies. The concept of policy management is well understood among vendors of centralized software systems [39]. Gartner defines the concept of entitlement management as follows [23]:

> "Entitlement management is technology that grants, resolves, enforces, revokes and administers fine-grained access entitlements (also referred to as 'authorizations', 'privileges', 'access rights',

'permissions' and/or 'rules'). Its purpose is to execute IT access policies to structured/unstructured data, devices and services. Entitlement management can be delivered by different technologies, and is often different across platforms, applications, network components and devices."

1.2. Entitlements in Decentralized Systems

In the context of DECODE, data entitlement could be described as the definition, management and application of authorization policies. The DECODE architecture is inherently distributed and as such the management of data entitlements may contain certain challenges not present in a traditionally centralized system. These challenges have been discussed below:

Challenge of embedded decisions. The authorization policy for a piece of data would need to be consistent wherever that data is stored. Data in a distributed system may be sharded (i.e., split by a logical organization) or replicated (many copies) or as is more usual, both. A change made to the authorization policy would need to be replicated alongside the data it governs to allow for local decision making on authorizations against the distributed data. A desirable property of this replication is consistency—all nodes holding the data need to enforce the same policy.

Challenge of lack of overview. Distributed authorization policies (e.g. those related to data created from other data) make it difficult to gather and understand policies governing the data. If the data is subject to a data entitlement policy regarding any derived data and any of its derived data, there are significant problems in tracing and tracking the policies and data.

Challenge of identity integration. A data entitlement system within the context of a distributed system may need to interface with one of many identity systems. These identity systems may or may not outlive lifetime of the data. Consider the impact of an identity provider ceasing to exist and orphaning access to data when a user cannot prove who they are.

Challenge of expression. A formal expression of a data entitlement should have a rich model for the expression of access and discovery. Rights can be expressed against the data, only part of the data, the derived data and to a singular party or groups of parties. It might capture aspects of temporal access (at certain times) or situational access (only in an emergency). It might wish to capture some aspects of differential privacy and anonymization.

1.3. Formal Policies

In a decentralized infrastructure, it is worthwhile considering a formal format for expressing the policy of a data entitlement. This format would, if complete, allow for decisions to be made local to the data without consultation of a centralised third party, as stated by Woo et al. [50]:

> "In most existing systems, authorization is specified using some low-level system-specific mechanisms, e.g., protection bits, capabilities and access control lists. We argue that authorization is an independent semantic concept that must be separated from implementation mechanisms and given a precise semantics. We propose a logical approach to representing and evaluating authorization. Specifically, we introduce a language for specifying policy bases. A policy base encodes a set of authorization requirements and is given a precise semantics based upon a formal notion of authorization policy."..."a set of authorization requirements is specified declaratively by a policy base. Unlike most existing approaches, the semantics of authorization is defined independently and is separate from implementation mechanisms."

1.4. Approaches to Entitlements

Although there is a wide range of related work this document will consider UNIX file permissions from Bell Labs, XACML from OASIS, P3P from the W3C, S4P which builds on the work of SECPal, both Microsoft Research, and Soutaei from Planning Systems and Fleet Numerical Meteorology and Oceanography Center as relevant to the DECODE project.

The UNIX [43] file system implements authorisation with such brevity and power of expression that is is worth considering although it does not contain the higher level constructs of the other systems we will examine. The UNIX file system applies the building blocks of read, write and execute for both the owner of the file, the group the file belongs to and for everyone else. Additionally through the application of the Set User ID (SUID) and Set Group ID (SGID) bit an operation can gain or lose temporary permissions. For each file, directory and device this information is held in 12 bits.

XACML (eXtensible Access Control Markup Language) [38] first appeared in 2001 and has gained adoption mostly in the enterprise. XACML is a XML based domain specific language that formalises an Attribute-Based Access Control system (ABAC). XACML systems have mostly been designed

and developed as centralised authorities within an organisation. XACML proposes a novel system of obligations. An obligation is a directive that must be carried out before, on or after the authorisation event. In the (pseudo-text) example below, actions are defined on the outcome of the access request.

```
Allow access to resource MedicalJournal with attribute patientID=x
               if Subject match DesignatedDoctorOfPatient
               and action is read
         with obligation
              on Permit: doLog_Inform(patientID, Subject, time)
              on Deny  : doLog_UnauthorizedLogin(patientID, Subject, time)
```

Whilst XACML is both an architecture and a policy specification it is the policy dimension that is particularly of interest with features such as obligation, time based access and high level predicates. Reliance on a centralized point of authority for authorizations is a common XACML pattern. The design is followed by the UMA (User Managed Access) project from Katara [29]. UMA builds on the OAUTH2 specification enabling "privacy controls that are individual-empowering" and might be thought of as a more modern form of the XACML based systems. XACML has also been used as the base for other privacy related works including PAPEL (A Language and Model for Provenance-Aware Policy Definition and Execution) [44]. PAPEL exposes a provenance model, for example a user's health records can be read but only for the purpose of research with their names anonymized. PAPEL also introduces a traceability of compliance dimension via the exposing of *attributes* and *reduced* facts.

P3P [17] proposed a machine readable format to express how an organization (accessed via the Internet) will attempt to protect and use third-party data. The organization states how it may treat data once it has been gathered using a fixed vocabulary. P3P is web centric in its scope and the fixed vocabulary limits its usefulness when applied to the definition of data entitlement policies. Furthermore it does not support the use case of the data owner being able to express a nuanced policy specific to their data.

