IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 25, 2020, accepted June 15, 2020, date of publication June 25, 2020, date of current version July 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3005020

Recent Advances in Smart Contracts: A Technical Overview and State of the Art

VICTOR YOUDOM KEMMOE[1], WILLIAM STONE[1], JEEHYEONG KIM[2], DAEYOUNG KIM[1], AND JUNGGAB SON[1], (Member, IEEE)

[1]Department of Computer Science, Kennesaw State University, Marietta, GA 30346, USA
[2]Department of Computer Science and Engineering, Hanyang University, Ansan 15588, South Korea

Corresponding author: Junggab Son (json@kennesaw.edu)

This research was supported by the MSIT (Ministry of Science, ICT), Korea, under the High-Potential Individuals Global Training Program) (2019-0-01601) supervised by the IITP (Institute for Information & Communications Technology Planning & Evaluation).

ABSTRACT Smart contracts, as an added functionality to blockchain, have received increased attention recently. They are executable programs whose instance and state are stored in blockchain. Hence, smart contracts and blockchain enable a trustable, trackable, and irreversible protocol without the need for trusted third parties which generally constitute a single point of failure. If a user creates and distributes a smart contract, others will be able to interact with it while the underlying blockchain ensures a trustable execution. In this paper, we aim to introduce state-of-the-art technologies of the smart contract protocol. We firstly introduce the history of blockchain and smart contracts followed by their step-by-step operations. Then, we introduce the survey results which are classified into four categories based on their purposes: cryptography, access management, social application, and smart contract structure. By presenting the most recent knowledge, this paper will contribute to the advances and proliferation of smart contracts.

INDEX TERMS Bitcoin, blockchain, smart contract, smart contract applications, smart contract operations, recent advances in smart contract.

I. INTRODUCTION

A. HISTORY OF SMART CONTRACTS

Although smart contract technologies have begun to receive more widespread attention from both industry and academia, there is a long history along the path from cryptocurrencies leading up to blockchain and smart contracts. In 1991, S. Haber and W.S. Stornetta proposed a hard-to-tamper system to timestamp digital documents [43]. In their scheme, a digital document is issued a certificate that contains the date on which it was created and information about previously issued certificates for other digital documents. Therefore, it creates a link that can be used to retrieve and prove the date on which a document was created. Later, in 2008, Satoshi Nakamoto proposed a decentralized payment system called Bitcoin, where the history of transactions is stored in a distributed ledger made of blocks [44]. Each block contains a set of transactions, a nonce, a timestamp, and a hash to a preceding block, which is similar to the approach used in [43]. It is the idea of Satoshi Nakamoto's Bitcoin that sparked what we know today as *blockchain*. A blockchain is a distributed data structure made of blocks of data in which a block is linked to another block through its hash value. It has two basic operations: read (to read the content of blocks) and append (to append new blocks). Although an adversary might try to remove/modify some blocks, it requires extremely expensive computations, which makes them nonviable [44]. The read-append property of blockchain has made it a suitable medium to develop different decentralized payment systems by removing the need for centralized institutions such as banks [44]–[46]. Any user can transfer currencies to others and participate in the verification process of transactions. With the growth of Bitcoin, people have recognized that blockchain could be used to develop decentralized solutions in other domains, such as in the food industry, healthcare, etc. However, the Bitcoin architecture, which only supports scripts to check and validate currency transactions was not sufficient to support such applications. Therefore, the smart contract protocol was introduced into blockchain.

The associate editor coordinating the review of this manuscript and approving it for publication was S. K. Hafizul Islam.

In 1994, Nick Szabo introduced *smart contracts*, which are computer programs that replicate the actions described in physical/traditional contracts [47]. Later in 1996, Szabo [48] defined the following objectives of a smart contract: observability, verifiability, privity, and enforceability. With its read-append property, Bitcoin and its scripts language showed that blockchain is a suitable platform to implement smart contracts. On a blockchain, a smart contract cannot be modified, it can be easily observed, verified, self-enforced, and depending on the access mode of the blockchain, privity can be achieved. Despite the fact that its script language only allows checking and validating transactions, Bitcoin can be considered as the first implementation of a smart contract on blockchain. Later in 2015, Vitalik Buterin [46] created Ethereum, a blockchain platform that features a decentralized payment system and a Turing complete language, which allows for the development of a wide variety of smart contracts on a blockchain.

B. RELATED WORK

To introduce the variety of smart contract technologies, many survey results collecting the advancements of smart contracts have been presented before. At the level of security, Liu *et al.* presented a survey that focuses on the security and verification of smart contracts [49]. Atzei *et al.* presented a survey of different attacks on smart contracts on the Ethereum platform [50]. Harz *et al.* presented a survey of different languages currently used to develop smart contracts and their security features, and they also presented different verification methods used to assess the security of smart contracts [51]. Murray *et al*. proposed a survey on different works using formal verification to detect vulnerabilities in smart contracts [52]. Angelo *et al*. proposed a survey of different tools used to analyze smart contracts on the Ethereum platform [53]. These surveys focus on security in the implementation of smart contracts, whereas our survey not only covers recent advancements in the implementation of smart contracts but also covers the use of smart contracts to improve security. At the level of general applications, Hu *et al.* presented a survey on applications of smart contracts in different domains [54]. However, there is a gap between the survey results and the recent advances in smart contract technologies. Rouhani *et al.* presented a survey that tackle four different aspects of smart contract: blockchain platforms supporting smart contract, security and performance issues at the level of smart contract, and applications of smart contracts in decentralized environments [55]. However, their survey does not lean in the technical aspect of different works that use smart contracts. Zheng *et al*. proposed a survey on challenges around the life-cycle of smart contracts, recent advancements in their analysis, and the status of current platforms [56]. Though exhaustive, it does not provide a view of the usage of smart contracts in some emerging fields such as quantum computing.

C. OUR SCOPE AND CONTRIBUTIONS

The objective of this survey is to provide the reader with a technical overview of a variety of emerging technologies and the role of smart contract in their applications. With this in mind, without disregarding the various publications on the topic of smart contract, we surveyed works from 2017 to 2020. After reviewing each of the collected works, we classified them into four categories with the representative terms: Cryptography, Access Management, Social Application, Smart Contract Structure.

Table 1 describes the areas and the summary of our survey. The category Cryptography introduces technologies that bring cryptography to the smart contract domain, overcome a drawback of a cryptosystem by leveraging smart contracts, and introduce attack methods valid on smart contracts. This can be further classified into four sub-categories: decentralized public key infrastructure (PKI), pseudorandom number generation, quantum cryptography, and attack methods. The category Access Management introduces access control and/or management schemes developed for diverse applications on the basis of the smart contract structure. This can be classified into four sub-categories of access management in the Internet of Things (IoT) environment, federated identity management, attribute-based access management, and role-based access management. The category Social Application introduces useful secure solutions that can be widely utilized for rich functionality in social applications, such as voting, auction, mobile payment, and energy distribution. Lastly, the category Smart Contract Structure introduces schemes focused on the development or redesigning of smart contract structure in a way to improve its performance and/or effectiveness for a given environment. This can be classified into the four sub-categories of microservice, service-oriented computing, research framework, and high-performance computing.

D. ORGANIZATION

The remainder of this paper is organized as follows: Section II presents how smart contracts work. Our main contributions, a survey of state-of-the-art smart contract technologies categorized by cryptography, access management, social application, and smart contract structure, will be highlighted in Section III, Section IV, Section VI, Section VII, respectively. Finally, we provide some future directions in Section VIII and a summary of this paper in Section IX.

II. HOW SMART CONTRACTS WORK

In this section, we introduce a high-level overview of a smart contract's process, starting from its development until the verification of transactions. Figure 1 provides a graphic representation of those different steps.

Before diving into the smart contract's process, we give a definition of the different terms that will be used:

TABLE 1. Table of references.

Category	Objective	Paper	Summary of Contribution
Cryptography	Decentralized PKI	[1], [2]	A cryptographic accumulator embedded in a smart contract used to store and verify the identity of entities
		[3]	A smart contract called SCPKI that emulates a web of trust model to authenticate attributes
	Pseudorandom Number Generation	[4], [5]	A PRNG on smart contract based on a contribution scheme
		[6]	An evaluation of PRNGs that use blockchain's variables as seed
		[7]	An external oracle used by smart contracts to obtain random numbers
	Quantum Cryptography	[8]	Improvement of currency's transactions processing on blockchain by using a quantum bank note
		[9]	A smart contract architecture using light-weighted quantum blind signature for security enhancement.
	Attacks on Smart Contracts	[10]	A tool called SMARTSCOPY that uses symbolic execution to detect vulnerabilities in smart contracts
		[11]	A tool called Easyflow whose objective is to track overflows in Ethereum smart contracts
		[12]	A list of vulnerabilities on the Ethereum platform and their severities
		[13]	A tool called SolidityCheck that uses regular expressions to detect potential vulnerabilities in smart contracts
Access Management	IoT access Management	[14]	A variant of OAuth 2.0 protocol that uses blockchain and smart contract
		[15]	An access control scheme based on 03 Smart Contract: Access control contract, judge contract, and register contract
		[16], [17]	capability-based access control frameworks for IoT devices that use smart contracts and support capability delegation
		[18]	An ABAC for IoTs implemented on a permissioned blockchain
		[19]	An ABAC for IoTs designed for Ethereum and based on four smart contracts
	Federated Identity Management	[20]	An identity management system where a user can send its attributes directly to a relying party, which then uses smart contract to verify authenticity.
	Attribute Based Access Management	[21]	An access control framework in which attribute tokens needed to access a resource are held by the respective smart contract of each attribute authority
	Role Based Access Management	[22]	RBAC implemented on smart contracts that can be used among multiple organizations
		[23]	RBAC implementable on smart contracts that supports user authentication and anonymity
		[24]	RBAC based on users' localization and implemented on Ethereum's smart contracts
Social Application	Voting	[25]	E-voting protocol that uses smart contracts to register and count votes
		[26]	E-voting system in which participants use smart contracts to check the integrity of their votes
		[27]	E-voting protocol on smart contracts based on threshold encryption and ring signature
	Auction	[26]	An e-biding system in which participants use smart contracts to check the integrity of their bids
		[28]	An e-auction system implemented using Ethereum' smart contracts
		[29]	an e-auction system that uses a private and a public blockchain
		[30]	A tool called CReam built on top of a smart contract to prevent collusion during e-auction
	Payment	[31]	A payment system relying on smart contracts to store transactions
		[32]	Two Payment-Channel Network protocols using a novel Hash Time Lock Smart Contract
	Energy distribution	[33]	A energy management system in which smart contract handles the selling and buying of energy from the grid
		[34]	Scheduling mechanism that incentivizes producer-consumer types to contribute to the grid
		[35]	A distributed energy management system in which smart contracts are used to monitor energy consumption and perform energy-related transactions
Smart Contract Structure	Micro-service	[36]	A micro-service architecture based on smart contracts
		[37]	A surveillance system using smart contracts based on a micro-service architecture
	Service Oriented Computing	[38]	An evaluation of current blockchain platforms on their capability to support service-oriented computing
	Research Framework	[39]	A layered architecture of smart contracts to ease research in the domain.
	High performance computing	[40]	A parallel execution mechanism for smart contracts based on software transactional memory and locks for memory access
		[41]	Concurrent execution model for smart contracts based on the swappability of transactions
		[42]	An empirical analysis of speculative parallel smart contract execution in Ethereum

