



# D-CHAIN

A COMMUNITY-DRIVEN OPEN-SOURCE  
BLOCKCHAIN PROTOCOL


# TABLE OF CONTENTS

- ..... 05
- ..... 05
- ..... 06
- ..... 07
- ..... 09
- ..... 11
- ..... 12
- ..... 14
- ..... 16
- ..... 17
- ..... 18
- ..... 19
- ..... 20
- ..... 21
- ..... 23
- ..... 25
- ..... 26
- ..... 28
- ..... 31
- ..... 32
- ..... 33

[D-Chain](#) is a decentralized, Open-source blockchain with a virtual machine that supports small, medium, and large-scale enterprises to implement their smart contracts. The coin of the D-chain is called Decos or D-Coin and the ticker for the same is called DCX. For decades, the concept of decentralized digital currencies, and other applications such as property registries, has been well-known. In the 1980s and 1990s, anonymous e-cash protocols relied on a cryptographic primitive called Chaumian blinding to provide a highly private currency. However, the protocols failed to gain traction due to their dependence on a central intermediary. Wei Dai's B-money was the first to propose creating money by solving computational puzzles. However, the proposal did not provide details about how this could be done. Hal Finney presented a concept called reusable Proofs of Work in 2005. This system combines ideas from b money with Adam Back's computationally challenging Hashcash puzzles to create an idea for a cryptocurrency. However, it again failed to meet the ideal because it relied on trusted computing as its backend. Satoshi Nakamoto introduced a decentralized currency in 2009 for the first time. This combination of established primitives for managing ownership via public key cryptography and a consensus algorithm to keep track of who has coins is known as "proof-of-work."

The screenshot displays the D-Wallet interface. On the left, there is a logo for 'wallet' and a heading 'D-Wallet Compatible Chains'. Below this, a sub-heading reads 'Helping users to install EVM powered Blockchain Networks'. A paragraph explains that D-Wallet supports a list of EVM networks and allows users to add them in one click. At the bottom left, there are two buttons: 'Add Your Network +' and 'Add Your RPC +'. The main part of the interface is a grid of network cards. Each card includes a search bar at the top with 'Search Networks ETH, Fantom, ...', a 'Testnets' toggle, and a 'Connect Wallet' button. The grid contains nine cards, each representing a different blockchain network with its ChainID and Currency.

Network Name	ChainID	Currency
Ethereum Mainnet	1	ETH
Binance Smart Chain ...	56	BNB
Avalanche C-Chain	43114	AVAX
Polygon Mainnet	137	MATIC
Fantom Opera	250	FTM
Cronos Mainnet Beta	25	CRO
Arbitrum One	42161	ETH
Aurora Mainnet	1313161554	ETH
Klaytn Mainnet Cypress	8217	KLAY



Because it simultaneously solved two problems, proof-of-work was a significant breakthrough in space. It provided a simple but moderately efficient consensus algorithm that allowed nodes to agree on a set canonical update to Bitcoin's Ledger. It also allowed for free entry to the consensus process. This solved the political problem of who could influence it, and prevented Sybil attacks. This is done by replacing a formal barrier to participation (such as the requirement to register as a unique entity to a list) with an economic one. The weight of a single vote in the consensus voting process will directly correlate to the computing power the node brings. An alternative approach, proof-of-stake, has been developed. It calculates the node's weight based on its currency holdings, not its computational resources. While each approach's merits are beyond this paper's scope, it is essential to note that both can be used as the backbone of a cryptocurrency.

Ethereum launched an alternative protocol to build decentralized applications. They believed this would be helpful for many classes of decentralized apps. Ethereum achieved this by creating the ultimate abstract foundational layer, a blockchain with a Turing-complete programming language. This allowed anyone to create smart contracts and decentralized apps where anyone could set their own rules for ownership, transaction formats, and state transition functions. Namecoin's bare-bones version can be coded in just two lines, while other protocols such as currencies and reputation systems like currencies can be created in less than twenty. Smart contracts, which are cryptographic "boxes" that have value and only unlock if certain conditions are met, were built on top of a blockchain. They became the first one to launch virtual machine on top of a blockchain, but failed to save energy as it launched Proof of Work consensus mechanism. Miners had spent more energy in the mining farms for mining Ethereum.

Bitcoin mining alone creates more CO2 emissions than the whole of Argentina every year. Many Cryptos had launched after launch of bitcoin which contributed more on green house effects and climate change. Cryptos with Proof-of-Work consensus contributes more to climate change and negative effect on environment. Cost for the Electric Consumption went difficult to users due to highly volatile market. Although blockchains need to be validated, there are new validation methods that offer comparable security levels through alternative methods of verification.

# PROOF OF STAKE: THIS IS A PROCESS THAT VALIDATES CRYPTO TRANSACTIONS BY STAKING



To validate their transactions, miners use their cryptocurrency to access mining rights proportional to the number of coins they own. To create a validator, they lock away their coins. This node can verify transactions. The blockchain selects a random validator to approve a block of information. The validator can verify the block and add it to the blockchain. They will lose some coins if they add an incorrect block. The most return is earned by those who have the most coins. D-Chain decided to use Proof-of-Stake (PoS),

a consensus mechanism. PoS is a mechanism that allows nodes to commit token "stakes" for a some time in exchange for the chance of being selected to produce the next block. The block rewards will be given to the "validator", the node that is chosen. This is the native token of PoS.

