Sustainable Blockchain-Enabled Services: Smart Contracts

Craig Wright
nChain , London , UK
corresponding author: craig@ncrypt.com

Antoaneta Serguieva
nChain , London , UK
antoaneta@ncrypt.com

***Abstract*—This paper introduces some of the interdependent components within the multifaceted solution our team is developing towards accelerating the functionality, complexity and versatility of blockchain-enabled services. The focus here is particularly on introducing and bringing together selected individual components of the solution to achieve a synergistic effect in expanding the functionality of blockchain-enforced smart contracts. The contributions of this paper include: (i) proposing a method for automated management of contracts with hierarchical conditionality structures through an hierarchy of intelligent agents and the use of hierarchical cryptographic key-pairs; (ii) proposing a method for efficient and secure matching and transfer of smart-contract underlyings (entities) among disparate smart contracts/subcontracts; (iii) proposing a method for producing an hierarchy of common secrets to facilitate hierarchical communication channels of increased security, and applying this method both in the context of method (i) and method (ii); and (iv) proposing the use of distributed hash tables DHT in building secure and optimized repositories in the context of method (i) and in the context method (ii), where the former involves a DHT repository of smart contracts and the latter involves a DHT repository of entities underlying smart contracts that are being exchanged among different smart contracts and subcontracts. The smart-contract focused methods introduced in this paper contribute to the overall goal towards a sustainable adaptive mechanism for processing evolving volumes, versatility, and complexity of blockchain transactions, traffic, and services. Blockchain-enabled services are efficient, secure, automated, and allowing worldwide distribution of resources. They present a more efficient and sustainable alternative to current service infrastructures within a range of domains, particularly the legal and financial domains. They also set a sustainable infrastructure for emerging Internet-of-things services.**

Keywords—multifaceted scaling solution, smart contracts, hierarchical contract conditionality, hierarchical encription keys, common sectret, sustainable blockchain-enabled services.

I. INTRODUCTION

Blockchain technology aims at increasing volumes of versatile traffic of evolving complexity. The target is to enable complex services securely, sustainably, efficiently. Therefore, the target is to accelerate/scale blockchain functionality. While aiming at this target, the introduction/identification of criteria for the solution provides grounds for and allows comparison of alternative solutions. We argue in [1] that part of the criteria should include: (i) a long-term rather than short-term perspective, (ii) accommodation of new types of business models rather than focusing predominantly on a single business model, (iii) multiple facets/components contributing synergetic effects to the overall growth solution, (iv) maintaining the integrity and auditability of the blockchain. Such criteria lead to the longevity of the new technology, and thus contribute to the sustainability of the services it enables. In a range of domains, Blockchain-enabled services provide a viable alternative to current underperforming service infrastructures of unreliable security.

In this paper, we propose components of the overall solution that synergistically contribute to the functionality of smart contracts. Smart contracts are currently an area of active interest and emerging research. Still, innovative elements of the proposed methods in this paper have not been considered in the literature. We can relate the proposed methods to open active questions in the literature. In [2], the challenges are addressed that are involved in the validation and verification of smart contracts running over blockchains, considering that they might encode legal contracts written in natural language. The current paper also addresses these challenges, and in Section II here, a contract model is proposed along with a method for automated management of smart contracts enforced through the blockchain. In [3], it is discussed how the blockchain-IoT combination facilitates the sharing of services and resources leading to the creation of a marketplace of services between devices. In the current paper, the contract model is presented in terms of structured control conditions, and thus the method we propose in Section II for automated management of such conditions is directly applicable and beneficial to emerging IoT services. The method proposed in Section III here, for efficient and secure transfer of entities underlying smart contracts, also contributes to a solution of the question addressed in [3] about creation of a marketplace of services between IoT devices. We agree with the authors' view that the blockchain-IoT combination is powerful and can cause significant transformations across several industries. In [4], the authors consider the potential role of blockchain technology towards innovation and transformation of governmental processes. Their conclusion is that the processes can be changed to benefit most of the technology, if blockchain applications are customized to fit with process requirements. The secure repository of smart contract templates, i.e. smart process templates, which we propose in Section II here, may contribute to achieving such benefit. Its mechanism involves each institution accessing the repository of smart templates to derive its institutional semi-templates and continuously amend them through reuse.

978-1-5386-2715-0/17/$31.00 ©2017 IEEE

The remainder of the paper is organized as follows: Section II considers the automated management of blockchain-enforced smart contracts, Section III is focused on smart contracts' efficiency and security, and Section IV states the conclusions and indicates further research focus.

II. Automated Management of Blockchain-Enforced Smart Contracts

A. Benefits

Blockchain-enforced smart contracts extend the functionality of Bitcoin and broaden the type of services facilitated through the blockchain technology. A range of legal and financial services, which are currently inefficient and potentially unsustainable, can be and maintained through the blockchain-enforced smart contracts. Newly emerging services within the Internet of Things (IoT) can also benefit from and accelerate through the automated management of blockchain-enforced structured control conditions. The structured conditions are referred to as 'contracts', using the broader definition of this terminology that is not restricted to legal contracts. The counterparties can be institutions, individuals, intelligent computing agents, or IoT devices. The method proposed here simplifies the management of such contracts, and thus provides for scaling the versatility and complexity of blockchain-enabled services. This method is a synergistic implementation and adaptation of key components [5][6][7][8] within a multi-faceted solution for sustainable scaling of blockchain functionality [1].

The proposed innovation provides:

- a security-enhanced control mechanism that permits or prohibits access to an off-chain resource in an intelligent manner, and allows contracts to be time-bound, condition-bound, or open-ended and rolling-over;
- a formalism for translating any contract or structured conditions into a corresponding $contract\_model$ and its deterministic finite automation DFA;
- a mechanism for implementing intelligent computing agents that follow and execute the logic embedded in $contract\_models$ and $DFAs$, and for holding a secure, public record of agents' code on the blockchain;
- a mechanism for turning any unspent crypto-transaction $UTXO$ into a smart contract, and to structure a hierarchy of subcontracts allowing control over different aspects of the overall contract to be partitioned;
- a mechanism to hold a secure, public record of contracts on the blockchain, in a manner that allows automated determination of their validity, and release of their details to authorized entities upon validation.

