Technical Paper

(Some details might be added)

Decentralized financial system

CREDITS

Version 1.6/25.10.2017

Contents

Peer-to-peer network

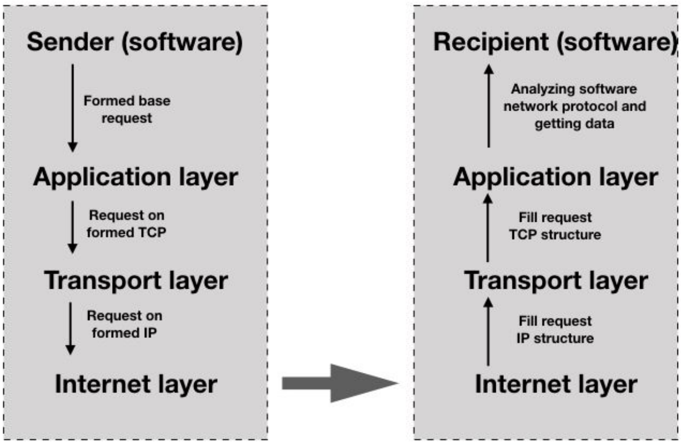
A peer-to-peer network is a peer-to-peer computer network that works over the Internet and uses the TCP/IP protocol as the primary transport protocol for data and commands within the network. The TCP/IP protocol stack includes four levels:

- Application layer
- Transport layer
- Internet layer
- Link layer

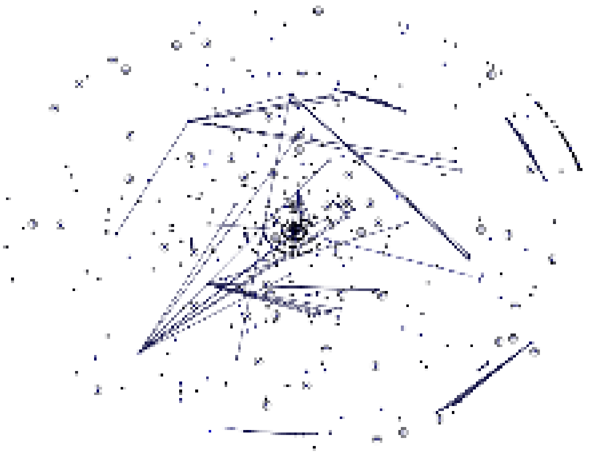
Protocols of these levels fully implement functionality of the OSI model. All user interaction in IP networks are built on the TCP/IP protocol stack. The stack is independent of the physical data transfer environment, which in particular, ensures completely transparent communication between wired and wireless networks.

To operate the peer-to-peer network of the Credits platform, the following are used:

1. Internet layer: Originally designed to transfer data from one network to another. The IP network packets contain the code indicating the protocol of the next layer to be used to extract data from the packet. This value is the unique IP number of the protocol.



2. Transport layer: Transport layer protocols can solve the problem of non-guaranteed delivery of messages ("did the message reach the addressee?"), and also guarantee the correct sequence of data arrival. In the TCP/IP stack, transport protocols determine which application this data is intended for. TCP (IP identifier 6) is a "guaranteed" transport mechanism with a pre-established connection, which provides the application with a reliable data stream that gives confidence in the accuracy of the received data, re-requests data in case of loss, and eliminates duplication of data. TCP adjusts the network load, as well as reduces the wait time for data transmission over long distances. Moreover, TCP ensures that the received data has been sent exactly in the same sequence.
3. Application layer: This layer provides data exchange between the network software.



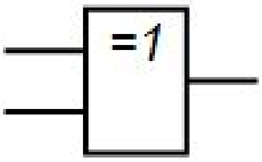
The concept of DHT (*distributed **h**ash **t**able*) is used for the constant and stable connection between nodes. DHT is a class of decentralized distributed search engine systems that operates like a hash table. As a data structure, the hash table can be an associative array containing pairs (key-value). Also, the term DHT refers to a number of principles and algorithms that allow writing data by distributing information among a certain set of storage nodes, and restoring them through distributed search by a key. A feature of the distributed table is the possibility to distribute information among some set of storage nodes in such a way that each participating node could find the value associated with this key. DHT is characterized by the following properties:

- Decentralization: the form of the system of collective nodes without coordination;
- Scalability: the system will work equally well for thousands or millions of nodes;
- Fault tolerance: the system will be equally reliable (in some sense) with nodes constantly switching, tripping and issuing errors.

The key technique for achieving the goal is that any node must coordinate the work with only a few nodes in the system – usually O (log*n*), where *n* is the number of participants (see below) – so that only a limited amount of work is done for each change in the number of participants.

The work protocol of the Credits platform peer-to-peer network determines the network structure that regulates communication between the nodes and the way information exchanges within it. Network nodes communicate with each other over the TCP transport layer protocol. Credits nodes store data via distributed hash tables (DHT). As a result, over the existing Internet network built on the basis of the IP protocol, a new virtual network is created, where each node is designated by a special number ("Node ID", represented as a hash value). A node that wants to join the network must undergo a bootstrap process. At this point, the node must know the address of the other node (received from the user or taken from the list) that is already included in the virtual network. If the connected node was not yet included in this network, then a random ID value is calculated, which does not belong to any node. ID is used until exit from the network. The network operation algorithm is based on calculating the "distance" between nodes by applying an exclusive OR operation to ID of these nodes.

Exclusive OR (**modulo 2 addition**) is a Boolean function, as well as a logical and bit operation. In case of two variables, the result of the operation is true, if and only if one of the arguments is true and the second is false. For a function of three or more variables, the result of the operation will be true only if the number of arguments equal to 1, constituting the current set is odd.



This "distance" has nothing to do with the geographical position. For example, nodes from Germany and Australia can be "neighboring" in a virtual network

The information in DHT is stored in the so-called "values". Each "value" is linked to a "key".

When searching for the value corresponding to the key, the algorithm examines the network in several steps. Each step brings the target node closer until full finding of the "value" or to the absence of such nodes. The number of contacted nodes depends logarithmically on the network size: if the number of participants increases twice, the number of requests will increase by just one.

Ledger

The ledger is a system for storing data on transactions (actions) conducted in the system made in the account (value) calculation system. The ledger is presented in the form of a dictionary Key=Value, where Key is a unique value assigned to the entry when registering in the system; Value is the transaction content to perform the action in the system.

A transaction pool formed from the approved transactions at the node voting stage is added to the ledger. A transaction pool is a list of approved transactions presented in a dynamic dictionary format.

The local ledger is stored on a local disk in a compressed (archived) format, made by a lossless compression algorithm called Deflate. Deflate is a lossless compression algorithm that uses a combination of LZ77 and Huffman algorithms.

Deflate-flow contains a series of blocks. There is a three-bit header before each block:

- One bit: the flag of the last block.
- 1: the last block.
- 0: not the last block.
- Two bits: the method of data encoding.
- 00: The data is not encoded (the output data is directly in the block).
- 01: The data is encoded using the static Huffman method.
  - 10: The data is encoded using the dynamic Huffman method.
  - 11: Reserved value (error).

Most of the blocks are encoded using method 10 (dynamic Huffman) which is an optimized Huffman code tree for each new block. The instructions for creating the Huffman code tree follow immediately after the block header.

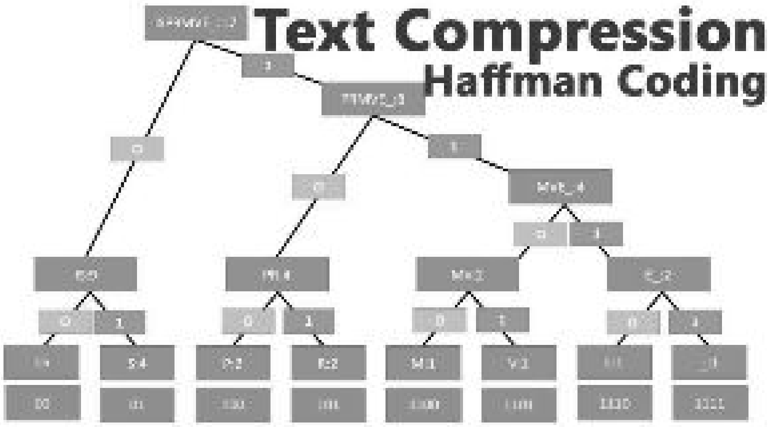
Compression is performed in two stages:

- Replacing duplicate lines with pointers (LZ77 algorithm);
- Replacing of characters with new characters based on the frequency of their use (Huffman algorithm).

Principle of LZ77 operation. The main idea of the algorithm is to replace the repeated entry of a line with a reference to one of the previous entry positions. The sliding window method is used to this end. A sliding window can be represented in the form of a dynamic data structure organized in such a way as to remember the previously “said” information and provide access to it. Thus, the process of compressing encoding according to LZ77 is similar to writing a program which commands allow access to the elements of a sliding window, and instead of values of a compressible sequence inserting references to these values of the sliding window. In the standard LZ77 algorithm, line matches are encoded with the pair:

- match length
- offset or distance

The encoded pair is treated exactly as a command for copying characters from the sliding window from a specific position, or literally as: "Return in the dictionary to the character offset value and copy the value of the character length starting from the current position". The feature of this compression algorithm is that the use of the encoded length-offset pair is not just acceptable, but also effective in cases where the length value exceeds the offset value. The example with the copy command is not quite obvious: "Return 1 character back in the buffer and copy 7 characters starting from the current position". How can we copy 7 characters from the buffer when at the moment there is only 1 character in the buffer? However, the following interpretation of the encoding pair can clarify the situation: every 7 subsequent symbols coincide (are equivalent) with 1 character before them. This means that each character can be uniquely identified by moving backwards in the buffer, even if the given character is still missing in the buffer at the time of decoding the current length-offset pair.



