ISSN 0005-1055, Automatic Documentation and Mathematical Linguistics, 2018, Vol. 52, No. 2, pp. 108–112. © Allerton Press, Inc., 2018.
Original Russian Text © A.Yu. Shcherbakov, 2018, published in Nauchno-Tekhnicheskaya Informatsiya, Seriya 2: Informatsionnye Protsessy i Sistemy, 2018, No. 4, pp. 00000–00000.

Development of Means for the Formation of a Corporate Distributed Register (Blockchain)

A. Yu. Shcherbakov[a]

[a] *Federal Research Center Computer Science and Control, Russian Academy of Sciences, Moscow, 119333 Russia*
e-mail: x509@ras.ru

Received February 19, 2018

Abstract—The problem of developing methods for forming the atoms of a corporate distributed register (blockchain), which provides secure distributed storage of data in an associated chain, is discussed; an example of the application of the proposed methods for the implementation of a conditional tax payment through a smart contract is considered.

Keywords: blockchain, distributed data storage, information security, e-voting, encryption, integrity control, smart contract, development tools for blockchain

DOI: 10.3103/S0005105518020085

INTRODUCTION

For system integrity, a blockchain must consist of separate elements, links, each of which in turn is divided into elementary components (in [1] these are called blockchain atoms). Atoms as elements that are associated or embedded in a computer system can be passive and active.

BLOCKCHAIN STRUCTURE

Blockchain atoms are characterized by an identifier (name) and have the properties that are required both to form links and an entire chain and to implement the properties of the computer (information) system that uses the blockchain, for example, the properties of privacy of user data placed in the distributed register. It should be noted that the currently existing blockchain systems do not provide the property of information confidentiality.

Let us identify the following types of blockchain atoms.

1. A boundary atom (the beginning or the end) of the blockchain: contains references to the end of the previous link or the beginning of the next one.

2. A structure atom defines and describes the vector of identifiers of atoms that exist in the link and logically follows the boundary atom (the beginning).

3. A data atom contains description of the data interpreted as a passive component of the link. A variety of data atoms are:

• An open data atom (OD atom) is characterized by length and contains data not subject to any processing;

• An integral data atom (ID atom) is characterized by the length of the data of the fixed integrity and an identifier or a reference to the data integrity control procedure;

• A signed data atom (SD atom) is characterized by the length of the signed data: an identifier or a reference to the signature verification procedure for data, as well as a reference to the signature verification key. The signature verification procedure can be specified externally with respect to the blockchain or be a subject atom;

• An encrypted (restricted) data atom (RD atom) is characterized by the length of the encrypted data: an identifier or a reference to the procedure of data encryption/decryption, as well as an encryption or decryption key;

• A signature atom contains the signature of an atom or several atoms: “signed data” with a given identifier (identifiers);

• A hash atom contains the integrity control standard for an atom or several “integral data” atoms with a given identifier (identifiers).

4. A subject atom contains description of the data interpreted as an active component of the link.

A subject atom can be:

• a scenario atom that contains the interpreted code for performing operations on data atoms;

• an executor atom that contains the compiled code for the actual processor, a hypervisor of the computer system, or a machine atom in which the blockchain is processed;

• a machine atom that contains the environment for the execution of the scenario atom or executor atom.

Next, let us consider the elementary modules that are necessary for the formation of the corresponding atoms. It is important to note that in this case we are talking about the corporate distributed register, since in general-purpose blockchains the atoms are formed in a different way. However, the proposed technology makes it possible to protect data in general-purpose blockchains as well, where the information is preformed with the modules considered here and then is placed in the blockchain.

ELEMENTARY MODULES FOR THE FORMATION OF CORPORATE BLOCKCHAIN ATOMS

A module for primary formation of an initial random number and a personal identifier for the user

InitUser

Format of use:

InitUser FileUserID UserPIN <RandomString>,

where:

FileUserID is the name of the file with the password-restricted personal identifier of the user (may be associated with the user's name),

UserPIN is the password (PIN code or method of its input, for example, reading from a USB-token) for restricting the personal identifier of the user,

<*RandomString*> is an optional parameter for improving the operation of the random-number generator ("accelerating line").

This module creates a random.bin file for further use of the random-number generator and a file with the password-restricted personal identifier of the user (in fact, a protected container for storing and transferring the user's personal identifier).