The initiation, stages of execution, and closure of a smart contract are recorded on the blockchain through the creation, broadcasting and recording of bitcoin transactions. This allows verifying the current or past existence of a contract by looking up corresponding blockchain transactions. The stage of execution of an existing contract can also be verified, by looking up recorded transactions corresponding to the initiation or closure of its subcontracts. The logic of the structured control conditions from the $contract\_model$ is embedded within the locking/unlocking scripts of transactions, as well as within other transaction elements such as the nLockTime field. The contract's logic is enforced through individual actions and overall behavior of one or several intelligent-agent applications. For accountability and for reuse, parts of the codified overall behavior of an $agent$, or links to off-chain repositories where the code is stored, are also embedded within the locking/unlocking scripts of transactions recorded on the blockchain. The access to these repositories, as well as to repositories storing the contract documents and the $contract\_models$, is selective, partial, and secure. The access to the code or parts of it, and the access to the contract or its subcontracts, matches the requirements and character of the contract and the preferences of the multiple counterparties involved.

B. Contract Model and Tokenisation

A repository of contracts can be implemented as a distributed hash table (DHT) [9] across storage resources within the Bitcoin network. A hash of a contract is generated and stored as metadata within a blockchain transaction, and serves as a DHT look-up key for referencing the contract from the blockchain. The use of a master encryption key and multiple sub-keys by each counterparty, proposed in Section III.A of this paper, allows for secure access to the contract and its subcontracts in the repository by authorized counterparties. Auditing authorities are also provided with access corresponding to the scope of each audit. For example, a building company in England enters into a contract with multiple counterparties to deliver a new development. The contract has multiple subcontracts, and one of them addresses the issuance of a plans certificate, as required by the relevant regulation. One of the counterparties for this subcontract is the building control department of a Local Authority. The control department have access to this and probably further subcontracts in the repository, but may not have access to subcontracts specifying remunerations for the pool of builders, as such information may be confidential. Another subcontract of the building contract concerns the issuance of a final certificate, as required by regulation, and one of the counterparties here is an approved inspector. His access is defined by analogy with the former case. The building company and its auditors have access to all of the subcontracts in the repository, and to all blockchain transactions enforcing the contract and its subcontracts. The auditing firm may not be a counterparty to any subcontract, and may access the repository after the completion of the contract. The auditors still are able to retrieve the relevant information and verify transactions and the past execution of the contract, and thus assess the performance of the building company.

The use of multiple encryption sub-keys also allows that trusted third parties may modify some of the conditionality and subcontracts of a stored contract. This translates into an amended behavior of the intelligent $agents$ enforcing the contract. The blockchain transactions, which the $agents$ create for the amended instantiation of the contract, include amended parameters in comparison with the transactions they

created for a previous instantiation. For example, the renewal of a lease contract or the renewal of a rental contract may involve amended amounts and rates. Multiple encryption sub-keys (see Section III.A) further facilitate establishing a $common\_secret$ [6] for each pair of counterparties on each subcontract. A $common\_secret$ based encryption allows for a secure channel of communication between a pair of counterparties, when it is necessary to negotiate values of parameters related to a subcontract such as lease rates and rental amounts. Differing $common\_secrets$ between the same counterparties on different subcontracts provide the additional security.

Having considered the mechanism of a DHT repository for smart contracts, we now turn to the $contract\_model$. Some of its elements are listed in the following Definition:

Definition 1: $contract\_model$ elements

- a codification scheme that allows a complete description of any type of contract, and is based on constructs such as XBRL, XML, JSON, etc.;
- a deterministic finite automaton DFA 'translating' the contract logic and conditionality, where DFA can be fully defined within the codification scheme and consists of:
  - a set of parameters and an indication where to source them;
  - a set of state definitions;
  - a set of transitions between the states, including the trigger for the transition and the rules followed during the transition;
  - rules definition table;
- definitions of the specific parameters for this instance of the contract;
- a 'compiler' converting the codification scheme into intelligent-agent code and bitcoin script.

DFA is the essential component of the $contract\_model$ and is implemented as an agent-based process. For complex contracts, the DFA implementation involves a sequence of processes or parallel sequences of processes. Processes access off-chain resources, and/or monitor the values of off-chain and on-chain parameters, and/or create different blockchain transactions under each conditionality step within the active contract or under different triggers and parameter values. Agent-based processes also send multisig transactions for signature by counterparties prior to broadcasting them, and/or communicate off-chain to inform counterparties or trusted third parties, and/or verify on-chain records related to the execution of past contracts. A $master\_agent$ can manage a hierarchy of $subordinate\_agents$ that carry out tasks defined in a smart contract. The $master\_agent$ controls, directs, monitors, and authorizes the activities of each $subordinate\_agent$, and also coordinates their activities. The $master\_agent$ and $subordinate\_agents$ communicate to execute the variety of tasks.

Having introduced the $contract\_model$, the first step in its implementation is to indicate the existence of a contract. The $master\_agent$ on this contract creates the first transaction T associated with the contract, broadcasts it to the Bitcoin network, monitors when it is recorded on the blockchain and extracts its ID. Thus the existence of the contract and the time when it became active are a permanent auditable record publicly available on the blockhcain, although the details of the contract may not be publicly available. The $master\_agent$ uses a pay-to-script-hash $P2SH$ address when creating transaction T. For such transaction to be spent, a recipient must provide a script matching the $P2SH$ script hash as well as data that makes the script evaluate to true. $P2SH$ is determined using the contract metadata. After T, a number of further transactions follow that are associated with the contract and its subcontracts. They are created by the $master\_agent$ or by $subordinate\_agents$.

A range of these transactions involve tokenization. In the rest of this Section II.B, we introduce and extend tokenization mechanisms from [10][11]. The use of master keys and hierarchies of sub-keys by the $master\_agent$ and $subordinate\_agents$ – where each agent has its own master key and sub-keys, in combination with the tokenization mechanism, allows for contract structure of any complexity to be created and implemented and for its subcontracts and schedules to be confirmed, triggered, executed, and terminated. In this context, a token can non-exhaustively be used to represent and detail, in the form of a bitcoin transaction, the transferable rights conferred by a specific contract or subcontract. The $token$ efficiently uses metadata comprising only three parameters:

- a number of units available overall, $total\_units$;
- a quantity of $transfer\_units$ to be transferred from a sender to at least one recipient;
- a $pegging\_rate$ for calculating a value for the $transfer\_units$ as pegged to the cryptocurrency.

Such $token$ can represent any type of transferable rights, and thus common algorithms are reused as parts of the codified behavior of different $agents$. The $token$ is either divisible or non-divisible, corresponding to the transfer of divisible or non-divisible rights. In the latter case, the value of the parameter $pegging\_rate$ is set to 0. For divisible rights, the tokenized value transferred in the transaction output is tied to the underlying bitcoin amount via a non-zero $pegging\_rate$, and the transferred rights are thus specified in terms of a $pegging\_rate$. An example of divisible tokens are those transferring some quantity of bearer shares, where a share is a percentage ownership, i.e. a pegging rate, of the company. An example of non-divisible tokens is those transferring bearer bonds, where a bond is redeemable for an exact amount of a fiat currency such as USD, GBP, etc. If some smart contracts or their subcontracts involve issuing and selling bearer shares or bonds then among the bitcoin transactions being created during the implementation of the contracts are also transaction representing the transfer of tokenized quantities of shares or bonds.