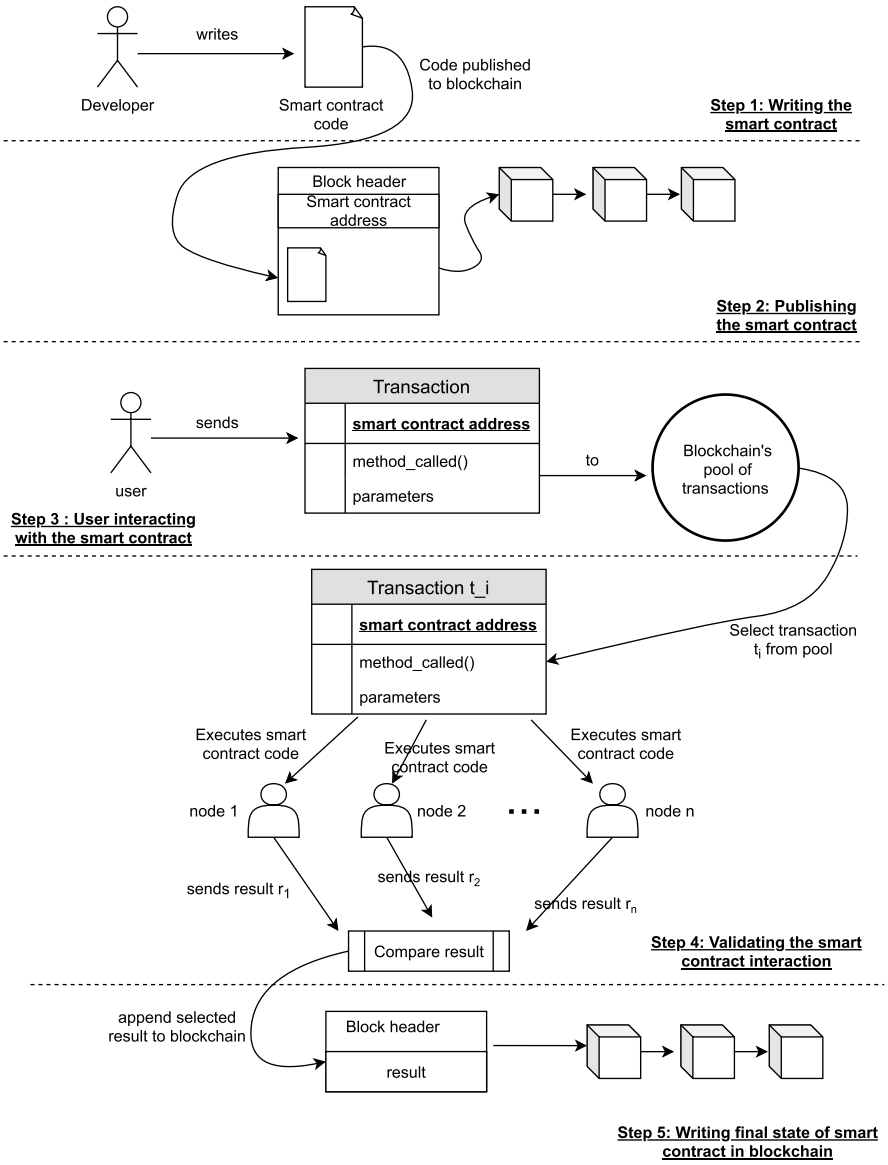

FIGURE 1. **Overview of smart contracts.**

- Developer: an individual who implements the logic of a smart contract using a specific set of instructions compatible with or provided by a blockchain platform
- User: any entity that uses the services of a smart contract
- Transaction: a query made to a smart contract program
- Blockchain platform: the set of applications and protocols used to maintain and manage a blockchain
- Node: an entity having an account on the blockchain platform that can execute and validate transactions
- Faulty node: a node susceptible to submitting false results after the execution of a smart contract

Following is a step-by-step process of the smart contract:

Step 1: Developers write the logic for the contract in a programming language supported by the blockchain platform they wish to use. Then, using a specific compiler (usually provided by the blockchain platform), they compile the source code representing their smart contract and obtain a byte code.

Step 2: After obtaining the byte code, they will publish it to the blockchain platform where it will be stored on the blockchain. Depending on the platform used, once the smart contract program is published, it will be either read-only or modifiable. For instance, Ethereum does not allow smart contracts to be modified [46], whereas EOSIO allows overwriting through the uploading of a new byte code [57]. In case it is read-only, to provide an update, the developers will need to publish a new version of the smart contract and redirect users to it. Once uploaded, the smart contract is at its initial state.

The initial state represents the initial values of the internal variables of that smart contract.

Step 3: Access to a published smart contract program depends on the blockchain platform. In the case of Ethereum [46] and Neo [58], the blockchain platform returns an address to the developers. The address will then be used to interact with the smart contract. In the case of EOSIO [59], the smart contract is published to an account (hosted on the blockchain platform) that was previously created by the developers. The identifier of the account will be used to access and interact with the smart contract. Once users obtain the address/identifier, they can begin sending transactions. Each transaction should contain the function of the smart contract that they wish to utilize and the function's arguments. If an amount of platform currency is needed to start the function's execution, that amount will be transferred alongside the transaction. The transaction will be stored in the blockchain platform's pool of transactions that await to be executed and validated. Step 3 in figure 1 is based on the functioning of [46], [58].

Step 4: From the pool of transactions, the blockchain platform will select a set of transactions to be executed and validated. During the execution phase, the functions of a smart contract that are specified in the transaction will be executed by a set of nodes. During the validation phase, the nodes that executed the transaction will compare their results and select the one to be kept according to a consensus protocol. For instance, in a Byzantine Fault Tolerant (BFT) consensus protocol based on [60], the blockchain platform will select n nodes to execute and validate a set of transactions $T = \{t_1, t_2, \ldots, t_q\}$, where t_i $(i \in \{1, \ldots, q\})$ is a transaction and $n \geq 3m + 1$ with m being the maximum number of possible faulty nodes. Each node k will execute the smart contract tied to each transaction and submit a set of results $r_k = \{r_{1_k}, r_{2_k}, \ldots, r_{q_k}\}$ where r_{i_k} represents the result of each transaction $t_i \in T$, with $k \in (1, \ldots, n)$. From the set of results $R = \{r_1, r_2, \ldots, r_n\}$, a result r' that was obtained by n' nodes with $n' > (n+m)/2$ will be considered as the valid result. Also, there is proof-based consensus where instead of a group of nodes agreeing on a final answer, each node has to prove that it has executed a certain operation, or it is in possession of a certain value. The first node to present a valid proof is elected as the leader and is allowed to attach the result of its execution to the blockchain [61]. In addition, there are hybrid consensus protocols which are based on both BFT and proof-based protocols [62], [63].

Step 5: Once the valid result has been selected, it will be inserted in a block that will be appended to the blockchain. Also, the initial state of each smart contract specified in set T will be updated, i.e., if a validated transaction altered the internal variables of a smart contract, those new values will now be considered as initial values by future transactions.

III. CRYPTOGRAPHY

Cryptography is a discipline that aims to develop means to secure and protect information and the channels over which the information is passed so that only authorized entities can have access to it. Blockchains and smart contracts are possible today thanks in part to some cryptographic primitives that protect the integrity of a blockchain. For instance, by using an asymmetric key scheme and digital signature, a party B can send an amount x to a party A by using A's public key pk_A as the address and signing the transaction with its secret key sk_B. By taking the advantages of blockchains and smart contracts, researchers and developers are looking for new ways to use smart contracts as a medium to improve previous cryptographic tools and develop new ones.

A. DECENTRALIZED PKI

Public Key Interface (PKI) is an entity or set of entities that help support public/asymmetric key based encryption systems by distributing, verifying, and revoking certificates that bind the identity of other entities in a network with their public key. One of the objectives of a PKI is to help prevent a man-in-the-middle attack during an exchange between entities in a network. Currently, the architecture of PKIs is centralized, which means they are controlled by only a few entities called certificate authorities. Certificate authorities in their current form represent a single point of failure since they can be hacked and forced to issue rogue certificates or have some of their certificates compromised and become rogue.

Hence, some researchers and developers are proposing to move from a centralized architecture to a decentralized architecture that uses blockchain and smart contracts. Patsonakis *et al.* propose a decentralized PKI scheme that uses smart contracts and an RSA-based accumulator [1]. One of the current problems of blockchain technology is the fact that the space needed to store a complete chain increases linearly with the amount of data. This makes the blockchain difficult to maintain as it scales up. To avoid such a situation, Patsonakis *et al.* use an RSA-based accumulator of constant size to store valid certificates in a smart contract, but since an accumulator may only be checked for added values, they propose to store a copy of the certificates in a trusted database. To check the authenticity of certificates, a user needs to obtain a copy from the trusted database and check if it was added to the RSA-based accumulator on the smart contract. However, removing an item from the accumulator requires access to its private key. To circumvent this issue, if one wishes to add an item δ to the accumulator, it adds $m = (\delta, i, a)$ to the accumulator, with a being the add operator and i representing the i^{th} time δ is added. To remove δ from the accumulator, it suffices to accumulate $m' = (\delta, i, d)$, with d being the delete operator. The removal of δ from the accumulator is equivalent to its revocation.

Still in the use of a cryptographic accumulator, as an improvement to [1], Patsonakis *et al.* propose the use of a hash tree-based accumulator over the RSA-based accumulator [2]. The hash tree-based accumulator allows for the addition and deletion of items to and from the accumulator without the need for the private key, thus offering a degree of improved security over the RSA-based accumulator and rendering the

inefficient method used in [1] to remove an item unnecessary. It should be noted that in [1], [2], the revocation of a certificate is left in the hands of its owner. Therefore, the lifetime of a certificate is infinite unless the owner decides to revoke it.

At last, Mustafa *et al.* propose a decentralized PKI based on a web of trust model [3]. In their proposed scheme, SCPKI, the authors implement a smart contract on the Ethereum platform in which an entity (people, organizations, etc.) can register its attributes, such as a certificate. An attribute can be stored on the blockchain (full version) or off the blockchain (light version). In the case of the light version, a link pointing to the location of the attribute is provided during the registration. After the registration, any entity except the owner of the attribute can sign the attribute as proof of endorsement. An entity can also revoke its signature on a signed attribute. Since all operations are publicly visible, anyone can see who signed what and base their trust on the information it has. However, Mustafa *et al.* fail to provide a clear mechanism to revoke an attribute. For instance, given a certificate that was signed by a state authority and a federal authority, if the federal authority revokes its signature, can we still consider the attribute as valid? This may lead to a situation of anarchy where an attribute is considered valid by an entity A, but invalid by an entity B. Furthermore, there is no definition on the lifetime of an attribute. Thus, unless one decides to not trust any signature over an attribute, the attribute's lifetime is infinite.

B. ATTACKS ON SMART CONTRACTS

Even though blockchain and smart contract architectures are secured and protected by cryptographic primitives and elaborated protocols, smart contract codes are not. Generally, their security is left in the hands of their developers, who commonly do not take the time and consideration to implement measures that will make smart contracts secure and, as a consequence, leave their implementations vulnerable to attacks. A well-known example is the DAO attack [64] that resulted from an error in the code of the smart contract leading to a third of DAO's assets being siphoned by an attacker. To help developers in the identification of those attacks, Perez *et al.* propose a list of smart contract vulnerabilities on the Ethereum platform and an evaluation of their severity [12].

As possible solutions to find vulnerabilities in written smart contract applications, Feng *et al.* propose a smart contract analysis tool called SMARTSCOPY [10]. It is a program that can be used to produce adversarial smart contracts that exploit vulnerabilities of a smart contract written in Solidity (the programming language used to implement smart contract programs on the Ethereum platform). To detect potential vulnerabilities, SMARTSCOPY uses a novel approach based on symbolic execution, which reduces the time it takes to find vulnerabilities in a smart contract in comparison to a brute force approach.

Memory overflow, also known as buffer overflow, is a phenomenon that occurs when an algorithm writes an amount of data to a memory location that is larger than what the memory location can hold and instead of stopping at the memory location's boundaries, the algorithm writes the surplus of data to adjacent memory locations. This attack is quite easy to perform and can put a smart contract in an undefined state. Gao *et al.* present a tool called EASYFLOW [11]. Its objective is to detect overflow vulnerabilities in smart contracts on the Ethereum Platform. EASYFLOW is made of 4 components:

- extended go-ethereum: detects possible overflow vulnerabilities in a smart contract and outputs logs
- log analyzer: analyzes logs generated by extended go-ethereum, sorts the results by category, sends potential overflow transaction to the transaction constructor, and sends the sorted results to the report generator
- transaction constructor: using the transactions that were flagged as potential overflow, it creates new ones by changing their data and sends them to extended go-ethereum for re-execution to reassess if an overflow can be triggered
- report generator: gathers all results and produces a brief analysis report

However, the proposed scheme is only capable of detecting integer overflow.

Zhang *et al*. propose a tool called SolidityCheck that uses regular expressions to detect potential vulnerabilities in smart contracts [13]. Their scheme assigns susceptible dangerous statements to different classes of specific vulnerabilities by using regular expressions. Once each susceptible dangerous statement is labeled, it is further analyzed to provide a possible solution.

C. PSEUDORANDOM NUMBER GENERATION

Pseudorandom Number Generation (PRGN) is a process that consists of using an algorithm that takes an initial input, generally called a *seed*, to generate a sequence of numbers that is probabilistically similar to a sequence of truly random numbers. This process is deterministic, i.e., if we use the same *seed* as input for the algorithm, we will obtain the same sequence of numbers. At the level of smart contracts, PRGNs can provide many benefits, such as the implementation of secure lotteries. Current blockchain platforms supporting smart contracts do not directly offer a PRGN because of consensus protocols [46], [58], [59]. The majority of nodes in a blockchain platform should obtain the same result after executing a transaction to make it valid. Hence, PRGNs may lead nodes to obtain different results, which makes the implementation of PRGNs in blockchain platforms challenging.

To circumvent this absence of PRGNs, some developers use properties of the blockchain, such as the number of blocks, to generate random numbers, but this approach has shown to be insecure by Reutov [6].

Another method to obtain a random number in a smart contract is by making a query to an oracle located outside of the blockchain platform [7], but one of the major drawbacks

of this solution is that the external oracle represents a single point of failure. If it is out-of-service, then requests for PRGN will not be fulfilled, which can cause some smart contract programs to crash.