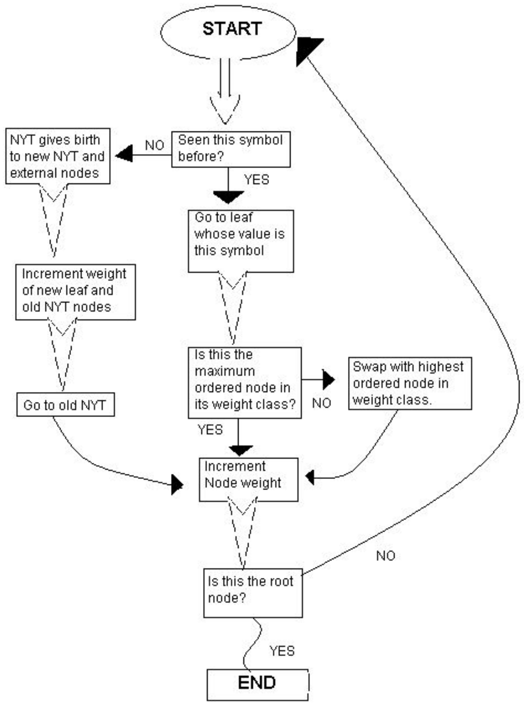
Huffman algorithm is a greedy algorithm for optimal prefix encoding of the alphabet with minimal redundancy. This encoding method consists of two main stages:

1. Building the optimal code tree.
2. Building the code-symbol mapping based on the built tree.

The classical Huffman algorithm at the input receives a table of frequencies of character occurrence in the message. Further on the basis of this table, a Huffman coding tree (H-tree) is built.

1. The characters of the input alphabet form a list of free nodes. Each leaf has a weight that can be equal to either the probability or the number of occurrences of the character in the compressible message.
2. Two free nodes of the tree with the smallest weights are selected.
3. Their parent is created with a weight equal to their total weight.
4. The parent is added to the list of free nodes, and its two descendants are removed from this list.
5. One arc leaving the parent is assigned bit 1, the other – bit 0. The bit values of the branches originating from the root do not depend on the weights of the descendants.
6. Steps, starting from the second one, are repeated until there is only one free node left in the list of free nodes. It will be considered the root of the tree.

This process can be represented as the construction of a tree, which root is a character with the sum of probabilities of the combined characters obtained by combining the characters from the last step, its n_0 descendants are the characters from the previous step, and so on.


START
Seen this symbol before?
NO
NYT gives birth to new NYT and external nodes
YES
Go to leaf whose value is this symbol
Increment weight of new leaf and old NYT nodes
Is this the maximum ordered node in its weight class?
NO
Swap with highest ordered node in weight class.
Go to old NYT
YES
Increment Node weight
NO
Is this the root node?
YES
END

To determine the code for each of the characters included in the message, we must go from the tree leaf corresponding to the current character to its root accumulating bits when moving along the branches of the tree (the first branch in the path corresponds to the lowest bit). The bit sequence obtained in this way is the code of this character written in the reverse order.

Since none of the obtained codes is a prefix of the other, they can be uniquely decoded when they are read from the stream. In addition, the most frequent message character is encoded with the smallest number of bits, and the rarest character – with the largest number.

Adaptive compression disallows transmitting the message model along with it and confines itself to one pass along the message, both during encoding and decoding.

In developing the adaptive Huffman coding algorithm, the greatest difficulties arise when developing the procedure for updating the model with the next character. Theoretically, one could simply insert the full construction of the Huffman coding tree inside this procedure, however, such a compression algorithm would have an unacceptably low speed, since building an H-tree is labor-intensive, and it is unreasonable to do this when processing each character. Fortunately, there is a way to modify an already existing H-tree so as to reflect processing of the new character. The most popular algorithms for reconstruction are the Faller-Gallagher-Knuth algorithm (FGK) and the Witter algorithm.

All algorithms for rebuilding a tree when reading the next character include two operations:

- The first is to increase the weight of tree nodes. First, increase the weight of the leaf corresponding to the read character by one. Then increase the weight of the parent to bring it into line with the new weight values of the descendants. This process continues until we reach the root of the tree. The average number of weight increase operations is equal to the average number of bits required to encode a character.
- The second operation – rearrangement of tree nodes – is required when increasing the weight of the node leads to a disruption of the ordering property, that is, when the increased weight of the node becomes larger than the weight of the next node. If we continue to process the weight increase, moving to the tree root, then the tree will cease to be a Huffman tree.

To preserve the ordering of the coding tree, the algorithm works as follows: Let the new increased weight of the node be W+1. Then we start moving along the list in the direction of increasing the weight until we find the last node with weight W. Permute the current and found nodes among themselves in the list, thus restoring the order in the tree (the parents of each of the nodes will also change). This completes the permutation operation.

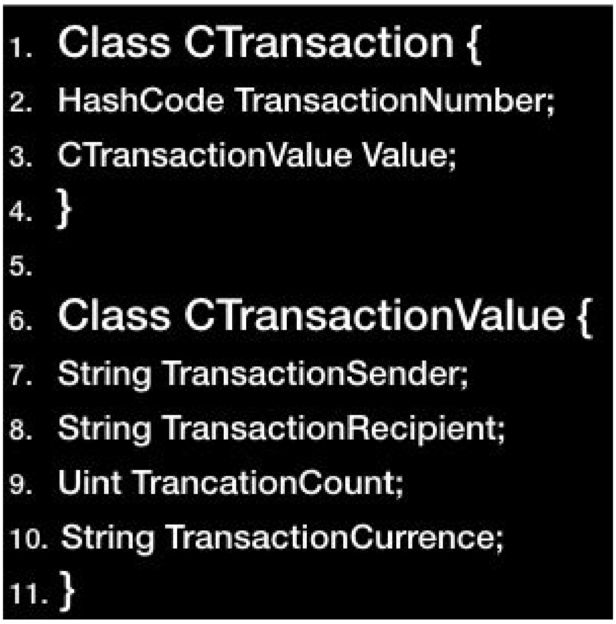
After the permutation, the operation of increasing the weight of the nodes continues further. The next node, which weight will be increased by the algorithm, is the new parent of the node which weight increase caused permutation.

The algorithm for updating the Huffman tree should be changed as follows: when increasing the weight, check it to achieve an acceptable maximum. If we have reached the maximum, then we need to "scale" the weight, usually by dividing the weight of the leaves by an integer number, for example 2, then recalculating the weight of all other nodes.

However, when dividing the weight in half, there is a problem associated with the fact that after performing this operation, the tree can change its form. This is explained by the fact that when dividing integers, the fractional part is discarded.

A properly organized Huffman tree after scaling may have a form that is significantly different from the original one. This is because scaling leads to a loss of the statistics accuracy. But with collection of new statistics, the consequences of these "errors" practically come to naught.

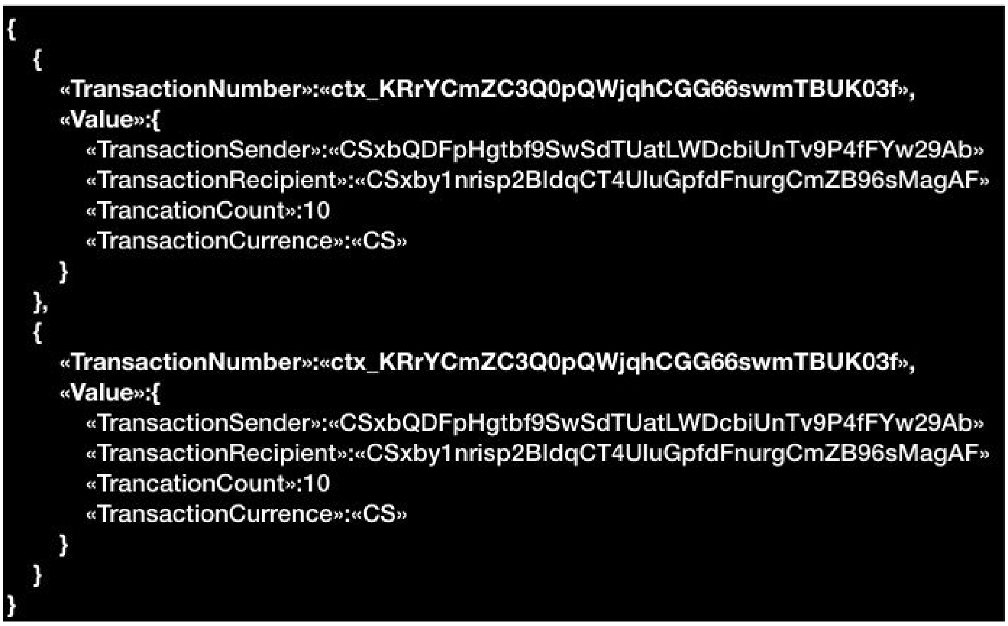
Data is synchronized by exchanging a transaction pool, which is sent by the main node, after the round of the federated voting on compilation of the white list of transactions.
The data format for synchronization is JSON.
Below is a data model with the necessary fields for implementing the process of analytics and voting in the network, required to announce a transaction.



```
Class CTransaction {
HashCode TransactionNumber;
CTransactionValue Value;
}

Class CTransactionValue {
String TransactionSender;
String TransactionRecipient;
Uint TrancationCount;
String TransactionCurrence;
}
```

The standardized JSON format (http://json-rpc.org) is used for data synchronization:

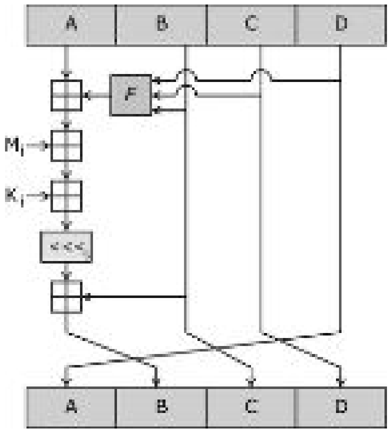


```
{
  {
    «TransactionNumber»:«ctx_KRrYCmZC3Q0pQWjqhCGG66swmTBUK03f»,
    «Value»:{
      «TransactionSender»:«CSxbQDFpHgtbf9SwSdTUatLWDcbiUnTv9P4fFYw29Ab»
      «TransactionRecipient»:«CSxby1nrisp2BldqCT4UluGpfdFnurgCmZB96sMagAF»
      «TrancationCount»:10
      «TransactionCurrence»:«CS»
    }
  },
  {
    «TransactionNumber»:«ctx_KRrYCmZC3Q0pQWjqhCGG66swmTBUK03f»,
    «Value»:{
      «TransactionSender»:«CSxbQDFpHgtbf9SwSdTUatLWDcbiUnTv9P4fFYw29Ab»
      «TransactionRecipient»:«CSxby1nrisp2BldqCT4UluGpfdFnurgCmZB96sMagAF»
      «TrancationCount»:10
      «TransactionCurrence»:«CS»
    }
  }
}
```

Nodes

A network node is a computer that has a full network client and a ledger storage connected to a common system participating in voting rounds for electing the processing node and trusted nodes, confirming/rejecting transactions and storing them in the ledger. The system subdivides all computers (nodes) into three categories:

- Main node is a node that provides analysis and confirmation of transactions (white list) adding transactions to the ledger.
- Trusted node is a node that provides transaction analysis and compilation of the primary white list.



