The CryptoDbSS Blockchain Core
Technical Concept Paper

Steeven Salazar
Steevenjavier@gmail.com

Preamble

The concept of digital currency has evolved significantly since its inception. David Chaum introduced the idea of digital cash in the 1980s, emphasizing privacy and security through cryptographic techniques. In 1982, he proposed a blockchain-like protocol in his dissertation titled "Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups," laying the groundwork for future developments in digital currencies. In 1998, Wei Dai advanced the concept of electronic cash by proposing "b-money," a decentralized digital currency. These innovations set the stage for the emergence of blockchain technology, which provides a secure, transparent, and decentralized framework for digital transactions.

Blockchain technology came to light in 2009 with Bitcoin, the first cryptocurrency created by the anonymous developer Satoshi Nakamoto, allowing transactions to be made in a decentralized manner without the need for an intermediary. Since its launch, Bitcoin and other cryptocurrencies have gained increasing attention and adoption globally, impacting finance, supply chain, and healthcare. In 2013, the value of Bitcoin surpassed that of an ounce of gold for the first time, generating significant interest from the public, governments, and central banks.

blockchain creates a shared, immutable record of transactions among network nodes, preventing fraud and unauthorized actions. Through end-to-end encryption and access permissions, blockchain effectively addresses privacy and information security concerns, By recording all transactions in a distributed manner, complete visibility of the history of an asset or process is provided to all authorized participants. This allows for greater transparency and traceability, which is especially useful in industries with counterfeiting issues or where consumers are concerned about environmental and human rights issues. It also automates and optimizes processes that traditionally required a lot of paperwork and third-party intervention. This translates into faster and more efficient transactions, as well as significant cost savings for organizations. By providing more accurate and complete information in real time, blockchain allows entities to make more informed business and investment decisions. This is because all participants have access to the same up-to-date data, reducing information asymmetries.

Emerging as one of the most significant innovations of recent decades, blockchain's transformative potential extends beyond finance. While challenges remain, its future development prospects are encouraging. In finance, blockchain can create more efficient and secure payment, transfer, and loan systems by eliminating traditional intermediaries, thereby increasing access to financial services for underserved populations and enhancing global economic inclusion.

The "CryptoDbSS Blockchain Core" builds on these foundational concepts, aiming to address existing limitations in scalability, security, and efficiency within current blockchain frameworks. By incorporating advanced consensus mechanisms and innovative data structures, CryptoDbSS seeks to provide a robust solution for the next generation of decentralized applications.

Steeven Salazar
Steevenjavier@gmail.com

The CryptoDbSS Blockchain Core

Technical Concept Paper

CryptoDbSS

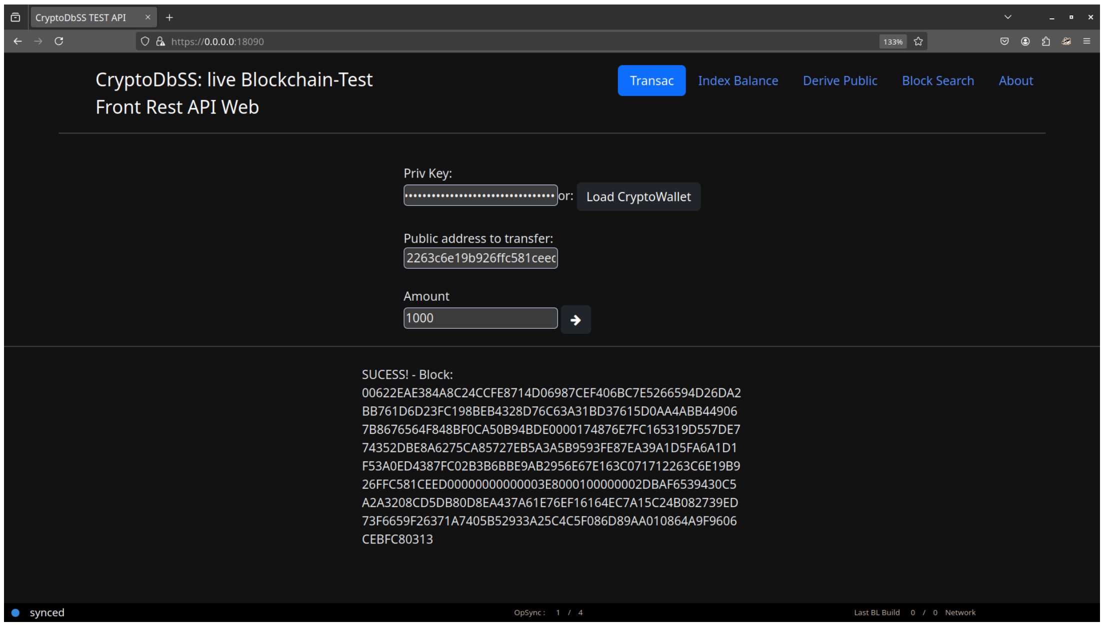
CryptoDbSS is a comprehensive framework that consists of software and protocols based on blockchain technology and cryptography. It serves as an engine for executing blockchains, decentralized applications (DApps), accounting ledgers, and asset tracing. The architecture incorporates robust security algorithms, high execution performance, and scalability, positioning it as an alternative to centralized and inefficient asset management systems. By facilitating the exchange of value and liquidity among individuals, as well as public, private, and mixed entities, CryptoDbSS aims to enhance operational efficiency and trust.