Still, on PRGN for smart contracts, the authors of [5] propose a smart contract called RANDAO that can generate random numbers directly on a blockchain platform. They use a contribution scheme in which a set of n users on the blockchain platform each selects a random number $a_{i,i\in(1,\ldots,n)}$ and computes a hashed value $h_i = \text{SHA3}(a_i)$. Then, each user i will select an amount m_i and send (h_i, m_i) as a transaction to RANDAO. After a period of time t, each user will send a'_i to RANDAO. For each a'_i received, RANDAO will compute $h'_i = \text{SHA3}(a'_i)$ and compare h'_i with h_i. If $h'_i = h_i \implies a_i = a'_i$, then user i was honest and RANDAO will keep a'_i and return the amount m_i. Otherwise, user i was dishonest and RANDAO will discard a'_i and keep m_i. Using a function f and the set of kept values $\{a'_1, \ldots, a'_j\}$ $(j \leq n)$ RANDAO will compute $r = f(a'_1, \ldots, a'_j)$ and return r. A smart contract k requesting a random number from RANDAO will send an amount m_k in exchange for r. RANDAO will use the amount m_k to reward honest users. We have identified two drawbacks in this approach: first, if more than one smart contract requests a random number from RANDAO before r is computed, they will all receive the same value r and second, there is a possibility that no users participate, which will result in a failure to provide r.

As an improvement to PRGNs based on contribution schemes, Chatterjee *et al.* propose a game theory approach to improve the incentive for participants to be honest [4]. For a random bit request with ID r_{id}, n participants each select a random bit b_i (where $i = \{1, \ldots, n\}$), a nonce n_i, and an amount m_i. Next, using a cryptographic hash function $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$, each computes and sends $h_i = H(b_i, n_i, i, r_{id})$ to the smart contract. After a period of time t, each sends b'_i and n'_i to the smart contract program which computes $h'_i = H(b'_i, n'_i, i, r_{id})$ and compares it with h_i. If $h_i = h'_i$, then the amount m_i is returned to participant i, otherwise, the smart contract sends m_i as payment to the requester and discards b_i. After collecting and keeping only valid random bits, the smart contract performs an XOR operation and sends the result to the requester. Using game theory, each honest participant's weight (utility/importance in the computation of the result) is computed. Then, the weight is used to determine its reward. Unlike [5], a new random bit is computed for each request. Even though this scheme also suffers from the risk of having no participants, the game theory approach helps mitigate this by offering a higher reward for honest participants. Furthermore, a request returns only one random bit; hence, to obtain a random variable with high entropy, multiple requests will be needed. This can be quite costly in terms of currency for the requester and also for the blockchain platform in terms of the number of transactions generated.

D. QUANTUM CRYPTOGRAPHY

Quantum cryptography is a sub-field of cryptography that leverages the properties of quantum mechanics to develop cryptographic tools. Some properties of quantum mechanics, such as superposition and entanglement, can tremendously increase the computational power and transmission speed of the information processing schemes that we currently utilize.

Blockchain platforms, such as Bitcoin, still have a low rate of transactions per second when compared to services like Visa [65]. This slow transaction processing is currently a scalability issue that prevents other services such as restaurants, retail stores, etc. from adopting blockchain-based payment systems. As a possible solution to this issue, Andrea Coladangelo presents a quantum money scheme that uses smart contracts to facilitate the verification process of banknotes [8]. Coladangelo's quantum money scheme is built upon Zhandry's quantum lightning scheme [66]. The quantum lightning scheme is comprised of two algorithms *(Gen, Ver)*. *Gen* is used to produce a quantum state $|\psi\rangle \in \mathcal{H}_\$$ and *Ver* either outputs $s \in \{0,1\}^\lambda$, a serial number, when it is applied to a quantum state $\in \mathcal{H}_\$$ or $\perp$. The output $(|\psi\rangle, s)$ represents a banknote b that can be used for payment. Smart contracts hold serial numbers, which are used for verification, and amounts of coins (the currency of blockchain), which are used to back quantum banknotes in case the quantum states are lost.

First, to establish his scheme, Coladangelo adds the following propositions to the definition of Zhandry's quantum lightning scheme:

1) Let H be a function defined as follows: $H : \{0,1\}^{l(\lambda)} \rightarrow \{0,1\}^\lambda \cup \{\perp\}$ (for some polynomial l_λ)
2) Let $\mathcal{A}_*$ be a polynomial-time quantum algorithm such that given a state $|\psi\rangle \in \mathcal{H}_\$$, $\mathcal{A}_*(|\psi\rangle) = (x, |\psi\prime\rangle)$, with $x \in \{0,1\}^\lambda$ and $H(x) = s$

Second, Coladangelo proposes a definition of a global ideal functionality $\mathcal{F}_{Ledg}$ that establishes the different features that a blockchain should have to implement the proposed scheme. The features that Coladangelo emphasizes are the abilities of a party P to register itself, to retrieve information about another party, to pay a party, to create a smart contract, and to query information about a smart contract. As to what blockchain platform one should use, Coladangelo proposes Ethereum as an example, but leaves the final choice to the developers. Following the definition of $\mathcal{F}_{Ledg}$, Coladangelo gives the basic structure of a smart contract named *banknote-contract* that is used by the scheme. A bank-note contract is defined by $\textit{bank-note contract} = \phi_\$(pid, w, t, (serial, ActivelostClaim), d))$ where:

- $\phi_\$$ or circuit represents the layout/structure of banknote-contract in $\mathcal{F}_{Ledg}$
- *pid* or party ID represents the ID of a party registering the contract. In the case of the Ethereum contract, it is similar to the address of an externally owned account

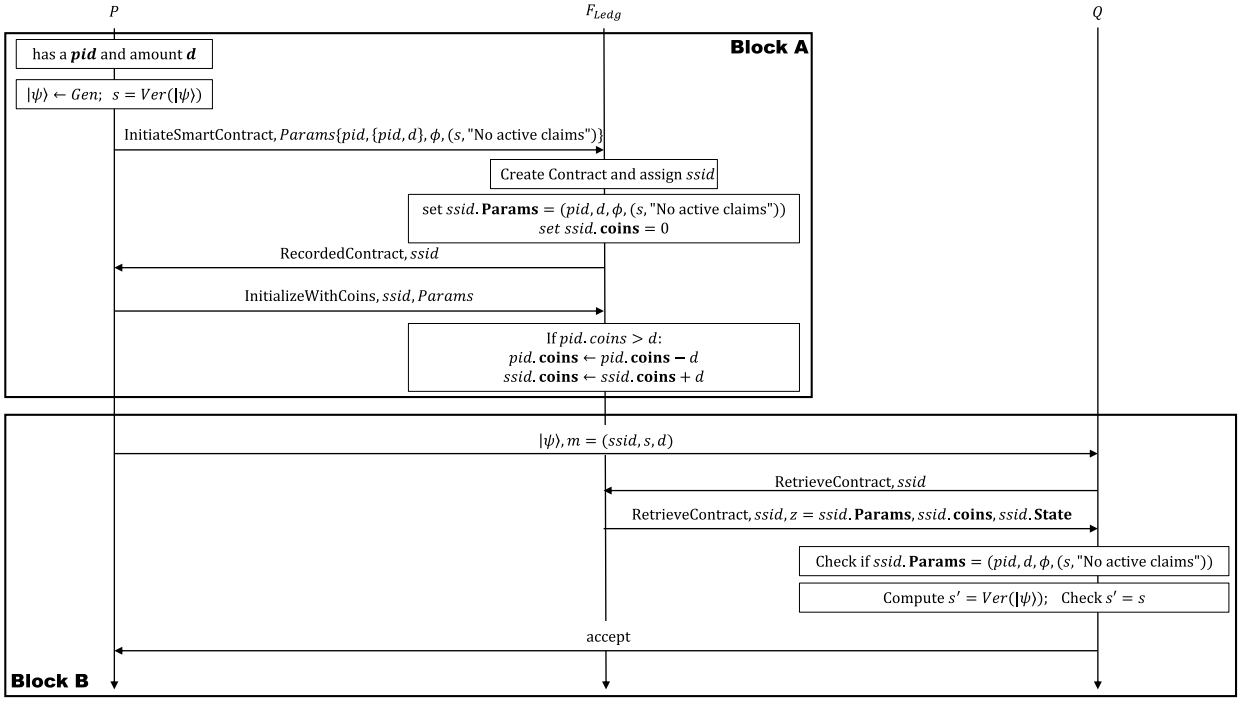

FIGURE 2. Valid quantum banknote generation and payment.

- w or witness represents the type of action that the banknote-contract should execute
- t or time represents an instance of time in $\mathcal{F}_{Ledg}$
- *(serial, ActiveLostClaim)* represents the state of the smart contract, *serial* represents a serial number as defined in [66], *ActiveLostClaim* represents the statue of a claim about the loss of a quantum state $|\psi\rangle$ such that $Ver(|\psi\rangle) = serial$. ActiveLostClaim has two values: "*No active claim*" that means no party initiated a claim and "*Claim by pid' at time* t_0" that means a party having $pid = pid'$ has initiated a claim at time $t = t_0$
- $d \in \mathbb{N}$ represents the initial amount deposited in the smart contract

Finally, Coladangelo gives a full picture of the proposed payment scheme. The scheme is made of 5 protocols, which combined together form a payment system. Those protocols are:

1) **Valid quantum banknotes generation:** To initiate a valid quantum banknote, a party P first generates a quantum state $|\psi\rangle$ by running *Gen* and its serial number s by running *Ver* on $|\psi\rangle$. Then, P sends the message (InitiateSmartContract, *Params*) to $\mathcal{F}_{Ledg}$, where $Params = pid, \{pid, d\}, \phi,$ (s,"No Active claim"). After receiving the message, $\mathcal{F}_{Ledg}$ creates a smart contract with *Params*, assigns an *ssid*, and sets $ssid$.coins $= 0$. Then, it sends the message(RecordedContract, *ssid*) to P as a confirmation. *ssid* represents the ID of the smart contract in $\mathcal{F}_{Ledg}$. In the case of the Ethereum platform, it corresponds to the address of the smart contract account. Once P receives the confirmation, it sends (InitializeWithCoins, *ssid*, *Params*) to $\mathcal{F}_{Ledg}$. Finally, once $\mathcal{F}_{Ledg}$ receives the message of P, it performs the following operations: first, it checks if pid.coins $\geq d$ is verified, then it performs $ssid$.coins $\leftarrow ssid$.coins $+ d$ and pid.coins $\leftarrow pid$.coins $- d$. Figure 2-**Block A** shows a depiction of this protocol.
2) **Payment**: For a party P to pay a party Q, P should hold $|\Psi\rangle \in \mathcal{H}_{\$}$, a quantum state, s, a serial number, and *ssid*, the ID of a smart contract in $\mathcal{F}_{Ledg}$, such that $ssid$.state =(s,"No active claim") and $ssid$.coins $= d$. Then, P sends $|\Psi\rangle$ through entanglement or over a quantum channel and a message $m = (ssid, s, d)$ over a classical channel to Q. Once Q receives $|\Psi\rangle$ and m, it sends the message (RetrieveContract, *ssid*) to $\mathcal{F}_{Ledg}$. After receiving the message, $\mathcal{F}_{Ledg}$ sends the message (RetrieveContract, *ssid*, z) back to Q, where $z = (ssid$.Params, $ssid$.State, $ssid$.coins) if $ssid \in \mathcal{F}_{Ledg}$ or $z = \perp$. Finally, once Q receives z, if $z = (ssid$.Params, $ssid$.State, $ssid$.coins), then Q checks if $ssid$.Params $= pid, \{pid, d\}, \phi,$ (s,"No Active claim"). If verified, Q calculates $s' = Ver(|\Psi\rangle)$ and compares it with s. If $s' = s$ then Q sends the message accept to P, otherwise if $z = \perp$, Q aborts. Figure 2-**Block B** shows a representation of this protocol.
3) **Changing serial number of a bank-note contract**: A set of qubits in superposition can collapse due

to decoherence and causes the loss of a quantum state. Since a quantum state held by party P can be lost, the aim of this protocol is to allow P to create a new quantum state and replace the old quantum state's serial number stored in a smart contract with the serial number of the new quantum state. Once P has generated a new quantum state and acquired the new serial number s', it sends the message (Trigger, $ssid$, BanknoteLost, d_0) to $\mathcal{F}_{Ledg}$. Then, on the smart contract corresponding to $ssid$, $\mathcal{F}_{Ledg}$ changes the $ssid$.state to ($s_{original\ serial\ number}$, ActiveLostClaim="Claim by pid at time t"), t being the time at which the claim was filed on $\mathcal{F}_{Ledg}$. After a period of time t_{tr}, P sends the message (Trigger, $ssid$, (ClaimUnchallenged, s'), 0) to $\mathcal{F}_{Ledg}$. If no one succeeds in challenging the claim of P (i.e., P was honest), then $\mathcal{F}_{Ledg}$ updates $ssid$.state to (s',ActiveLostClaim="No active claim") and sends d_0 back to P.