- Normal node is a node participating in selection of the main and trusted nodes.
- Each node in the system is represented by a unique code that is the result of computing the MD5 hash function.

MD5 is a 128-bit hash algorithm developed by Professor Ronald L. Rivest. It is intended for the creation of "prints" or digests of the message of any length and the subsequent check of their authenticity. The input of the algorithm receives an input data stream, which hash must be found. The message length can be any (including zero). Write the length of the message in L. This number is an integer and non-negative. Multiplicity to any numbers is optional. After the data is received, the stream is then prepared for calculations.

Below are the 5 steps of the algorithm:

Step 1: Stream alignment

First, a single bit is added to the stream end (byte 80h), then the required number of zero bits. The input data is aligned so that their new size L′ is comparable to 448 modulo 512, (L′=512×N+448). Alignment occurs even if the length is already comparable to 448.

Step 2: Adding message length

At the end of the message, a 64-bit representation of the data length (the number of bits in the message) is appended until alignment. First, write the lower 4 bytes, then the higher ones. If the length exceeds 2^{64}-1, then only the lower bits are added (equivalent to taking modulo 2^{64}). After that, the stream length will be a multiple of 512. The calculations will be based on the representation of this data stream in the form of a 512-bit word array.

Step 3: Buffer initialization

For calculations, 4 variables with a size of 32 bits are initialized, and the initial values are given in hexadecimal numbers (the order of bytes is little-endian, the low-order byte first):

A = 01 23 45 67; // 67452301h

B = 89 AB CD EF; // EFCDAB89h

C = FE DC BA 98; // 98BADCFEh

D = 76 54 32 10. // 10325476h

The results of intermediate calculations will be stored in these variables. The initial state ABCD is called the initializing vector.

Also define functions and constants that we need for computations.

- We will need 4 functions for four rounds. Introduce functions of three parameters – words; the result will also be a word:

Round 1: FunF(X,Y,Z)=(X∧Y)∨(¬X∧Z),

Round 2: FunG(X,Y,Z)=(X∧Z)∨(¬Z∧Y),

Round 3: FunH(X,Y,Z)=X⊕Y⊕Z,

Round 4: FunI(X,Y,Z)=Y⊕(¬Z∨X),

where ⊕,∧,∨,¬ are bitwise logical operations XOR, AND, OR and NOT, respectively.

- Define the table of constants T[1...64] – a 64-element data table, constructed as follows: T[n]=int(232 · |sinn|).
- Each 512-bit block goes through 4 stages of computation in 16 rounds. For this, the block is represented as array *X* of 16 words of 32 bits. All rounds are of the same type and have the form: [abcd k s i], defined as a=b+((a+Fun(b,c,d)+X[k]+T[i])⋘s), where *k* is the number of 32-bit word from the current 512-bit message block, and …⋘s is a cyclic left shift by *s* bit of the obtained 32-bit argument. The number *s* is set separately for each round.

Step 4: Computation in a cycle

We put element *n* from the array of 512-bit blocks into the data block. Values A, B, C, and D remaining after operations on the previous blocks (or their initial values, if the block is the first) remain.

AA = A

BB = B

CC = C

DD = D

Stage 1

```
/* [abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 7  1][DABC  1 12  2][CDAB  2 17  3][BCDA  3 22  4]
[ABCD  4 7  5][DABC  5 12  6][CDAB  6 17  7][BCDA  7 22  8]
[ABCD  8 7  9][DABC  9 12 10][CDAB 10 17 11][BCDA 11 22 12]
[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]
```

Stage 2

```
/* [abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  1 5 17][DABC  6 9 18][CDAB 11 14 19][BCDA  0 20 20]
[ABCD  5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA  4 20 24]
[ABCD  9 5 25][DABC 14 9 26][CDAB  3 14 27][BCDA  8 20 28]
[ABCD 13 5 29][DABC  2 9 30][CDAB  7 14 31][BCDA 12 20 32]
```

Stage 3

```
/* [abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  5 4 33][DABC  8 11 34][CDAB 11 16 35][BCDA 14 23 36]
[ABCD  1 4 37][DABC  4 11 38][CDAB  7 16 39][BCDA 10 23 40]
[ABCD 13 4 41][DABC  0 11 42][CDAB  3 16 43][BCDA  6 23 44]
[ABCD  9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA  2 23 48]
```

Stage 4

```
/* [abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */
[ABCD  0 6 49][DABC  7 10 50][CDAB 14 15 51][BCDA  5 21 52]
[ABCD 12 6 53][DABC  3 10 54][CDAB 10 15 55][BCDA  1 21 56]
[ABCD  8 6 57][DABC 15 10 58][CDAB  6 15 59][BCDA 13 21 60]
[ABCD  4 6 61][DABC 11 10 62][CDAB  2 15 63][BCDA  9 21 64]
```

Sum with the result of the previous cycle:

```
A = AA + A
B = BB + B
C = CC + C
D = DD + D
```

After the end of the cycle, we need to check if there are more blocks for computations. If so, go to the next item of array (*n* + 1) and repeat the cycle.

Step 5: Result of the calculations

Result of the calculations is in buffer ABCD, this is a hash. If we output bytewise, beginning with low byte A and ending with high byte D, then we get MD5-hash. 1, 0, 15, 34, 17, 18…

Below is a table of the main differences and functions of all types of nodes

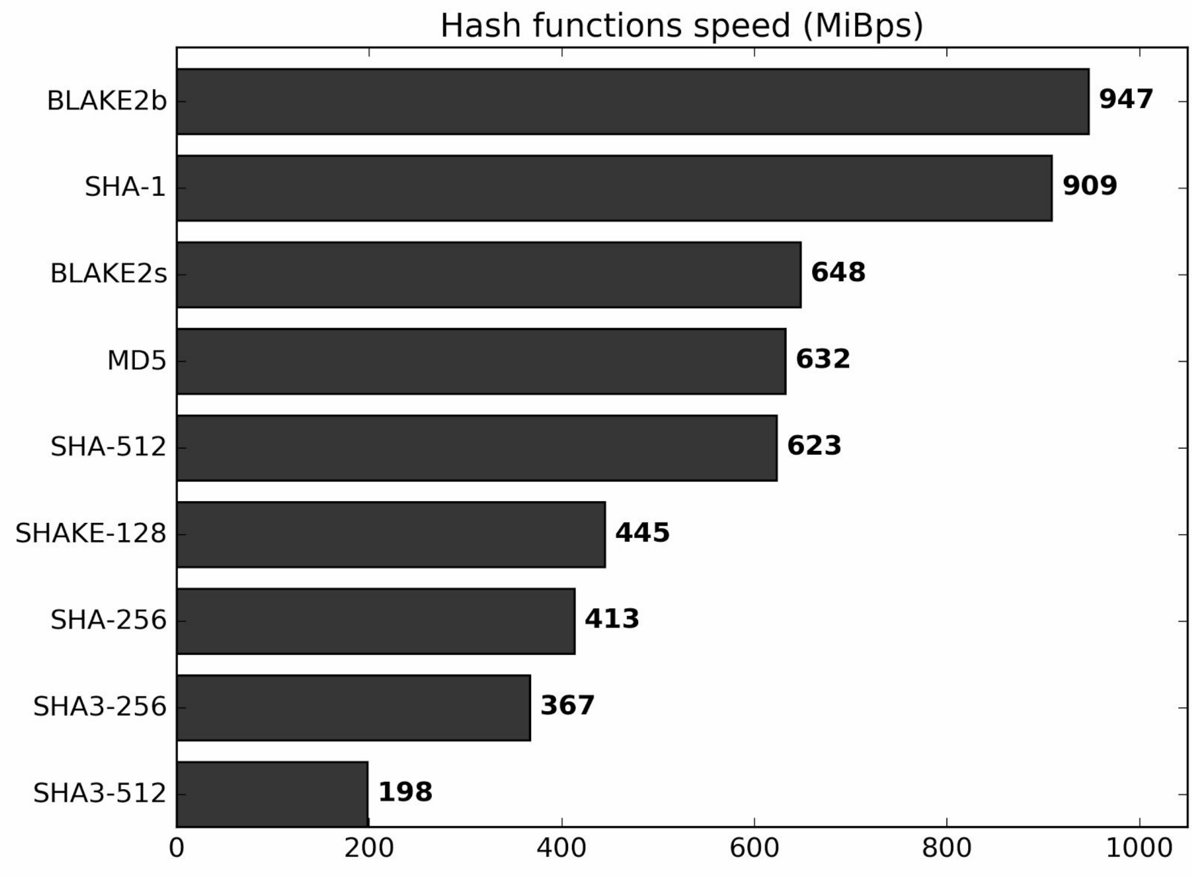
Parameter	Common node	Trusted node	Main node
Direction of exchange	1. With all common nodes of the peer-to-peer network 2. With the main nodes	1. With the main node of the current round for exchange of lists 2. Receiving ledger updates from the peer-to-peer network	1. With trusted nodes of the current round 2. With the main nodes for synchronizing the transaction list in order to identify a double expenditure 3. Receiving ledger updates from the peer-to-peer network
Data for exchange	1. Hashcode of the node and checksum of the ledger to participate in the round of selection of the main and trusted nodes 2. Hashcode of the node for registering on the network and getting the current version of the ledger	1. Hashcode of the node and the generated white list for sending to the main node and compiling the final white list 2. A ranked list of nodes* admitted to voting for becoming a trusted or main node	1. List of candidate transactions for sending to trusted nodes 2. White list of transactions for sending to all nodes of the network for adding the newly generated transaction pool to the local copies of the ledger 3. Filtered list of nodes*