Furthermore, the number of units available overall in the tokenized rights is either limited or unlimited. In the former case, the parameter $total\_units$ is fixed and always greater than 0. An example of limited units is the shared ownership of a race horse, such as $total\_units = 10$ and $pegging\_rate = 10\%$, or $total\_units = 25$ and $pegging\_rate = 4\%$. An example of unlimited overall units

is the inventory of a product in a warehouse, as the inventory can be increased at any time and allow an increase in the tokenized amount of product units. Bearer shares are also an example of potentially unlimited units, as the company can issue more shares. In some cases of unlimited units, the current total number of units does not matter for the transfer of ownership, and the value of parameter $total\_units$ in the token is set to 0. Such is the inventory example, where one unit is one instance of the transferable product, i.e. a T-shirt in the warehouse stock of T-shirts. In other cases, the current number of potentially unlimited total available units matters. As this number is variable, a $subordinate\_agent$ monitors it and identifies its correct value for each instantiation of such divisible token. In case of bearer shares, transferring tokenized ownership rights involves parameter values as follows:

$$total\_units = \text{current number of issued and non-redeemed shares}$$

$$pegging\_rate = \frac{1}{total\ units}\%$$

A $subordinate\_agent$ monitors the number of issued shares and the number of redeemed shares, and identifies the current value of $total\_units$.

A final point here is that in the Bitcoin protocol, every transaction output must have a non-zero bitcoin (฿) amount to be considered valid. As the token is in the form of a bitcoin transaction, it has an underlying bitcoin value, which is the ฿ amount attached to the output. This amount is arbitrary, though at least equal to the set minimum called 'dust', as such transaction is only a facilitator of ownership transfer. The true value of transferred rights is found through the metadata parameters. Non-divisible tokens are carried by $dust$. Divisible tokens, under the proposed specification, use underlying ฿ amounts in a meaningful manner by linking these amounts to the $pegging\_rates$. An underlying ฿ amount is chosen so that when a divisible token is split into several transaction outputs, the minimum of the split divisions is carried by $dust$. Having considered the contract model and a relevant tokenization mechanism in this Section, the next Section II.C will focus on a contract's conditionality and subcontracts.

C. Master Contracts' Conditionality and Subcontracts

A $master\_contract$ is interpreted as remaining in effect, as long as there is a valid unspent transaction output $UTXO$ representing the existence of this master contract. That unspent state is influenced and altered by the behavior of the $master\_agent$ and $subordinate\_agents$. Agents' behavior is controlled through conditions in the $master\_contract\ model$ that translate provisions and stipulations from the contract document. For example, a condition may involve that the contract expires when the values of some variables reach specified thresholds. Transactions associated with a contract are a permanent, unalterable public record of the contract's existence and current status. The termination of a contract is also recorded on the blockchain, as a spent output in a crypto-transaction. Anyone can use a software module to determine at what stage of its execution a contract is or whether it has been terminated.

In this context, a $subcontract$ is a contract that is directly related to an existing $master\_contract$, where the metadata in transactions associated with the $subcontract$ contain a pointer or a reference, along with its hash, to the location of the $master\_contract$ within the DHT repository. The existence of a $subcontract$ is implemented, similarly to the $master\_contract$'s existence, as an $UTXO$ with a deterministic redeem script address. The $subcontract$ is interpreted as being completed when this $UTXO$ is spent. The mechanisms used for creating the deterministic addresses in the $P2SH$ transactions associated with $subcontracts$, within the $master\_contract$'s conditionality structure, include the following:

- derive a new public sub-key using seed information;
- if an entry for a $subcontract$ does not exist in the repository of contracts, then create an entry so that:
  - the entry is a description of this $subcontract$;
  - the description is in compliance with the codification scheme for describing contracts (see Definition 1)
  - this description includes a reference to the $master\_contract$ entry in the repository
- once the $subcontract$ entry is created or if such entry already exists in the repository, then add the reference to this entry to the metadata of transactions associated with the $subcontract$;
  - the metadata may also include a reference to the $master\_contract$ entry;
- use the amended metadata to create $P2SH$ addresses.

A use-case for creating a subcontract is described in Table 1.

TABLE I. ISSUING A SUBCONTRACT BASED ON AN EXISTING CONTRACT

Step	Details
one	The $master\_agent$ derives, using a seed value, a new public sub-key from its master public key used to create the $master\_contract$. The $master\_contract\_issuer$ derives, using the same seed, a new public sub-key from his master public key used to sign the $master\_contract$. The $master\_contract\_issuer$ can be an institution or an individual responsible off-chain for the master contract. The seed value is based on information from the $master\_contract$. Examples of appropriate seeds include: -Transaction ID of the $UTXO$ published on the blockchain to indicate the existence of the $master\_contract$; -Redeem script hash securing the $master\_contract$ and created by the $contract\_issuer$ or the $master\_agent$ in an m-of-n multi-signature structure, where at least the public keys of the $contract\_issuer$ and the $master\_agent$ must be supplied to this script. Depending on the terms of the $master\_contract$, other signatures may also be required, including the signatures of a $subordinate\_agent$ and a $subcontract\_issuer$, where the $subcontract\_issuer$ has responsibilities for the $subcontract$ in the off-chain world. The number of all these signatures is m, while n further includes the number of metadata blocks. *Note: If a $sub\_subcontract$ is being created instead of a $subcontract$, then this step may include a $subordinate\_agent$ deriving a new public sub-key, though a seed value, from its master public key used to sign the parent $subcontract$. All the $master\_agent$, the $master\_contract\_issuer$, the $subordinate\_agent$, and the $subcontract\_issuer$ use the same seed to derive a sub-sub-