4) **Challenge claim about loss of bank-note**: Since any party P can interact with $\mathcal{F}_{Ledg}$, a malicious party M having $pid = pid'$ may acquire the $ssid$ of a bank-note contract and try to change its stored serial number s by launching protocol 3. To avoid that, party P, rightful holder of quantum state $|\psi\rangle$ having the serial number equal to s, sends the message (RetrieveContract, $ssid$) at regular interval time $t_r - 1$ to $\mathcal{F}_{Ledg}$. $\mathcal{F}_{Ledg}$ replies with the message (RetriveContract, $ssid$, z). Then, P checks if z=(Params,(s,"Claim by pid' at time t"),d). If verified, P executes the following procedure:
   - Compute $x = \mathcal{A}_*(|\psi\rangle)$ where $\mathcal{A}_*$ is the algorithm defined in requirement 2, $x \in \{0,1\}'$ and $H(x) = s$ with H being the hash function defined in requirement 1. This operation will destroy the quantum state $|\psi\rangle$.
   - Generate a new state $|\psi'\rangle \leftarrow Gen$ and new serial number $s' = Ver(|\psi'\rangle)$
   - Message (Trigger, $ssid$, (ChallengeClaim, x, s'), 0) is sent to $\mathcal{F}_{Ledg}$. If P is honest, i.e., $H(x) = s$, then $\mathcal{F}_{Ledg}$ will update the state of $ssid$: $ssid$.state $\leftarrow$(s',"No active claim") and send d_0, the coins deposited by M when launching protocol 3 to P

5) **Change quantum banknote to coins**: a party P can change a quantum state $|\psi\rangle$ to coins if it holds $|\psi\rangle$, s, $ssid$ and $ssid$.state =(s,"No active claim"). If verified, P executes the following procedure:
   - Compute $x = \mathcal{A}_*(|\psi\rangle)$
   - Send message (Trigger, $ssid$, (RecoverCoins, x), 0) to $\mathcal{F}_{Ledg}$. If $H(x) = s$ then $\mathcal{F}_{Ledg}$ releases d the initial coins deposited when executing protocol 1 to P.

Due to the fact that this scheme requires quantum memories and quantum channels to store and transfer quantum states, respectively, implementation represents a daunting challenge because these quantum technologies are not yet readily available on the market. Nevertheless, it shows how smart contracts on classical architectures can be used in conjunction with quantum cryptography.

To reinforce the security of blockchain and smart contracts against quantum attacks, Cai *et al.* propose a smart contract architecture that employs a light-weighted quantum blind signature [9]. The blind signature scheme allows miners to sign on smart contracts without learning their content, which allows for improvements to confidentiality. However, similar to [8], it suffers from the lack of viable quantum channels and quantum memories.

IV. ACCESS MANAGEMENT

Access Management is the set of procedures that define and manage what entity has access to what resources in a domain. Current access management systems are built around a centralized architecture where key entities are in charge of granting, declining, and revoking access control of resources to other entities. This centralization slows down the treatment of requests to access resources and also represents a security caveat. Depending on the distribution of power among key entities, if the most powerful entity is breached, then an attacker can freely tamper with access controls in a domain. For instance, in an operating system, if an attacker is able to breach the root account, then it will be able to overwrite the access control of all other non-root users. To improve the treatment of requests and to reduce breach points, researchers and developers are turning to smart contracts to develop new secure and decentralized access management systems.

A. IoT ACCESS MANAGEMENT

The number of IoT devices is steadily increasing [67]. With their augmentation, the attack surface of systems using IoT is also increasing. Hence, there is a need for better systems to manage their access and prevent unauthorized users from accessing them.

OAuth 2.0 is an authorization protocol that allows a client to access a protected resource, owned by a resource owner (RO), through an authorization server (AS). However, it requires a client, the protected resource, the RO, and the AS to be online at the moment of the access request, which is often difficult to fulfill when the protected resource is an IoT device with intermittent network access. As a potential solution, Siris *et al.* proposed a variant of OAuth 2.0 for IoT devices: a two-fold scheme which uses blockchain and smart contract technology [14]. Firstly, they proposed a scheme in which the client requests the access for an IoT to an AS. Then, the AS sends the portion of the credentials which are necessary to access the IoT and the price the client has to pay. Once the client makes its payment through the blockchain, the AS sends the remaining credentials. Using complete credentials, the client can access IoT. Secondly, they propose a scheme in which instead of making a request directly to the AS, the client makes the request to a smart contract. Then, the smart contract sends the request to the AS

which, in return, sends part of the credentials and a price. Next, the client pays the required price to the AS through blockchain. Once the AS receives the payment, it sends the remainder of the credentials directly to the client. After that, the client can access the IoT. In both schemes, the RO and the protected resource are not required to be online, which provides more flexibility. However, to replace the consent of the RO, a payment is used. This can become an issue since anyone with the required amount can access the IoT. Also, there is no mechanism to revoke access once granted.

Zhang *et al.* propose an access control framework for IoT based on smart contracts, which is self-regulated [15]. The proposed framework is comprised of three contracts:

- a set of access control contracts which define the access policies for a specific resource and oversee client misbehavior
- a judge contract which analyzes any misbehavior reported by the access control contract and delegates penalties
- a register contract which maintains a lookup table for different access controls and misbehavior judging methods

To gain access to a resource, a client queries the register contract to obtain the access control contracts associated with that resource. Having obtained the list of access control contracts, the client initiates the access request to each access control contract, which returns access results to both the client and the resource. Depending on the returned access results, the client can or cannot access the resource. We note that the capacity of the judge contract can be extended to completely revoke the access of a client to a certain resource.

Capability-based access control (CapBAC) is a type of access control in which an unforgeable token (also known as capability) that contains a set of access rights for a restricted resource is used to grant access control. Compared to role-based access control and attribute-based access control, it can offer the possibility to craft more agile access policies, which can be advantageous in an environment where IoTs are constantly added and removed. However, the use of a centralized authorization server can reduce the overall performance. As a possible improvement, Xu *et al.* propose a CapBAC for IoTs, which uses a smart contract to easily update access control status [16]. Once a client is granted an access token for an IoT, that access token is also uploaded in a smart contract. Hence, the IoT will just need to update its local blockchain copy, and the client will be able to access it. Furthermore, an entity (delegator) can delegate its capabilities to another entity (delegatee), but the delegation process must be authorized by a trusted third party known as a delegation authorization center. To revoke an access, BlendCAC uses trusted entities that can modify the smart contract and execute the revocation process. Nakamura *et al.* present a CapBAC for IoTs implemented on the Ethereum platform that supports multi-delegation [17]. Nakamura *et al.* uncover an issue with the delegation process presented in [16]: it is not possible for two entities Alice and Bob to delegate their capabilities to the same entity Carl. Therefore, they propose to split each operation (such as read, execute, etc.) into a single capability instead of bundling all of them into one capability. Thus, it allows a delegatee to receive a delegation from more than one delegator.

With the caveats of public blockchains, such as the possibility for anyone to access the stored data, Islam *et al.* propose the use of a permissioned blockchain to implement attribute-based access control (ABAC) for IoTs [18]. Since it is not uncommon for IoTs to handle sensitive data, Islam *et al.* turned to a permissioned blockchain to better ensure the confidentiality of data. In their proposed scheme, they used Hyperledger Fabric as the underlying blockchain platform. Resource owners issue attributes which provide access to its resource. Access attributes are registered on the blockchain through a smart contract, which also handles the evaluation of an access request.

Wang *et al.* propose an ABAC scheme for IoTs designed for Ethereum [19]. The proposed scheme uses four types of smart contracts: a subject contract that is used to register device manufacturers and IoTs, an object contract that is used to handle the attributes of IoTs, a policy contract that is used to define the access policy of an IoT, and an access control contract that is used to handle access requests.

B. FEDERATED IDENTITY MANAGEMENT

Federated identity management (FIM) is a specific type of access management that allows an entity to use the same access control to access resources located in different domains. In its current form, a special entity called a credential service provider (CSP) is trusted by other entities (relying parties) belonging to the same or different domains to identify and validate the access controls of clients wishing to interact with those relying parties. In this current setting, CSPs represent a single point of failure for FIM because if they are not available then the relying parties will not be able to validate the access control of a client. In addition, CSPs can breach the privacy of clients since there is a possibility for them to identify the relying party a client is interacting with.

To solve the issue with CSPs, Mell *et al.* propose an Identity Management System (IDMS) where a client can send the different attributes needed for an access control directly to a relying party without relying on a CSP [20]. In their scheme, Mel *et al.* identify five entities that interact through a smart contract to provide an FIM:

- **IDMS Owner**: the entity in charge of the IDMS, its role is to grant and remove access to an account manager on the smart contract. To some extent, it can also modify the smart contract code.
- **Account Manager**: grants and removes access to a user on the smart contract. Also, if ordered by the IDMS owner, it can perform a check of identity on users to confirm that they are who they claim to be. It can also add attributes that will help a user identify its account.

- **Attribute Manager**: adds and removes attributes to users' accounts. Adding an attribute to a user account requires its permission.
- **User**: the entity interacting with the relying party, it can delete attributes which do not serve as its identity from its account. It can also completely delete its account from the IDMS.
- **Relying Party**: an entity that can read data stored in a smart contract.

On a smart contract, the IDMS owner authorizes some account managers to identify and register users. Once a user is registered, an attribute manager can add attributes to its account. To ensure the privacy of data in those attributes, the attribute manager uploads an encrypted version of the data to the smart contract. If a relying party requests certain attributes from a user, the user will send those attributes to it directly, and by using a copy of the state of the smart contract, the relying party will compare the attributes received by the user with those stored on the smart contract.

C. ATTRIBUTE BASED ACCESS MANAGEMENT

Attribute-based access management, also known as ABAC, is a type of access management where access rights to a resource are determined through the evaluation of given attributes by a set of defined policies. The evaluation of those attributes requires their authentication to make sure that they were not forged by an adversary. This authentication process is handled by a trusted attribute authority, but it's possible that for an access right, the different attributes required are shared between different attribute authorities. This distribution can render the evaluation process complex and time-consuming. In [21], Guo *et al.* propose an improvement to attribute-based access management through the use of smart contracts on the Ethereum platform. Guo *et al.* identify three actors in their scheme:

- Data Owner (DO), the entity in charge of a resource, whose role is to define the required attributes and policies that should be fulfilled to obtain access rights to that resource.
- Data User (DU), an entity in need of access rights to a resource. A data user will communicate with a data owner and attribute authorities to obtain access rights.
- Attribute Authority (AA), an entity in charge of issuing and validating attributes necessary to obtain access rights to a resource.

Figure 3 shows a graphical representation of the process that a data user must undergo to obtain access rights for a certain resource, the encircled numbers represent the order in which operations are executed, and a similar number on different operations means that the execution of those operations happen, roughly, at the same time.

First, a data owner has a resource M that it wishes to share only with the entities it has approved. To ensure that the resource cannot be accessed by anyone, the DO generates a fresh key k and encrypts M with k using an AES encryption mechanism. Then, it sends $C = AES(k, M)$, the encrypted version of M, to a shared database. Afterward, the DO defines the set of policies to access C. The DO also creates a smart contract *DO* that contains the functions *verifyAT()* and *sendKey().verifyAT()* are used to validate the attribute tokens sent by a DU that requests access rights for C, if those attribute tokens are validated then *verifyAT()* will invoke *sendKey()* to send $C' = E(pk_{DU}, k)$ to DU, where *C'* is the cipher-text of k encrypted using pk_{DU} which is DU's public key.

Second, a DU willing to access C will create a smart contract *RequestAT* that contains the function *checkA()*. Also, each attribute authority creates a smart contract *AT* that contains the functions *CheckAttribute()* and *sendToken()*. The DU will execute the smart contract *RequestAT* to have its attributes validated by attribute authorities and obtain attribute tokens. The function *CheckA()* will invoke the function *CheckAttribute()* of each smart contract *AT* of each attribute to validate the DU's attributes. If the DU's attributes are validated, *CheckAttribute()* will invoke the function *sendToken()* to send the attribute token corresponding to the validated attribute. Also, the DU creates a smart contract called *RequestKey*, which contains the function *CheckAT()*. It will be executed once the DU has collected enough attribute tokens to request access rights to the DO.

Finally, once every smart contract programs have been deployed, the DU executes *RequestAT* to collect the attribute tokens from the attribute authorities. Once the attribute tokens have been obtained, DU executes *RequestKey* and passes the different attribute tokens as arguments. *RequestKey* will invoke the method *checkAT()*, which will invoke the method *verifyAT()* from the smart contract *DO*. This is to validate the authenticity of the submitted attribute tokens and verify that they comply with the set of policies defined. If everything is in order, *verifyAt()* will invoke the function *sendKey()* that will send *C'* to DU. Once in possession of *C'*, DU obtains the key k by deciphering *C'*: $k = D(sk_{DU}, C')$, with key k, granting DU access rights to M.

V. ROLE BASED ACCESS MANAGEMENT

Role-Based Access Management, also known as Role-Based Access Control (RBAC), is a type of access management in which rights to access a restricted resource are assigned to a role or entity's position in a system. Once a user is assigned a role, it is granted the rights related to that role.