			admitted to become a main or trusted node for sending to trusted nodes of this round 4. Generated list of trusted nodes* and the main node for the next round
Main functions	1. Synchronization of the local copy of the ledger 2. Generation of the checksum of the ledger for participation in voting	1. Synchronization of the local copy of the ledger 2. Forming a list-offer of nodes* that have the opportunity to become a main or trusted node for the next round and sending to the main node 3. Forming the transaction list by checking the transaction pool sent by the main node and confirming or rejecting transactions from the pool and sending to the main node	1. Synchronization of the local copy of the ledger 2. Forming a list-offer of nodes* that have the opportunity to become a main or trusted node for the next round and sending to the main node of the current round 3. Forming a final list of nodes* containing the list of trusted nodes and the main node for work in the next round 4. Forming a list of transaction candidates for confirmation/rejection for entry into the ledger, and sending them to the trusted nodes of the current round 5. Synchronization with previous transactions processed by other main nodes to exclude the possibility of double expenditure 6. Processing the generated lists from trusted nodes and forming a final white list for adding to the ledger across the entire network 7. Processing incoming inquiries from other main nodes to verify transactions and

			provide information about other transfers from/to the account

* - the list of nodes is a list containing hashcodes of nodes

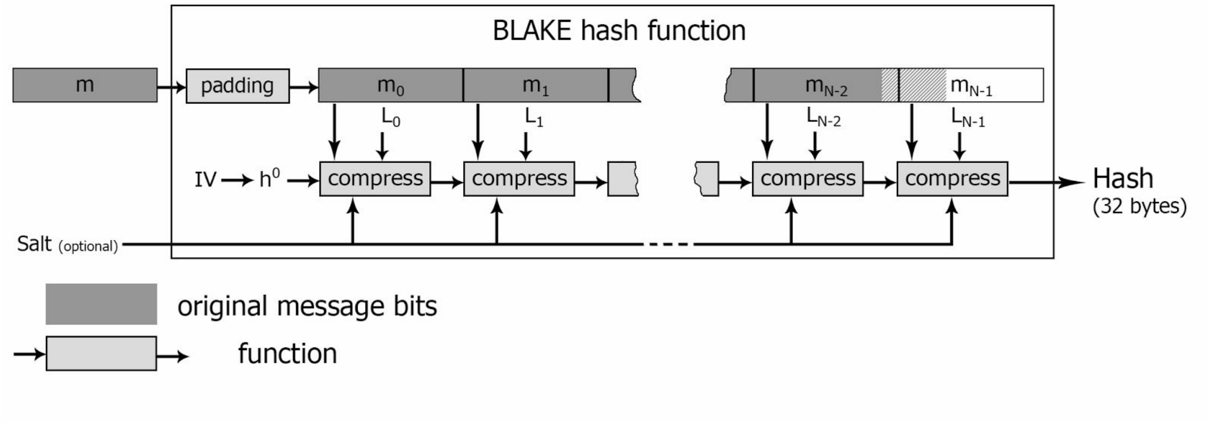
Since the network is public, there is a possibility of adding non-legalized nodes into the network (nodes aimed at changing the probability of making a decision when deciding on a consensus in favor of {a specific value}).The node validation method is used to minimize this probability.
Node validation is a method (process) that proves that the node is real and has the required resources to support the network operation. The node confirms its operability by calculating the checksum of the locally stored ledger and sending to the network for subsequent participation in voting to elect the trusted and main nodes in the new round upon a signal in the network about the start of a new vote (a new round). The checksum is searched using the blake2s algorithm. Comparison of the speed of finding the result of how Hash functions work with other algorithms is given below:



The BLAKE hash function is built from three previously known components:

- HAIFA iteration mode provides resistance to attacks of the second kind
- Internal local wide-pipe structure provides collision protection
- Compression algorithm for BLAKE is a modified version of the well-parallelized ChaCha stream cipher, which security is thoroughly analyzed

The process of hashing is given graphically on the block diagram:



In BLAKE2, there is no addition of constants in the round function. The shift constants are also changed, the addition is simplified, a block of parameters is added, which is summed with the initializing vectors. In addition, the number of rounds is reduced from 16 to 12 in BLAKE2b (similar to BLAKE-512) and from 14 to 10 in BLAKE2s (similar to BLAKE-256). As a result, the number of beats per bit decreased from 7.49 for BLAKE-256 and 5.64 for BLAKE-512 to 5.34 and 3.32 for Blake2s and Blake2b, respectively.

The following is the result of running Hash calculation for the parameters in parentheses for this algorithm:

BLAKE2s-256("") = 69217A3079908094E11121D042354A7C1F55B6482CA1A51E1B250DFD1ED0EEF9

BLAKE2s-256("The quick brown fox jumps over the lazy dog") = 606BEEEC743CCBEFF6CBCDF5D5302AA855C256C29B88C8ED331EA1A6BF3C8812

BLAKE2s-128("") = 64550D6FFE2C0A01A14ABA1EADE0200C

BLAKE2s-128("The quick brown fox jumps over the lazy dog") = 96FD07258925748A0D2FB1C8A1167A73

Consensus solution

CREDITS consensus is a method of group decision-making, with the purpose of the development of final solutions acceptable for all network nodes.

Consensus comparison

For comparison of different types of consensus, define the principles of the CREDITS decentralized ledger:

- Ledger availability (nodes can write data to the ledger and read them from it at any time);
- Modifiability by all participating nodes in the network;
- Consistency of all the nodes of the system (all nodes see an absolutely identical version of the ledger which is updated after the changes);
- Stability to separation (if one node becomes inoperable, this does not affect the entire ledger).

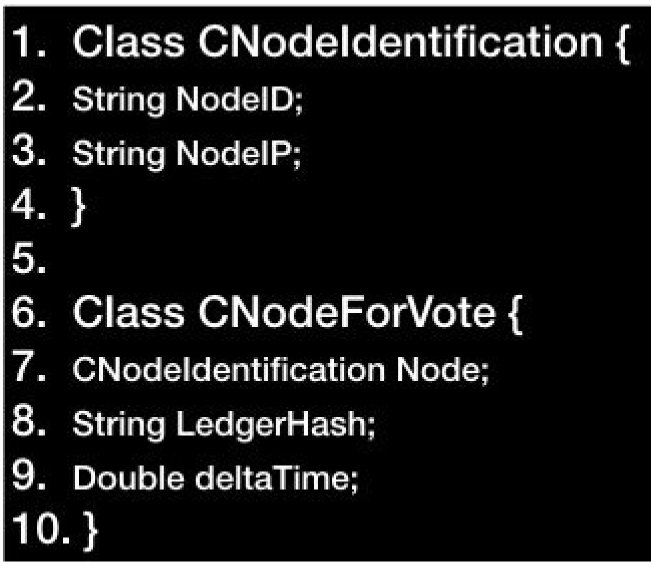
Compared parameter	Credits specific DPOS	PoW	PoS

The principle of identifying the node that created the block	Calculation of mathematical function. Confirmation of storage of the latest copy of the ledger	Performing an iterative calculation of mathematical function with varying complexity	Search among the participants (competing nodes) of the maximum stack
Attack 51%	Unlikely, because it is necessary to have full ledger on the resources and the computational power for calculation, and the choice of trusted nodes is dynamic	Probable, but very expensive it terms of the use of resources	Probable, but expensive, because of the need to increase own stack
Compensation for the work done on the node for adding to the ledger/blockchain	Calculated automatically, depends on the commission for the operation	Is provided fixed the block mining	Is provided fixed the block mining

The process of searching the main and trusted nodes

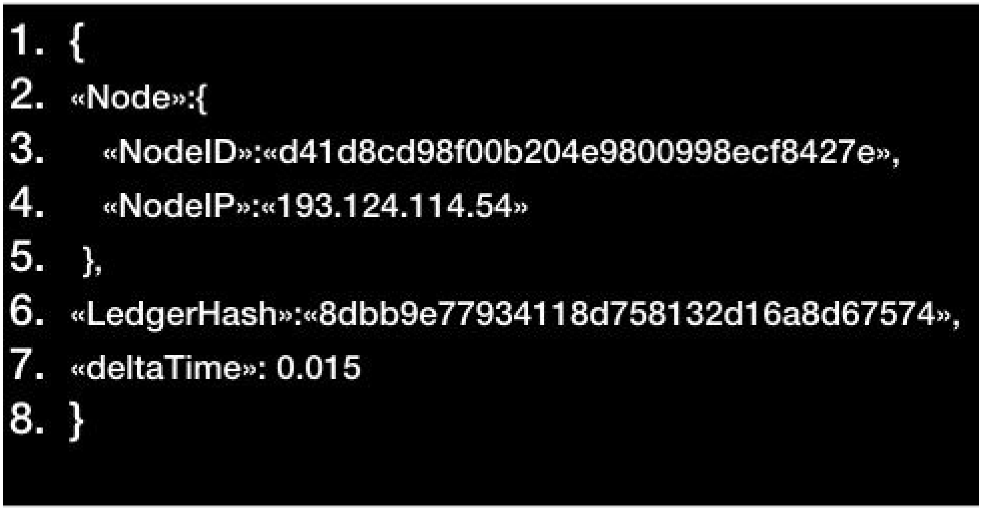
The process of the decision making system operation on adding transactions to the common decentralized ledger can be given in the form of an iterative (cyclic) approach. The cycle (round) can be represented by the following algorithm:

1. Calculation of the checksum (BLAKE2s algorithm described above) of the ledger on the nodes not participating in the previous rounds (iterations) and sending the result to the network for participation in voting. The data model and format are shown below:



```
Class CNodeIdentification {
String NodeID;
String NodeIP;
}

Class CNodeForVote {
CNodeIdentification Node;
String LedgerHash;
Double deltaTime;
}
```



```
{
«Node»:{
   «NodeID»:«d41d8cd98f00b204e9800998ecf8427e»,
   «NodeIP»:«193.124.114.54»
 },
«LedgerHash»:«8dbb9e77934118d758132d16a8d67574»,
«deltaTime»: 0.015
}
```

2. A primary list is generated on the main node of the last round and sent to the trusted nodes to assign a ranking for each node.
3. At the trusted node, each node receives a numerical value obtained on the basis of the function to obtain a random value:

```
boost::random::mt19937 gen;
boost::random::uniform_int_distribution<> dist(0, [count_list_nodes-1]);
int x = dist(gen);
```

4. Processing and generating a ranked list of nodes based on lists received from trusted nodes of the last round, list sorting by decrease of the assigned rank and excluding from the list those that are below the maximum allowable amount for this round. The maximum value of the permissible number of possible nodes in the list is calculated according to the mathematical model and the justification based on the network complexity and number of nodes (the mathematical model will be given in the next edition).
5. The main node sorts the list in accordance with the increase of the time to generate the checksum of the ledger. The result will be a list of nodes headed by the node with the least checksum generation time; this node will be the main node in the next round, all other nodes are trusted.
6. The main node of the current round sends a generated list of nodes to the network. The main node of the next round waits for connection of trusted nodes, after which the next iteration begins. If the trusted node from the list is not connected for some reason, then the node tries to connect itself, if the attempt fails, then the node is excluded from the list, and the invitation is sent to the next node beyond the line.