Since its development began in July 2023, CryptoDbSS has focused on core principles: security, efficiency, usability, scalability, and transparency. As of January 2025, it includes innovative features in blockchain architecture design:

- Indexing and Search Algorithms: Enhancing data retrieval and management.
- Transaction Compression: Reducing transaction sizes for improved efficiency.
- High Throughput: Supporting a large number of transactions per block with asynchronous processing.
- Logical Verification and Integrity: Ensuring data accuracy and reliability within the database.
- Unique Consensus Algorithm: Providing a secure and efficient method for achieving agreement among network participants.
-

These features position CryptoDbSS as a blockchain schema with significantly enhanced characteristics, contributing to its unique value in the landscape of blockchain-based software.

This document addresses all these aspects in more depth and in a descriptive manner, and others of important mention, explained in language that does not fall into so much technicality so that it can be understood concisely by enthusiasts, developers, investors (etc.) and so that a practical idea of the concept of operation, consensus and security is had.



CryptoDbSS: live Blockchain-Test
Front Rest API Web

Priv Key: •••••••••••••••••••••••••••••••••• or:

Public address to transfer: 2263c6e19b926ffc581ceec

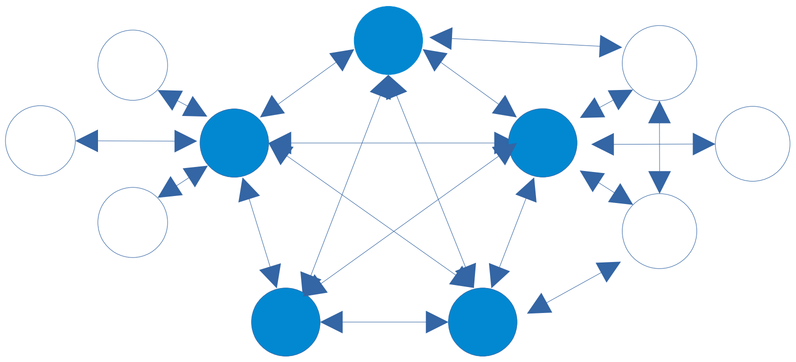
Amount: 1000

```
SUCESS! - Block:
00622EAE384A8C24CCFE8714D06987CEF406BC7E5266594D26DA2
BB761D6D23FC198BEB4328D76C63A31BD37615D0AA4ABB44906
7B8676564F848BF0CA50B94BDE0000174876E7FC165319D557DE7
74352DBE8A6275CA85727EB5A3A5B9593FE87EA39A1D5FA6A1D1
F53A0ED4387FC02B3B6BBE9AB2956E67E163C071712263C6E19B9
26FFC581CEED0000000000003E800010000002DBAF6539430C5
A2A3208CD5DB80D8EA437A61E76EF16164EC7A15C24B082739ED
73F6659F26371A7405B52933A25C4C5F086D89AA010864A9F9606
CEBFC80313
```

synced   OpSync : 1 / 4   Last BL Build 0 / 0 Network

Steeven Salazar
Steevenjavier@gmail.com

Consensus: MatchMin



A fundamental aspect in the blockchain area, if not the most important, is the consensus of the network of participants. It constitutes the basis of what is permissive and what is not in the network, which must be taken into account for the correct verification of the data, and how to act accordingly following a logic and algorithm through a common agreement between the participants.