Roles defined in RBAC can also be used across different systems to grant various rights to a given user. For instance, in a university, a user assigned the role of a student may have the right to ''read'' the content of a lecture, whereas in a conference, a user having the role of a student may have the right to ''apply'' a promotion code to access a discounted price. However, in the case of a conference, a user may be required to submit a scanned copy of a student ID or a student email for identity verification, which can be easily forged. To solve this issue, Cruz *et al.* propose an RBAC system implemented using Ethereum's smart contracts that can be applied

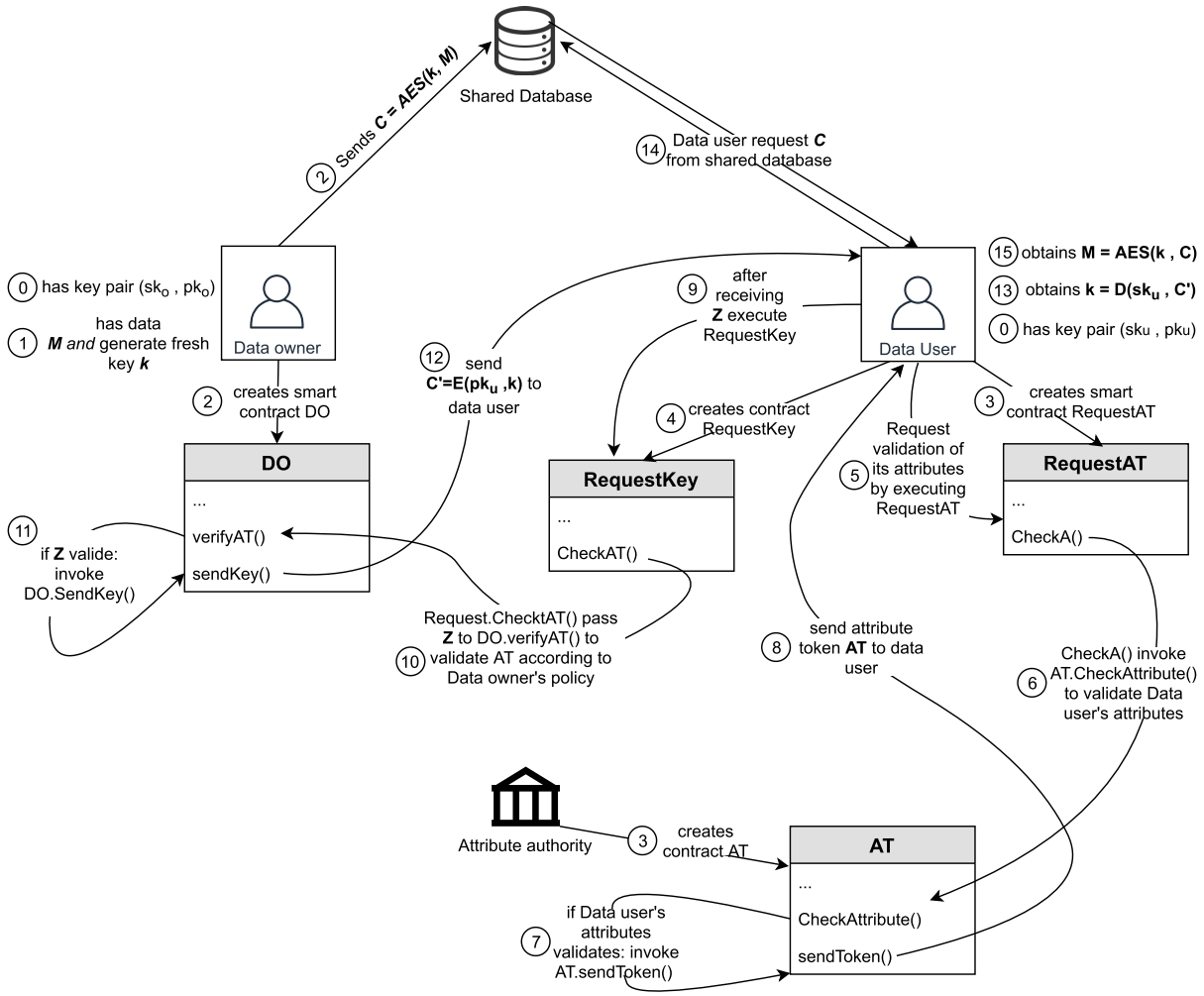

FIGURE 3. Multi-attributes based smart contract.

across multiple organizations [22]. In their proposed system, the authors identify three principal actors: role-issuers which are entities capable of creating roles (i.e., university, government, etc.), service-providers which are entities outside the domain of role-issuers that use those roles to grant specific rights (i.e., conference, restaurant, etc.), and users which are entities to whom roles are assigned. The role-issuer creates a smart contract in which it manages users and their roles. To preserve users' privacy, only their blockchain addresses, roles, and notes, such as the expiration date of the role, are saved in the smart contract. Using the smart contract created by the role-issuer, a service provider can query the information provided by a user willing to access its services. Once a match is found, the service provider sends a message to the user who will then digitally sign the message using its private key. The service provider will check the validity of the signature by using the user's public key stored in the role-issuer's smart contract.

RBACs need an authentication mechanism to prevent unauthorized users from using roles and their related rights. This can be resolved by using a username and password to authenticate a given user, but this method is inefficient since users and developers are not good at handling passwords [68], [69]. As a possible solution for an RBAC, Lee *et al.* propose an RBAC that supports user authentication and anonymity, and that can be implemented using Ethereum's smart contracts [23]. For their scheme, Lee *et al.* list three actors: role-issuers (RI) which are entities managing roles, role owners (RO) which are entities to whom roles are assigned, and role verifiers (RV) which are entities that authenticate role owners and provide permissions to use role owners' roles in service. Each entity has an account on the blockchain. Through a smart contract, a role issuer can issue a role pass (RP) to a role owner, with $RP = H(\text{RO's address}||\text{role}_{id})$, where H is a cryptographic hash function. Using RP, RO can request permission to access a service. The validation of such a request is handled by the RV, which will check if RP is valid and if the rights haven't expired. The use of RP allows for user authentication while preserving its anonymity.

To improve the security of RBAC systems, Rahman *et al.* propose an advanced RBAC model based on users' locations and implemented using Ethereum's smart contracts [24]. When a user's account is compromised, an adversary can log into a system using that account and have access to the same rights as the legitimate user. Adding the user's location as one of the attributes used to determine access rights can help mitigate such an attack since the adversary will have to be located in a valid area. In the scheme of Rahman *et al.*, we have four principal actors: a role manager, an entity that manages roles and assigns them to data users, a data user, an entity that needs access to restricted resources, a resource owner, an entity that owns a restricted resource, and a data provider, an entity in charge of handling access requests from data users. Each actor has an account on the Ethereum platform. Through a smart contract, the role manager can create/delete roles, assign those roles to data users, and set approved locations. Also, using the same smart contract, resource owners can add/remove permissions based on roles to access restricted resources. When a data user needs access to a restricted resource, it will send a request to a data provider. Using the smart contract, the data provider checks if the role of the data user matches the role stored in the smart contract. If true, it will check if the location data and role match with the requirements of the permission. If the permission requirements are fulfilled, the data provider will grant access to the data user. However, instead of data users providing their location data, a trusted third party such as a localization server is used. This is to avoid location data forgery by data users.

VI. SOCIAL APPLICATION

With such a large amount of users and data, applications revolving around societies and social interactions face the constant challenge of providing availability to its users while simultaneously securing their data against breaches and tampering by attackers. As we know, blockchain and smart contract platforms offer not only hard-to-tamper storage but also an architecture resistant to denial of service attacks. Therefore, developers and researchers are proposing new social applications that take advantage of the benefits of these technologies.

A. VOTING

An important application that helps maintain order in a society is voting. Through voting, people select their next leader in a time of elections or the next path to take when faced with a dilemma. However, one of the most important aspects of voting is integrity, or to make sure that the votes of all participants have been taken into account and were not tampered with. Traditional voting systems contain several points of failure. For instance, in the case of voting through cards, it is easy for counters to dismiss or change the entry of a participant during the counting process. Therefore, some researchers and developers are turning to blockchain and smart contracts to develop future systems.

McCorry *et al.* propose an e-voting protocol aimed at boardrooms that use smart contracts to register users' votes and count the tally [25]. The voting protocol consists of two rounds. During the first round, after the execution of the setup process, which selects the different parameters of a Diffie-Hellman group, each voter i uniformly randomly selects x_i, then it computes g^{x_i} and $\delta_i(x_i)$, a non-interactive zero-knowledge (NIZK) proof that proves the knowledge of x_i. Then, the values g^{x_i}, $\delta_i(x_i)$, and an amount m are sent to the smart contract. Once each voter has uploaded their share, the smart contract computes $g^{y_i} = \Pi_{j=1}^{i-1} g^{x_j} / \Pi_{j=i+1}^{n} g^{x_j}$. During the second round, each voter selects $v_i \in \{1, 0\}$ as its vote and sends $[g^{x_i y_i} g^{v_i}, \theta(v_i)]$ to the smart contract, where $\theta(v_i)$ is a NIZK proof which proves $v_i \in \{0, 1\}$. Once each vote is sent, the smart contract returns the amount m and computes the final result as follows: $R = \Pi_i g^{x_i y_i} g^{v_i} = g^{\sum_i v_i}$ because $\sum_i x_i y_i = 0$. To extract the actual outcomes of the voting process, i.e., $\sum_i v_i$, the smart contract performs a brute force search on $g^{\sum_i v_i}$. From an initial analysis, sending $g^{x_i y_i} g^{v_i}$ may seem to protect the anonymity of the voter and its vote. However, we note that if the smart contract is able to perform a brute force search on $g^{\sum_i v_i}$ to obtain $\sum_i v_i$, then it is capable of breaking the discrete logarithm problem (DLP). Thus, it is also possible to perform a brute force search on $g^{x_i y_i} g^{v_i}$ and obtain v_i. Therefore, the anonymity of the vote is not ensured.

Tso *et al.* present an e-voting system where users participating in a vote can use a smart contract to check if their vote was validated [26]. If an anomaly is detected in their votes, they can directly report to the election committee.

Lyu *et al.* present a privacy-preserving e-voting protocol that uses threshold encryption to enable tallying and linkable ring signature, a type of digital signature, to identify voters and prevent double-voting [27]. Given a group of users $G = \{U_0, U_1, \ldots, U_n\}$ where each user $U_i \in G$ has a secret-public key pair (sk_i, pk_i), and a linkable ring signature $\texttt{Sig}(m)_{U_j}$ issued by $U_j \in G$ on a message m, it is possible for an outsider $U' \notin G$ to know that $\texttt{Sig}(m)_{U_j}$ was issued by a member in G without learning which member issued it. The linkable part means that it is possible to retrace all signatures generated by U_j without learning the identity of U_j. By using those properties of linkable ring signature, Lyu *et al.* are able to prevent double-voting and the participation of unapproved users in an election. During the voting process, each voter uploads its encrypted vote using the public key generated by the threshold encryption and its linkable ring signature to the smart contract. After the voting process is completed, each voter uploads its share of the secret key generated by the threshold encryption. Once the smart contract receives enough shares, it reconstructs the secret key, decrypts the uploaded votes, and computes the final result. This prevents any user from knowing the final result before the vote-counting process.

B. AUCTION

In auctions, participants have found multiple methods to organize themselves into collusion and manipulate prices [70].

It is also more difficult for auctioneers to enforce participants to pay the bade amount as promised.

Chen *et al.* propose an e-auction platform implemented using Ehtereum's smart contracts [28]. They take advantage of the properties of blockchain and smart contracts to reinforce the integrity of auction operations. Since blockchain is read-append only, it is not possible for a bidder to deny a bid. Also, if a participant U_i bids an amount m_i, that amount will be held in the smart contract. If U_i is the winner at the end of the auction, m_i will be automatically transferred to the tenderer. However, it does not prevent collusion from bidders.

Desai *et al.* propose an e-auction system that uses public and private blockchains to ensure users' privacy and accountability [29]. Their system is comprised of three smart contracts: an auction contract, PublicDeclare smart contract, and CongressFactory contract. The auction contract is hosted on a private blockchain whose access permission is managed by the auctioneer. The PublicDeclare and CongressFactory contracts are both hosted on a public blockchain. Each user must own an account on both blockchains to participate in an auction. The auctioneer is in charge of handling a hash table that maps the address of a participant on the public blockchain with its address on the private blockchain. The auction is handled by the auction smart contract. To ensure accountability, every time a participant submits a bid, it uploads $H(n, m)$ to the CongressFactory where m is the bade amount, n is a random number, and H is a cryptographic hash function. At the end of the auction, participants send the random numbers used to generate the different commits to the auctioneer through the auction smart contract. Then, the auctioneer checks the consistency between the values received by the participants, and the commits are stored in the CongressFactory. Once the verification phase completes, the auctioneer compares bids and declares the winner through the PublicDeclare smart contract. The use of a private blockchain to run the auction ensures the privacy of the participants, with only the auctioneer being aware of the bade amounts. In the event that the auctioneer is dishonest and declares the wrong winner, participants can use the CongressFactory to check the values used by the auctioneer and the commits that were made. During this operation, the auctioneer must send the bids to the CongressFactory, making it possible for a participant to learn the bids of others. To mitigate this, participants seeking to verify the auctioneer must place a stake in some amount to access the information. Should the auctioneer prove to be honest, then the staked amount is confiscated.