## D-CHAIN — CONSENSUS ALGORITHM:

We must first define Proof-of-Stake to answer the question, "What is Proof-of-Stak?".





D-Chain allows decentralized distributed ledgers of transactions. There is no central server that controls the network. Therefore, everyone must agree on what transactions are valid. Fake transactions would be possible, otherwise it would not be possible to make them.

Nodes are the servers that make up a blockchain. Nodes are responsible for processing transactions. Some nodes can add blocks to the chain of transactions, maintaining and growing its ledger. These nodes are called "miners" in Proof-of-Work (PoW), networks such as Bitcoin.

PoS is when nodes contribute funds to the network. This process is known as "Staking."

D-Chain Nodes compete with one another to become the next block writers. Nodes that add blocks to PoS networks are known as "validators". These are people who verify transactions on a blockchain. Every validator is eligible to be selected to write the next block, and receive its rewards.

It works in a lottery-like fashion: the more significant the stake, the better the chance that a node will be chosen. "The pseudo-random selection of the next block writer or validator is determined by the amount of stake you, as the user, have given to the network."

## D-CHAIN ACCOUNTS

The state of the D-Chain is composed of objects called "accounts". Each account has a 20-byte address. State transitions are direct transfers of value between accounts. A D-Chain account has four fields.

- **THE NONCE IS A COUNTER THAT ENSURES EACH TRANSACTION CAN ONLY EVER BE PROCESSED ONCE**
- **THE CURRENT DECOS BALANCE OF THE ACCOUNT.**
- **IF PRESENT, THE CONTRACT CODE OF THE ACCOUNT.**
- **THE STORAGE ACCOUNT (EMPTY BY DEFAULT).**

"Decos", the main internal cryptocurrency-fuel of D-Chain and used to pay transaction fees, is called. There are two types: externally-owned accounts controlled by private keys and contract account controlled by contract codes. Externally owned accounts do not have codes. One can send messages to an externally owned one by signing transactions. Contract accounts activate their code every time they receive a message. This allows them to access internal storage, read other messages, and create contracts.

It is important to note that D-Chain "contracts" should not be viewed as something that must be "fulfilled" and "complied with". Instead, they should be recognized as "autonomous agents" who live within the D-Chain execution environment. They execute a specific code piece when "poked" via a message or transaction and have direct control over their Decos balance and key/value stores to track persistent variables.

## MESSAGES AND TRANSACTIONS

In D-Chain, "transaction" refers to the signed data package containing a message to send from an externally-owned account. Transactions include:

- **THE MESSAGE'S RECIPIENT**
- **SIGNATURE IDENTIFYING THE SENDER**
- **THE AMOUNT OF DECOS THAT THE SENDER WISHES TO TRANSFER TO THE RECIPIENT**
- **AN OPTIONAL DATA FIELD**
- **THE START GAS VALUE IS THE MAXIMUM NUMBER OF COMPUTATION STEPS A TRANSACTION CAN TAKE.**
- **THE GASPRICE VALUE IS THE AMOUNT THE SENDER PAYS FOR EACH COMPUTATIONAL STEP.**



These are the standard fields in all cryptocurrencies. Although the data field does not have a function by default, the virtual machine does have an opcode that allows a contract to access the data. For example, if a contract serves as an on-blockchain domain registry service, it might want to interpret the data it receives as having two fields. The first is the domain to register, and the second is the IP address to register it. These values would be read by the contract and stored in appropriate storage.

GAS PRICE and STAR TGAS are critical to D-Chain's anti-denial-of-service model. Each transaction must limit the number of computational steps it can use to avoid hostile infinite loops and other code wastage. Gas is the fundamental unit of computation.

A computational effort usually costs one gas. However, some operations can cost more gas due to their computational complexity or the increased amount of data they must store as part of the state. For every

transaction data byte, there is a fee of five gas. The fee system requires attackers to pay proportionately for all resources they use, including computation and bandwidth.

Therefore, any transaction that causes the network to consume more of these resources will be subject to a gas fee approximately proportional to the increase.

# MESSAGES

Contracts can send messages to other contracts. Messages can be described as virtual objects that cannot be serialized and only exist in the D-Chain execution environment. A message can contain the following::

- **THE MESSAGE SENDER (IMPLICIT)**
- **THE MESSAGE'S RECIPIENT**
- **THE MESSAGE AND THE AMOUNT OF DECOS THAT WILL BE TRANSFERRED**
- **AN OPTIONAL DATA FIELD**
- **A START GAS VALUE**



A message is a transaction, except that it is produced by a contract, not an outside actor. A message is created when code in a contract executes the CALL operator, which develops and implements a message. A message is similar to a transaction. It leads to the recipient account running its code. Contracts can also have connect with external actors in the same way as external actors.

The gas allowance assigned to a transaction or contract only applies to the total amount of gas consumed by the transaction and any sub-executions. If an external actor A sends B a transaction with 1000 gas B then sends C a message, which consumes 600 gas. C's internal execution consumes 300 gas before it returns, so B can still spend 100 gas until running out.