The Matchmin scheme is the following: the network is made up of 2 types of participants, Validators and Clients, each node is identified with a unique ID that the rest of the participants know, this is a cryptographic public key which it itself derived before joining the network and who only knows and uses its private key.

Further to recognizing each other within the network, the key is used to sign and corroborate the data that is sent between nodes and certify the authenticity of the sender. If someone tried to falsify data in advance, they would not be able to prove its authenticity, due to the requirement of the signature of the author node as a security condition in addition to other parameters that the signature message must contain.

Validators are responsible for corroborating whether the information provided by participants is correct and permissible within the network, decided by the majority following a common criterion. They also decide, based on a pseudo-random selection, who among them will build the next block of transactions/operations, in addition to obtaining commissions as an incentive for processing and adding the transactions or operations to the new block. When the Validator builder has filled the maximum transaction space or after a time limit has elapsed or he does not wish to process more transactions, the integrity of the block will continue to be verified in the network with the counterpart of the rest of the participants, expecting at least 51% affirmation, in order to guarantee that the data has been correctly propagated in the network. This verification, being true, marks the block as validated and immutable.

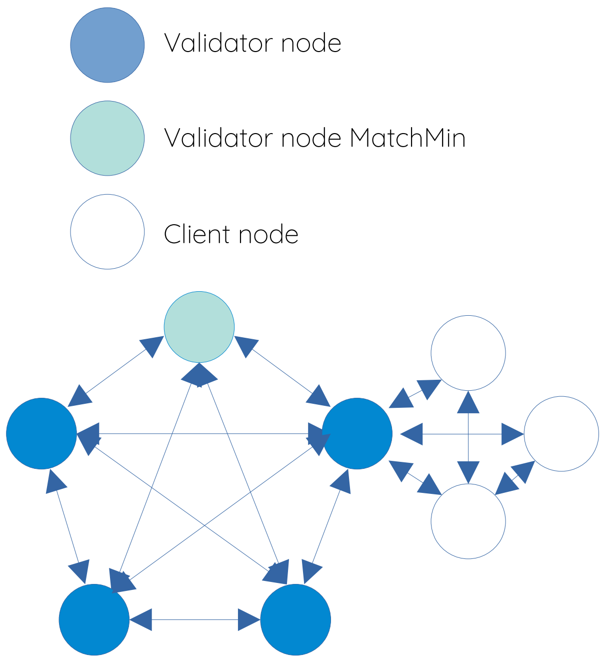
If the validator wishes, he can serve the network publicly without registering a client, providing more decentralization in the network. These parameters can vary depending on the context in the configuration with which the blockchain is to be operated (public, mixed or private).

The Client can make requests for processing new transactions to the validators, which are subsequently signed and verified after having verified that the status of the account in transaction is valid for said operation. He can also request resources in the network and its states such as the indexing of accounts, transactions and general information supplied through the network.

Steeven Salazar
Steevenjavier@gmail.com

Pseudorandom choice algorithm

To start building a new block of transactions, First it is necessary to start a round from a list-queue of all Validators that participating in the blocks construction , they'll follow this order having only one construction turn per round, whoever's turn it is will the queue of the new transactions with rest of the network that make up the new block and reward the charge commission for it. This list is built in a pseudo-random way as follows:



1) Each Validator node must choose a random number that only it knows. We will call this number "Antecedent." This antecedent will have a Consequent, which is the result or product of implementing a function that makes it complex and very difficult to recover its antecedent (such as the SHA function). Each node will share its Consequent on the network without first revealing its Antecedent.

2) When all the validator nodes have contributed the product of their function with each participant, they proceed to make their antecedent public. It must also be verified that the one contributed by the others is correct.