To prevent collusion among participants, Wu *et al.* propose a collusion resistant decentralized platform for e-auction called CReam built on top of smart contract technology [30]. In the CReam, participants first submit a hash version of their bids to the smart contract during the first round. In the second round, participants reveal their bids to the smart contract. After the second round, the smart contract computes the winning bidder and punishes dishonest bidders by redistributing their coins to honest bidders. The smart contract also forces sellers to send the auctioned resources and the winning bidder to pay the bade amount. Collusion resistance is ensured by the use of a consensus estimation function during the determination of the winner.

In [26], Tso *et al.* also propose an e-bidding system based on smart contracts that allow participants of bidding to check if their bids were correctly taken into account. Their system also ensures that participants can pay the amount they bid by using a system of bonds.

C. PAYMENT

To ensure better traceability in transactions involving money and to avoid the problems that come with physical cash, society is moving towards cashless payment [71]. Companies like MasterCard, Apple, Samsung, Google, etc. are each developing their own mobile payment systems. Since there is no widely accepted standard, the burden lies on the side of merchants who have to implement each service in their business. To solve this situation, Yeh *et al.* propose a robust mobile payment using smart contracts as a transaction repository [31]. This allows clients and merchants to easily query smart contracts and check the validity of a transaction in case of a dispute.

Blockchain platforms have offered a transaction rate, which is very low compared to traditional systems such as Visa [65]. To circumvent this issue, some use payment-channel networks. Payment-channel networks (PCN) are off-chain networks that allow parties to perform multiple transactions without uploading them to the blockchain. Once a PCN is closed, the final balance is uploaded to the blockchain. However, PCNs suffer from concurrency issues (multiple transactions involving the same payees may enter into deadlock or starvation), and in case there are multiple intermediaries in a transaction, some of them can collude to reveal the identity of the sender and the receiver. As a possible remedy, Malalvolta *et al.* propose two PCN protocols relying on a novel Hash Time Lock (HTL) smart contract [32]. Our interest was centered on the proposed HTL smart contract called Multi-Hop HTLC. During the setup phase of Multi-Hop HTLC with a set of users $U = \{u_0, u_1, u_2, \ldots, u_n\}$ where u_0 is the sender and u_n the receiver, u_0 samples n random strings x_i and generates

$$\forall x_i \in \{0,1\}^{\lambda}; y_i = H(\bigoplus_{j=i}^{n} x_j), \text{ with } i = \{0, \ldots, n\}$$

Then u_0 sends $\{x_n, y_n\}$ to the receiver, and it sends $\{x_k, y_k, \pi_k\}$ to u_k, with $k \in \{1, n-1\}$, π_k being the proof issued by a pre-chosen NIZK protocol. The NIZK is used by intermediary nodes to ensure that x_i is legit. Next, using y_{i+1}, each pair set (u_i, u_{i+1}) set an HTL. Since the values (x_i, y_i) are unique and sent through private channels, it is impossible for intermediary nodes to collude and reveal the identity of the sender and the receiver.

D. ENERGY DISTRIBUTION

Renewable energy allows individuals to gain independence from the grid and produce their own energy, but occasionally,

the amount of energy produced may exceed or fall short of the current demand. A quick solution is to allow individuals in need to make a request to those who possess a surplus through a dedicated grid. Wang *et al.* propose a decentralized distribution energy system where users who have an excess of energy can easily sell it to the grid [33]. In their approach, users use smart contracts to automatically or manually sell their excess energy through the grid. However, some participants with an excess of energy may decide to opt out and not sell their excess to the grid, which could benefit those in need. As a possible solution to this issue, Amanbek *et al.* propose a scheme that uses a scheduling mechanism to sell energy to the grid that benefits prosumers that contribute the most by using smart contracts [34].

In an effort to completely automate grid management, Afzal *et al*. propose a distributed energy management system that uses smart contracts to monitor energy demands and perform energy-related transactions [35]. In the proposed scheme, they used sensors that communicate with a smart contract to compute energy consumption. Furthermore, they proposed a game theory model which can help ensure fair pricing between parties.

VII. SMART CONTRACT STRUCTURE

The smart contract is a concept that is still in its infancy, with each new blockchain platform proposing a structure that is slightly or completely different from others. For instance, in Ethereum, once a smart contract has been published, it cannot be changed, whereas in EOSIO, the owner of a smart contract can make alterations. Researchers and developers are also looking for ways to combine or change the structures of smart contracts so that they can be used to benefit other applications. In this subsection, we are going to look into some proposals that assess and challenge the current architecture of smart contracts.

A. MICRO-SERVICE

Micro-service is a design approach in software engineering that consists of breaking an application into sets of independent services that will communicate together to solve clients' problems. This approach generally requires always-on units to run those services and an always-on message passing interfaces to allow communication between them. Those requirements are easily satisfied by blockchain platforms that support Turing-complete language for smart contracts such as [46], [58], [59]. Hence, in an effort to show the feasibility of this proposition, Tonelli *et al.* propose a concrete implementation of a micro-service architecture using the implementation of smart contracts in Ethereum [36]. For simplicity, each micro-service is represented by a single smart contract. To simulate the REST architecture, which is often the underlying architecture to allow communication between services in a micro-service architecture, the authors use the address of each contract, which can be considered analogous to the URI. Functionalities provided by micro-services are implemented through public getter and setter methods embedded in the smart contract. However, since all kinds of information stored in a public blockchain are visible to anyone, such a framework should not be used for applications handling sensitive data. We also note that the delay caused by the validation of each transaction in a blockchain downgrades the overall performance.

Micro-service architecture also influences how schemes using smart contracts are developed. In [37], Nagothu *et al.* propose a surveillance system using a smart contract implementation based on micro-service architecture. Nagothu *et al.* assign each functionality of their surveillance system, such as facial recognition, to a micro-service. Those micro-services are loaded onto low-powered IoTs, such as cameras, and send their output to a database. To ensure that the outputs produced by micro-services are not tampered with, a hashed version is stored on a blockchain. Access control policies for features of the system are handled by smart contracts.

B. SERVICE-ORIENTED COMPUTING

Software oriented computing, also known as software-oriented architecture, is a design principle similar to micro-service architecture that consists of building an application around a set of services that will communicate together to offer the required functionalities. Compared to micro-service architecture, services designed for software-oriented architecture encompass more functionalities and can share databases [72]. As in the case of micro-service, implementing this design may seem daunting, but blockchain and smart contracts seem to be an effective platform for this. Daniel *et al.* provide more insight into the possibility of implementing a software-oriented architecture on a blockchain using the smart contract in [38]. The authors study four blockchain platforms supporting smart contracts to see if the technologies can be used to implement a service-oriented computing architecture. At the end of their survey, they present different properties that could make them suitable for the implementation of service-oriented computing. Finally, they conclude that none of these platforms completely fulfill the requirements to build service-oriented computing on top of a blockchain and make propositions on what should be addressed next to allow future generations of platforms to offer that architecture.

C. RESEARCH FRAMEWORK

Smart contracts and blockchain offer great potential to solve problems in diverse fields, but with no concrete structure, developers keep creating new applications and platforms. During this process, they sometimes discard the advantages or recreate the holes of previous versions. This situation promotes anarchy in the field and can be considered analogous to the beginning of computer networks before the presentation of the OSI model: each engineer was creating different network models that were not able to communicate with others.

Wang *et al.* propose a layered architecture to easily organize research on smart contract technology [39]. This layered architecture can also be used as a reference to develop future

blockchain and smart contract platforms. Their architecture is comprised of 6 layers:

- **Applications layer**: represents the application of smart contracts to specific domains such as finance, food industry, etc.
- **Manifestations layer**: represents the different forms smart contracts applications can take, such as decentralized autonomous organizations (DAOs)
- **Intelligence layer**: this layer encapsulates elements that can make a smart contract intelligent by learning and adapting to presented situations
- **Operations layer**: this layer is focused on operations that deal with the form of a smart contract
- **Contracts layer**: this layer encapsulates the logic of the smart contract (i.e., how they are going to work)
- **Infrastructures layer**: consists of all infrastructures supporting the development of smart contracts and their applications

Wang *et al.* also propose a set of challenges that current blockchain and smart contract implementations face and a set of future development trends.

D. HIGH PERFORMANCE COMPUTING

In the current architecture of blockchain and smart contracts in platforms such as Ethereum, the execution and validation of smart contract transactions are sequential. Miner nodes select a set of transactions to be processed and execute them one after another. This approach not only limits the number of transactions that can be processed per second, but it also does not take advantage of the multiple central processing unit (CPUs) architectures of modern computers. Therefore, moving from a sequential execution to a parallel execution can solve this problem. Some smart contracts can access the same resources during their execution. In a parallel execution setting, this can lead to problems of deadlock, race-condition, cache coherence, etc. Also, at the level of consensus mechanisms, it can increase the likelihood for miners to obtain different results, impeding the ability to reach a consensus.

As a possible parallel architecture for smart contract execution, Dickerson *et al.* propose a parallel architecture based on software transactional memory (STM) and modeled after the Ethereum consensus mechanism [40]. In the Ethereum consensus mechanism, there are two types of miner nodes: *miner nodes*, which are in charge of executing transactions and creating new blocks and *validator nodes*, which are in charge of checking if a miner node has correctly executed a transaction and if the block it proposes should be appended to the blockchain [46]. For a validator node to validate the result of a transaction executed by a miner node, it needs to re-execute that transaction and compare its result with the result submitted to the miner node. Using the STM approach, Dickerson *et al.* encapsulate each smart contract transaction in a *transaction*. The term *transaction* used here refers to a thread or a process in the context of multiprocessing. To avoid confusion between a smart contract transaction and a *transaction*, we will refer to the *transaction* as a thread.

Each thread is equipped with an *abstract lock* that is used to lock the access to a shared memory location, and it has access to an *inverse log* that is used to record the inverse of an operation performed on a shared memory location. It should be noted that the mechanism of the abstract lock and inverse log is invisible to developers of smart contract programs, the tasks are handled automatically by the virtual machine running those threads. If an instruction in a thread p needs to access a shared memory location m, p will launch a request to set an abstract lock on m. If the request is granted, other threads that also requested m will have to wait for p's instruction to terminate so that p can release its abstract lock on m. As we can see in Figure 4, we have three contracts: A, B, and C that are concurrently executed. At phase (b), A and C are both requesting access to the shared memory location 0×011, but it is A that is able to set its abstract lock on 0×011. Therefore, C is obliged to wait, and its status switches to idle.

For each action that a thread performs on a shared memory location, it records the inverse operations in an inverse log. During its execution, if a thread aborts (fail/crash), the abstract locks stay effective until all the inverse operations it has recorded in the inverse log are executed to return the shared memory locations used in their initial states prior to the interaction. Once the memory locations are back to their initial states, the abstract locks are released, and the thread is rolled back (set to be re-executed from the beginning). This process is carried-out by miner nodes only. If validator nodes were to execute this same process, the order in which their threads obtain abstract locks can differ from that of miner nodes and will lead to different results. To overcome this issue, Dickerson *et al.* introduce a *log of abstract locks* in which the order in which the threads acquired abstract locks is recorded. Using the log of abstract locks and the set of threads executed, the miner nodes generate a *happened-before graph*, which is a graph showing the dependency between threads and a serial representation of that graph. After obtaining the happened-before graph and the serial representation, the miner nodes send them to the validator nodes that will use them to reconstruct the same parallel execution. However, we note that the proposed scheme suffers from efficiency issues due to a lack of group exclusive access for READ operations: it is not possible for two or more threads to perform READ operations on a given memory location in parallel since threads cannot set abstract locks for the same memory location concurrently. Also, there is a possibility for a conflicting transaction to be restarted, which creates significant overhead.