# D-CHAIN STATE

# TRANSITION FUNCTION

The D-Chain state transition function, is called  $APPLY(S.TX \rightarrow S')$ , which can be described as follows:

1. Verify that the transaction is formatted correctly (e.g., The transaction has the correct number of values, the signature is valid and the nonce matches that in the sender's account.) If not, it will return as an error.
2. Calculate the transaction fee in  $START\ GAS * GAS\ PRICE$  and calculate the sending address using the signature. Add the fee to the sender's account balance, and increase the sender's nonce. Return as an error if there isn't enough balance.
3. Initialize  $GAS = START\ GAS$  and deduct a certain amount of gas per byte to pay for the transactions.

4. Transfer the transaction value to the recipient from the sender account. If the receiving account is not yet created, it should be. If the receiving account contains a contract, you can run the contract's code until it is completed or ends.

5. If the value transfer fails because the sender does not have enough money or the code execution runs out of gas, you can reverse all state changes, except for the payment of fees, and add them to the miner's account.

6. If not, send all fees associated with gas consumption to the Validator and refund any gas fees to the sender.

**LET'S SAY, FOR EXAMPLE, THAT THE CODE OF A CONTRACT IS:**

**IF !SELF.STORAGE[CALLDATALOAD(0)]:**

**SELF.STORAGE[CALLDATALOAD(0)] = CALLDATALOAD(32)**

The contract code in reality, is written in DVM code. This example was written in Serpent, one of our high-level languages. It can be compiled into DVM code. Let's say that the contract's storage starts empty. A transaction is sent with 10 Decos value, 2000 Gas, 0.001 Decos Gas Prices, and 64 Bytes of Data. Bytes 0-31 represent the number 2, and 32-63 the string CHARLIE. In this example, the state transition function is performed as follows:

1. Verify that the transaction is legal and properly formed.
2. Verify that the transaction's sender has at least  $2000 * 0.01 = 2$  Decos. Add 2 Decos to the account of the sender if it is.
3. Initialize gas = 2000. Assuming the transaction is 170 bytes in length and the byte fee is 5, subtract 850 to get 1150 gas.
4. Add ten more Decos to the account of the sender and subtract them from the account of the contract.
5. Run the code. This code will check if the index 2 storage is available for the contract. If it does not, it will set the index 2 storage to the value CHARLIE. This will take 187 gas. The remaining gas will be 1150.
6. Add  $963 * 0.01 = 0.963$  to the sender's account and return the result.

If the transaction was not completed by a contract, the total transaction fee would equal the GAS PRICE multiplied with the length of the transaction. The data that was sent along by the transaction would then be insignificant.

In terms of reverts, messages are equivalent to transactions: If a message execution runs low on gas, then all executions will trigger from that execution revert. Parent executions don't need to be reversed. It is safe for a contract to call another contract. For example, if A calls B with gas, then A's execution will lose only G gas. Last but not least, there is an opcode called CREATE that creates a contract. Its' execution mechanics are similar to CALL, except that the output of execution determines the code for a newly crea-

# CODE EXECUTION

The code used in D-Chain contracts is written using a low-level stack-based bytecode language. Also known as "D-Chain Virtual Machine Code" or "DVM Code," the code is written in an "D-Chain virtual code" (or "DVM code") format. Each byte in the code represents an operation. Code execution is an endless loop. It consists of repeating an operation at the counter program count (which starts at zero) and then incrementing it by one until the end of the code or an error, STOP, and RETURN instruction is detected. Three types of storage space are available to the operations:

- **THE STACK IS A CONTAINER THAT LASTS IN FIRST OUT AND CAN HOLD VALUES.**
- **MEMORY IS AN INFINITELY EXPANDABLE ARRAY OF BYTE ARRAYS**
- **THE LONG-TERM STORAGE OF THE CONTRACT, WHICH IS A KEY/VALUE BANK, IS ALSO INCLUDED. STORAGE IS UNLIKE MEMORY AND STACK, WHICH ARE RESET AFTER COMPUTATION ENDS.**

Storage is unlike memory and stack, which are reset after computation ends.

The code can access the sender, value and data of the incoming messages as well as the block header data. It can also return a bytes array of data as an output.

It is simple to create an DVM code formal execution model. The tuple is the block\_state, transaction message, code, memory and the stack of the D-Chain virtual machine. Block\_state contains all accounts, balances, storage, Moreover, it is also used to define the entire computational state. The current instruction is determined at the beginning of each round of the execution by finding the PCth byte of code (or 0, if  $pc \geq \text{len}(\text{code})$ .) Each instruction has its definition of how it affects tuple. ADD, for example, removes two items from the stack, decreases gas by 1, increments pc 1, and causes their sum. and storage. on the other hand, pushes the top two stack items off the stack and inserts the second item in the contract's storage at index 1. A basic implementation of the D-Chain can be accomplished in just a few hundred lines.