Step	Details
	key or a sub-key, within each one's hierarchy of public keys, from the corresponding parent key. A parent key for the different signatories in this case may be either their master key or their sub-key.
two	Depending on the nature of the $subcontract$ being created, the $master\_agent$ either: -uses the location and hash of the $master\_contract$ entry in the repository of contracts; or -embeds a link to the $master\_contract$ entry within the $subcontract$ entry of the repository, and stores the location of the subcontract entry and secure hash of it for later use. *Note: the contract repository can be public, private or semi-private, depending on the nature of $contracts$.
three	The $master\_agent$ creates a redeem script covering the $subcontract$ being secured, in an m-of-n multi-signature structure, where m is the number of compulsory signatures and n further includes the number of metadata blocks. The number of metadata blocks is at least two, the reference to the $master\ contract$ repository entry and the reference to the $subcontract\_entry$.
four	The $master\_agent$ or the $maste\_contract\_issuer$ pays a nominal ฿ amount to the redeem script calculated in step ***three***, through a standard pay-to-script-hash $P2SH$ transaction.
five	The $master\_agent$ waits until the $subcontract$ transaction has been published onto the blockchain and extracts its ID.
six	***six A*:** For a fixed-duration $subcontract$, the $master\_agent$ then creates a new transaction, with a lock-time set to the expiry time of the $subcontract$, paying the output from step ***five*** back to the $master\_agent$'s public sub-key hash or to the $master\_contract\_issuer$'s public sub-key hash.
	six B*:** For a $subcontract$ with no fixed duration, the repay script in the new transaction created at step ***six is not time-locked but implemented as an m-of-n multi-signature element. This transaction requires a sign-off from a $subordinate\_agent$ monitoring the termination conditions for the $subcontract$, and may be a sign-off from a further third party. The multi-signature element may state "subject to sign-off by <x>". The new transaction is then circulated to the required signatories to sign, which include at least <x>. Such transaction has two outputs: the fee to <x> + the payment of the generated $UTXO$.

The mechanism described at step *six* in Table 1 is also used to monitor further types of conditions within a given $master\_contract$. For example, if a contract is worth Z฿, with Z_1฿, ⋯ , Z_k฿ to be paid at checkpoints 1 through k, then this is implemented as a $master\_contract$ plus k $subcontracts$. Each of the $subcontracts$ is marked as complete using the same or different signatories ($agents$, notaries, surveyors, brokers). Thus, a public record is maintained showing which of the conditions attached to the $master\_contract$ have been met and which are yet to be met. For $i\epsilon\{1, \cdots, k\}$, a $subordinate\_agen_i$ monitors the state of $subcontract_i$ and triggers payment Z_i, once the monitoring confirms that $subcontract_i$ is complete.

Bitcoin transactions implementing an example scenario of contract conditionality are shown in Fig. 1. This scenario corresponds to the building contract from Section II.B. The contract includes at least two conditions requiring a planning approval through the issuance of plans certificate and a building-standard approval through the issuance of a final certificate, correspondingly. The building company often enters in such multiple-counterparty contracts to delivery new buildings. Therefore, it is reasonable to assume that the $contract\ model$ of such contracts already exists, and that there is an entry in the contract repository that can be reused/reissued by instantiating it with amended counterparties and parameters. The 'template' contract can be reused simultaneously by several active instantiations, when the building company works in parallel on several projects that target the delivery of different properties. The simultaneous instantiations may also be due to different building companies having an active project each, or having more than one active project each. When a building company reuses the repository entry for the $template\_contract$ for the first time, it creates a new repository entry that acts as the company's own template from then on. That latter template, or rather semi-template, may embed a link to the repository record of the former template. When reusing the semi-template next, the company only appends a line of metadata to the repository entry for that semi-template, and do not create a new repository entry. The appended metadata plays a key role in creating, monitoring and spending Bitcoin transaction that implement the corresponding instantiation of the contract. The metadata in such transactions include a reference to the company's $semi\_template\_contract$, and a pointer to the line in it containing the specific metadata for this instantiation of the semi-template.

Representation of the Existence of a New Instance of the Property Building Contract
Transaction-ID: $master\_agent$-S5-T1
Version number
Number of inputs: 1
Previous Transaction Output: <$master\_agent's$ previous unspent BTC output - assume Y Satoshi>
Previous Transaction Output Index: IDX-00
Script length
ScriptSig: Sig-$master\_agent$ PubK $master\_agent$
Sequence number
Number of outputs: 2
First Output value: Z_1 <Z_1 is less than Y>
First Output script length
First Output script: OP_HASH160 <redeem script hash> OP_EQUAL Redeem Script: requires 2 out of $building\ company$ and $master\_agent$ to conclude: OP_1AssetMetaDataA AssetMetadataB PubK- $master\_agent$ PubK- $building\ company$ OP_4 OP_CHECKMULTISIG**u**
Second Output value: Z_2 <$Z_1 + Z_2$ is less than Y>
Second Output length
Second Output script: OP_DUP OP_HASH160 <PubK-$master\_agent$ Hash> OP_EQUALVERIFY OP_CHECKSIG
LockTime
Creation of a Subcontract by the Master Agent using his first derived key to confirm planning approval
Transaction-ID: $master\_agent$-S5-T2
Version number
Number of inputs: 1
Previous Transaction Output: $master\_agent$-S5-T1
Previous Transaction Output Index: IDX-01
Script length
ScriptSig: Sig-$master\_agent$ PubK $master\_agent$
Sequence number
Number of outputs: 2
First Output value: Z_3 <Z_3 is less than Z_2>
First Output script length

First Output script: OP_HASH160 <redeem script hash> OP_EQUAL Redeem Script requires *building control department* to approve and *building company* to approve: OP_2AssetMetaDataA AssetMetadataB PubK-*master_agent* SK1 PubK-*building control department* PubK-*building company* OP_5
Second Output value: Z_4 <$Z_3 + Z_4$ is less than Z_2>
Second Output length
Second Output script: OP_DUP OP_HASH160 <PubK-*master_agent* Hash> OP_EQUALVERIFY OP_CHECKSIG
LockTime
Creation of a Subcontract by the Master Agent using his second derived key to confirm building-standard approval
Transaction-ID: *master_agent*-S5-T3
Version number
Number of inputs: 1
Previous Transaction Output: *master_agent*-S5-T2
Previous Transaction Output Index: IDX-01
Script length
ScriptSig: Sig-*master agent* PubK *master_agent*
Sequence number
Number of outputs: 2
First Output value: Z_5 <Z_5 is less than Z_4>
First Output script length
First Output script: OP_HASH160 <redeem script hash> OP_EQUAL Redeem Script requires *approved inspector* to approve and *building company* to approve: OP_2AssetMetaDataA AssetMetadataB PubK-*master_agent*SK2 PubK-*approved inspector* PubK-*building company* OP_5
Second Output value: Z_6 <$Z_5 + Z_6$ is less than Z_4>
Second Output length
Second Output script: OP_DUP OP_HASH160 <PubK-*master_agent* Hash> OP_EQUALVERIFY OP_CHECKSIG
LockTime
Planning Authority Sign-off
Transaction-ID: *master_agent*-S5-T4
Version number
Number of inputs: 1
Previous Transaction Output: *master_agent*-S5-T2
Previous Transaction Output Index: IDX-01
Script length
ScriptSig: Sig-*building control department* Sig-*building company* OP_2AssetMetaDataA AssetMetadateB PubK-*master_agent* SK1 PubK- *building control department* PubK-*building company* OP_5 OP_CHECKMULTISIG
Sequence number
Number of outputs: 1
Output value: Z_7 <Z_7 is less than Z_3>
Output script length
Output script: OP_DUP OP_HASH160<*building control departmen*t Hash>OP_EQUAL VERIFY OP_CHECKSIG <The *building control departmen*t is paid a fee in Satoshi>
LockTime

Figure 1. Creating crypto-transactions corresponding to contract and subcontract start, execution and completion.