Bartoletti *et al.* proposed a concurrent execution model for smart contracts based on a swappability factor of transactions [41]. They defined two transactions $t_x, t'_x | t_x \neq t'_x$ as swappable if executing them in any order produces the same global state of the blockchain. Using this definition, they define a more powerful swappability concept, strong swappability, as follows: given two transactions t_x, t'_x and their respective set of WRITE and READ memories accessed

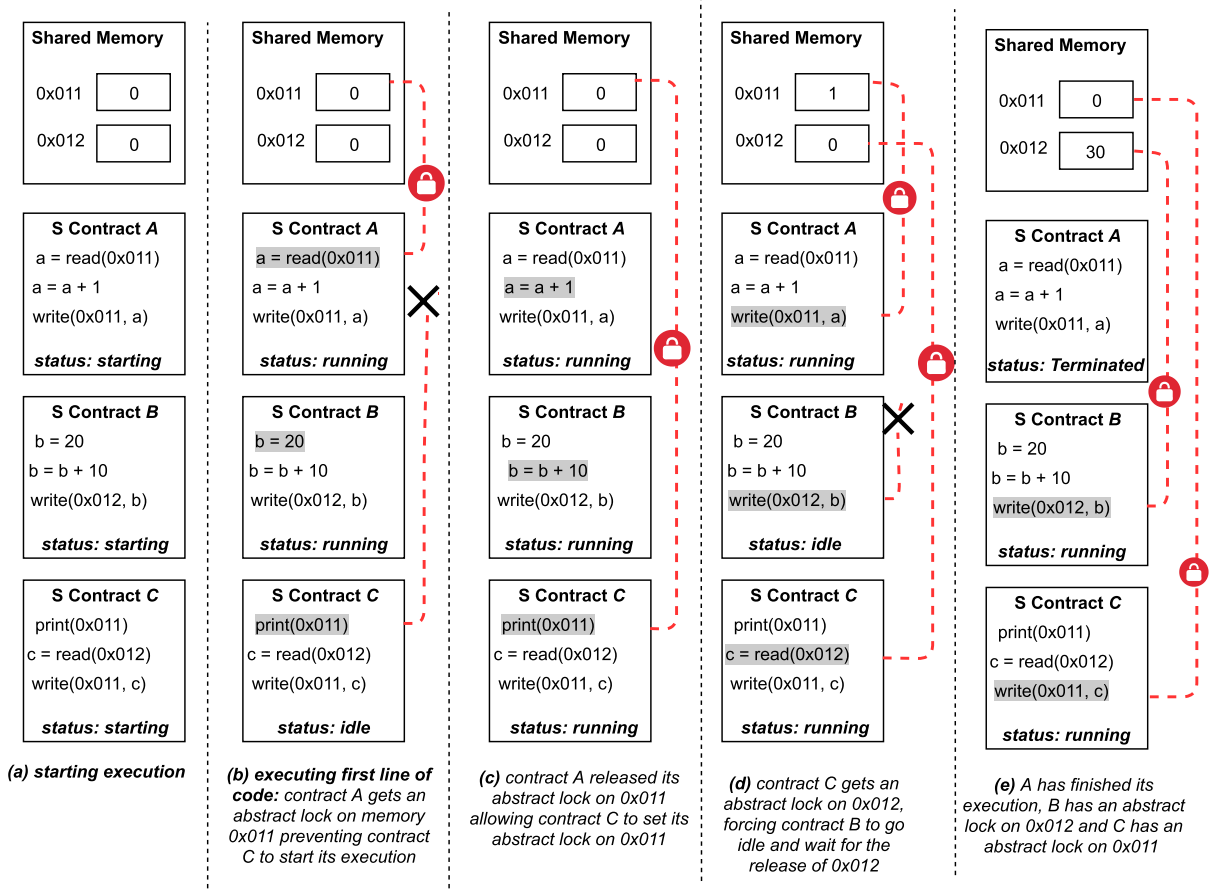

FIGURE 4. Parallel smart contract execution using abstract locks.

(W, R), (W', R'), t_x and t'_x are strongly swappable if

$$(R \cup W) \cap W' = (R' \cup W') \cap W = \emptyset,$$

i.e., none of these transactions write in a memory location accessed by the other. Using the strong swappability concept, they define an *occurrence net*, which shows the relationship between transactions and their execution order. Basically, if two transactions are strongly swappable, they are executed in parallel, else they are executed in series. This occurrence net can be produced by a miner and shared with validators. We note that this offers a possible solution to the issue of conflicting transactions in [40] and is capable of offering concurrent READ operations on given memory access. Though, the creation of an occurrence net can be difficult for a set of transactions involving complex programming syntax, such as recursion.

With Ethereum being the avant-garde of blockchain platforms supporting smart contracts, Saraph *et al*. propose an empirical analysis of speculative parallel smart contract execution [42]. During their analysis, they used a custom engine to run past Ethereum's transactions in two phases. During the first phase, all transactions are executed in parallel. Only transactions that fail to terminate are sent to the second phase, where they are executed sequentially. This produces an execution history that can be shared between a miner and a validator. To avoid READ/WRITE conflicts, they used a read-write memory lock. This lock is used by WRITE operations. Hence, multiple simultaneous READ operations can be executed unless a WRITE operation has set a lock. At the end of their analysis, they concluded that for a block containing a transaction whose execution time is significantly greater than others, the speed-up gained by using speculative parallel execution could be insignificant. They also concluded that speculative parallel execution is disadvantageous when multiple smart contract applications access the same memory location.

VIII. FUTURE RESEARCH DIRECTIONS

Following a thorough analysis of the related papers, we would like to suggest the following future research directions:

- **Smart Contract Applications:** Smart contracts provide verifiable and trustworthy protocols without the need for a trusted third party. Due to this property, it can be utilized for many applications involving resource-constrained devices, such as IoT, vehicular ad-hoc networks (VANET), smart home products, and so forth. Since those devices have a limited capacity of storing verified protocols, including cryptographic

tools, smart contracts will be a potential source of rich functionality.

- **Parallelization of Code**: Despite the efforts of proposing parallel smart contract execution frameworks [40]–[42], we believe that parallelization in smart contract execution can be pushed further to allow task parallelism and a limited degree of data parallelism. For instance, in the case of task parallelism, a smart contract that handles a loan application may have to check if a client has enough money for the prepayment and an insurance before proceeding, which could be performed concurrently by different threads if the code allows. In the case of a limited degree of data parallelism, a for-loop that only reads from a given memory location and does not suffer from loop-carried dependency can be executed by multiple threads with each thread handling a given number of iterations. Traditional frameworks such as OpenMP, OpenCL, or CUDA can be regarded as potential templates to produce a possible solution. Furthermore, since such parallelization is only at the level of the smart contract code, theoretically, there is no risk of an issue with a consensus protocol. Hence, the existence of such a framework could improve the execution time of a transaction, improve the CPU utilization of miners, and possibly reduce the execution cost. This could also be a possible solution to solve the issue raised by [42] on transactions with significant execution time.
- **Distributed Execution of Smart Contracts by IoT Devices:** we remarked that when IoTs are used in conjunction with smart contracts, they mostly perform push transactions (upload data to a blockchain via smart contract) or pull transactions (get data from a blockchain via smart contract). They are not used to execute smart contracts, due in part to their low computing power. Therefore, we think that distributing the execution of a smart contract program among a set of nodes instead of having one node executing the whole program can not only allow the execution of a smart contract by a set of IoTs but also open the door for the implementation of smart contract based solutions in more restricted environments.
- **Revocation of Certificates and Tokens**: throughout our analysis, we observed that schemes that issue certificates or tokens either do not provide a mechanism to revoke them (certificates and tokens can have a limitless validity duration) or centralize the revocation mechanism by delegating it to trusted entities (which may leave a scheme open to adversary attacks). Using an attribute-based signature scheme in which the expiry date of a certificate/token is one of the attributes might be a possible solution. With such a scheme, one will be able to prove that a certificate/token is still valid if the associated signature is verified. Kumar *et al*. proposed an encryption scheme that uses such an approach to revoke delegated keys [73].
- **Random Number Generation**: Chatterjee *et al.* proposed a PRNG for smart contracts based on a contribution scheme [4]. However, a request results in the return of one bit. As a possible improvement, we propose the use of the result returned after XORing the hash value of retained numbers sent by honest participants. Depending on the hash function used, this can provide a random number of high entropy in one round.
- **Smart Contract Execution with NIZK as Proof of Correctness:** In our studies, we observed that unless the access type of the blockchain platform is private, it is difficult to conserve the confidentiality of data being handled. Even with private access, the total confidentiality of data is not guaranteed since certain trusted entities still have access to them. Hence, to provide stronger confidentiality while allowing any party to check the soundness of operations, we believe that a blockchain platform in which the execution of smart contracts results in the production of a NIZK proof could provide a possible solution. This could allow any party to check that a smart contract was correctly executed without the need to access the data.

IX. CONCLUSION

This paper introduced state-of-the-art technologies in the smart contract protocol. Specifically, we surveyed the most recent advances, then classified them into four categories based on their goals and purposes to effectively deliver knowledge of the current status of smart contracts in domains such as cryptography, access management, and social applications, as well as the structure of smart contracts. In each of these categories, we highlighted the shortcomings of some recent works. We also proposed interesting future research directions that may help improve the current status of smart contracts and solve some of the identified shortcomings.

REFERENCES

[1] C. Patsonakis, K. Samari, M. Roussopoulos, and A. Kiayias, "Towards a smart contract-based, decentralized, public-key," in *Proc. 16th Int. Conf., CANS*. Hong Kong: Springer, Nov./Dec. 2017, pp. 299–321.

[2] C. Patsonakis, K. Samari, A. Kiayiasy, and M. Roussopoulos, "On the practicality of a smart contract PKI," in *Proc. IEEE Int. Conf. Decentralized Appl. Infrastruct. (DAPPCON)*, Apr. 2019, pp. 109–118.

[3] M. Al-Bassam, "SCPKI: A smart contract-based PKI and identity system," in *Proc. ACM Workshop Blockchain, Cryptocurrencies Contracts BCC*, 2017, pp. 35–40.

[4] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2019, pp. 403–412.

[5] *Randao: A Dao Working As Rng of Ethereum*. Accessed: Oct. 25, 2019. [Online]. Available: https://github.com/randao/randao

[6] A. Reutov. (2018). *Predicting Random Numbers in Ethereum Smart Contracts*. [Online]. Available: https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-c%ontracts-e5358c6b8620

[7] *A Scalable Architecture for on-Demand, Untrusted Delivery of Entropy*. Accessed: Oct. 25, 2019. [Online]. Available: https://provable.xyz/papers/random_datasource-rev1.pdf

[8] A. Coladangelo, "Smart contracts meet quantum cryptography," 2019, *arXiv:1902.05214*. [Online]. Available: http://arxiv.org/abs/1902.05214