SecPAL [7] proposed a high level authorization format. The format consists of the application of one or more statements and predicates. Similar to the expressions found in a constraint logic programming language, the intent is expressed in an almost natural language format. SecPAL's major advancement is to guarantee that as long as certain syntactic conditions were met that the logic would terminate (e.g., be computed). In the example below, Admin is an authority of the system (e.g., locally or centrally) and Alice can

only access the data if she is a student. The policy is valid for the period on 1 year from when it has been set.

```
Admin says Alice is a student until 31/7/2007
Admin says Alice is entitled to access temperature data from my sensor" if
Alice is a student till date,
            currentTime () <= date,
            date - currentTime () <= 1 year
```

Implicit in the effective application of the above example is the means to revoke Alice's role as a student if for instance Alice had to leave the university unexpectedly. SecPAL suggests giving each authorization a unique identifier (ID) and introduces the "revoke" keyword.

```
Admin says Admin revokes ID if currentTime () > 31/6/2007
```

Multiple policy idioms can be expressed within SecPAL. It is possible to define a policy allowing other users to delegate their credentials to other users. In the example below, Bob gives Alice the ability to read the file foo/bar:

```
FileServer says user can say x can access resource if user can access resource
FileServer says Bob can read file://foo/
Bob says Alice can read file://foo/bar
```

SecPal introduces a verb "can act as" to implement role membership and role hierarchies. In the example below, Alice can read `file://docs/`:

```
NHS says FoundationTrainee can read file://docs/
NHS says SpecialistTrainee can act as FoundationTrainee
NHS says SeniorMedPractitioner can act as SpecialistTrainee
NHS says Alice can act as SeniorMedPractitioner
```

Soutei [42] is a logic-based trust-management system that proposes that policies live within a Closed World Assumption—a formal system of logic based on the presumption that a statement that is true is also known to be true and conversely what is not known to be true is false. A core design idea behind Soutei is that an entity or a right needs to be defined before usage. The paper states that:

> "Soutei policies and credentials are written in a declarative logic-based security language and thus constitute distributed logic programs. Soutei policies are modular, concise, and readable. They support policy verification, and, despite the simplicity of the language, express role- and attribute-based access control lists, and conditional delegation."

Soutei describes a system of efficiently resolving authorization requests without loading the complete archive of authorization rules and maintaining resilience against careless, incomplete or malicious rule sets. An example rule set is given below that allows access only from the IP range 192.168.0.0/8 (the first example is followed by its pesudo-text equivalent):

```
may(?access) :-
          application says ip-address(?IP),
          application says ip-of(?IP, #n192.168.0.0/8),
          administrator(?admin),
          ?admin says may(?access).
known-access(read).
known-access(write).

Define access as read
Define access as write
If request has an IP address
          If request is in range 192.168.0.0/8
                    If there is an administrator called admin
                              Admin says they can read or write
```

S4P [6] builds on SecPAL and proposes a formalized language to distinguish between service behaviours (how user data is going to be used) and user preferences (how users would like their data to be used). S4P suggests that this separation would allow for some interesting capabilities. A user could calculate if the the service would meet their own preferences and a service would know the reverse—do existing policies match the user's expectations? S4P proposes this separation would allow a service to evolve their policies in a way that would remain in compliance with the user's policy.

2. Blockchain Technologies

DECODE provides a distributed and scalable architecture for decentralized data governance and federated identities. In a decentralized system, control of information stored or processed on the system is not concentrated in any single entity. We explore feasibility of blockchain technology to implement DECODE's decentralized backend. Blockchain represents an immutable and decentralized database that facilitates transparent management of data. In this section, we present concepts and components related to blockchains, and then provide an overview of design approaches with focus on scalability.

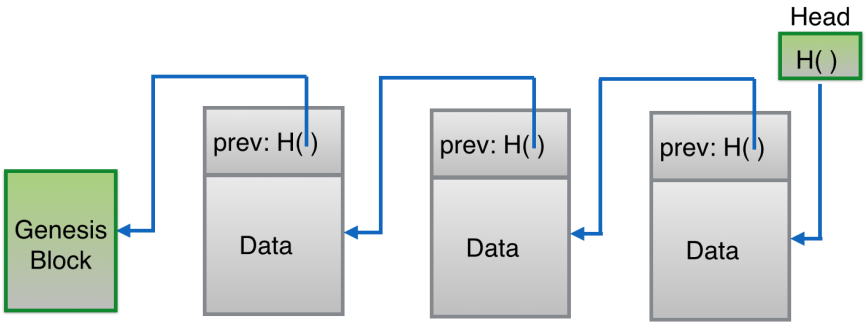

Figure 1: Blockchain is a linked list of hash pointers.

2.1. Background and Concepts

In this section, we discuss key components of blockchains and related concepts. To provide concrete examples, we refer to Bitcoin [37]. Bitcoin is a widely known cryptocurrency based on blockchain that organizes nodes in a peer-to-peer (p2p) network; any node can join and become part of the network. If a node receives new information, it broadcasts it to rest of the network. While all nodes listen to and broadcast information, only special nodes can append information to the blockchain.

Blockchain. The building block of blockchain is hash pointer (Figure 1). A hash pointer is simply some information (typically called transaction) along with a hash of the information. The hash serves to identify the information, and also allows verification of its integrity. A blockchain is a linked list of hash pointers, where pointers have been replaced with hash pointers. The first block in the chain points to a special block called the genesis block. Each block contains a hash of the previous block and information specific to the current block. A key result of iterative hashing is that a block implicitly verifies integrity of the entire blockchain before it. Thus given the current head of the blockchain, any party can independently verify the entire blockchain by generating hashes from the beginning of the chain up until the end. The final hash should match that in the head, otherwise the blockchain has been tampered with. Effectively, blockchain can be thought of as a tamper-evident log where data can be appended to end of the log, and tampering with previous data in the log can be detected.

Transaction. A transaction specifies some operation on one or more previous blocks in the blockchain. The result of this transformation, subject to pass-

ing validity and verification tests, is a candidate block for being appended to the blockchain. In other words, a transactions represents a function that, if valid, changes the state of the blockchain. In Bitcoin, a typical transaction is transferring money from payer(s) to payee(s)—or spending money with certain inputs (payers) and certain outputs (payees). Payers and payees are identified by public keys; the payer digitally signs the transaction. Special nodes must perform four checks before adding a new transaction to the blockchain. First, they must verify that the payer is authorized to conduct this transaction by checking that the transaction's digital signature corresponds to public key of the payer. Second, the nodes must verify that the sum of outputs is equal to the sum of inputs: payers cannot pay out more than they own. Note that unlike traditional financial transactions, a Bitcoin transaction's aggregate input value must be completely consumed; if the payer wants to only spend part of the input amount and retain the balance, then he should include himself as one of the payees in the transaction output. Third, nodes must check that the transaction itself is well-formed, and that hash of the previous block is correct. This requires verifying all previous blocks in the blockchain. Finally, nodes must ensure that none of the inputs is being double-spent since the payer is allowed to spend an amount only once. This can be verified by traversing back in the blockchain to when the input value was created, and then traverse forward all the way to the current transaction—ensuring that the input has not been previously spent. Note that usually multiple transactions grouped into a 'block' are added to the blockchain rather than individual transactions. Also, for convenience we use the terms input and output values, but in reality Bitcoin money has no physical existence: the fact that Bob owns a certain amount corresponds to the fact that majority of nodes in the system believe this to be the truth. This brings us to our next topic, consensus.