3) After this, a hash is derived from a chain that contains all the antecedents together with the hash of the last block, this derived hash must be interpreted as an integer.

4) the list is built in such a way that the node whose hash of its public address is algebraically closest to the hash derived by the algorithm will be placed next on the list in ascending order and so on with the rest, in this order it is decided who and how will build the following blocks and reward fees.

This in turn could be integrated with a PoS scheme or similar in the registry of validator nodes at a network with publics settings.

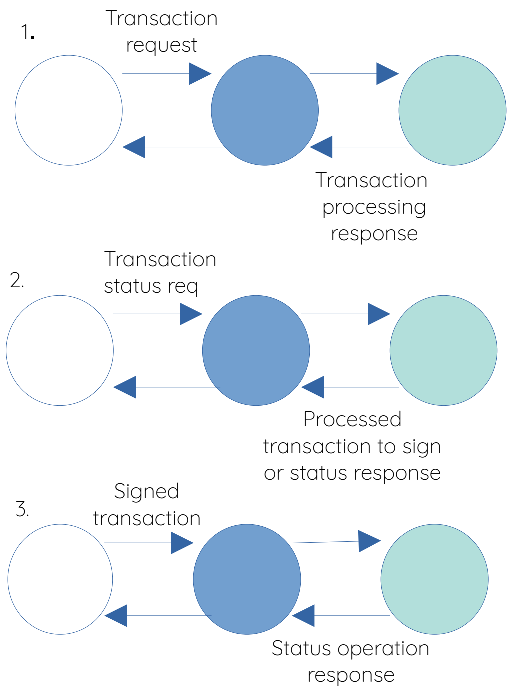
In terms of attack mitigation, trying to manipulate the list to alter the order to an advantage would not make sense since the malicious node would have to know all the antecedents beforehand. In a network where the majority are trustworthy, this should not happen and would be sufficient. Otherwise, it could try to crack the SHA, would require a very high computing force that ends up mitigating the profitability of an attack. However, let's suppose that in the remotely hypothetical case that this happened, the structure of the scheme is to follow a previously assembled queue, once its turn ends, it must wait for the others to build the block that is theirs. This reason makes no sense to attempt fraudulent actions in this area to gain an advantage in mining.

This is what Matchmin relies, this algorithm proves not to require the computing power used in consensus such as PoW (bitcoin), having as a characteristic to be efficient, and continuing to maintain security and transparency in the network allowing everyone to participate equally, without requiring large investments in energy infrastructure or hardware in this area.

Steeven Salazar
Steevenjavier@gmail.com

Synchronizing transactions across the network

Once the round has started, the node proceeds to listen to the requests to write the new transactions that make up the new block. For a transaction to be valid and processed, the algorithm consists of the following steps:



1.
Transaction request
Transaction processing response
2.
Transaction status req
Processed transaction to sign or status response
3.
Signed transaction
Status operation response

1. Request for Transaction from Client to Validator:

A Client or Validator requests a space for the new transaction by passing the basic information of this, provides the address of the issuing and receiving account, the amount to be transacted as well as a cryptographic signature of this data together with the hash of the last block, then the accounts are indexed and verified that the issuing has sufficient balance to carry out the transaction (including commissions). In the construction of this, the metadata of the block under construction are added as well as a number assigned in increasing sequential order of the spaces available for transactions (0-65500), this makes each transaction only valid if it is in the block and its assigned space number.

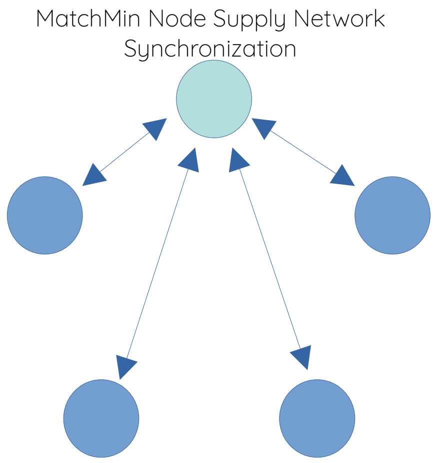
2. Transaction Status Check:

While indexing the accounts to build the transaction, the client must wait for it. The status of the transaction will be requested periodically and when the transaction is ready, the client node receives the data, verifies whether its data is correct and proceeds to sign it and obtain its subsequent validation.