## BLOCKCHAIN AND MINING

Although the D-Chain blockchain is very similar to Bitcoin it has some differences D-Chain is different from Bitcoin in terms of blockchain architecture. D-Chain blocks have a copy both of the transaction list as well as the most recent state. The block also stores two additional values, the block number (and the difficulty) This is the basic block validation algorithm for D-Chain:

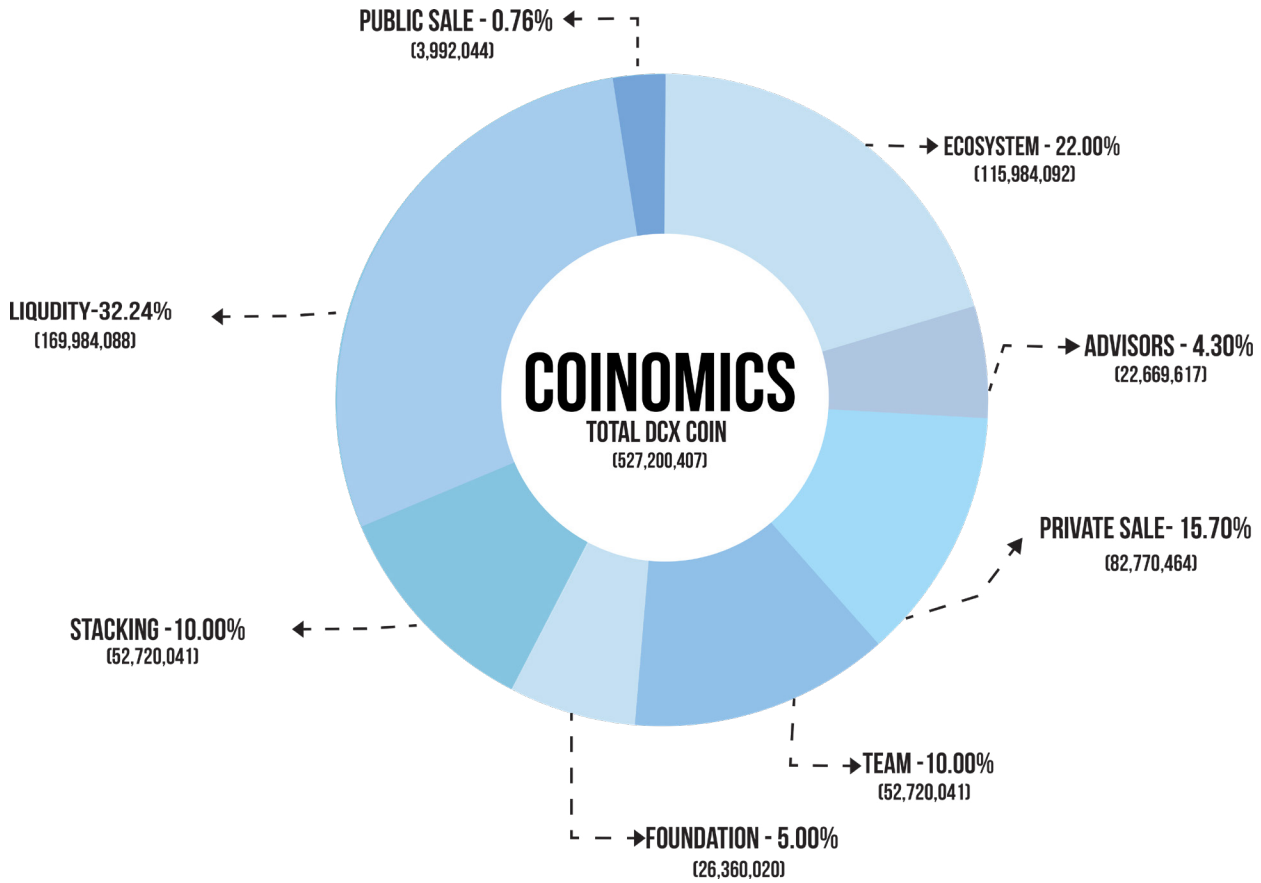
- **VERIFY THAT THE BLOCK YOU ARE REFERRING TO IS STILL VALID.**
- **ENSURE THAT THE TIMESTAMP FOR THE CURRENT BLOCK IS NOT GREATER THAN THE REFERENCED BLOCK'S AND THAT IT IS LESS THAN 15 MINUTES IN THE FUTURE.**
- **VERIFY THAT BLOCK NUMBER, DIFFICULTY, AND TRANSACTION ROOT ARE ALL VALID.**
- **VERIFY THAT THE PROOF OF WORK ON THE BLOCK IS VALID.**

- LET  $S[0]$  REPRESENT THE STATE AT THE BLOCK'S END.
- LET  $TX$  BE THE BLOCK'S TRANSACTION LIST WITH  $N$  TRANSACTIONS. FOR ALL  $I$  IN  $0 \dots N-1$ , SET  $S[I+1] = \text{APPLY}(S[I], TX[I])$ . RETURN AN ERROR IF ANY APPLICATION RETURNS AN ERROR OR IF THE TOTAL AMOUNT OF GAS CONSUMED IN THE BLOCK UNTIL THAT POINT EXCEEDS THE GASLIMIT.
- LET  $S\_FINAL$  BE  $S[N]$ , WHILE ADDING THE BLOCK REWARD TO THE MINER.
- VERIFY THAT THE MERKLE TREE ROOT FOR THE STATE  $S\_FINAL$  EQUALS THE LAST STATE ROOT IN THE BLOCK HEADER. IF IT IS, THEN THE BLOCK IS VALID.