This module returns typed errors that are needed to integrate module calls into a smart contract.

Returned errors:

- 1, an error during testing of protection modules,
- 2, a call-format error,
- 3, the identifier file already exists,
- 4, a random-number generation error,
- 5, an random-number updating error,
- 6, a user-file writing error,
- 7, a user-file control read error.

Modules of formation of blockchain atoms

GenAtomX

This module generates a closed data atom from open data using the user's personal identifier, an X-type atom.

Format of use:

GenAtomX FileUserID UserPIN file_or_string AtomFile,

where:

FileUserID is the name of the file with the password-restricted personal identifier of the user,

UserPIN is the password for restricting the user's personal identifier,

file_or_string is the information to be placed in the RD (restricted data) atom, restricted in the user's personal identifier,

AtomFile is the file into which the atom is written.

This module employs a file with the password-restricted personal identifier of the user, opens it and builds a blockchain atom according to the protocol.

Returned errors:

- 1, an error during testing of protection modules,
- 2, a call-format error,
- 3, the user's identifier file does not exist,
- 4, a random-number generation error,
- 5, the atom file already exists,
- 6, an incorrect PIN code,
- 7, an atom-writing error,
- 8, an atom control reading error,
- 9, a random-number updating error.

GenAtomY

This module generates a restricted data atom from open data using random data to achieve a specified time (PoW), a Y-type atom.

Format of use:

GenAtomY file_or_string AtomFile Power,

where:

A *file_or_string* is the information to be placed in the RD (restricted data) atom, restricted in a random number of a given length,

AtomFile is the file into which the atom is written,

Power is a two-digit number that provides PoW (18–22 is recommended for testing).

Returned errors:

- 1, an error during testing of protection modules,
- 2, a call-format error,
- 4, a random-number generation error,
- 5, a random-number updating error,
- 6, the atom file already exists,
- 7, an atom-writing error,
- 9, a random-number updating error.

GenAtomH

This module generates a hash atom (integral data atom) from open data using the user's personal identifier, an H-type atom.

Format of use:

GenAtomX FileUserID UserPIN file_or_string AtomFile, where:

FileUserID is the name of the file with the password-restricted personal identifier of the user,

UserPIN is the password for restricting the user's personal identifier,

file_or_string is the information to be placed in the hash atom, a function from open data, calculated with the help of the user's personal identifier,

AtomFile is the file into which the atom is written.

Returned errors:

• 1, an error during testing of protection modules,
• 2, a call-format error,
• 3, the user's identifier file does not exist,
• 4, a random-number generation error,
• 5, the atom file already exists,
• 6, an incorrect PIN code,
• 7, an atom-writing error,
• 8, an atom control reading error,
• 9, a random-number updating error.

Modules for extracting data from blockchain links

ExcX

A module for extracting data from an X-type atom.

Format of use:

ExcX FileUserID UserPIN AtomFile Result,

where:

FileUserID is the name of the file with the password-restricted personal identifier of the user,

UserPIN is the password for restricting the user's personal identifier,

AtomFile is the file into which the RD atom is placed,

Result is the file with recovered data.

This module has the purpose of testing of their actions by a user and audit of transactions by a user.

Returned errors:

• 1, an error during testing of protection modules,
• 2, a call-format error,
• 3, the user's identifier file does not exist,
• 4, a random-number generation error,
• 5, the atom file does not exist,
• 6, an incorrect PIN code,
• 7, an atom-writing error,
• 8, an atom control reading error.

ExcY

A module for extracting data from a Y-type atom.

Format of use:

ExcY AtomFile Result,

where:

AtomFile is the file into which the RD atom is placed,

Result is the file with recovered data.

This module has the function of allocating restricted data with a specified labour intensity PoW.

Returned errors:

– 1, an error during testing of protection modules,
– 2, a call-format error,
– 3, the user's identifier file does not exist,
– 4, a random-number generation error,
– 5, the atom file does not exist,
– 7, an atom-writing error.

CAtomH

This module checks the integrity of data using a precomputed hash atom via the user's personal identifier.

Format of use:

GenAtomX FileUserID UserPIN file_or_string AtomFile,

where:

FileUserID is the name of the file with the password-restricted personal identifier of the user,

UserPIN is the password for restricting the user's personal identifier,

file_or_string is the information for integrity control,

AtomFile is the file into which the atom was written that contains the control information from the data.