3. Send Signed transaction:

If the Validator considers that all the data is valid, then it adds the new transaction to its assigned space (nonce) to be hosted in the new block under construction.

Synchronize Transactions Across the Entire Network in Real Time



Periodically the validators will ask to the Matchmin node for the queue transactions to validate them and have synchronization in the network, so when a new transaction is requested the rest will also be processing it, and they should not wait for the block to be built to check it, this means that all together they will build the block while verifying the validity of the transactions, the queue of transactions has a sequential order according to its transaction number, first the first available space is assigned (between 1-65500), this does not yet have its cryptographic signature but the node verifies that the data of this is correct to later receive the signature of the issuing account, when this occurs the transaction is considered processed and validated successfully in the network and will be added to the new block, otherwise if the Matchmin does not receive the signature or the data is invalid for some reason it will report it in the queue in its assigned space and it will be discarded to be added to the block, and the new account states will not be applied in the transaction

Construction and validation of new blocks

Upon having the minimum number of transactions confirmed by the node (varies depending on the node parameter <65000) or the maximum allowed by the algorithm (the maximum is constant 65000) the Validators proceed to build the new block with the metadata belonging to it along with the confirmed transactions, then, the hash is verified on the network for validation between the participants with a minimum of 51% of approve, if this happens the new block is permanently added to the blockchain and be immutably, if not, the block will be considered invalid for some reason (incorrect metadata, data corruption, fraudulent activity, etc.) after one or another of the scenarios occurs, the next chosen Matchmin node is proceeded with the next and the process is repeated again.

DB Architecture and Storage

CryptoDbSS transactions have the following format in their structure, each data is written in sequence and is represented by a byte string:

Transaction Type ID	1 byte
Issuing Address	64 bytes
Amount of Issuing Address	8 bytes
Receiving Address	64 bytes
Amount of receiving Address	8 bytes
Transaction Number ID	2 bytes
Feed	4 bytes
Signature	64 bytes
ID Blockchain	32 bytes
SHA of Node Matchmin	32 bytes
Block Header	32 bytes
Total	311 bytes

the total length of this is reduced by compression algorithms based on redundancy elimination.

the first byte of each transaction defines the type of transaction, more in explanation, is how the data of this will be interpreted, this in practice serves for scalability and to be able to provide new features in the future, a clear example is the asynchronous processing between transactions characteristic of CryptoDbSS, in the DB scheme this also defines the type of compression. This space can be used to add new types of transactions, or others cryptographic signature schemes, ballots, patches, interoperability with EVM, etc.

Asynchronous Account Statement Definitions for Transactions

if an account is being transacted with its status defined but not yet confirmed and another transaction is attempted to be made that has the same account, it should be waited for it to be confirmed to proceed with the next one in the case of defining its status, since if the first one were to fail and the second is validated it would cause an error between the sums of the previous statuses of the account, therefore this should be synchronous until the transaction is processed and confirmed; in order to break this blockage, It becomes asynchronous by not defining the status of the transacting accounts in the transactions after writing pending confirmation, this works because instead of placing the status of the accounts in the transaction, the amount of the operation to be transacted is placed, having previously verified that the account has the balance to carry it out, also debiting the transactions to be confirmed. When a new transaction is requested to be processed, 4 scenarios can occur that will define the transaction (0x00, 0x04, 0x06, 0x08) this definition will tell how the data will be interpreted, below is the linear structure that all transactions have, but on this occasion we will only focus on the first 5 elements that compose it, which are essential in this explanation:

transaction type	Acc issuing	value	Acc receiving	value

Steeven Salazar
Steevenjavier@gmail.com

Having the scheme focused, let's show how the data changes in them depending on the scenario that the transaction should take.