Consensus. The issue of consensus in distributed systems in the presence of faulty or malicious nodes long predates its application in cryptocurrencies and distributed ledgers. In a network with n honest nodes that each receive input values and share them with rest of the network, the consensus protocol enables agreement between all n honest nodes on the set of input values generated by honest nodes. In the Bitcoin context where nodes broadcast transactions as part of a p2p network, nodes need to reach consensus on exactly which transactions took place and in what order—that is, the nodes must agree on state of the blockchain. Consensus is challenging because

nodes might have different views of blockchain due to latency in propagation of transactions over the p2p network, nodes randomly failing, and malicious nodes trying to suppress valid transactions and push invalid transactions to the blockchain. Most consensus protocols have the concept of a leader. The leader is responsible for coordinating with other nodes to reach consensus, and for appending a final, committed value to the blockchain. The leader is usually effective only for a period of time called *epoch*, after which a new leader is elected.

Mining and Incentivization. So far we said that only special nodes can propose blocks to append to the blockchain. To stop dishonest nodes from bringing the system to a stall, these nodes should be chosen in a random way. A common way to enforce random selection of special nodes is through mining or proof-of-work. In Bitcoin, this involves solving a hash puzzle. To propose the next block, a node needs to find a random number (nonce) such that hash of the concatenation of the nonce, the hash pointer to previous block, and content of the current block falls within a specified range.

$$H(nonce||prev\_hash||tx||tx||\ldots||tx) < target$$

Bitcoin miners are busy calculating hashes all the time; if a miner gets lucky by finding a nonce that satisfies the hash puzzle, it proposes the next block. The nonce is included in the block and can be trivially verified by any node. To incentivize miners to solve hash puzzles and propose next blocks, the system should reward them in some way. In Bitcoin, there are two kinds of rewards. The miner that solves the puzzle and proposes the next block gets to pay some amount to itself (block reward). The second incentive mechanism is transaction fee. Recall that we said that the sum of input values must be equal to the sum of output values in a transaction. There is an exception to this rule: the creator of the transaction can make the output value less than the input value; so a miner that proposes the current blocks gets the difference as transaction fee.

Forks. If two miners find solutions to hash puzzles and propose next blocks within a small time interval, nodes will either append block proposed by the first miner or the second, depending on which one they first received on the network. This creates a fork—nodes having different views of state of the blockchain. Forks defy consensus; so some mechanism needs to be in place to resolve such conflicts and get majority of the nodes to agree on state of the

blockchain. This means that until a resolution has been reached, some miners will waste resources mining on top of part of the blockchain that might be eventually discarded (orphaned blocks). Thus it is desirable to minimize forks to curb wastage of mining power and enable the system to make progress. In Bitcoin, the rule is that miners should always extend the longest valid branch of the blockchain. All honest miners are incentivized to follow the same rule so that their proposed blocks end up on the blockchain and they receive their reward. To avoid frequent forks, the system automatically parametrizes hash puzzles in such a way that time between successive blocks (inter-block time) remains at an average of 10 minutes. To be sure that a transaction is included in the consensus chain, a node must wait for several confirmations—announcements from different nodes in the network that the block containing the transaction has been appended to the blockchain. Usually the heuristic of 6 confirmations is used.

Script. Bitcoin has a simple stack-based internal scripting for language for transactions called Script [48]. Scripts are used in transaction outputs to define what data is needed to spend a transaction, by returning *True* on the correct input. For example, the script for a standard Bitcoin transaction returns *True* when presented with a valid signture showing evidence of the private key corrosponding to the public key specified in the script. More advanced scripts may, for example, check multiple signatures and require two of three valid signatures representing three keys.

Smart Contract. Script is very limited in its functionality and does not have features such as while loops, and is not Turing complete. Consequently, other systems such as Ethereum [1] have created a blockchain with an extended and Turing complete scripting system, to allow for more use cases. Ethereum scripts are written in a high-level language (such as Solidity, a Javascript-like langauge for writing smart contracts) and are compiled to low-level Ethereum Virtual Machine (EVM) code. In Ethereum, each transaction has a *gas* cost associated with it that is calculated based on how many computational steps a transaction has. Transactions must then be funded with sufficient *Ether*—Ethereum's cryptocurrency token—to cover the gas cost. These scripts are called *smart contracts*, because they effectively act as traditional contracts that are written in computer code rather than natural language. For example, it is possible to create a smart contract that defines a loan that a debtor can withdraw or deposit into, with the interest calculated automatically and the rules enforced by the blockchain network. However, it is also

possible to create many different kinds of non-financial applications using a smart contract, to achieve decentralization and transparency. For example, it is possible to create a decentralized domain name registration system by defining a smart contract that enables users to register domain names [2].