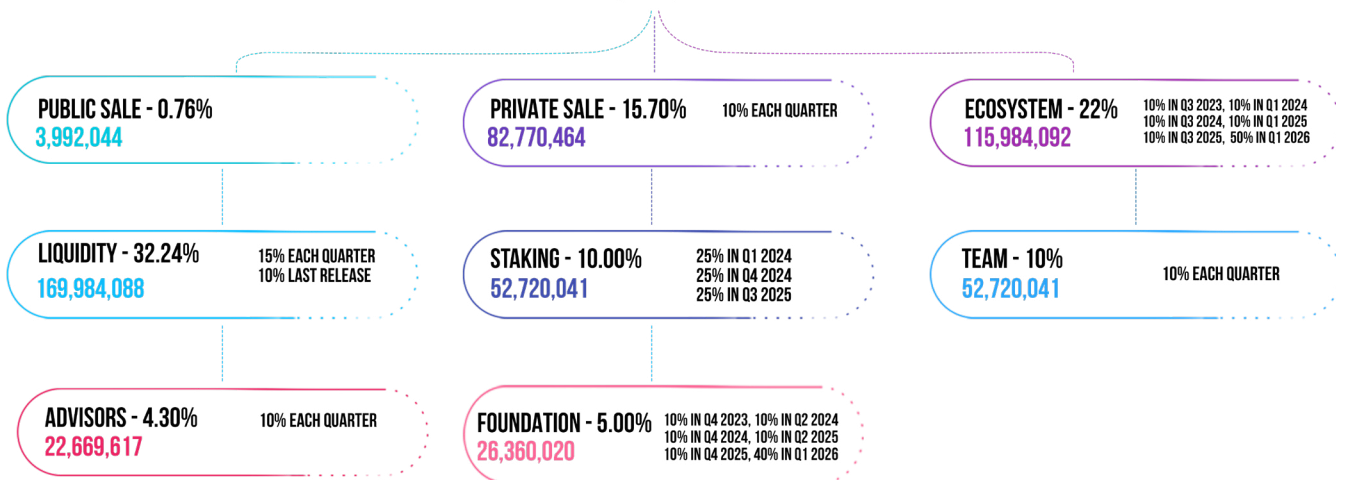
Although the approach might seem inefficient because it stores all of the state with every block, in reality, efficiency should be similar to Bitcoin. Because the state is stored in a tree structure, only a small portion of the tree must be modified after each block. The vast majority of the tree should look the same between adjacent blocks. Therefore, the data can be stored once, but referenced twice with pointers (e.g., Hashes of subtrees. This is possible with a special type of tree called a "Patricia Tree.""). It also incorporates a modified Merkle tree concept which allows nodes to be efficiently added and removed, not just modified. Because all state information is included in the last block, it is not necessary to store all blockchain history. This strategy, if applied to Bitcoin can result in a 5-20x reduction in space.

One common question is "Where" contract code is executed regarding physical hardware. The simple answer is that contract code execution is part of the state transition function definition. This block validates block B. If a transaction is added to block B, the code execution generated by that transaction will be executed now by all nodes that have downloaded and validated block B.

# COINOMICS AND VESTING ALLOCATION










## TOTAL DCX COIN 527,200,407



# APPLICATIONS

There are generally three types of applications that can be built on top of D-Chain. The first is financial apps, which provide users with powerful ways to manage and enter into contracts using their money. These include sub-currencies and financial derivatives as well as savings wallets, wills, hedging, savings, and even full-scale employment contracts. Semi-financial apps are those where money is involved, but there is also a heavy non-monetary side to what's being done. A perfect example of this is self-enforcing bounty for solving computational problems. There are also applications like online voting or decentralized governance, which are not financially related. D-Ecosystem is about to use this Chain for Launching Decentralised Ecosystem for Users.

## DETAILS OF PRODUCTS

 <b>D-WALLET</b>	 <b>SWAP &amp; FARM</b> (D-CHAIN, BSC, ETH, POLYGON)	 <b>DEX</b> D-EXCHANGE	
 <b>D-NFT</b> NFT MARKETPLACE	 <b>METAVVERSE</b>	 <b>D-MESSENGER</b>	
 <b>D-APP</b> DECENTRALIZED SERVICES		 <b>P2P</b> <b>D-P2P</b>	
 <b>DUSD</b> STABLECOIN BACKED BY XAUT		 <b>NETWORK BRIDGE</b>	
 <b>IPFS SERVICES</b>		 <b>IDO BRIDGE</b>	
 <b>LAUNCHPAD &amp; LAUNCHPOOL</b> (CROSS CHAIN IDO & ON ALL EVM CHAINS)		 <b>VALIDATOR TOOL OF D-CHAIN</b>	 <b>SPOT &amp; MARGIN</b>
 <b>FUTURES &amp; OPTION</b>		 <b>RESEARCH, ACADEMY &amp; CHARITY</b>	 <b>BETTING + LOTTERY</b>

# IDO DETAILS

PHASE -1

PHASE -2

PHASE -3

## TOKEN SYSTEMS

Many applications can be made of on-blockchain token systems. These include sub-currencies that represent assets like USD or gold, company stocks, individual tokens representing smart property, secure unforgeable coupon systems, and token systems without any ties to traditional value. In D-Chain, token systems are straightforward to implement. It is important to remember that a token system or currency is simply a database that performs one operation: subtract A units from A, and give B X units. However, A must have at least X units prior before the transaction. (2) The transaction has been approved by A. To implement a token system, you only need to put this logic into a contract.