When no transaction account is being previously operated, the first element transaction type is defined with the byte 0x00, and the values are defined with the states of the accounts e.g. (Bob has a balance of 5000, and alice has 2500, Bob sends 500 to Alice, the structure is as follows:

0x00	Bob	4500	Alice	3000

When one of the accounts is being operated and has not yet been confirmed, its status cannot be defined for the reasons mentioned above, so its value field, instead of being defined by its status, is defined by the value that the transaction takes (in this example and the following ones, the values used in the previous one will be taken)

This time Bob, has a pending transaction, and Alice has none, Bob's value is defined by the amount of the operation (500), and Alice's value is defined by the state of her account added to the amount that is credited to her (previously 2500, now 3000), the transaction type is defined with 0x06 and the structure is the following:

0x06	Bob	500	Alice	3000

Bob does not have any pending transactions but Alice does, Bob's value is defined by the status of his account (previously 5000, now 4500), and Alice's value is defined by the amount of the operation, the transaction type is defined with 0x08 and the structure is the following:

0x08	Bob	4500	Alice	500

When both have pending transactions, both values are defined by the amount of the operation. This case is defined as 0x04 and the structure is:

0x04	Bob	500	Alice	500

Structure of a CryptoDbSS block

Element	size
Block header	32 bytes
Block Type ID	1 byte
Blockchain network ID	32 bytes
SHA of Node Matchmin	32 bytes
Address to feed	64 bytes
New value of address to feed	8 bytes
Block number	8 bytes
Qtty of transactions	2 bytes
Tansacs content	may vary
Space check dumper	2 bytes
Uncompressed integrity data SHA	32 bytes

Steeven Salazar
Steevenjavier@gmail.com

Compression of Transaction Data

When writing extensively in a database, it is essential to employ algorithms that reduce operational costs, hosting requirements, and overall data size without compromising data integrity. In CryptoDbSS, processing a transaction with its signature occupies 311 bytes, of which 96 bytes are metadata from the block header. By eliminating this metadata, the transaction size is reduced to 215 bytes, even before applying further compression.

To assess the weight of a block containing the maximum number of confirmed transactions, we calculate as follows:

(215 bytes×65,500 transactions)+(213 bytes of block metadata)=14.1MB

This calculation reflects the size of the block without compression

While this may seem substantial, it is important to note that many blockchain systems do not achieve a throughput of 65,500 transactions. For example, Bitcoin typically processes transactions averaging 250 bytes. A standard Bitcoin block, constrained by its 1 MB limit, can accommodate approximately 4,000 transactions.

Note: The data presented and its precision may change in the future as technologies and methodologies evolve.

In Ethereum, the transaction capacity is influenced by its gas system, which measures the computational costs. On average, a block can handle about 30 million gas, with typical usage around 15 million. Each transaction consumes approximately 21,000 gas, resulting in an average of about 1,428 transactions per block, totaling around 1.77 MB—this includes the implementation of the Snappy compression algorithm.

By applying the highest level of compression available in CryptoDbSS, which is 83 bytes per transaction, we can estimate the following for a Bitcoin block:

83 bytes x 4,000 transactions = 332 KB

This represents an approximate 62% reduction in data size. Such optimization significantly alters the landscape of blockchain data management, showcasing the efficiency that CryptoDbSS contributes to the field.

compression levels

A transaction can have several levels of compression, these are defined by 3 conditionals:

1. how many blocks ago the transacting accounts performed an operation (if it is in a range < 65500 previous to the indexing prior to this one and decompressed).

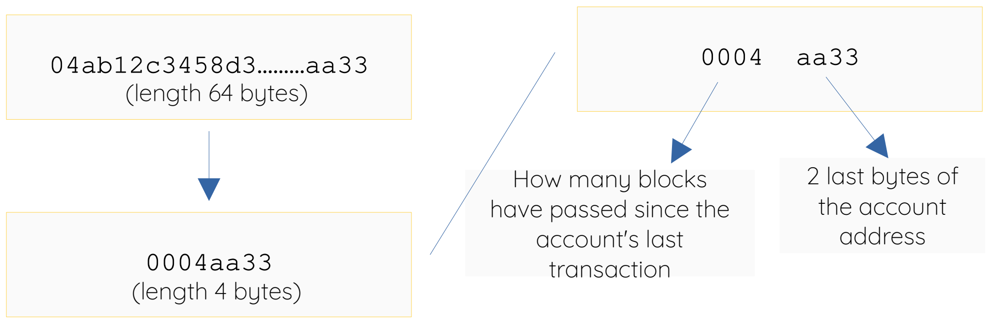
2. the amount of the accounts

3. If the transaction is defined as type 0x04, (amounts only).

In case 1, each account is verified. If one meets the condition, it is compressed by reducing its size since 64 to only 4 Bytes, this leaving only the last 2 bytes of its public address and adding it to the beginning as many blocks ago as the current indexed one has been. The following diagram will serve as an illustration.