2.2. Scalable Blockchains

Bitcoin's transaction throughput depends on block size and inter-block interval. With Bitcoin's current block size of 1MB and 10 minute inter-block interval, the maximum throughput is capped at about 7 transactions per second. Moreover, a client that creates a transaction has to wait for at least 10 minutes to be sure that the transaction is included in the blockchain. In contrast, mainstream payment processing companies like Visa confirm transactions within few seconds, and have high throughput of 2000 transactions per second on average, peaking up to 56,000 transactions per second [3]. Current research in the area is focused on developing solutions to significantly improve blockchain performance, so that it is at least at par with mainstream payment systems while retaining its decentralized nature. More specifically, the following aspects of blockchain could be potential performance bottlenecks (adapted from work by Croman et al. [18]):

- **Maximum Throughput:** The maximum rate at which the blockchain can confirm transactions (Bitcoin has 3.3–7 transactions per second).
- **Latency:** Time to confirm that a transaction is included in the blockchain. The minimum time for Bitcoin is 10 minutes assuming single confirmation; but it is recommended to wait for 6 confirmations to have high confidence that the transaction will be included in the blockchain, which translates to latency of about an hour.
- **Bootstrap Time:** Time taken for a new node to download and process part of the blockchain necessary to validate the system's current state. In Bitcoin, bootstrap time is linear in the size of the blockchain, estimated at about 4 days computed on a mid-grade computer [18].
- **Transaction Validation:** Time taken for a node to validate that a transaction can spend the inputs. In bitcoin, this requires traversing back in the blockchain to when the output value referenced by the inputs was created, and then traverse forward until the current transaction is reached—checking that the referenced output value has not

been previously spent. A recent study [18] estimates that transaction validation comprises 0.2% of overall cost incurred by all nodes in Bitcoin.

- **Network Bandwidth:** How much network bandwidth is consumed by nodes to share information (transactions, blocks, metadata) with each other. This is closely related to properties of the consensus mechanism employed by the system.
- **Transaction-Validation Storage:** The amount of storage required per node to enable transaction validation.
- **Bootstrap Storage:** The amount of storage required for new nodes that join the network to store (part of the) blockchain to validate state of the system. In bitcoin, new nodes have to store and validate entire blockchain history which is currently (June, 2017) about 20GB.

Reparametrization of Bitcoin system metrics, block size and inter-block interval, can improve performance to a limited extent—estimated at 27 transactions per second and 12 seconds, respectively. However, significant improvement in performance requires fundamental redesign of blockchain paradigm. Next we identify different design themes that improve blockchain scalability.

2.3. Multiple Blocks per Leader

In this approach, the leader appends multiple blocks to the blockchain until another leader is elected.

Bitcoin-NG. Bitcoin-NG [22] is based on the same trust model as Bitcoin, but improves performance by breaking up Bitcoin's blockchain operation into leader election and transaction serialization. Leader election is similar to Bitcoin, and performed randomly and infrequently via proof-of-work. However, unlike Bitcoin where the leader (miner who solved the puzzle) can only propose 1 block to append to the blockchain, in Bitcoin-NG time is divided into epochs and a leader can unilaterally append multiple transactions to the blockchain for the duration of its epoch which ends when a new leader is elected. The system has two kind of blocks: keyblocks and microblocks. Keyblocks contain solution to the puzzle and are used for leader election. Keyblocks contain a public key which is used to sign subsequent microblocks generated by the leader. Every block contains a reference to the previous microblock and keyblock. Fee is distributed between the current

leader (40%) and the next leader (60%). As in Bitcoin, forks are resolved by extending the longest branch aggregated over all keyblocks. Note that microblocks do not contribute to length of a branch as these do not contain proof-of-work. To penalise a leader that creates forks in the microblocks generated, a subsequent leader can insert a special poison transaction after its keyblock that contains header of the first block in the pruned branch as a proof-of-fraud. This invalidates the malicious leaders reward, a fraction of which is paid to the reporting leader. Forks can also happen when a new leader has been elected but the previous leader has not yet heard about it and continues to generate microblocks. However, such forks are resolved as soon as announcement of election of the new leader reaches all nodes. Bitcoin-NG s performance is evaluated against original Bitcoin client on an emulation testbed comprising 1000 nodes (about 15% of the current operational Bitcoin network). The study compares Bitcoin and Bitcoin-NG in terms of various metrics related to systems performance and security while varying block frequency and block size. While reparametrization improves Bitcoin s performance, its security deteriorates. Bitcoin-NG achieves similar performance while retaining security grantees, qualitatively outperforming Bitcoin [22].

2.4. Sharding Transactions

In this approach, the nodes are organized into groups: each group is responsible for handling only a subset of transactions (shards).

RSCoin [20] is a permissioned blockchain (where part of the system is trusted). The central bank controls all monetary supply, while mintettes (nodes authorized by the bank) manage transactions. Transactions have identifiers, and each mintette is responsible for a subset (shard) of transactions such that a shard can potentially overlap across mintettes for security and reliability. A mintette maintains information about outputs of the transactions it manages, whether these have been spent and if so in which transactions. A user who wants to append a transaction to the blockchain first gets signed clearance from majority of the mintettes corresponding to each input value in the transaction. Next the user sends the transaction and signed clearance from input owners to mintettes corresponding to output values in the transaction. The mintettes check validity of the transactions and verify signed evidence from input mintettes that the transaction is not double-spending any inputs. If the checks pass, the mintettes send evidence to the user that the transaction will be included in the blockchain

(this evidence can be used to implicate the mintettes if the transaction does not appear in the blockchain). The system operates in epochs: at the end of each epoch, mintettes send all cleared transactions to the central bank which collates transactions into blocks that are added to the blockchain. As communication between mintettes takes place indirectly through the user, RSCoin has low communication overhead and improved performance. The transaction throughput scales linearly with the number of mintettes.

Elastico [34] is another system that improves performance by sharding transactions in a permissionless setting, that is assuming a completely decentralized system with no trusted component. Nodes in the network are partitioned into committees, where each committee is responsible for managing a subset (shard) of transactions. Within a committee, nodes run a byzantine consensus protocol (e.g. PBFT) to agree on a block of transactions. If the block has been signed by enough nodes, the committee sends it to a final committee. The final committee collates sets of transactions received from committees into a final block, and runs a byzantine consensus protocol between its members to get agreement and broadcast the final block to other committees. The system operates in epochs: assignment of nodes to committees is valid only for duration of the epoch. At the end of the epoch, nodes compute solution to a puzzle seeded by a random string generated by the final committee and sends the solution to the final committee to be assigned to a committee. As a result, in each epoch a node is paired with different nodes in a committee managing a different set of transactions. The number of committees scales linearly in the amount of computational power available in the system, but the number of nodes within a committee is fixed. Thus the block throughput scales up almost linear to the size of the network. As more nodes join the network, transaction throughput increases without adding to latency as messages needed for consensus are decoupled from computation and broadcast of final block to be added to the blockchain.