Returned errors:

• 1, an error during testing of protection modules,
• 2, a call-format error,
• 3, the user's identifier file does not exist,
• 4, a random-number generation error,
• 5, the atom file already exists,
• 6, an incorrect PIN code,
• 7, an atom-writing error,
• 8, an atom control reading error,
• 9, a random-number updating error.

The Power calculation module

Cpower

Format of use:

Cpower Time

This module returns the number of bits required to provide the labor intensity of *Time* minutes.

All modules generate text logs with the name of the module.

THE CONCEPT OF SMART CONTRACTS

The first ideas about smart contracts were proposed in 1996 by Nick Szabo [2]. Practical implementations became possible due to the emergence of blockchain technology in 2008. Some principles of smart contracts were formed in the protocol of the first blockchain currency (crypto currency), the Bitcoin; however, they were not implemented in the client software, did not have Turing completeness due to security reasons, and were not used in practice. With the development of the blockchain technology, there have been ideas that various higher-level protocols can be created over the Bitcoin protocol, including full smart contracts, similar to how many application-level protocols exist over TCP/IP.

Smart contracts were first applied in practice in the Ethereum project. The idea to create the project occurred in 2013. At that time, V. Buterin, the founder of Bitcoin Magazine, came to the conclusion that the Bitcoin is poorly suited as a basic protocol since it was not originally designed for this task; in one of his articles he wrote about the idea of creating such a protocol from scratch.

We will assume that a smart contract is an executable code with fixed integrity that is placed in an executor atom and operates with atoms and blockchain links. At the same time, the result of the work of a smart contract is always placed in a new atom or a blockchain link. Otherwise, the ideology of the invariability of the blockchain links will be disturbed.

AN EXAMPLE OF THE FORMATION AND OPERATION OF A SMART-CONTRACT CONSTRUCTED FROM ELEMENTARY MODULES

We assume that the user *nlp100* (taxpayer-100) has a wallet with cryptocurrency (for paying taxes) *cnal100*, which has 500 cryptorubles.

For test creation of a wallet, we use a smart contract:

genatomx nlp100_1 privet1 500 cnal100_u

genatomx nlp100_3 privet3 500 cnal100_b.

In a real payment system, creation of a wallet will require carrying out actions coordinated with the authorized bank or cryptocurrency operator to identify the user, create a wallet, and credit the cryptocurrency.

The user *fns* (tax service) made a tax payment invoice for *nlp100* of 50 cryptorubles to the bank *bank*, which was issued as a smart contract *fns*100_b.*bat*.

Initially the user creates their private key containers for working with the wallet nlp100_1 to receive smart contracts and notices from the tax service nlp100_2 and to exchange with the bank nlp100_3 using the *inituser* module.

A smart contract is a sequential call of methods for creating blockchain atoms *genatomx*, which deduct 50 cryptorubles from taxpayer-100's wallet and credit it to the bank account, forming the atoms *a1* and *a2*:

genatomx nlp100_1% 1 -50 a1

genatomx nlp100_3% 2 +50 a2.

It should be noted that a password or private key is required to work with the wallet of the taxpayer, which is indicated by the argument %1 (the password or the key is entered by the user), respectively, the second password or private key is needed for crediting funds to the bank.

The tax service creates a container for exchange with taxpayer-100, *fns100*, the bank opens a container for exchange with taxpayer-100, *bank100*, and for sending receipts for the tax service the bank creates a container *bfns*.

To create containers, the following smart contract is used:

inituser nlp100_1 privet1

inituser nlp100_2 privet2

inituser nlp100_3 privet3

inituser fns100 privet2

inituser bank100 privet3

inituser bfns privet4.

Passwords or methods for entering a private key are used by the user to access their wallet: *privet*1, a private key unknown to anyone other than the user; *privet*2, for exchange with the tax service; *privet*3, for exchange between the user and the bank; and *privet*4, for exchange between the bank and the tax service.

The tax service forms the atom a0, which contains the encrypted text of the smart contract *fns100_b.bat*:

genatomx nlp100_2 *privet*2 fns*100_b*.bat a0.