This is the basic code to implement a token system within Serpent:

```
def send(to) value:
```

```
    if self.storage[msg.sender] >= value:
```

```
        self.storage[msg.sender] = self.storage[msg.sender] - value
```

```
        self.storage[to] = value
```



This is a literal implementation of the "banking system" state transition function, which we discussed in this document. To distribute the currency units, a few additional lines of code are required. Additionally, there are some edge cases. Ideally, a function is added that allows other contracts to query the address's balance. It is possible that D-Chain-based token systems that act as sub-currencies could include another feature that on-chain Bitcoin meta-currencies do not have: the ability to pay transaction fees directly in the currency. This would work by the contract maintaining a Decos balance, with which it would reimburse senders for fees paid. It would then replenish this balance by taking in fees and selling them in a continuous running auction. The contract would then refund the money each time it was refunded. Users would need to "activate" their accounts with Decos.

## FINANCIAL DERIVATIVES AND STABLE-VALUE CURRENCIES



Financial derivatives are one of the most popular uses of "smart contracts" and the easiest to implement in code. Implementing financial contracts presents a challenge because most need to reference an external price ticker. A smart contract that protects against volatility of Decos or another cryptocurrency with respect to the US Dollar is one example of this. However, the contract must know the USD/DCX value. This can be done by using a "data feed" agreement maintained by one party (e.g., NASDAQ) is designed to allow the party to update the contract at will. It also provides an interface that allows other contracts to send messages to the contract and receive a reply that includes the price.

This is the key ingredient of a hedging agreement:

- **WAIT FOR PARTY A, TO INPUT 1000 DECOS**
- **WAIT FOR PARTY B, TO INPUT 1000 DECOS**

In storage, record the USD value for 1000 Decos. This can be calculated by querying the data-feed contract.

After 30 days, A or B can "reactivate" the contract to send \$x worth of Decos to A and B.

## TOKENS STANDARDS

These are the most widely used token standards globally.



- DEC20: is a standard interface for interchangeable tokens such as voting tokens and virtual currencies.
- DEC 721: A standard interface for non-fungible tokens, such as a deed for artwork and a song.
- DEC777: - lets you add functionality to tokens, such as a mixer contract to improve transaction privacy or an emergency recovery function to help you out in case you lose your private keys.
- DEC 1155: allows you to trade more efficiently and bundle transactions, thereby saving money. This token standard permits the creation of utility tokens such as \$BNB and \$BAT, and non-fungible tokens such as CryptoPunks.
- DEC 46626- Tokenized vault standard that optimizes and unifies technical parameters for yield-bearing vaults

This contract could have great potential for crypto-commerce. The main problem with cryptocurrency is its volatility. While many people and merchants want to be able to deal with it securely and conveniently, they may not want to risk losing 23% of their funds within a single day. The most popular solution to this problem has been issuer-backed assets. This is where an issuer creates a currency in which they can issue and revoke units and provide one unit to anyone (offline) who provides them with one unit (eg., gold, USD. gold, USD). If the issuer receives one unit of crypto-asset, the issuer promises to give it back. This allows any non-cryptographic asset (subject to trust) to be "uplifted" into cryptographic

However, in practice, issuers may not always be trustworthy, and the banking infrastructure may be too weak or hostile to allow such services. Financial derivatives provide an alternative. Instead of one issuer funding an asset, there is a decentralized market made up of speculators. They bet that the cryptographic reference asset will rise in price. DCX plays this role. Because the hedging contract holds the funds in escrow, speculators cannot default, unlike issuers. This approach is not entirely decentralized because a trusted source still needs to provide the price ticker. However, this is an enormous improvement in infrastructure requirements (unlike being an issuer, issuing price feeds requires no licenses and can likely qualify as free speech) and reducing fraud potential.

## IDENTITY AND REPUTATION SYSTEMS

The first alternative cryptocurrency Namecoin tried to use a Bitcoin-like Blockchain to provide a name registration service where users could register their names in a public databank alongside other data. A DNS system that maps domain names such as "bitcoin.org" or, in Namecoin's case, "bitcoin.bit")

an IP address is the most popular use case. Email authentication is another use case, and as more advanced reputation systems. This is the basic contract that will provide a D-Chain based Namecoin like system for name registration.

```
def register (name, value)
  if !self.storage[name] is:
```

The contract is simple. It is simply a database within the D-Chain network that can be added or deleted. Anybody can register a name that has a value. The registration will remain indefinitely. An advanced name registration contract will have a function clause that allows other contracts to query it and a mechanism for the "owner" of the name (i.e., The first person to register a name can change data or transfer ownership. You can also add web-of-trust functionality and reputation.