OmniLedger [31] uses RandHound [46]—a decentralized randomness protocol—to randomly assign nodes to a shard, making it difficult for an adversary to compromise individual shards. OmniLedger uses a block-DAG (Directed Acyclic Graph) rather than a blockchain, effectively creating multiple blockchains in which consensus of transactions can take place in parallel. To realize parallel consensus, dependencies between transactions are identified from their inputs and outputs. Moreover, transactions are organized in such a way that the block containing a transaction must be a member of the blockchain corresponding to the transaction's inputs. To enable se-

cure validation of cross-shard transactions—transactions that have inputs or outputs corresponding to multiple shards—OmniLedger uses an atomic commit protocol across shards. To commit a transaction to the blockchain, the client first sends the transaction to the network. The leader of each shard that is responsible for the transaction's inputs (input shard) must valdiate the transaction and return a proof-of-acceptance (or proof-of-rejection). The transaction's inputs are then considered to be locked. To unlock transaction inputs, the client collects and sends proof-of-accepts to the output shards, whose leaders add the transaction to the next block to be appended to the blockchain. In case the transaction fails the validation test, the client can send proof-of-rejection to the input shards to roll back the transaction and unlock the inputs.

2.5. Sharding Proof-of-Work

This is an approach for allowing multiple leaders to extend different parts of the blockchain by working on different puzzles in parallel.

Bitcoin has a linear process of transaction verification, where all miners try to solve proof-of-work in parallel. The one that gets lucky proposes the next block; all miners then get busy mining to propose the next block. The framework proposed by **Boyen et al.** [9] parallelizes this process by forgoing the concepts of 'blocks' and 'chain' in favour of a graph of cross-verifying transactions. Each transaction validates two previous transactions (its parents), some payload (e.g. cryptocurrency) and proof-of-work. A transaction can be potentially validated by multiple children nodes. Additionally, each transaction also carries a reward to be collected by the transaction that validates it. Value of the reward decreases as more nodes directly or indirectly validate it, thus new nodes have more incentive to validate recent transactions. The system has been shown to have convergence property, that is at some point there is a transaction that connects to (and thus implicitly verifies) all transactions before it. As a result of the graph structure, different branches of the transactions graph can be extended in parallel by miners who get reward for useful work even for verifying the same transactions. Normal (non-miner) nodes in the system verify transactions as they receive them. In addition to standard checks on correctness of proof-of-work and structural validity of the transaction and its parents, the node also checks that the transaction is not a double-spend by accepting as valid the well-formed transaction that has the largest amount of work attached to it (height).

2.6. Strong Consistency via Collective Leaders

A number of systems employ multiple leaders to collectively and quickly decide if a block should be added to the blockchain. This offers strong consistency to a client that a submitted block will remain on the blockchain. Another advantage is that the blockchain remains fork-free, as all leaders instantly agree on block validity.

ByzCoin [30] improves Bitcoin's transaction latency by replacing its probabilistic transaction consistency guarantees with strong consistency. It builds this design on Bitcoin-NG to achieve high transaction throughput. Recall that Bitcoin-NG operates in epochs where miners compete to find solution to a puzzle, and the winner becomes the current leader that appends blocks to the blockchain for duration of epoch until a new leader is announced. The system has two kinds of blocks: keyblocks announce a new leader and includes proof-of-work, while microblocks are generated by a leader for duration of the epoch to be appended to the blockchain. ByzCoin modifies how keyblocks are generated: a consensus group, instead of a solo leader, generates a keyblock followed by microblocks. The consensus group is dynamically formed by a window of recent miners. Each miner has voting power proportional to the number of mining blocks it has in the current window, which is proportional to its hash power. When a miner finds solution to the puzzle, it becomes a member of the current consensus group and receives a share in the current window which moves one step forwards (ejecting the oldest miner). ByzCoin uses the same incentive model as Bitcoin, however instead of the most recent miner receiving all reward and fee, it is shared between members of the consensus group in proportion to their shares. The cosensus group is organized into a communication tree where the most recent miner (the leader) is at the root. The leader runs the Practical Byzantine Fault Tolerant (PBFT) protocol [13] to get all members to agree on the next microblock. However, it replaces PBFTs $O(n^2)$ MAC-authenticated all-to-all communication with a primitive called scalable collective signining (CoSi) that reduces messaging complexity to $O(n)$. The outcome of running two rounds of PBFT with CoSI is a fixed 64 byte collective signature that proves that at least two-thirds of the consensus group members witnessed and attested the microblock. A node in the network can verify in $O(1)$ that a microblock has been validated by the consensus group. This design addresses a limitation of Bitcoin-NG where a malicious leader can create microblock forks: in ByzCoin this would require two-third majority of consensus group members to be malicious which is unlikely. Similarly, Bitcoin-NG suffers

from a race condition where an old leader that has not heard about the new leader may continue to generate microblocks which will be eventually orphaned (assuming that the new leader mined on top of older microblocks). In ByzCoin, consensus group members ensure that a new leader builds on top of the most recent microblock. In experiments with simulated consensus groups, a consensus group of 144 miners (formed over 24 hours of mining) have transaction latency less than 20 seconds on blocks of size 1MB (Bitcoin s current maximum size). If block size in the former setting is increased to 32MB, transaction throughput reaches 974 transactions per second with a transaction latency of 68 seconds. For a consensus group of 1008 members (formed over a week of mining) with 8MB block size has transaction latency of 90 seconds.

Similar to ByzCoin, **PeerConsensus** [21] also achieves strong consistency by allowing previous miners to vote on blocks. A *Chain Agreement* tracks the membership of identities in the system that can vote on new blocks. The difference between PeerConsensus and ByzCoin is that PeerConsensus uses a standard Practical Byzantine Fault Tolereance (PBFT) protocol [13], which requires participants to receive $O(n)$-size messages, whereas ByzCoin uses a modified PBFT protocol using collective signatures (CoSi) that only requires participants to receive $O(1)$-size messages.