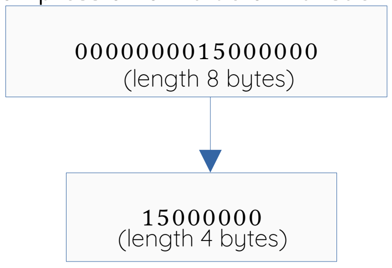
Steeven Salazar
Steevenjavier@gmail.com

Compressing account addresses in transactions


04ab12c3458d3.........aa33
(length 64 bytes)
0004aa33
(length 4 bytes)
0004 aa33
How many blocks have passed since the account's last transaction
2 last bytes of the account address

For case 2, the value associated with each account is taken and if it is less than 4,294,967,295, CryptoDbSS manages the account values internally with a memory space of 8 bytes. If the conditional is met, the figure can be represented with 4 bytes, doubling the size it occupies

Compression of value of transactions



For case 3, the transaction must be processed as type 0x04, which means that the transaction's statuses are not defined, but only the transaction amount is provided in the digits space. This only says how much is subtracted from the transient account and how much is added to the transacted account, so one of the digits spaces is eliminated when it is saved in the block to reduce its size.

The next table shows all the compression levels for each type of transaction, L stands for issuing account and R for receiving account, The transaction types are 0x00, 0x04, 0x06, 0x08 and the items below them are the compression type applicable when the condition is met, byte length tells the final size of the compressed transaction.

type of transaction	0x00	0x04	0x06	0x08	byte length
compressed element	Compress ID				
L acc	0x0B	0x1A	0x29	0x38	155-147
R acc	0x0C	0x1B	0x2A	0x39	155-147
L Value	0x0D	0x1C	0x2B	0x3A	211-203
R value	0x0E		0x2C	0x3B	211
L & R Value	0x0F		0x2D	0x3C	207
L Acc & L Value	0x10	0x1F	0x2E	0x3D	151-143
L Acc &R Value	0x11		0x2F	0x3E	151
L Acc & L&R value	0x12		0x30	0x3F	147
R Acc & L Value	0x13	0x22	0x31	0x40	151-143
R Acc & R Value	0x14		0x32	0x41	151
R Acc & L&R Value	0x15		0x33	0x42	147
L&R Acc	0x16	0x25	0x34	0x43	95-87
L&R Acc & L value	0X17	0X26	0X35	0X44	91-83
L&R Acc & R Value	0X18		0X36	0X45	91
L&R Acc & L&R Value	0X19		0X37	0X46	87

Steeven Salazar
Steevenjavier@gmail.com

The CryptoDbSS Blockchain Core

Technical Concept Paper

S. S.

CryptoDbSS is a set of software and protocols based on blockchain and cryptography that serves as an engine for the execution of blockchains, accounting books and asset tracing, it has its own architectural schemes implementing robust, efficient and scalable security algorithms.

The code is written in C++, at a medium and low level of abstraction in elements,
The database is managed directly with byte arrays, this makes it have a powerful performance in indexing, processing and querying transactions and states.

The source code of the architecture, consensus, algorithms and structure of the database, logic and network infrastructure were developed by Steeven J. Salazar.

Steeven Salazar
Steevenjavier@gmail.com

References

1. Chaum, David Blind Signatures for Untraceable Payments.
2. Chaum, David Computer Systems Established, Maintained, and Trusted by Mutually Suspicious Groups.
3. Wei, Dai b-money
4. Wikipedia contributors. Blockchain.
5 .Wikipedia contributors. Elliptic-curve cryptography