## DECENTRALIZED STORAGE

There have been a lot of online file storage startups over the years. The most popular is Dropbox. They allow users to upload backups of their hard drives and store them on the server. Users can then access the backup for a monthly charge. The file storage market is still inefficient at this stage. A quick look at different solutions shows that at the 20-200GB level, where neither enterprise-level discounts nor free quotas kick in, the monthly cost for mainstream file storage costs can be more than the price of the entire hard disk in a single month. D-Chain contracts could create a decentralized file storage system where users can rent out their hard drives for small amounts of money. Unused space can also be used to lower file storage costs.

We call this the "decentralized Dropbox agreement," which is the crucial component of such a device. The contract works in the following way: the first step is to split the data into blocks and encrypt each. Next, you create a Merkle tree from it. The contract is then

formed with the rule that for every N block, the contract would choose a random index from the Merkle tree. This will use the previous block hash (accessible from contract code) as a source to random and give X Decos the first entity that supplies a transaction that includes a simplified payment verification-like proof that the block at that index is owned by the user. A micropayment channel protocol is available to users who wish to re-download the file. To recover the file, users will need to pay 1 szabo for every 32 kilobytes. The most cost-effective approach is to have the payer replace the transaction with one that is slightly more lucrative with the same nonce at 32 kilobytes.

One important aspect of the protocol is the fact that even though it might seem that one is trusting many random Nodes not to forget the file file, you can reduce that risk to almost zero by secret sharing the file and then watching the contracts to see if any piece is still in the possession of any node. Cryptographic evidence that the file is still in someone's possession is provided by contracts that are still paying money.

# DECENTRALIZED AUTONOMOUS ORGANIZATIONS

A "decentralized autonomous organization" is a virtual entity with a set of shareholders or members. These members, possibly 67%, would have the power to spend the entity's funds and change its code. They would decide how to allocate the funds together. The methods for allocating funds to a DAO could include salaries, bounties, and even more exotic mechanisms like an internal currency to reward hard

work. It replicates the legal trappings and enforcement of a traditional nonprofit or company but uses cryptographic blockchain technology instead. DAOs have been dominated by the capitalist model of a "decentralized autonomy corporation" (DAC), with dividend-receiving shareholders and tradable shares

An alternative possibly called a "decentralized community," would allow all members to have equal decision-making power and require 67% of members to consent to adding or removing a member. The group must enforce the requirement that only one person can have one membership.

Here's a general guideline for how to code DAO. The simplest design is a small piece of self-modifying software that can be modified if the members agree to a change. Code is theoretically immutable. However, it's possible to get around this by having code chunks in separate contracts and the addresses of which contracts to call stored within the modifiable storage. Three transaction types would result from a DAO contract implementation. These are distinguished by the transaction data:

- $[0, i, K, V]$  to submit a proposal to index  $i$  for a change of address at storage index  $K$  value  $V$
- $[1, i]$  To cast a vote for proposal  $i$
- $[2, i]$  To finalize the proposal  $i$ , if sufficient votes have been received

Each clause would be added to the contract. It would keep a record of open storage changes and a list of those who voted for them. It would also keep a list of all members. A finalizing transaction can be executed if a storage change is approved by two-thirds of the members. A more advanced skeleton may include voting capabilities such as sending transactions adding or removing members, and may even allow for Liquid democracy style vote delegation. Anyone can vote for anyone. Assignment is transitive, so if A assigns someone and B someone else then A gets C's vote. This would allow the DAO, as a decentralized community to grow organically. It also allows people to delegate the task to specialists to filter out members. However, unlike the current system, specialists can pop in and out as community members move around.

A decentralized corporation is an alternative model. Each account can have zero shares or more, and only two-thirds of shares must be present to make a decision. The skeleton would include asset management functionality, the possibility to offer shares to be bought or sold, and the ability for the company to accept offers (preferably using an order-matching mechanism within the contract). Liquid Democracy-style delegation would also be available, extending the idea of a "board" of directors.

## ADDITIONAL APPLICATIONS

1. Savings accounts. Imagine Alice is anxious about keeping her money safe but worried that someone might hack her private key.

- Alice can only withdraw a maximum amount of 1% per day.
- Bob can only withdraw 1% of the funds each day. Alice, however, can make transactions with her key closing off.
- Alice and Bob can both withdraw anything together.

Alice can usually withdraw 1% each day Bob can help her if she needs more. Alice can call Bob to transfer the funds to a new contract if her key is stolen. Bob will eventually get the funds out of Alice's account if she loses her key. If Bob is malicious, she can disable his withdrawal ability.

2. Crop insurance. A weather data feed can be used to create a financial derivatives contract. A derivative that pays inversely on Iowa's precipitation will pay if there is drought. If there is rain, the farmer will receive money. This insurance can also be extended to include natural disaster coverage.

3. A decentralized data feed. It may be possible to decentralize financial contracts using a protocol called "[SchellingCoin](#)." SchellingCoin works like this: N parties each put into the system the value for a given datum ((e.g., The DCX/USD price, the values are sorted, and everyone between the 25 and 75th percentile receives one token as a reward.) Everyone is motivated to give the correct answer, but there's only one value, that all players can agree on: the truth. This allows the decentralized protocol to provide values such as the DCX/USD currency, Berlin's temperature, or the result of complex computation.