Algorand [35] is a new type of blockchain that attempts to achieve Sybil-resistance in a decentralised way without the use of proof-of-work, with strong consistency. It proposes a faster *graded* Byzantine fault tolerance protocol, that allows for a set of nodes to decide on the next block. One key aspect of Algorand is that these nodes are selected randomly using algorthimic randomness based on input from previously generated blocks. When validating nodes sign new blocks, they destroy the key which was used to sign it, introducing a notion of forward integrity that makes it difficult for an adversary to retrospectively corrupt nodes that have destroyed their keys.

Hyperledger Fabric [10] is a permissioned blockchain system designed by IBM, that allows organizations to setup their own blockchain networks to run smart contracts. It is designed around the idea of a 'consortium' blockchain, where a specific set of nodes are designated to validate transactions as part of a consortium, rather than random nodes in a decentralized network. Smart contracts are called *chaincode* in Hyperledger Fabric, and each chaincode has its own set of *endorser* nodes that re-execute transactions for chaincode received from submitters of transaction, to validate that they are valid and correct. A *consensus service* then orders the endorsed transac-

tions, produces an ordered stream of transactions and filters out transactions that are not endorsed by enough endorsers. Hyperledger Fabric uses *modular consensus*, meaning that the consensus protocol used by endorsers to decide which transactions are valid is replaceable depending on the requirements. For example, Apache Kafka or SBFT (a simple implementation of Practical Byzantine Fault Tolerance protocol) may be used.

3. Privacy

DECODE is a decentralized data repository that provides privacy and transparency through ABC (Attribute Based Cryptography) and other privacy enhancing technologies. As the platform supports multiple, diverse contexts of data ownership, DECODE supports privacy by design to enable flexible and extensible data governance. This section presents a survey of privacy enhancing technologies relevant to blockchains, and DECODE architecture overall.

3.1. Zero-Knowledge Protocols

Zero-knowledge proofs represent a class of protocols that allow a prover to demonstrate to a verifier knowledge of some secrets that fulfils some statements, without disclosing the secrets themselves. These proofs can be used for a variety of applications, from data aggregation and smart metering to verifiable computations.

Example of usages of zero-knowledge protocols. Borges et al. [8] propose a set of protocols between a provider, a user and a tamper-resistant piece of hard- ware (in this case, an electricity meter) to ensure the provider that the bill provided by the user is correct while preserving the user's privacy. The user computes the total bill from measurements provided by the meter according to the provider's policy. The user then sends it to the provider along with a zero-knowledge proof ensuring correctness of the bill's computation. Specifically, during the billing period, the meter sends homomorphic commitments of the readings to the user along with a valid signature on those commitments. At the end of the billing period, the user employs the policy signed by the provider and the meter's commitments to compute the total fees. Then, the user sends to the provider the meter's commitments, a commitment to the total price, and a zero-knowledge proof stating that she knows the opening of the signed commitments, the opening of the commitment to the total price, and the required signatures. Next, the provider

verifies the signatures, verifies the proof, and use the homomorphism of the commitments to aggregate them to the price. Finally, the provider checks the commitments' openings to the total fees. Consequently, the provider is assured that the user paid the correct amount of fees, without seeing the user's current consumption.

Private verifiable computations. Pinocchio [40] is an efficient system for verifiable computations. It lets a client outsource resource-intensive computations to a well-provisioned server while verifying correctness of its computations by performing only a small amount of work. Pinocchio employs zero-knowledge proofs, to allow some inputs to stay private. A key strength of this scheme is that the proof-of-correctness is constant size regardless of the computation performed. Similar works in the area of verifiable computations are [24, 41, 25].

3.2. Privacy on Blockchain

Next we review key researches in privacy enhancing technologies relevant to blockchains.

A fully anonymous currency transaction. Zerocoin [36] presents an extension to Bitcoin that allows fully anonymous currency transaction. The main objective of this scheme is to break linkability between individual Bitcoin transactions without relying on trusted third parties. Zerocoin uses Bitcoin as an append-only bulletin board and as a backing currency. Zerocoin identifies coins by commitments c of a serial number s and an opening value r. Once a user wants to spend a coin, she reveals s and provides a zero knowledge proof-of-knowledge of r for any c among all coins commitments in the ledger $(c_0, \ldots, c_{n-1})$. The scope of this zero knowledge proof is to guarantee that the users knows the opening of one of the commitments $c \in (c_0, \ldots, c_{n-1})$ (without revealing which one). Moreover, in order to prevent double-spending, the user is also required to show the coin's serial number s, so that the ledger can verify it has not appeared in a previous transaction. A significant limitation of Zerocoin is the difficulty to efficiently prove that a commitment c is part of a given set $(c_0, \ldots, c_{n-1})$. To solve this problem the authors employ a "public" one-way accumulator based on, strong RSA, A. Therefore, all commitments have to be prime numbers from a given interval. This also causes the zero knowledge proof to involve a double-discrete logarithm proof, which requires large proof sizes (size that exceeds 45KB).

Improving performances. To address the limitations of Zerocoin, Pinocchio Coin [19] shows a variant to Zerocoin that use elliptic curves and bilinear pairing to greatly improve efficiency. Pinocchio Coin reduces the proof size from about 45KB to 344B.

A fully decentralised private cryptocurrency. Zerocash [45] is a fully decentralized digital currency supporting strong privacy guarantees. It presents the notion of *Decentralised Anonymous Payment* (DAP) scheme based on zk-SNARKs. As a result, unlike systems discussed previously, Zerocash also hides the amount and metadata of the transaction. A DAP scheme is a tuple of algorithms (setup, CreateAddress, Mint, Pour, VerifyTransaction, Receive). In Zerocash, the setup is executed by a trusted party, only once. The trust in the setup algorithm is needed for the transaction non-malleability and balance properties but not for the anonymity properties. Each user generates an address key pair $(addr_{pk}, addr_{sk})$ during the $CreateAddresst$ phase. The public address $addr_{pk}$ is used to receive coins and the secret key $addr_{sk}$ is used to redeem coins. Next, the *mint* operation creates a coins that can be spent only with knowledge of an associated key a_{sk}. A corresponding mint transaction $txMint$ is appended to the ledger only if the user has paid v BTC. Due to the nested nature of the Zerocash commitments, anyone can verify that the commitment in $txMint$ is a coin commitment of value v but cannot discern the owner (by learning the address key a_{pk}) or the coin's serial number. These coins are spent using the *pour* algorithm, which takes a set of input coins, to be consumed, and "pours" their value into a set of fresh output coins: we can say that Zerocash do not "transfer" ownership of coins (contrarily to Bitcoin), but instead it destroys old coins to create new ones. When creating a *pour* transaction, the user can also specifies a nonnegative amount v_{pub} that is publicly declared, as a classic Bitcoin transaction. One of the key points of Zerocash is that the *pour* transaction reveals no information about the value of the consumed coin, nor which coin commitment corresponds to the consumed coin, nor the address public keys to which the two new fresh coins are targeted. The $VerifyTransaction$ algorithm allows the ledger to verify the zk-SNARKS proofs in order to accept or decline an upcoming transaction, and finally, the *Receive* algorithm is used to scan the ledger and retrieve unspent coins paid to a particular user address.