The taxpayer extracts the smart contract from the atom:

excx nlp100_2 *privet*2 a0 *user*100.*bat*

and executes the contract *user*100.*bat* by entering passwords or private keys instead of %1 and %2 to access their wallet and make a transaction to the bank.

The contract creates blockchain links a1 and a2, which contain an altered state of the taxpayer's wallet (a decrease of 50 cryptorubles) and an order to transfer the required amount to the wallet of the bank.

The bank extracts from the atom the taxpayer's order for the account

excx nlp100_3 privet3 a2 b1.

The bank credits money to its wallet (crypto account) and sends a confirmation to the taxpayer by forming the next atom

genatomx nlp100_3 privet3 zachisleno=50 a3.

On the side of the taxpayer, the amount credited to the bank is automatically deducted from the wallet:

genatomx nlp100_1 privet1 450 cnal100_u

genatomx nlp100_3 privet3 450 cnal100_b.

A new copy of the wallet is created with the value 450 = 500 – 50 according to the notification of the bank.

The bank has the opportunity to verify in *cnal*100_b that the amount in the wallet has changed:

excx nlp100_3 privet3 cnal100_b cnal100_t2.

On the side of the client (taxpayer), transactions with the wallet are carried out only provided the balance of funds occurs, that is, where the wallets of the client and bank contain the same amounts.

The balance is checked using the following smart contract:

```
excx nlp100_1 privet1 cnal100_u cnal100_t1
excx nlp100_3 privet3 cnal100_b cnal100_t2.
```

The last operation is the formation of the confirmation atom a4:

```
genatomx bfns privet4 prinjato=nlp100 = 50 a4.
```

The tax service extracts the atom a4

```
excx bfns privet4 a4 b3
```

and receives a confirmation line of tax payment.

For real use, one must also enter a confirmation of the bank payment to the treasury.

To demonstrate the voting system, the following smart contract can be executed:

```
genatomx nlp100_1 privet1 Ivan_Ivanov g1
genatomx nlp100_2 privet2 Petr_Petrov g2
genatomy Ivan_Ivanov g3 19
genatomy Petr_Petrov g4 20
excx nlp100_1 privet1 g1 golos11
excx nlp100_2 privet2 g2 golos21
excy g3 golos12
excy g4 golos22.
```

For convenience, containers that were already created are used.

First, votes for *Ivan_Ivanov* and *Petr_Petrov* are formed, which are formed and available only to users who have voted and then are available to everyone, but with disclosure at the specified time, where for *Petr_Petrov* the disclosure time is on average twice as long. The votes are then disclosed into the corresponding files and compared.

The log of the *excy* function shows an approximate difference in the vote extraction times: approximately 12 s and 31 s for the same percentage of searches (about 30):

```
00:39:01 05.02.2018:Start excy
00:39:01 05.02.2018:Success Protect Function
00:39:01 05.02.2018:Ok Test Random
00:39:13 05.02.2018:RealCicle = 157705.000000
[Full cycles = 524288.000000] 30.079842 percent of full cycles
Full time = 11.875000 sec
00:39:13 05.02.2018:Ok AtomFileY
00:39:13 05.02.2018:Start excy
00:39:13 05.02.2018:Success Protect Function
00:39:13 05.02.2018:Ok Test Random
00:39:44 05.02.2018:RealCicle = 410876.000000
[Full cycles = 048576.000000] 39.184189 percent of full cycles
Full time = 30.861000 sec
00:39:44 05.02.2018:Ok AtomFileY.
```

CONCLUSIONS

The proposed technology makes it possible to form almost the entire necessary range of blockchain atoms and perform operations with them using the technology of smart contracts. Cryptographic operations with atoms should be implemented on the basis of domestic crypto algorithms to ensure the warrant properties and the possibility of subsequent certification and validation of the developed solutions by state regulators.

REFERENCES

1. Biktimirov, M.R., Domashev, A.V., Cherkashin, P.A., and Shcherbakov, A.Yu., Blockchain technology: Universal structure and requirements, *Autom. Doc. Math. Linguist.,* 2017, vol. 51, no. 6, pp. 235–238.
2. Szabo, N., Clever contracts (Fourth Value Revolution), *Komp'yuterra,* 1998, no. 38, pp. 12–19.

Translated by K. Lazarev

SPELL: OK