Analysis and decision making to add a transaction to the white list

To analyze and decide to add a transaction to the white list, we propose using a modification of the federal voting algorithm, called BTF (Byzantine Fault Tolerant).

Figuratively, this task can be described as follows:

Byzantium. On the night before the great battle, the Byzantine army has n legions. Each of them obeys their general. The whole Byzantine army has a commander-in-chief, leading generals. The empire is in decline and there may be traitors among the generals, including the commander-in-chief. Throughout the night, each of the generals receives from the leader the order of actions for the morning. This can be one of two options: to "attack" or "retreat". If all honest generals attack – they will win. If everyone retreats, they will be able to retain the army. If part of them attack, and part of them retreat – they are defeated. If the commander-in-chief is a traitor, he can give different orders to different generals, hence his orders should not be obeyed unquestionably. If each general acts independently of others, the results of the battle can also be deplorable. Therefore, generals need to exchange information with each other in order to come to a consensus.

Definition

There are n generals. They communicate by means of reliable communication (for example, telephone). Of n generals, m are traitors and try to prevent an agreement between loyal generals. The consensus is that all loyal generals have learned about the number of all loyal armies and come to the same conclusions (let them be false) regarding the state of treacherous armies. (The latter condition is important if the generals plan to develop a strategy based on the data obtained, and it is necessary that all generals develop the same strategy).

As a result:

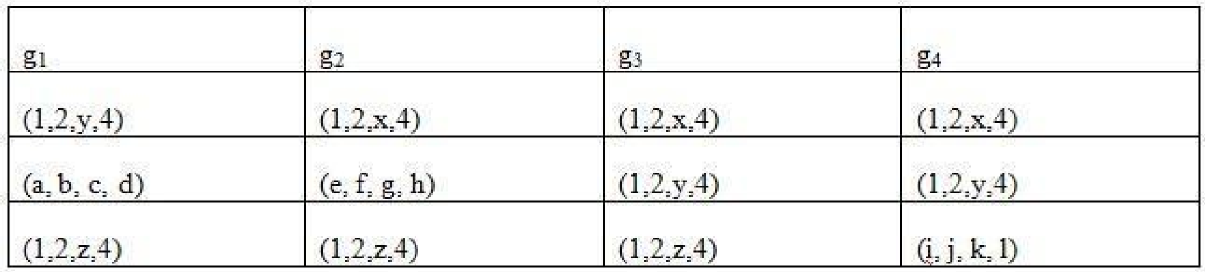
Each loyal general must receive a vector of length n, where the i-th element either necessarily contains the numerical force of the i-th army (if the commander is loyal), or contains an arbitrary number otherwise. Vectors should be completely identical for all loyal generals.

Solution algorithm

Step 1: Each of the generals sends a message to the others, indicating the numerical force of his army. Traitors can specify different numbers in different messages, and loyal generals specify the true number. General g1 specified 1 (one thousand warriors), general g2 – 2, general g3 respectively specified to the three remaining generals x, y, z, general g4 – 4.

Step 2: Each of the generals computes its vector from the information received from the rest of the generals. We derive vect1 (1,2,x,4), vect2 (1,2,y,4), vect3 (1,2,3,4), vect4(1,2,z,4).

Step 3: Generals send their vectors to others. Generator g3 again sends arbitrary values. The following vectors are obtained:

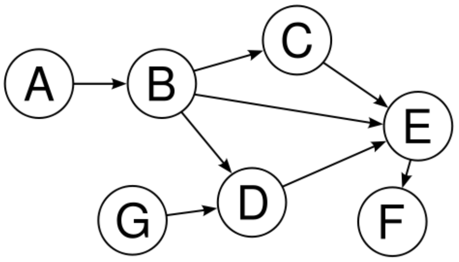


g_1	g_2	g_3	g_4
(1,2,y,4)	(1,2,x,4)	(1,2,x,4)	(1,2,x,4)
(a, b, c, d)	(e, f, g, h)	(1,2,y,4)	(1,2,y,4)
(1,2,z,4)	(1,2,z,4)	(1,2,z,4)	(i, j, k, l)

Step 4: Each of the generals checks each element in the resulting vectors. If one value coincides in at least two vectors, it is placed in the result vector, otherwise, the corresponding item is marked as "unknown". As a result, all generals will receive one vector (1, 2, unknown, 4). Consequently, consensus has been reached. For n=3 and m=1, consensus will not be reached.

The algorithm is the same for the real working system, but the general is a node, and messages (the aligned vector) represent a transaction pool where each transaction is assigned a value (1 – Confirm, 0 – Fault).

Search for transaction participants



The CREDITS peer-to-peer network can be represented as a graph, with user accounts in the form of vertices and a multitude of possible transactions in the form of directed edges that connect two vertices (account). Since all edges have an initial and a terminal vertex, you can always construct an oriented graph (orgraph). If we take the following conditions for identity:

- Any transaction always has a sender and a receiver;
- Any vertex (account) can always be connected to another vertex (account) with a directed edge (transaction);
- Any vertex of the graph (account) has a finite number of directed edges (incoming and outgoing transactions).

In connection with the foregoing, we can say that the orgraph may contain the required route (a finite sequence of vertices in which each vertex, except the last one, is connected to the next vertex in the sequence by an edge) for fulfilling the necessary transaction conditions, and construct a simple chain (a simple chain is the route in the orgraph without repeated vertices).

Since the graph is not known in advance, proceeding from the graph theory – the route will have to be built according to an unknown oriented graph. As is known for the class of such graphs, the girth length is $\Theta(nm)$, where n is the number of vertices, and m is the number of edges of the graph. For any graph, there exists a girth of length $O(nm)$ and there exist graphs with the minimum girth length $\Omega(nm)$. The girth of an unknown graph means that its topology is not known in advance, and we will know it only in the course of moving along the graph. In each vertex, we can see edges that outgo from it, but we can see which vertex the edge leads to just by following it. This is similar to the task of traversing a labyrinth by a robot inside it and not having a labyrinth plan. If the number of states is limited, then the robot is the finite-state automaton. Such robot is an analog of the Turing machine: the tape is replaced by a graph, and its cells are tied to the vertices and edges of the graph.

Add a restrictive rule – the edge orientation must satisfy the transaction/contract conditions. If the orientation does not correspond to the desired values, the transition along the edge is not performed.

So, let’s set a task to construct a route with a simple chain.

The oriented graph where the robot works can be defined as $G=(V,E,\alpha,\beta,\gamma,\delta,X,\chi)$, where:

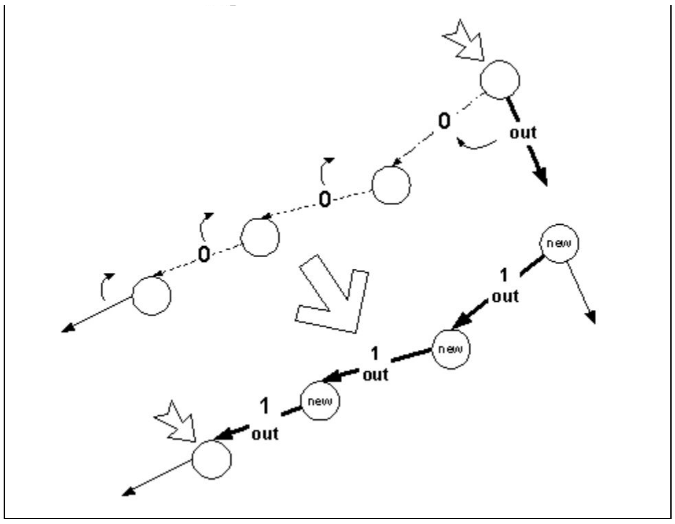
- V – is a set of vertices;
- E – is a set of edges (for convenience we will assume that $E\cap V=\varnothing$);
- $\alpha:E\to V$ – is a function that determines the initial vertex (beginning) of the edge;
- $\beta:E\to V$ – is a function that determines the final vertex (end) of the edge;
- $\gamma:V\to E$ – is a function that determines the first arc in the cycle of outgoing edges, with the following condition:
    - $\forall v\in V\ dout(v)>0 \Rightarrow \alpha(\gamma(v))=v$;
- $\delta:E\to E$ – is a function that determines the first arc in the cycle of outgoing edges, with the following condition:
    - $\forall e\in E\ \exists\ k=0..dout(\alpha(e))-1\ \delta k(e)=\gamma(\alpha(e))$, where $\delta k=\delta\circ\delta\circ\ldots\circ\delta$ and the superposition sign is applied k-1 times;
- X – is a set of symbols that can be stored in cells of vertices and edges;
- $\chi:V\cup E\to X$ – is a function that defines the symbols stored in cells of vertices and edges.

A graph is finite if sets V and E are finite. The number of vertices of the finite graph is denoted by $n=|V|$. Two edges e and e` are adjacent, if $\beta(e)=\alpha(e`)$. A route or a path is a sequence of adjacent edges. The route

passes through a vertex if it is the beginning or end of some edge route. The beginning of the first edge of the route is called the beginning of the route, and the end of the last edge is called the end of the route.
A simple path is a route that does not pass through one vertex more than once.
A cycle is a route with a coinciding beginning and end. In a simple cycle, the beginning and the end are the only coinciding vertices.
A girth is a route containing all edges of the graph. A graph is strongly connected if any pair of vertices is connected by some route. The robot on graph G is defined as R=(Q,X,T), where:

- Q is a set of the states;
- X is a set of input symbols (coincides with the set of symbols in the graph cells);
- $T \subseteq Q\times X\times X\times Q\times X\times X\times\{i,o\}$ is a set of girths.