A private and decentralised smart contract system. Built upon Zerocash, Hawk [32] is a decentralised smart contract system that enforces privacy

on the transaction stored on the ledger: it allows transfers of money with programmability. In addition, Hawk allows a programmer to write a smart contract with little worry of the underlying cryptographic constructions. The Hawk compiler compiles the program to create a protocol between the user and the ledger. The resulting contract contains two parts. The first part, ϕ_{priv}, is private and takes the parties' inputs and performs computations to determine the payee. The second part is the public part, ϕ_{pub}, which deals with public data. Hawk smart contract involves a special party called the *manager* who is trusted to not disclose user's private data but cannot compromise the contract's correct execution. Hawk achieves this by employing three primitives. First, the $freeze$ primitive allows parties to commit coins and data (the committed coins are "frozen" in the contract). In the second step, *compute*, parties open their commitments to the manager that determines the payout distribution by running ϕ_{priv}. Finally, the $finalise$ primitive allows the manager to submit to the blockchain the result of ϕ_{priv} along with a proof of correctness of its computation. The public part of the contract ϕ_{pub} is run in order to check the correctness of the manager's inputs and to redistribute the previously frozen coins. Hawk also introduces three time periods T_1, T_2, T_3. All parties have to complete the $freeze$ operation by time T_1 and the manager is required to run the *compute* operation by time T_2. Finally, users can reclaim their frozen coins to the blockchain after time T_3. These timeouts permits to enforce financial fairness by redistributing the frozen coins accordingly in case one of the parties aborts.

3.3. Anonymous Credentials Systems

Anonymous credential systems allow users to selectively disclose certain attributes (or functions of these attributes) about themselves, without revealing their identity. In fact, these credentials are unlinkable: a verifier cannot tell that the same credential is involved in two different transactions

Anonymous credentials' schemes have been an active area of research [11, 12, 16, 15]. In this survey we focus on Anonymous Credentials Light (ACL) [5] due to its efficiency and single-usage (that is, credentials can be used only once). ACL leverages hardness of the decisional Diffie-Hellman problem, and has been designed to work in low-resource settings such as mobile phones and RFIDs.

ACL employs blind signatures to enable users to get a particular attribute signed by an authority. Specifically, ACL modifies Abe's blind signature scheme [4] to provably encode attributes in the Abe signature. The idea

is that only the signer possesses a private key x, associated with a public key $y = g^x$ that allows to sign attributes. Moreover, ACL is a single-use anonymous credential scheme (in contrast to multi-use schemes [11, 15]) : if the credentials are used again, the user's identity is revealed and she can be penalised. This property can be applied to detection of double-spending in the context of blockchain transactions (Section 2).

3.4. Attribute-Based Cryptography

Attribute-based cryptography (ABE) allows schemes in which each user is identified by a set of attributes instead of by an unique identity string. Then, some functions of those attributes are used to determine whether each ciphertext can be decrypted or not [14]. Exploring these principles, Goyal et al. [26] present the concept of Key-Policy Attribute-Based Encryption (KP-ABE) for fine-grained sharing of encrypted data and demonstrates its applicability to sharing of audit-log information. More specifically, ciphertexts are associated with sets of descriptive attributes and users' keys are associated with policies.

Then, Waters et al. [47] introduced Ciphertext-Policy Attribute Encryption (CP-ABE): a system for realising complex access control on encrypted data. The purpose of this construction is to keep confidential encrypted data even if the storage server is untrusted, while remaining secure against collusion attacks. The innovation of CP-ABE is that their attributes are used to describe users' credentials, and the party encrypting data determines who can decrypt them. In other words, it is the encryptor that decides who can have access to the encrypted data, and not the key-issuer (as in KP-ABE). Therefore, CP-ABE can be used to handle secure logs by encrypting it with attributes which match recipients' attributes. Works on CP-ABE have been continued by Waters et al. [47] in order to present a solution allowing any encryptor to specify access control in terms of any access formula over the attributes in the system. Moreover, their methodology, ciphertext size, encryption, and decryption time scale linearly with the complexity of the access formula.

Multi-authority ABE allows independent authorities to monitor attributes and distribute secret keys. Therefore, an encryptor can encrypt a message such that a user can only decrypt it if he has enough attributes from each authority. An example of multi-authority ABE has is presented by Chase et al. [14] by proposing a scheme that handle any polynomial number of authorities and that can tolerates an arbitrary number of corrupt authorities. In

the work of A. Lewko and B. Waters [33], any party can become an authority and there is virtually no need for any global coordination (except for the creation of an initial set of common parameters). In fact, anyone can simply act as an authority by creating a public key and issuing the corresponding private keys to different users reflecting their attributes. Successively, a user can encrypt data over attributes issued from any chosen set of authorities.

Finally, Jung et al. [28] group many of the previous concepts to achieve anonymous cloud data access and fine-grained privilege control by using multi-authority in cloud computing systems. The basic idea behind their scheme is that users' pseudonyms are tied to their private key, but the key generators never know about these keys. Therefore, they are not able to link multiple pseudonyms belonging to the same user or to recognise the same user in different transactions.

References

[1] Ethereum. `https://ethereum.org/`.