[9] Z. Cai, J. Qu, P. Liu, and J. Yu, "A blockchain smart contract based on light—Weighted quantum blind signature," *IEEE Access*, vol. 7, pp. 138657–138668, 2019.
[10] Y. Feng, E. Torlak, and R. Bodik, "Precise attack synthesis for smart contracts," 2019, *arXiv:1902.06067*. [Online]. Available: http://arxiv.org/abs/1902.06067
[11] J. Gao, H. Liu, C. Liu, Q. Li, Z. Guan, and Z. Chen, "EASYFLOW: Keep ethereum away from overflow," 2018, *arXiv:1811.03814*. [Online]. Available: http://arxiv.org/abs/1811.03814
[12] D. Perez and B. Livshits, "Smart contract vulnerabilities: Does anyone care?" 2019, *arXiv:1902.06710*. [Online]. Available: http://arxiv.org/abs/1902.06710
[13] P. Zhang, F. Xiao, and X. Luo, "SolidityCheck: Quickly detecting smart contract problems through regular expressions," 2019, *arXiv:1911.09425*. [Online]. Available: https://arxiv.org/abs/1911.09425
[14] V. A. Siris, D. Dimopoulos, N. Fotiou, S. Voulgaris, and G. C. Polyzos, "OAuth 2.0 meets blockchain for authorization in constrained IoT environments," in *Proc. IEEE 5th World Forum Internet Things (WF-IoT)*, Apr. 2019, pp. 364–367.
[15] Y. Zhang, S. Kasahara, Y. Shen, X. Jiang, and J. Wan, "Smart contract-based access control for the Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1594–1605, Apr. 2019.
[16] R. Xu, Y. Chen, E. Blasch, and G. Chen, "BlendCAC: A smart contract enabled decentralized capability-based access control mechanism for the IoT," *Comput.*, vol. 7, no. 3, pp. 1–27, 2018.
[17] Y. Nakamura, Y. Zhang, M. Sasabe, and S. Kasahara, "Exploiting smart contracts for capability-based access control in the Internet of Things," *Sensors*, vol. 20, no. 6, p. 1793, 2020.
[18] M. A. Islam and S. Madria, "A permissioned blockchain based access control system for IOT," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 469–476.
[19] P. Wang, Y. Yue, W. Sun, and J. Liu, "An attribute-based distributed access control for blockchain-enabled IoT," in *Proc. Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2019, pp. 1–6.
[20] P. Mell, J. Dray, and J. Shook, "Smart contract federated identity management without third party authentication services," in *Open Identity Summit*, vol. 293. Gaithersburg, MD, USA: National Institute of Standards and Technology, 2019.
[21] H. Guo, E. Meamari, and C.-C. Shen, "Multi-authority attribute-based access control with smart contract," in *Proc. Int. Conf. Blockchain Technol. ICBCT*, 2019, pp. 6–11, doi: 10.1145/3320154.3320164.
[22] J. P. Cruz, Y. Kaji, and N. Yanai, "RBAC-SC: Role-based access control using smart contract," *IEEE Access*, vol. 6, pp. 12240–12251, 2018.
[23] Y. Lee and K. M. Lee, "Blockchain-based RBAC for user authentication with anonymity," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Sep. 2019, pp. 289–294, doi: 10.1145/3338840.3355673.
[24] M. U. Rahman, B. Guidi, F. Baiardi, and L. Ricci, "Context-aware and dynamic role-based access control using blockchain," in *Advanced Information Networking and Applications*, L. Barolli, F. Amato, F. Moscato, T. Enokido, and M. Takizawa, Eds. Cham, Switzerland: Springer, 2020, pp. 1449–1460.
[25] P. McCorry, S. F. Shahandashti, and F. Hao, "A smart contract for boardroom voting with maximum voter privacy," in *Financial Cryptography Data Security*, A. Kiayias, Ed. Cham, Switzerland: Springer, 2017, pp. 357–375.
[26] R. Tso, Z.-Y. Liu, and J.-H. Hsiao, "Distributed E-voting and E-bidding systems based on smart contract," *Electronics*, vol. 8, no. 4, p. 422, Apr. 2019.
[27] J. Lyu, Z. L. Jiang, X. Wang, Z. Nong, M. H. Au, and J. Fang, "A secure decentralized trustless E-Voting system based on smart contract," in *Proc. 18th IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun./13th IEEE Int. Conf. Big Data Sci. Eng. (TrustCom/BigDataSE)*, Aug. 2019, pp. 570–577.
[28] Y.-H. Chen, S.-H. Chen, and I.-C. Lin, "Blockchain based smart contract for bidding system," in *Proc. IEEE Int. Conf. Appl. Syst. Invention (ICASI)*, Apr. 2018, pp. 208–211.
[29] H. Desai, M. Kantarcioglu, and L. Kagal, "A hybrid blockchain architecture for privacy-enabled and accountable auctions," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 34–43.
[30] S. Wu, Y. Chen, Q. Wang, M. Li, C. Wang, and X. Luo, "CReam: A smart contract enabled collusion-resistant e-Auction," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 7, pp. 1687–1701, Jul. 2019.
[31] K.-H. Yeh, C. Su, J.-L. Hou, W. Chiu, and C.-M. Chen, "A robust mobile payment scheme with smart contract-based transaction repository," *IEEE Access*, vol. 6, pp. 59394–59404, 2018.
[32] G. Malavolta, P. Moreno-Sanchez, A. Kate, M. Maffei, and S. Ravi, "Concurrency and privacy with payment-channel networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 455–471, doi: 10.1145/3133956.3134096.
[33] X. Wang, W. Yang, S. Noor, C. Chen, M. Guo, and K. H. van Dam, "Blockchain-based smart contract for energy demand management," *Energy Procedia*, vol. 158, pp. 2719–2724, Feb. 2019, doi: 10.1016/j.egypro.2019.02.028.
[34] Y. Amanbek, Y. Tabarak, H. S. V. S. K. Nunna, and S. Doolla, "Decentralized transactive energy management system for distribution systems with prosumer microgrids," in *Proc. 19th Int. Carpathian Control Conf. (ICCC)*, May 2018, pp. 553–558.
[35] M. Afzal, Q. Huang, W. Amin, K. Umer, A. Raza, and M. Naeem, "Blockchain enabled distributed demand side management in community energy system with smart homes," *IEEE Access*, vol. 8, pp. 37428–37439, 2020.
[36] R. Tonelli, M. I. Lunesu, A. Pinna, D. Taibi, and M. Marchesi, "Implementing a microservices system with blockchain smart contracts," in *Proc. IEEE Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Feb. 2019, pp. 22–31.
[37] D. Nagothu, R. Xu, S. Y. Nikouei, and Y. Chen, "A microservice-enabled architecture for smart surveillance using blockchain technology," in *Proc. IEEE Int. Smart Cities Conf. (ISC)*, Sep. 2018, pp. 1–4.
[38] F. Daniel and L. Guida, "A service-oriented perspective on blockchain smart contracts," *IEEE Internet Comput.*, vol. 23, no. 1, pp. 46–53, Jan. 2019.
[39] S. Wang, L. Ouyang, Y. Yuan, S. Member, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-enabled smart contracts: Architecture, applications, and future trends," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 49, no. 11, pp. 2266–2277, Nov. 2019.
[40] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2017, pp. 303–312, doi: 10.1145/3087801.3087835.
[41] M. Bartoletti, L. Galletta, and M. Murgia, "A true concurrent model of smart contracts executions," 2019, *arXiv:1905.04366*. [Online]. Available: http://arxiv.org/abs/1905.04366
[42] V. Saraph and M. Herlihy, "An empirical study of speculative concurrency in Ethereum smart contracts," in *Proc. Int. Conf. Blockchain Econ., Secur. Protocols (Tokenomics)*, vol. 71, V. Danos, M. Herlihy, M. Potop-Butucaru, J. Prat, and S. Tucci-Piergiovanni, eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, pp. 4:1–4:15. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2019/11394
[43] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *J. Cryptol.*, vol. 3, no. 2, pp. 99–111, Jan. 1991, doi: 10.1007/BF00196791.
[44] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[45] I. Miers, C. Garman, M. Green, and A. D. Rubin, "Zerocoin: Anonymous distributed E-Cash from bitcoin," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 397–411.
[46] *Ethereum-White Paper*. Accessed: Jun. 5, 2019. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[47] N. Szabo. (1994). *Smart Contracts*. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literat%ure/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html
[48] Szabo. (196). *Smart Contracts: Building Blocks for Digital Markets*. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literat%ure/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html
[49] J. Liu and Z. Liu, "A survey on security verification of blockchain smart contracts," *IEEE Access*, vol. 7, pp. 77894–77904, 2019.
[50] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)," in *Princ. Secur. Trust*, M. Maffei and M. Ryan, Eds. Springer Berlin Heidelberg, 2017, pp. 164–186.
[51] D. Harz and W. Knottenbelt, "Towards safer smart contracts: A survey of languages and verification methods," 2018, *arXiv:1809.09805*. [Online]. Available: http://arxiv.org/abs/1809.09805
[52] Y. Murray and D. A. Anisi, "Survey of formal verification methods for smart contracts on blockchain," in *Proc. 10th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Jun. 2019, pp. 1–6.
[53] M. di Angelo and G. Salzer, "A survey of tools for analyzing ethereum smart contracts," in *Proc. IEEE Int. Conf. Decentralized Appl. Infrastruct. (DAPPCON)*, Apr. 2019, pp. 69–78.

[54] Y. Hu, M. Liyanage, A. Mansoor, K. Thilakarathna, G. Jourjon, and A. Seneviratne, "Blockchain-based smart contracts–applications and challenges," 2018, *arXiv:1810.04699*. [Online]. Available: http://arxiv.org/abs/1810.04699

[55] S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7, pp. 50759–50779, 2019.

[56] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Gener. Comput. Syst.*, vol. 105, pp. 475–491, Apr. 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X19316280

[57] *Eosio-Cleos/Cleos-Set*. Accessed: Jul. 10, 2019. [Online]. Available: https://developers.eos.io/eosio-cleos/v1.2.0/reference#cleos-set

[58] *Neocontract White Paper*. Accessed: Oct. 24, 2019. [Online]. Available: https://docs.neo.org/docs/en-us/basic/technology/neocontract.html

[59] *Eosio Api*. Accessed: Oct. 24, 2019. [Online]. Available: https://eosio.github.io/eosio.cdt/latest/modules

[60] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *J. ACM (JACM)*, vol. 27, no. 2, pp. 228–234, Apr. 1980.

[61] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain," *J. Inf. Process. Syst.*, vol. 14, no. 1, pp. 101–128, 2018.

[62] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *Proc. 31st Int. Symp. Distrib. Comput. (DISC)*, in Leibniz International Proceedings in Informatics (LIPIcs), vol. 91, A. W. Richa, Ed. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum Fuer Informatik, 2017, pp. 39:1–39:16. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/8004, doi: 10.4230/LIPIcs.DISC.2017.39.

[63] Y. Wu, P. Song, and F. Wang, "Hybrid consensus algorithm optimization: A mathematical method based on POS and PBFT and its application in blockchain," *Math. Problems Eng.*, vol. 2020, pp. 1–13, Apr. 2020, doi: 10.1155/2020/7270624.

[64] (2017). *Report of Investigation Pursuant to Section 21(a) of the Securities Exchange Act of 1934: The DAO*. Securities Exchange Act Of 1934. [Online]. Available: https://www.sec.gov/litigation/investreport/34-81207.pdf

[65] *Bitcoin Vs. Visa Blockchain.Us*. Accessed: Jun. 19, 2019. [Online]. Available: https://blockchain.us/bitcoin-vs-visa/

[66] M. Zhandry, "Quantum lightning never strikes the same state twice," in *Proc. 38th Annu. Int. Conf. Theory Appl. Cryptograph. Techn.*, in Lecture Notes in Computer Science, vol. 11478. Darmstadt, Germany: Springer, May 2019, pp. 408–438.

[67] A. Nordrum. *Popular Internet of Things Forecast of 50 Billion Devices by 2020 is Outdated*. Accessed: Oct. 25, 2019. [Online]. Available: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-o%f-things-forecast-of-50-billion-devices-by-2020-is-outdated

[68] W. C. Summers and E. Bosworth, "Password policy: The good, the bad, and the ugly," in *Proc. Winter Int. Synposium Inf. Commun. Technol. (WISICT)*, Dublin, Ireland: Trinity College Dublin, 2004, pp. 1–6.

[69] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong?: A qualitative usability study," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 311–328, doi: 10.1145/3133956.3134082.

[70] R. C. Marshall and M. J. Meurer, "The economics of auctions and bidder collusion," in *Game Theory and Business Applications* (International Series in Operations Research & Management Science), vol. 35, K. Chatterjee and W. F. Samuelson, Eds. Boston, MA, USA: Springer, 2002. [Online]. Available: https://link.springer.com/chapter/10.1007/0-306-47568-5_11#citeas

[71] D. Döderlein. (2019). *Moving Towards a Cashless Society in a Cash-Reliant World*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.forbes.com/sites/danieldoderlein/2019/03/18/moving-towards-% a-cashless-society-in-a-cash-reliant-world/#78bd855c236a

[72] *Microservices Vs Soa : What's the Difference*. Accessed: Mar. 25, 2020. [Online]. Available: https://www.edureka.co/blog/microservices-vs-soa/

[73] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: Many-to-many end-to-end encryption and key delegation for IoT," in *Proc. 28th USENIX Secur. Symposium (USENIX Secur.)*. Santa Clara, CA, USA: USENIX Association, Aug. 2019, pp. 1519–1536. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/kumar-s%am

VICTOR YOUDOM KEMMOE received the Bachelor of Science degree in computer science from Kennesaw State University, Georgia, USA, in 2018. He is currently pursuing the Master of Science degree in computer science with Kennesaw State University. He is also a Graduate Research Assistant with the Information and Intelligent Security Laboratory, Kennesaw State University. His research interests include applied cryptography, cybersecurity, blockchain and smart contracts, machine learning in cybersecurity, and quantum computing.

WILLIAM STONE received the B.S. degree in biomedical engineering from Mercer University, Macon, GA, USA, in 2013. He is currently pursuing the M.S. degree in computer science with Kennesaw State University, Marietta, GA, USA. He is currently working under the direction of Dr. Junggab Son as a Research Assistant at Kennesaw State University. His current interests include heavily in the domain of artificial intelligence and machine learning, focusing on but not limited to the implementation of data science strategies to solve cybersecurity problems. His other research interests include data privacy, graph theory, social networking, game theory, and econometrics.

JEEHYEONG KIM received the B.S.E. degree in computer science and engineering from Hanyang University, South Korea, in 2015. He is currently pursuing the M.S.-leading-to-Ph.D. degree in computer science and engineering with Hanyang University, South Korea. Since 2015, he has been with the Computer Science and Engineering, Hanyang University of Engineering, South Korea. His research interests include machine learning in cybersecurity and next wireless communications.

DAEYOUNG KIM received the B.S. degree in electronic and computer engineering and the M.S. degree in computer science and engineering from Hanyang University, South Korea, in 2005 and 2007, respectively, and the Ph.D. degree in computer science from Rutgers University, in 2019. He is currently a limited-term Assistant Professor of Computer Science with Kennesaw State University. His research interests include computer security and privacy, and mobile computing.

JUNGGAB SON (Member, IEEE) received the B.S.E. degree in computer science and engineering from Hanyang University, Ansan, South Korea, in 2009, and the Ph.D. degree in computer science and engineering from Hanyang University, Seoul, South Korea, in 2014.

From 2014 to 2016, he was a Post-Doctoral Research Associate with the Department of Math and Physics, North Carolina Central University. From 2016 to 2018, he was a Research Fellow and a limited-term Assistant Professor with the Department of Computer Science, Kennesaw State University, where he has been an Assistant Professor, since 2018. His research interests include applied cryptography, cybersecurity, privacy protection, blockchain and smart contract, and data science in cybersecurity. He is an Associate Editor of IEEE Access.