At each time, the robot is located in the current vertex $v \in V$, on the current edge $e \in E$, outgoing from v, that is, $\alpha(e)=v$. The robot is in the state $q \in Q$, reads the vertex symbol $xv=\chi(v)$ and the edge symbol $xe=\chi(e)$.
Transition $(q,xv,xe,q`,x`v,x`e,i/o) \in T$ means that the robot goes to the state q`, writes symbols to the cell of vertex $\chi(v)=x`v$ and to cell of edge $\chi(e)=x`e$.
With an internal transition (i), the robot remains at the same vertex v, but moves to the next edge in v-cycle $\delta(e)$.
With an external transition (o), the robot moves along edge e to its final vertex $\beta(v)$ to the first edge outgoing from it $\gamma(\beta(v))$. The robot is finite if sets Q and X are finite. The robot is determinate if for every three $(q,xv,xe) \in Q\times X\times X$, there is not more than one transition $(q,xv,xe,q`,x`v,x`e,i/o) \in T$.
If there is no transition for a certain three (q,xv,xe), the robot stops in state q at the vertex with symbol xv on the edge with symbol xe. We will assume that one character $\varepsilon \in X$ is allocated as an initial, which is in all cells of vertices and edges at the beginning of the robot operation. The vertex where the robot begins to operate will be called the initial and denoted by v0. The initial edge is the first outgoing edge $\gamma(v0)$. The sequence of external transitions that robot R makes on graph G from the beginning of operation obviously determines the route in G, which we will call the passed through route. This route is finite if the robot stops.



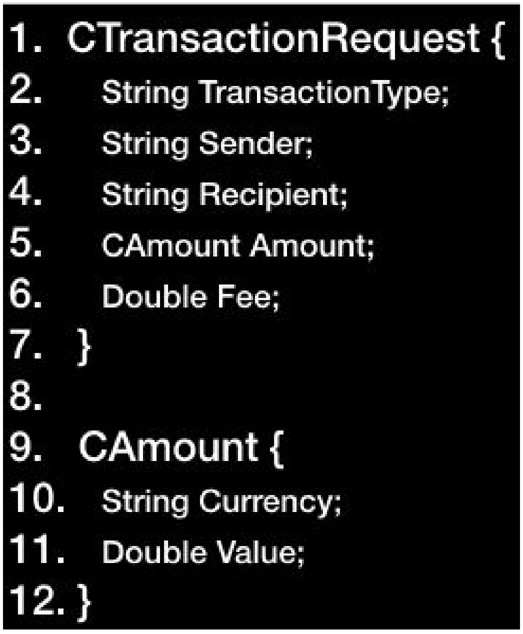
The robot passes through the graph if it stops at this graph and the passed route is a girth. If each edge outgoing from vertex v belongs to route P, this vertex will be called completely passed (in route P). Obviously, for strongly connected graphs, the route passed through is a girth if and only if all its vertices are completely passed through.
To solve the above problem, it is proposed to use the passage algorithm by not passed through edges. Imagine that we are in open-vertex v and, if at least one edge outgoes from it, the current edge p – the first in the v-cycle is not passed through. We move along not passed through edges until the passed through vertex or to the terminal vertex. To do this, we first check whether v is a terminal vertex. If vertex v is terminal, then we return from the procedure with response "came to the terminal vertex". Note that in a graph, the terminal vertex can only exist if it is the only vertex of the graph. In this case, obviously, the robot has passed through the graph. If vertex v is not terminal, the not passed through edge p=(v,w) outgoes from it. Mark vertex v with the "new" tag, change the status of edge p from 0 to 1, mark it as an

out-edge and go along it to its end w. If vertex w is passed through, we return from the procedure with response "chord passed through". Otherwise, mark it as root-vertex and repeat the procedure for vertex w.

Transactions

A transaction is the minimum unit of the system informing the platform of the execution of contract methods or direct transfers between accounts without creating a smart contract, followed by placement of the result in the peer-to-peer network.

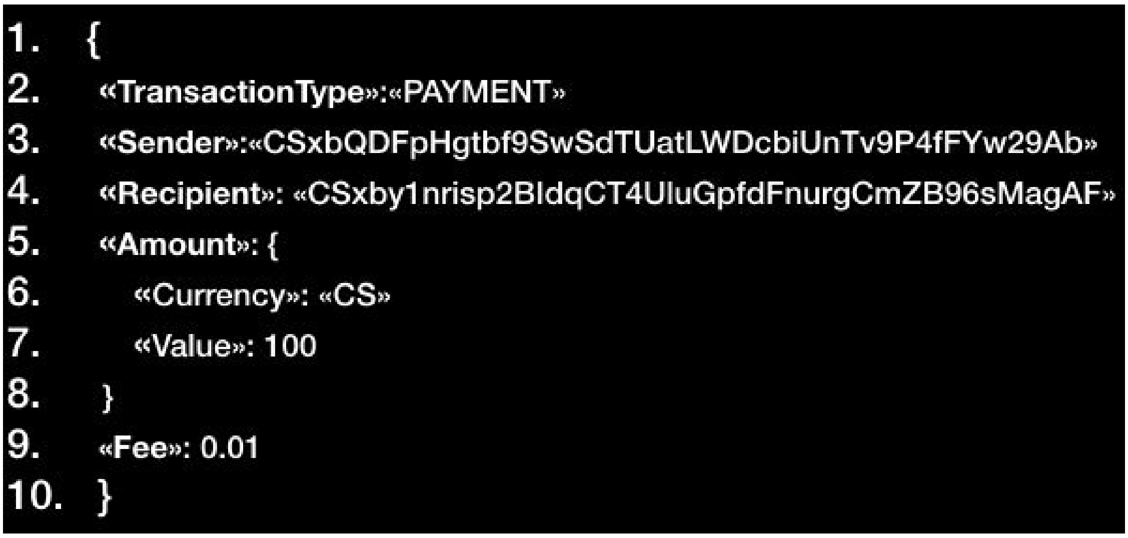
The transaction is generated on the client side according to the model:



```
CTransactionRequest {
    String TransactionType;
    String Sender;
    String Recipient;
    CAmount Amount;
    Double Fee;
}

 CAmount {
    String Currency;
    Double Value;
}
```

In accordance with the above model, you can display the contents of the transmitted data:



```
{
 «TransactionType»:«PAYMENT»
 «Sender»:«CSxbQDFpHgtbf9SwSdTUatLWDcbiUnTv9P4fFYw29Ab»
 «Recipient»: «CSxby1nrisp2BldqCT4UluGpfdFnurgCmZB96sMagAF»
 «Amount»: {
   «Currency»: «CS»
   «Value»: 100
 }
 «Fee»: 0.01
}
```

After generation, the transaction is signed with PrivateKey (the ecdsa25519 algorithm is described in the security section), encrypted using the homomorphic encryption algorithm (the algorithm is described in the security section) and sent to the network for further processing.

After the transaction enters the network, its hash is generated based on the Blake2s algorithm (described above).

Security and possible threats

The algorithm of homomorphic encryption is used to improve the cryptographic strength and reliability of the system, which increases the speed of analysis and transaction processing.

Homomorphic encryption is a form of encryption that allows performing certain mathematical actions with encrypted text and obtaining an encrypted result that corresponds to the result of operations performed with plain text. For example, one person could add two encrypted numbers, and then another person could decipher the result without using any of them. Homomorphic encryption would allow combining different services into a unified whole, without providing data for each service.
There are partially homomorphic and completely homomorphic cryptosystems. While a partially homomorphic system allows simultaneous execution of only one of the operations – addition or multiplication, completely homomorphic cryptosystems support simultaneous performance of both operations, which allows us to homomorphically compute arbitrary logical circuits.

Completely homomorphic encryption is the encryption, which for a given ciphertext $\pi_1, \ldots, \pi_t$ allows anyone (not just the key holder) to obtain the ciphertext of any desired function $f(\pi_1, \ldots, \pi_t)$, as long as this function can be efficiently computed.

Ciphertext is the result of an encryption operation. It is also often used instead of the term "cryptogram", although the latter emphasizes the very fact of message transmission, rather than encryption.

When considering the ciphertext as a random variable Y=f(X,Z), depending on the corresponding random variables of plain text X and encryption key Z, you can define the following ciphertext properties:

- Uniqueness of encryption:

$H(Y|XZ)=0$

- From chain equalities it follows that

$H(ZYX)=H(Z)+H(Y|Z)+H(X|YZ)=H(Z)+H(Y|Z)+0$ (from the deciphering unambiguity property)

$H(ZXY)=H(Z)+H(X|Z)+H(Y|XZ)=H(Z)+H(X)+0$ (from the principle of plain text independence on the key and the deciphering unambiguity property)

then

$H(Y|Z)=H(X)$ this equation is used to derive the formula for the uniqueness distance.

- For an absolutely reliable cryptosystem

$I(Y,X)=0$, i.e. $H(Y)=H(Y|X)$

The function of homomorphic encryption *E(k,m)*, where *m* is the plain text, *k* is an encryption key, is homomorphic with respect to operation * *(multiplication)* over plain texts if there exists an effective algorithm *M* that upon receipt of any pair of cryptograms in the form $E(k,m_1),E(k,m_2)$, produces cryptogram *C* such that when decrypting c, a plain text m_1*m_2 will be obtained. Similarly, we define a homomorphism with respect to operation + *(addition)*.

While partially homomorphic cryptosystems are homomorphic only with respect to one operation on the plain text (addition or multiplication), completely homomorphic systems support homomorphism with respect to both operations (addition and multiplication). That is, the following conditions are fulfilled for them:

$$\begin{cases} Dec(Enc(m_1) \otimes Enc(m_2)) = m_1 * m_2 \\ Dec(Enc(m_1) \oplus Enc(m_2)) = m_1 + m_2 \end{cases}$$

Moreover, homomorphism with respect to addition and multiplication operations is sufficient for the system to be completely homomorphic.

Security of most completely homomorphic schemes is based on the difficulty of solving the learning problem with errors. Only in LVT (a cryptosystem on NTRU created by Lopez-Alt, Tromer and Vaikuntanahan), the security scheme is implemented on the option of the computational NTRU problem. All these systems, in contrast to the earlier schemes, have a slower noise increase in the process of homomorphic computations. As a result of additional optimization, a cryptosystem with almost optimal asymptotic complexity was obtained: complexity of calculating *T* operations over encrypted data with security parameter *k* has complexity of computing of just $T \cdot polylog(k)$. These optimizations are built on the Smart-Vercauteren technique, which allows compressing a set of text variables into one ciphertext and working on these variables in a single stream. Many achievements for homomorphic systems were also used in the cryptosystem on integers.