[2] Ethereum Wiki. `https://github.com/ethereum/wiki/wiki/White-Paper#applications`.

[3] How a Visa transaction works. `http://web.archive.org/web/20160121231718/http://apps.usa.visa.com/merchants/become-a-merchant/how-a-visa-transaction-works.jsp`.

[4] Masayuki Abe. A secure three-move blind signature scheme for polynomially many signatures. In *Eurocrypt*, volume 2045, pages 136–151. Springer, 2001.

[5] Foteini Baldimtsi and Anna Lysyanskaya. Anonymous credentials light. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 1087–1098. ACM, 2013.

[6] Mo Becker, Alexander Malkis, and Laurent Bussard. S4p: A generic language for specifying privacy preferences and policies. Technical report, April 2010.

[7] Moritz Y. Becker, Cdric Fournet, and Andrew D. Gordon. Secpal: Design and semantics of a decentralized authorization language. Technical report, In Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF, 2006.

[8] Fábio Borges, Denise Demirel, Leon Böck, Johannes Buchmann, and Max Mühlhäuser. A privacy-enhancing protocol that provides in-network data aggregation and verifiable smart meter billing. In *Computers and Communication (ISCC), 2014 IEEE Symposium on*, pages 1–6. IEEE, 2014.

[9] Xavier Boyen, Christopher Carr, and Thomas Haines. Blockchain-free cryptocurrencies. a rational framework for truly decentralised fast transactions. Cryptology ePrint Archive, Report 2016/871, 2016. Accessed: 2017-02-20.

[10] Christian Cachin. Architecture of the hyperledger blockchain fabric. `https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf`, 2016. Accessed: 2016-08-10.

[11] Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 93–118. Springer, 2001.

[12] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Annual International Cryptology Conference*, pages 56–72. Springer, 2004.

[13] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[14] Melissa Chase. Multi-authority attribute based encryption. In *Theory of Cryptography Conference*, pages 515–534. Springer, 2007.

[15] Melissa Chase, Sarah Meiklejohn, and Greg Zaverucha. Algebraic macs and keyed-verification anonymous credentials. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1205–1216. ACM, 2014.

[16] David Chaum. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, 28(10):1030–1044, 1985.

[17] Lorrie Cranor, Brooks Dobbs, Serge Egelman, Giles Hogben, Jack Humphrey, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, Joseph M. Reagle, Matthias Schunter, David A. Stampley, and Rigo Wenning. The platform for privacy preferences 1.1 (p3p1.1) specification. World Wide Web Consortium, Note NOTE-P3P11-20061113, November 2006.

[18] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, and Emin Gün. On scaling decentralized blockchains. In *3rd Workshop on Bitcoin and Blockchain Research, Financial Cryptography 16*, 2016.

[19] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, pages 27–30. ACM, 2013.

[20] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. In *Network and Distributed System Security*. The Internet Society, 2016.

[21] Christian Decker, Jochen Seidel, and Roger Wattenhofer. Bitcoin meets strong consistency. In *Proceedings of the 17th International Conference on Distributed Computing and Networking*, ICDCN '16, pages 13:1–13:10, New York, NY, USA, 2016. ACM.

[22] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*, NSDI'16, pages 45–59, Berkeley, CA, USA, 2016. USENIX Association.

[23] Gartner. Entitlement Management. `http://www.gartner.com/it-glossary/entitlement-management`.

[24] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *Annual Cryptology Conference*, pages 465–482. Springer, 2010.

[25] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer, 2013.

[26] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[27] Usman Haque. Managing Privacy in the Internet of Things. `https://hbr.org/2015/02/managing-privacy-in-the-internet-of-things`, 2015.

[28] Taeho Jung, Xiang-Yang Li, Zhiguo Wan, and Meng Wan. Privacy preserving cloud data access with multi-authorities. In *INFOCOM, 2013 Proceedings IEEE*, pages 2625–2633. IEEE, 2013.

[29] Inc. Kantara Initiative. User Managed Access. `https://kantarainitiative.org/category/uma/`.

[30] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, Austin, TX, 2016. USENIX Association.

[31] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger. *IACR Cryptology ePrint Archive*, 2017:406, 2017.

[32] Ahmed Kosba, Andrew Miller, Elaine Shi, Zikai Wen, and Charalampos Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 839–858. IEEE, 2016.

[33] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 568–588. Springer, 2011.

[34] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 17–30, New York, NY, USA, 2016. ACM.

[35] Silvio Micali. Algorand: The efficient and democratic ledger. http://arxiv.org/abs/1607.01341, 2016. Accessed: 2017-02-09.

[36] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[37] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. `https://bitcoin.org/bitcoin.pdf`, Dec 2008. Accessed: 2015-07-01.

[38] Oasis. eXtensible Access Control Markup Language (XACML) Version 3.0. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html`.

[39] Oracle. Oracle Entitlements Server. `http://www.oracle.com/technetwork/middleware/oes/overview/index.html`.

[40] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 238–252. IEEE, 2013.

[41] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference*, pages 422–439. Springer, 2012.

[42] Andrew Pimlott and Oleg Kiselyov. Soutei, a logic-based trust-management system. In *Functional and Logic Programming, 8th International Symposium, FLOPS 2006, Fuji-Susono, Japan, April 24-26, 2006, Proceedings*, pages 130–145, 2006.

[43] Eric S. Raymond. *The Art of UNIX Programming*. Pearson Education, 2003.

[44] Christoph Ringelstein and Steffen Staab. Papel: Provenance-aware policy definition and execution. *IEEE Internet Computing*, 15(1):49–58, 2011.

[45] Eli Ben Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on*, pages 459–474. IEEE, 2014.

[46] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. *IACR Cryptology ePrint Archive*, 2016:1067, 2016.

[47] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *International Workshop on Public Key Cryptography*, pages 53–70. Springer, 2011.

[48] Bitcoin Wiki. Script. `https://en.bitcoin.it/wiki/Script`.

[49] Wikipedia. Authentication—Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Authentication`, 2017.

[50] Thomas Y. C. Woo and Simon S. Lam. Authorization in distributed systems: A new approach. *J. Comput. Secur.*, 2(2-3):107–136, March 1993.