Within the automated management of a building company's *semi_template_contract*, a *master_agent* associated with the semi-template monitors for a new line of parameters being appended. New lines are appended by the building company. The mechanism involves the company routinely allocating some amount to the *master_agent*, so that the agent can activate at any time the first steps in its algorithm on issuing a new instance of the contract. The first step in that algorithm is the creation and broadcast of the transaction shown in dark shade in Fig. 1, and extracting its ID T_1 after the transaction is recorded on the blockchain. Thus T_1 becomes a secure, immutable, and publically available electronic record of the existence of the new contract in the physical world. The amount Y accessed for this step by the *master_agent* can be small. The amount allocated by the building company for access by the master agent is reviewed at routine intervals. At the end of an interval, the balance (excluding a set minimum) is automatically returned to the building company, and it is assessed if and what amount to make available to the agent in the next period. When the activity of a company is more versatile and it is captured through several semi-templates of different type, then the allocation and reallocation of ฿ amounts to the *master_agents* of the templates is managed and optimized by a higher-hierarchy agent called *templates_manager*. The reallocation is based on the evaluation of the prevailing performance and usual needs of the *master_agents*. That performance and needs are linked to the type of business line a semi-template is supporting within the company's business portfolio, and the performance of the company along the different business lines.

Following T_1, Fig. 1 presents next (in light grey) transactions related to two of the subcontracts that the *master_agent* of this template manages within an instantiation of the *template_contract*. First, T_2 indicates that a $subcontract_1$ for getting a planning approval exists, and next, T_3 indicates that a $subcontract_2$ exists for getting a building-standard approval. On the other hand, transaction T_4 confirms that a planning approval is received, and pays the fees to the local authority's building control department. Therefore, the first $subcontract_1$ is now closed. Of course, the complete conditionality structure a building contract is more complex, involves are larger number of *subcontracts*, and may involve *sub_subcontracts* and *subordinate_agents*. However, even in Fig.1 the hierarchical structure emerges, and shows that the *master agent* derives a secret (private) sub-key SK_1 for managing $subcontract_1$ and a secret sub-key SK_2 for managing $subcontract_2$. Section III next discuses hierarchical structures in more detail.

III. SMART CONTRACTS' EFFICIENCY AND SECURITY

A. Hierarchical Structures of Contracts, Crypto-keys, and Common Secrets

The automated management of smart contracts [5] introduced in Sections II is one of the components in the multifaceted solution for scaling blockchain functionality. Section II.C indicates that managing contract conditionality is assisted by a hierarchical structure of public/private key-pairs. The derivation of crypto-key hierarchies constitutes another component [6][12] of the overall solution, and that component assists and accelerates the function related components. We emphasize that synergistic inter-dependencies within the multifaceted solution accelerate the overall functionality and efficiency.

Let us consider complex contract conditionality implemented through an hierarchy of *subcontracts* and *sub_subcontracts*, and assisted through an hierarchy of sub-keys and sub-sub-keys. Fig. 2 shows a tree structure in blue representing the hierarchical contract conditionality, and a corresponding three structure in red representing hierarchical keys needed to assist the implementation of the complex contract. Each element in the red tree corresponds to a public/private key-pair, created by adding multiply-rehashed relevant information. That information may include IDs of existing transactions or metadata from existing entries in the contracts repository. Though only the secret (private) keys SK are indicated in the red tree, for every derived secret key, a corresponding public key PK is also derived. Therefore, the tree corresponds to a hierarchy of asymmetric public/private key-pairs PK/SK. For clarity of introducing the mechanism, it is assumed that the *master_agent* MA manages all *subcontracts* and *sub_subcontracts*. In practice, some of these elements of the blue structure can be managed by *subordinate_agents*. Notice that each element of the blue structure is implemented by at least two transactions, i.e. indicating that a new (sub-sub-)contract exists and then terminating it. This is the case with transactions T_2 and T_4 in Fig. 1, for example. Therefore, the pair in each element of the red structure is used to sign and redeem scripts in at least two transactions.

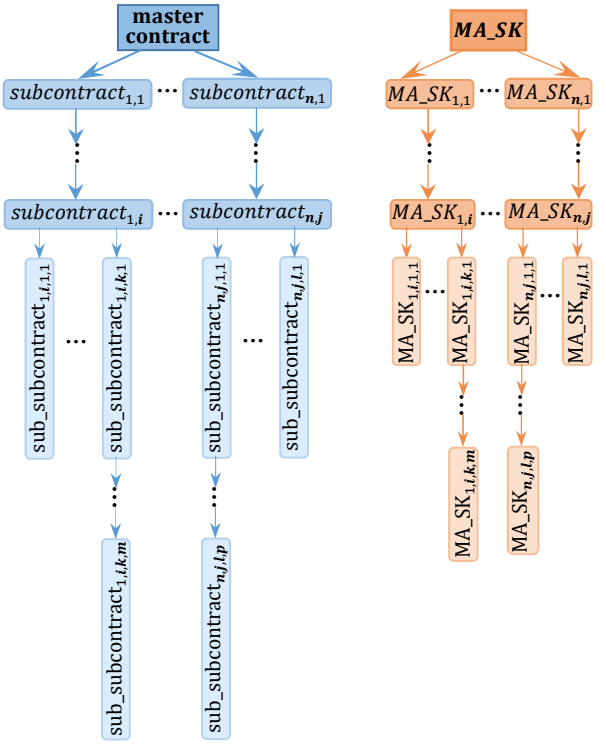

Figure 2. Hierarchical contract conditionality and correspomding derived tree-structure of keys.