The scheme of completely homomorphic encryption can be considered using the example of calculations in Z_2.

Encryption

The process of data encryption can be represented as follows:

1. Choose an arbitrary odd number $p=2k+1$, which is a secret parameter. Let $m \in \{0,1\}$.

2. The number $z \in Z_2$ is compiled such that $z=2r+m$, where *r* is an arbitrary number. This means that $z=m \bmod 2$.

3. In the process of encryption, any *m* is associated with a number $c=z+pq$, where *q* is chosen arbitrarily. Thus, $c=2r+m+(2k+1)*q$. It is easy to see that $c \bmod 2=(m+q) \bmod 2$, and therefore, the attacker will only be able to determine the parity of the encryption output.

Decryption

Let the encrypted number *c* and secret *p* be known. Then the process of decrypting data should contain the following actions:

1. Decryption using the secret parameter p: $r=c \bmod p=(z+pq) \bmod p=z \bmod p+(pq) \bmod p$, where $r=c \bmod p$ is called the noise and $z \in (-p/2,p/2)$.

2. Obtaining the original encrypted bit: $m=r \bmod 2$

Justification

Let there be two bits $m_1,m_2 \in Z_2$ and a pair of numbers $z_1=2r_1+m_1$ and $z_2=2r_2+m_2$ are put into correspondence. Let the secret parameter $p=2k+1$ be taken and data encryption: $c_1=z_1+pq_1$ and $c_2=z_2+pq_2$ is performed.

The sum of these numbers is calculated:

$$c_1+c_2=z_1+pq_1+z_2+pq_2=z_1+z_2+p(q_1+q_2)=2r_1+m_1+2r_2+m_2+(2k+1)(q_1+q_2)$$

For the sum of these numbers, the decrypted message is the sum of the original bits m_1,m_2:$((c_1+c_2) \bmod p) \bmod 2=(2(r_1+r_2)+m_1+m_2) \bmod 2=m_1+m_2$.

But without knowing p, it is not possible to decipher the data: $((c_1+c_2) \bmod p) \bmod 2=m_1+m_2+q_1+q_2$.

The multiplication operation is checked similarly:

$$c_1c_2=(z_1+pq_1)(z_2+pq_2)=z_1z_2+p(z_1q_2+z_2q_1)+p_2q_1q_2=(2r_1+m_1)(2r_2+m_2)+(2k+1)((2r_1+m_1)q_2+(2r_2+m_2)q_1)=4r_1r_2+2(r_1m_2+r_2m_1)+m_1m_2+2k(2r_1q_2+m_1q_2+2r_2q_1+m_2q_1)+2r_1q_2+m_1q_2+2r_2q_1+m_2q_1$$

The decryption procedure shall be applied to the results obtained, resulting in the following:

$$((c_1c_2) \bmod p) \bmod 2=(4r_1r_2+2(r_1m_2+r_2m_1)+m_1m_2) \bmod 2=m_1m_2.$$

To obtain keys, elliptic curves built on ecdsa25519 are used.

The elliptic curve over field K is a nonsingular cubic curve on the projective plane over K' (the algebraic closure of field K) given by a third-degree equation with coefficients from field K and a "point at infinity". In suitable affine coordinates, its equation is reduced to the form:

$$y2+a1xy+a3y=x3+a2x2+a4x+a6.$$

<u>Conceptual mathematical model</u>

If *charK* (characteristic of the field *K*) is not equal to 2 or 3, then the equation can be reduced to the canonical form (Weierstrass form) by coordinate substitution:

$$y^2=x^3+ax+b.$$

If char *charK=3*, then the canonical form of the equation is:

$$y^2=x^3+a_2x^2+a_4x+a_6.$$

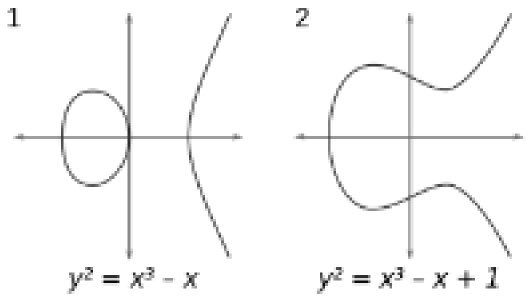
If *charK=2*, the equation is reduced to one of the following types:

$y^2+y=x^3+ax+b$ — supersingular curves

or

$y^2+xy=x^3+ax^2+b$ — nonspersingular curves.

Elliptic curves over real numbers



Since the characteristic of the field of real numbers is *0*, and not *2* or *3*, the elliptic curve is a plane curve determined by an equation of the following form:

$y^2=x^3+ax+b$,

where *a* and *b* are real numbers. These kind of equations are called the **Weierstrass equations**.

The definition of an elliptic curve also requires that the curve has no singular points. Geometrically, this means that the graph should not have cusps and self-intersections. Algebraically, it suffices to verify that the discriminant

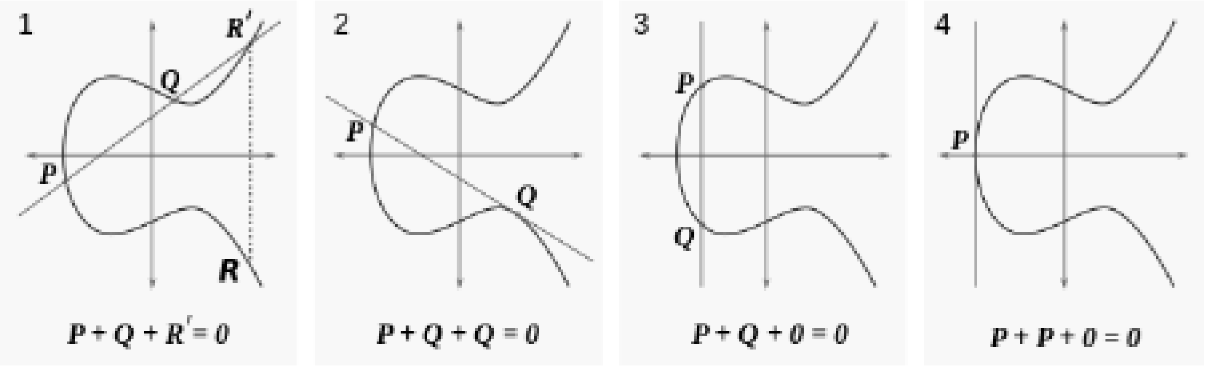
$\Delta=-16(4a^3+27b^2)$

is not equal to zero.

If the curve has no singular points, then its graph has two connected components, if the discriminant is positive, and one – if it is negative. For example, for the graphs above in the first case the discriminant is *64*, and in the second case it is *-368*.

Group law

By adding a "point at infinity", we obtain a projective version of this curve. If *P* and *Q* are two points on a curve, then it is possible to uniquely describe the third point – the point of intersection of the given curve with the line drawn through *P* and *Q*. If the line is a tangent to the curve at a point, then such a point is taken into account twice. If the line is parallel to the ordinate axis, the third point will be the point at infinity.



Thus, we can introduce the group operation *"+"* on a curve with the following properties: the point at infinity (denoted by the symbol *O*) is the neutral element of the group, and if the line intersects this curve at points *P*, *Q* and *R'*, then $P+Q+R'=O$ in the group. The sum of points *P* and *Q* is point $R=P+Q$, which is symmetric to point *R'* relative to axis *Ox*. It can be shown that with respect to the operation thus introduced, the points on the curve and point *O* form an Abelian group; in particular, the associativity property of operation *"+"* can be proved using the theorem on 9 points on a cubic curve (a cube).

This group can be described algebraically. Let $y^2=x^3+ax+b$ be a curve over field *K* (its characteristic is neither *2* nor *3*), and points $P=(x_P,y_P)$ and $Q=(x_Q,y_Q)$ on the curve; assume that $x_P\neq x_Q$. Let $s=(y_P-y_Q)/(x_P-x_Q)$; since *K* is a field, *s* is strictly defined. Then we can define $R=P+Q=(x_R,y_R)$ as follows:

$x_R=s^2-x_P-x_Q$,

$y_R = -y_P + s(x_P - x_R)$.

If $x_P = x_Q$, there are two options: if $y_P = -y_Q$, the sum is defined as *0*; then the point inverse to any point on the curve can be found by reflecting it with respect to axis *Ox*. If $y_P = y_Q \neq 0$, then $R = P + P = 2P = (x_R, y_R)$ is defined as follows:

$s = (3x^2_P + a)/2y_P$,

$x_R = s^2 - 2x_P$,

$y_R = -y_P + s(x_P - x_R)$.

If $y_P = y_Q = 0$, then $P + P = O$.

The element inverse to point *P*, denoted by $-P$ and such that $P + (-P) = 0$, in the group considered above is defined as follows:

- If coordinate y_P of point $P = (x_P, y_P)$ is not *0*, then $-P = (x_P, -y_P)$.
- If $y_P = 0$, then $-P = P = (x_P, y_P)$.
- If $P = O$ is a point at infinity, then $-P = O$.

Point $Q = nP$, where *n* is an integer, is defined (at $n > 0$) as$Q = P + P \cdots + P$. If $n < 0$, than *Q* is the inverse element to $|n|P$. If $n = 0$, then $Q = 0 \cdot P = O$. For example, let's show how to find point $Q = 4P$: it is represented as $4P = 2P + 2P$, and point *2P* is found by the formula $2P = P + P$.

ecdsa25519 is a cryptographic function based on elliptic curves, developed using an elliptic curve based on the Diffie-Hellman algorithm.

The curve is mathematically justified by the following graphical function $y^2 = x^3 + 486662x^2 + x$. ECDSA25519 is designed in such a way that it avoids many potential implementation errors. By its structure, it is immune to time-attacks and takes any 32-byte string as a valid public key and does not require verification.

The Diffie-Hellman protocol on elliptical curves is a cryptographic protocol that allows two parties that have public/private key pairs on elliptical curves to obtain a common secret key using a communication channel unprotected from listening through. This secret key can be used for both the encryption of further exchange, and for the generation of a new key, which can then be used for subsequent exchange of information using symmetric encryption algorithms. This is a variation of the Diffie-Hellman protocol using elliptical cryptography.

Algorithm description

Let there be two subscribers. Suppose the first subscriber wants to create a secret key common with the second subscriber, but the only channel available between them can be listened through by a third party. Initially, a set of parameters *((p, a, b, G, n, h)* for the general case and *(m,f(x),a,b,G,n,h)* for the characteristic field 2) must be agreed. Also, each party must have a key pair consisting of a private key *d* (a randomly selected integer from interval [*1, n-1*]) and a public key *Q* (where $Q = d \cdot G$ is the result of doing summation of element *G d* times). Let then the key pair of the first subscriber be (d_A, Q_A), and the key pair of the second subscriber (d_B, Q_B). Before executing the protocol, the parties must exchange public keys.

The first subscriber calculates $(x_k, y_k) = d_A \cdot Q_B$. The second subscriber calculates $(x_k, y_k) = d_B \cdot Q_A$. The general secret is x_k (the x-coordinate of the resulting point). Most standard protocols based on ECDH use the key generation functions to obtain a symmetric key from x_k.

The values calculated by the participants are equal, since $d_A \cdot Q_B = d_A \cdot d_B \cdot G = d_B \cdot d_A \cdot G = d_B \cdot Q_A$. Of all the information related to its private key, the first subscriber communicates his public key only. Thus, no one except the first subscriber can determine its private key, except for a participant capable of solving the discrete logarithmation problem on an elliptical curve. Bob's private key is similarly protected. Nobody except the first subscriber or the second subscriber can calculate their common secret, except for the participant who can solve the Diffie-Hellman problem.

Public keys are either static (and confirmed by a certificate) or ephemeral (abbreviated ECDHE). Ephemeral keys are used temporarily and do not necessarily identify the sender, so if identification is required, authentication must be obtained in a different way. Identification is necessary to exclude possible attack by the intermediary. If the first subscriber or the second subscriber uses a static key, the threat of the intermediary's attack is excluded, but neither direct privacy nor resistance to substitution with compromising the key can be provided, as well as some other properties of resistance to attacks. Users of static private keys are forced to check someone else's public key and use the function of generating a key to a shared secret to prevent leakage of information about a static private key. The MQV protocol is often used for encryption with other properties.

When using a shared secret as a key, it is often desirable to hash the secret to get rid of the vulnerabilities that arise after applying the protocol.

MQV (**Menezes-Qu-Vanstone**) is an authentication protocol based on the Diffie-Hellman algorithm. MQV provides protection against active attacks by combining static and temporary keys. The protocol can be modified to work in an arbitrary finite commutative group, and in particular in groups of elliptic curves, where it is known as **ECMQV**.

MQV is included in the project for standardization of public-key cryptosystems – IEEE P1363.

The first subscriber has a static key pair (W_a, w_a), where W_a is its public key and w_a is its secret key. The second subscriber has a static key pair (W_b, w_b), where W_b is its public key and w_b is its secret key. Define R'. Let $R=(x,y)$ be a point on an elliptic curve. Then $R'=(x \bmod 2^{L})+2^L$, where $L=(\log_2 n+1)/2$ and n is the order of generator of point P. So, R' is the first L bits of coordinate x for R. In addition, introduce cofactor h, defined as $h=|G|n$, where $|G|$ is the order of group G, and it should be taken into account that for technical reasons, the requirement $gcd(n,h)=1$ must be met.

Example

The elliptic curve E over field $GF(2^{163})$ is of order $2 \cdot P49$, where $P49$ is a prime number consisting of *49* digits in the decimal notation.

$E: Y^2+XY=X^3+X^2+1$

Choose an irreducible polynomial

$1+X+X^2+X^8+X^{163}$

And take the point of the elliptic curve

P=(d42149e09429df4563ec1816488c92de89f93a9b2, ccd18d6cc3042c4c17a213506345c80965ac19476)≠0.

Check that its order is not equal to *2*

2P=(ccd18d6cc3042c4c17a213506345c809b5ac1d476, 835a2f56b88d6a249b4bd2a7550a4375e531d8a37).

Hence, its order is equal to the order of group $2 \cdot P49$, namely the number $P49$, and it can be used to build the key. Let $k_A=12$, $k_b=123$. Then the public keys of the protocol participants are calculated as

$k_A \cdot P=12 \cdot P=$(bd9776bbe87a8b1024be2e415952f527eee928b43, c67a28ed7b137e756c37654f186a71bf64e5ac546).

$k_B \cdot P=123 \cdot P=$(a5684e246044fc126e9832d17513387e474290547, 568b4137f09f5f79a8a6b0fe44cdf41d8e68ae2c6).

And the shared secret will be:

$k_B \cdot k_A \cdot P=k_A \cdot k_B \cdot P=12 \cdot 123 \cdot P=$(bb7856cece13c71919534878bcb6f3a887d613c92, f661ffdfe1ba8cb1b2ad17b6550c65aa6d4f07f41).

The following value (or its part) is used as the key of the symmetric system x=bb7856cece13c71919534878bcb6f3a887d613c92.

Possible lists of security threats and options for their prevention:

№	Threat	Solutions
1	Attack of the intermediary	Using static keys and user identification (user confirmation)
2	Time attack	Using the ecdsa25519 algorithm
3	Attack 51%	The attack probability will decrease with an increase in the number of nodes, since when creating a node, the program complex should confirm by means of obtaining the checksum of the ledger file, its (ledger's) real physical local existence
4	Hacking of Public - Private keys bundle	Using modern key generation algorithm based on elliptic curves by the ecdsa25519 algorithm
5	Attacking the environment of (legal) node by compromised nodes	To minimize the likelihood of such an attack, the algorithm to confirm the node legality is used (providing the checksum of the ledger file located on the local disk space), together with random selection of the possible environment, by ranking the nodes contained in the list and their further processing
6	Participation in the decision-making round on inclusion of a transaction by the Byzantine Fault Tolerant algorithm of the compromised node	The system does not allow nodes to make decisions on transactions that are unable to confirm the relevance and real existence of the ledger and software complex on the local storage by generating a checksum (the blake2s algorithm). The checksum (hash) is checked when the white list of nodes is formed for the next stage

7	Wallet is poorly protected from theft	Any information transmitted on the network is encrypted through homomorphic encryption
8	Sybil Attack	The nodes synchronize the network address table with the remaining nodes and establish connections with a random set of nodes, which avoids substitution of packages and the environment of legitimate nodes
9	The problem of double expenditure.	Minimization of the likelihood of such an attack by reducing the time to form a transaction pool, resulting in a shorter decision-making time; also when working in multi-server transaction processing mode (parallel processing of transactions), synchronization of lists on the main nodes is used.
10	Interception and change of information/transaction while passing through the network.	Using the latest elliptic curve algorithms (ecdsa25519) to implement a more crypto-resistant signature and reliable protection, to increase reliability of data, a homomorphic encryption algorithm is used, increasing the speed by processing information without having to completely decode it.
11	Intervention in work with memory.	Using the C++ development language to fully control the memory and, if necessary, completely manage the ledgers and check their contents in order to intercept the attempt to compromise data that is being processed in memory.
12	Changing the local repository for the purpose of substituting legal values.	Recording of encrypted data and compression with a modified deflate algorithm, which will provide an increased level of reliability of storage and a higher coefficient of crypto-resistance of the local storage system.
13	Centralization of processing.	In a time interval determined mathematically, the node cannot be the main or trusted twice. This concept also avoids the node environment.

Smart Contracts

A smart contract in the CREDITS system is an electronic algorithm that describes a set of conditions by which actions and events in the real world or digital systems can be associated.

To implement self-controlled smart contracts, a decentralized environment that completely excludes the human factor is required, and to use the transfer of the cost in a smart contract, a cryptocurrency independent of the central authority is required.

Entities

The smart contract is developed using the interpreted JAVA language, which makes it possible to develop and conduct testing on absolutely any platform without having to install a special development environment.

To perform smart contracts, the Java Virtual Machine is used (hereinafter referred to as JVM supplied under a freely distributed license with the platform software).

A smart contract in CREDITS consists of the following entities:

1. Property (public variables) – the system entity storing public data necessary for work of the contract in the CREDITS system.
2. Method is the CREDITS system entity responsible for observing the logic and sequence of actions when conducting the transaction (actions under the contract).

Participants in the CREDITS system sign the smart contracts using the method call that modifies the contract properties, by launching the processes for verifying compliance with conditions and coordination. A smart contract comes into force after signing by the parties. To ensure automated fulfillment of obligations, an environment of existence is required, fully automating the execution of contract terms. This means that smart contracts can exist only within an environment that has unimpeded access of the executable code to the smart contract items.
All contract terms must have a mathematical description and clear logic of execution. Thus, the main principle of a smart contract is complete automation and reliability of contractual relations between the parties.

Smart contract method

The CREDITS smart contract method is the system entity responsible for compliance with the logic and sequence of actions during transaction (actions under the contract).

The logic and sequence of actions is described by a program code (module) containing commands; their sequential execution allows for obtaining the desired result. This code can handle system commands (for example, the assignment command), user commands (separately written functions), contract properties (statically or dynamically initialized variables available from any contract method), and methods from any other third-party contract available only to the owner of the connected (third-party) contract. For more popularization, the development is provided in JavaScript.
The method (program code) allows the use of all widely used scripting language operators (commands) (assignment, conditional and unconditional jumps), creation of functions and procedures (subroutines), and connection of third-party libraries.

Libraries for transfer transactions to the network, for use on third-party software

The protocol HTTP/HTTPS (work on 80/443 port) is used to work with third-party services for the sending of information about the operation (transaction) from the network. The system uses the standard protocol definition: the header and the body of the request. Any request is sent using the POST method, which makes it possible to increase the reliability of the transmitted data, than in case of public transmission through the GET method.

Additional conditions for identification of the sender include indication in the field of:

Authorization = [ACCOUNT_HASH]

The request body is formed in accordance with the output parameters of the services.

Libraries developed in C++ or JAVA languages are used to work with CREDITS peer-to-peer network.

Functions supported by libraries are divided into four groups.

1. System-wide, providing connection and support of connection to the network, checking the operability and filling fields in the system requests sent to the network, converting data in accordance with the algorithms required by the system;
2. Working with the account. Creating and unblocking the account, providing account information (balance, number of transactions, etc.);
3. Work on formation of transactions sent on behalf of the account;
4. Functions for working with public data. Request for common transaction data, account data.