In Fig. 2, MA_SK refers to the master private key of the *master_agent*, and $MA\_SK_{1,1}$ to $MA\_SK_{n,1}$ are private sub-keys of the same agent, where $subcontract_{1,1}$ to $subcontract_{n,1}$ can be executed in parallel. On the other hand, $MA\_SK_{1,1}$ and $MA\_SK_{1,i}$ are the private sub-keys of this agent, where $subcontract_{1,1}$ to $subcontract_{1,i}$ can only be executed in sequence. Next, $MA\_SK_{1,i,1,1}$ to $MA\_SK_{1,i,k,1}$ are MA 's private sub-sub-keys, where $sub\_subcontract_{1,i,1,1}$ to $sub\_subcontract_{1,i,k,1}$ can be executed in parallel only after $subcontract_{1,1}$ to $subcontract_{1,i}$ are executed in sequence. Finally, $MA\_SK_{n,j,1,1}$ to $MA\_SK_{n,j,l,p}$ are MA's private sub-sub-keys, where $sub\_subcontract_{n,j,1,1}$ to $sub\_subcontract_{n,j,l,p}$ can only be executed in sequence and only after $subcontract_{n,1}$ to $subcontract_{n,j}$ are executed in sequence. Notice that some *subcontracts* can serve as *master_contracts* for the *sub_subcontracts* that follow below them in the hierarchical structure. Thus, $subcontract_{1,i}$ can act as a *master_contract* or rather as a *submaster_contract* for $sub\_subcontract_{1,i,1,1}$ to $sub\_subcontract_{1,i,k,1}$ as well as for $sub\_subcontract_{1,i,k,1}$ to $sub\_subcontract_{1,i,k,m}$.

In the general case, any information can serve the role of a seed. However, the information may also be meaningful in the contexts that the hierarchy of keys is used. We choose here as *aster_seed* M , the redeem script hash securing the *master_contract* and created in an m-of-n multi-signature structure. Further, a *sub_master* seed SM is chosen as the redeem script hash securing a *sub_master_contract* and created in an m-of-n multi-signature structure. From Fig. 2, it can deducted that at least sub-master seeds $SM_{1,i}$ and $SM_{n,j}$ must be chosen, corresponding to $submaster\_contract_{1,i}$ and $sub\_master\_contract_{n,j}$. The seeds are involved in producing the generator values $GV_{1,1}$ to $GV_{n,j,l,p}$, when deriving the tree of asymmetric cryptographic key-pairs. Key derivation starts with the *master_agent* selecting a random value for the base point B , and communicating it to the *contract_issuer* . The base point is applied, as described below, to derive a public key from a corresponding private key, in order to complete an asymmetric cryptographic key-pair using Elliptic Curve Cryptography (ECC).The base point can also be communicated to any other signees on transactions created in implementing the hierarchical contract conditionality, particularly if they have a significant role in the structure and that role involves communications/negotiations in relation to a number of elements in the structure. We will introduce the mechanism first focusing on the *master_agent* and the *contruct_issuer*, and their meaningful hierarchies of cryptographic key-pairs and *common_secrets* . However, this mechanism can be applied accordingly when other signees also derive their hierarchies of key-pairs and *common_secrets*. The mechanism can further be adapted to the case when some branches of the conditionality structure are managed by *submaster_agents*.

Considering Fig. 2, the *master_agent* MA starts with its valid secret key MA_SK. Any 256-bit number from 0×1 to $0 \times$ *FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140* is a valid ECDSA private key [13], where ECDSA abbreviates the Elliptic Curve

Digital Signature Algorithm. Then, MA derives its hierarchy of private keys as follows:

$$\left.\begin{aligned} MA\_SK_{r,1} &= MA\_SK + GV_{r,1} \\ GV_{r,1} &= SHA\_256\big(M, L_{r,1}\big) \end{aligned}\right\} \text{ for } 1 \le r \le n \quad (1)$$

$$MA\_SK_{1,r} = MA\_SK_{1,r-1} + SHA\_256^{r-1}(M) \quad \text{for } 2 \le r \le i \quad (2)$$

$$\left.\begin{aligned} MA\_SK_{1,i,r,1} &= MA\_SK_{1,i} + GV_{1,i,r,1} \\ GV_{1,i,r,1} &= SHA\_256\big(SM_{1,i}\,, L_{1,i,1,1}\big) \end{aligned}\right\} \text{ for } 1 \le r \le k \quad (3)$$

$$MA\_SK_{n,j,1,r} = MA\_SK_{n,j,1,r-1} + SHA\_256^{r-1}\big(SM_{n,j}\big) \quad \text{for } 2 \le r \le p \quad (4)$$

where

$$SHA\_256^{r}(M) = SHA\_256\big(SHA\_256^{r-1}(M)\big) \quad (5)$$

For $1 \le r \le n$, generator value $GV_{r,1}$ is produced using the concatenation $\big(M, L_{r,1}\big)$ of the redeem script hash M of the $master\_contract$ and the $template\_contract$ hash $L_{r,1}$. Here, $subcontract_{r,1}$ is an instantiation of the $semi\_template\_contract$ which is being amended from the $template\_contract_{r,1}$. Further, for $1 \le r \le k$, generator values $GV_{1,i,r,1}$ use the $submaster\_seed$ $SM_{1,i}$ instead of the $master\_seed$ M. Also, $GV_{1,i,r,1}$ are produced by analogy to $GV_{r,1}$, as $sub\_subcontracts_{1,i,r,1}$ for $1 \le r \le n$ are executed in parallel, similarly to the way $sub\_subcontracts_{r,1}$ for $1 \le r \le k$ are executed in parallel. On the other hand, generator values $GV_{1,r}$ are produced by rehashing the $master\_seed$ M for $2 \le r \le i$, and values $GV_{n,j,1,r}$ are produced by rehashing the $submaster\_seed$ $SM_{n,j}$ for $2 \le r \le p$.

Next, Elliptic Curve Cryptography (ECC), properties of elliptic curve operations, and the base point B are used to complete the asymmetric cryptographic key-pairs and derive the public keys of the $master\_agent$. The operator + in (6-10) stands for scalar addition and the operator × refers to elliptic curve point multiplication. Having that elliptic curve cryptography algebra is distributive, hierarchy of public keys of the $master\_agent$ is produced as follows:

$$MA\_PK = MA\_SK \times B \quad (6)$$

$$MA\_PK_{r,1} = MA\_PK + GV_{r,1} \times B\,, \text{ for } 1 \le r \le n \quad (7)$$

$$MA\_PK_{1,r} = MA\_PK_{1,r-1} + SHA\_256^{r-1}(M) \times B \quad \text{for } 2 \le r \le i \quad (8)$$

$$MA\_PK_{1,i,r,1} = MA\_PK_{1,i} + GV_{1,i,r,1} \times B \text{ for } 1 \le r \le k \quad (9)$$

$$MA\_PK_{n,j,1,r} = MA\_PK_{n,j,1,r-1} + SHA\_256^{r-1}\big(SM_{n,j}\big) \quad \text{for } 2 \le r \le p \quad (10)$$

where $GV_{r,1}$ are the generator values used in (1) and $GV_{1,i,r,1}$ are the generator values used in (3). By analogy with the $master\_agent's$ hierarchy of key-pairs, the public/private key pairs of the $contract\_issuer$ CI are derived as follows:

$$CI\_SK_{r,1} = CI\_SK + GV_{r,1} \text{ for } 1 \le r \le n \quad (11)$$

$$CI\_SK_{1,r} = CI\_SK_{1,r-1} + SHA\_256^{r-1}(M) \quad \text{for } 2 \le r \le i \quad (12)$$

$$CI\_SK_{1,i,r,1} = CI\_SK_{1,i} + GV_{1,i,r,1}\,, \text{ for } 1 \le r \le k \quad (13)$$

$$CI\_SK_{n,j,1,r} = CI\_SK_{n,j,1,r-1} + SHA\_256^{r-1}\big(SM_{n,j}\big) \quad \text{for } 2 \le r \le p \quad (14)$$

$$CI\_PK = CI\_SK \times B \quad (15)$$

$$CI\_PK_{r,1} = CI\_PK + GV_{r,1} \times B\,, \text{ for } 1 \le r \le n \quad (16)$$

$$CI\_PK_{1,r} = CI\_PK_{1,r-1} + SHA\_256^{r-1}(M) \times B \quad \text{for } 2 \le r \le i \quad (17)$$

$$CI\_PK_{1,i,r,1} = CI\_PK_{1,i} + GV_{1,i,r,1} \times B \text{ for } 1 \le r \le k \quad (18)$$

$$CI\_PK_{n,j,1,r} = CI\_PK_{n,j,1,r-1} + SHA\_256^{r-1}\big(SM_{n,j}\big) \quad \text{for } 2 \le r \le p \quad (19)$$

In scripts and transactions related to different $subcontracts$ or $sub\_subcontracts$, the $master\_agent$ and the $contract\_issuer$ use different corresponding keys within their hierarchies. This increases security, as even if a transactions related to a $(sub\_)subcontract$ is compromised, the integrity of the rest of the contract structure is preserved. Furthermore, an element in the hierarchy of public keys can be produced in advance of the execution of the corresponding $(sub\_)subcontract$, as the relevant generator value is available and known to the $master\_agent$ and the $contract\_issuer$ before that execution. Notice that the generator values can be produced in a different way, and (1-19) present just one alternative. However, any version should allow the evaluation of the current generator value before the current element of the conditionality structure. Thus, the $master\_agent$ evaluates each of the public keys $CI\_PK_{r,1}$ to $CI\_PK_{n,j,1,r}$ of the $contract\_issuer$ at the same time at which the $contract\_issuer$ evaluates them. The vice-verse is also true, and the $contract\_issuer$ evaluates each of the public keys $MA\_PK_{r,1}$ to $MA\_PK_{n,j,1,r}$ of the $master\_agent$ at the same time at which the $master\_agent$ evaluates them. Also, the pairing private keys are produced by their owner at the same time, i.e. the earliest step, he can produce the corresponding public keys.

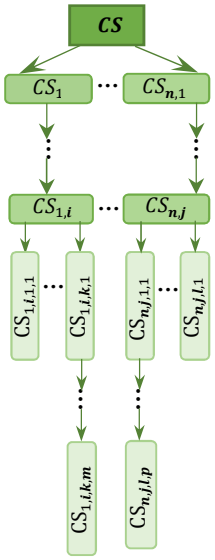

Figure 3. A hierarchical of common secrets.

As a result, the $contract\_issuer$ and the $master\_agent$ produce independently the same hierarchy of $common\_sercrets$ CS, as presented in Fig. 3. The $contract\_issuer$ produces the $common\_secrets$ as:

$$\begin{aligned} CS_{r,1} &= CI\_SK_{r,1} \times MA\_PK_{r,1} \\ &\vdots \\ CS_{n,j,1,r} &= CI\_SK_{n,j,1,r} \times MA\_PK_{n,j,1,r} \end{aligned} \quad (20)$$

and the $master\_agent$ produces the same $common\_secrets$ as:

$$\begin{aligned} CS_{r,1} &= MA\_SK_{r,1} \times CI\_PK_{r,1} \\ &\vdots \\ CS_{n,j,1,r} &= MA\_SK_{n,j,1,r} \times CI\_PK_{n,j,1,r} \end{aligned} \quad (21)$$

Now, each $common\_secret$ $CS_{r,1}$ to $CS_{n,j,1,r}$ serves as a basis for a symmetric encryption key securing a channel for communication between CI and MA regarding a corresponding $subcontract_{r,1}$ to $sub\_subcontract_{n,j,1,r}$. For example, the communication may confirm parameters or prioritize preferences.

B. *Efficient and Secure Transfer of Smart Contracts and Underlying Entities*

Further solution components provide methods and systems for efficient and secure transfer of entities on a blockchain [7][8]. These methods have wide effect on services but are particularly beneficial to smart-contract functionality. That increased functionality, in turn, helps sustain blockchain-enabled services of varying complexity. We focus in this Section III.B on the case when the entities being transferred are underlying smart contracts. A smart contract can also be an underlying of another smart contract. For example, the ownership of a tokenized financial instrument is transferred through a smart contract, and the structure of the financial instrument is implemented through another smart contract. The underlyings can include physical assets and IoT devices manipulated through the contracts, or virtual assets – such as rights on physical assets or rights on particular services – that are controlled through the contract. The control of smart IoT devices is a further example.

The transfer of entities underlying smart contracts is facilitated trough tokenization techniques. Enhanced optimization of memory usage in the electronic transfers, and improved security and data integrity are achieved through hashing techniques. Steps in the transfer involve:

- Generating a script S_k that comprises:
  - A set of metadata D_k associated with an invitation for the exchange of an entity E_k, where E_k is one of the underlyings of a smart $master\ contract$ or $subcontract$. The meta-data includes a pointer or other reference to the location of that contract.
  - The derived public key $CI_A\_PK_k$ associated with the $contract\_issuer_A$ and used in scripts, transactions and communication in relation to the exchange of entity E_k owned by A. In a hypothetical example, pension funds offer a variety of structured pension products and clients can hold a portfolio of different structured products from different funds. For each of the structured pension products a client holds, he may also select the proportion of the elements within the product. Clients of different funds are allowed to exchange (parts of) their holdings under certain conditions. The conditions differ among funds in level of detail and restrictive constraints, and so the exchange is not standardized. In the context of pensions, the actions are of relatively low frequency and based on long term perspective. When a client would like to exchange parts its holdings, he acts as a contract issuer. He has a different pair of keys derived from his master keys, where each pair is used for one of the structured products he holds. That pair is used for scripts/transactions related to the exchange of this product, and for communication about this product with the intelligent agent through a $common\_secret_{A,k}$.
  - The derived public key $MA_A\_PK_k$ of the $master\_agent_A$ managing the contract issued by A, where $MA_A\_PK_k$ is used only in relation to entity E_k.
- Hashing S_k and publishing S_k and its hash on a distributed hash table (DHT), which is distributed across a (worldwide) network and the script hash serves as a DHT look-up key.
  - This DHT resource differs from the DHT repository of contract discussed in Section II.
- Generating an invitation transaction TI_k for inclusion on the blockchain, where the transaction comprises the hash of S_k and an indication of an entity E'_k to be transferred in exchange for E_k.
- Scanning through the plurality of DHT entries, where each entry comprises:
  - an invitation to perform an exchange of an entity E_n underlying a smart contract; and
  - a link to an invitation transaction TI_n on the blockchain.
- (Partial) matching of the set of metadata D_k from the initial invitation-entry in the DHT repository of invitations, to a set of metadata D_m in another invitation-entry. Each set D_k and D_m comprises:
  - an indication of entities to be exchanged, E_k for E'_k and E_m for E'_m, correspondingly, where $E'_k \approx E_m$ and $E'_m \approx E_k$, and
  - conditions for the exchange that also (partially) match.
- Generating, broadcasting, and recording on the blockchain of an exchange transaction TE that includes:
  - The script S_k, signed with the derived private key $MA_A\_SK_k$ of the $master\_agent_A$, where $MA_A\_SK_k$ corresponds as cryptographic pairing to the public key $MA_A\_PK_k$. The script S_k may also be signed by the

$contract\_issuer_A$ using the private key $CI_A\_PK_k$.

- The script S_m of the (partially) matching DHT invitation-entry, signed with the private key $MA_B\_SK_m$ corresponding to public key $MA_B\_PK_m$, and signed with the private key $CI_B\_SK_m$ corresponding to $CI_B\_PK_m$. These keys are associated, respectively, with the $master\_agent_B$ and $contract\_issuer_B$.
- A first input provided from an output of invitation transaction TI_k .
- A second input provided from an output of invitation transaction TI_m .
- A first output indicating a quantity of the (tokenized) entity E_k to be transferred to the control of $smart\_$, and to the ownership of $contract\_issuer_A$.
- A second output indicating a quantity of the (tokenized) entity E_m to be transferred to the control of $smart\_contract_B$ and ownership of $ontract\_issuer_B$.

This method provides for data integrity and optimization of memory, and using DHT contributes to these qualities. Further, DHT invitation-entries, initiated at different stages of the conditionality structures of a variety of smart contracts, may be matched worldwide or within a scope indicated in/required by the smart contracts. The method enables disparate/distinct smart contracts (subcontracts) to identify/ match each other, and to securely exchange their underlying entities. This does not require alteration of the blockchain protocol, while embedding metadata in scripts associated with blockchain transactions.

IV. Conclusion

This paper introduces some of the interdependent components in a multifaceted solution for accelerating the functionality of blockchain-enabled services through distributed scalability and agent-based automation [1]. The focus here is particularly on components that enhance the functionality of blockchain-enforced smart contracts. These include the method/system for automated management of smart contracts with hierarchical conditionality structures, and the method/system for efficient and secure matching and transfer of contract underlyings among diverse smart contracts and subcontracts. Services enabled by implementing blockchain-enforced smart contracts are efficient, secure, automated, and allow (worldwide) resource distribution. They present a sustainable and efficient alternative to some of the current service infrastructures, particularly when some of them are underperforming or with unreliable security.

The methods introduced in this paper are not restricted to using SHA-256 and other hash algorithms from the Secure Hash Algorithm (SHA) family may be used, such as instances in the SHA-3 subset. Further hash algorithms may also include those in the RACE Integrity Primitives Evaluation Message Digest (RIPEMD) family, and in the families based on Zemor-Tillich hash function or on knapsack-based hash functions. The introduced methods are further not restricted to Bitcoin, and can be implemented in increasing the functionality of any blockchain-enforsed smart contracts.

We have emphasized the synergistic effect of implementing the interdependent components of the multifaceted solution we are developing towards increasing the complexity and versatility of blockchain-enabled services and accelerating the functionality of blockchain-enforced smart contracts. Some of these components are introduced in this paper. Our research focus is next on big-data analysis of the potential effects on the Bitcoin network performance due to: (i) adopting each of the innovative components of the solution, (ii) their rate of adoption, and (iii) the sequence in which they are adopted.

Acknowledgment

The authors are grateful for the creative and supportive environment at nChain Limited, within its scientific and applied research teams.

References

[1] C. Wright and A. Serguieva, "Sustainable blockchain-enabled services: Distributed scalability and agent-based automation," in Applications of Data-Centric Science to Social Design, A.-H. Sato, Ed., Agent-Based Social Systems Series, H. Deguchi, Ed.-in-chief, Springer, 2018, forthcoming.

[2] D. Magazzeni, P. McBurney and W. Nash, "Validation and Verification of Smart Contracts: A Research Agenda," Computer, vol. 50(9), pp. 50-57, 2017.

[3] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," IEEE Access, vol. 4, pp 2292-2303, 2016.

[4] S. Olnes, J. Ubacht and M. Janssen, "Blockchain in government: Benefits and implications of distributed ledger technology for information sharing," Government Information Quarterly, vol. 34(3), pp355-364, 2017.

[5] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050865," filed Feb. 23, 2016.

[6] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050856," filed Feb. 23, 2016.

[7] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050859," filed Feb. 23, 2016.

[8] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050861," filed Feb. 23, 2016.

[9] G. Urdaneta, G. Pierre, and M. van Steen, "A survey of DHT security techniques," ACM Computing Surveys, vol. 43(2), pp. 1-49, Jan. 2011. Available from: doi:10.1145/1883612.1883615 [Accessed on: Oct. 20, 2017]

[10] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050818," filed Feb. 23, 2016.

[11] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050819," filed Feb. 23, 2016.

[12] nChain Holdings, "International Patent Application No. PCT/IB2017/ 050815," filed Feb. 23, 2016.

[13] BitcoinWiki, "Range of valid ECDSA private keys,", BitcoinWiki Category: Private Key, Available from: http://en.bitcoin.it/ wiki/Private_key [Accessed on: Oct. 1, 2017].

[14] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," Bitcoin Project: Resources and Developer Documentation, pp. 1–9, 2008. Available from http://bitcoin.org/bitcoin.pdf [Accessed on: May 22, 2017].

[15] Nakamoto Institute, Code: the first three available Bitcoin codebases written by Satoshi Nakamoto, 2009. Available from: http://satoshi.nakamotoinstitute.org/code [Accessed on: Jun. 27, 2017].