4. Smart multi-signature escrow. Multi-signature transaction contracts are possible with Bitcoin. For example, three of five keys can be used to spend funds. D-Chain offers more flexibility. For example, four of five keys can spend all funds, while three of five can spend as much as 10% a day. Two out of five keys can also spend 0.5% per day. D-Chain multi-signature can be used to send transactions. Two parties can sign on the blockchain at different times, and the last signature will send it.

5. Cloud computing. DVM technology can be used to create verifiable computing environments. Users can ask others to perform computations and request proof that the computations at selected checkpoints were correct. This creates a cloud computing marketplace where anyone can participate with their computer, laptop, or specialized server. Security deposits and spot-checking can be used to verify that the system is reliable (i.e., Nodes cannot profitably cheat. This system is not suitable for all tasks. For example, functions that require high levels of inter-process communication cannot be accomplished on large clouds of nodes. However, other tasks are easier to parallelize.

6. Peer-to-peer gambling. Peer-to-peer gambling protocols such as Frank Stajano's [Cyberdice](#) can all be implemented on the D-Chain. The simplest gambling protocol can be described as a contract for difference on each block hash. More advanced protocols can be developed to create gambling services that charge almost zero fees and have no cheating ability.



7. Prediction markets. Prediction markets can be implemented with SchellingCoin or an oracle. They may also be the first widespread application of futarchy to decentralized organizations as a governance protocol.

8. Decentralized marketplaces on-chain, using the identity system and reputation as a base.

## MISCELLANEA AND CONCERNS

### MODIFIED HTMLOST IMPLEMENTATION

Yonatan Sompolinsky, Aviv Zohar, and Aviv Zohar introduced the "Greedy Heavy Observed Subtree" (GHOST) protocol in December 2013. GHOST was created because blockchains with fast confirmation times suffer from lower security. Because blocks take a specific time to propagate through a network, Validator A will mine a block, but Validator B will mine another block before Validator A's block propagates. This will cause Validator B to lose his block, which will be detrimental to network security. There is also a centralization problem: If Validator A has 30% mining power and Validator B has 10%, A has a 70% chance of producing a dead block. This is because A produces the last block 30% of the time and will receive mining data immediately.

B, on the other hand, will be at risk of producing dead blocks 90% of the times. Because A is more extensive it will be more efficient if the block intervals are short enough for a high stale rate. These two effects together make it more likely that blockchains that quickly produce blocks will lead to one mining pool with enough network hash power for de facto control of the mining process.

### SOMPOLINSKY & ZOHAR

GHOST addresses the first problem of network security loss. It includes stale blocks as part of the calculation for which block is the "longest". This means that not only the block's parent and further ancestors, but also the block's ancestor are included in the calculation to determine which block has the most proof-of-work backing it. We go beyond Zohar and Sompolinsky's protocol and offer block rewards to stales. A stale block gets 87.5%, and the nephew with the stale blocks the 12.5%. Uncles are not eligible for transaction fees.

D-Chain implements a simplified version of GHOST that only goes down seven levels. It is defined as follows:

A block must identify a parent and must also specify 0 or more uncles

The following properties must be present in an uncle who is included in Block B:

It must be the child of the  $k^{\text{th}}$  generation ancestral of B.  $2 \leq k \leq 7$ .

It can't be an ancestor of B.

An uncle must have a valid block header. However, he does not need to have a block that has been verified previously.

An uncle must be different from all uncles included in previous blocks and all other uncles included in the same block (non-double-inclusion)

Block B's uncle U earns an additional 3.125% of his coinbase reward, while the Validator from U receives 93.75% of the standard coinbase reward.

This version of GHOST was limited to only seven generations, and uncles were not included. Unlimited GHOST would add too many complications to calculating which uncles are valid for a block. Unlimited GHOST with compensation, removes the incentive to mine on the mainchain, not the public attacker's chain.



Every transaction published to the blockchain incurs the cost of downloading and verifying it. To prevent abuse, there needs to be some regulatory mechanism. Typically, this will involve transaction fees. In Bitcoin, the default approach is to use purely voluntary fees. This relies on Validators to act in their role as gatekeepers and sets dynamic minimums. This market-based approach allows supply and demand to determine the price.

It has been very well received by the Bitcoin community. This reasoning has a problem. Transaction processing is not a marketplace. Although it may seem appealing to consider transaction processing as a service the Validator offers to the sender, it will require that every transaction the Validator includes be processed by all nodes in the network. Therefore, the majority of transaction processing costs are borne by third parties, not the miner. Hence, tragedy-of-the-commons problems are very likely to occur.

This flaw in market-based mechanisms, when given an incorrect simplifying assumption, magically disappears. This argument can be summarized as follows. Let's say that: Transactions lead to  $k$  operations. The reward  $kR$  is offered to any Validator who includes it, where  $R$  has been set by the sender.  $R$  and  $k$  are (roughly speaking) visible to the Validator before the transaction.

A C operation costs per node (i.e., All nodes are equal in efficiency)

There are  $N$  Validators, each with an identical processing power (i.e.,  $1/N$  total)

There are no full-mining non-mining nodes.

If the expected reward exceeds the cost, a Validator will process the transaction. The expected reward for a Validator is  $kR/N$ , since he has

a  $1/N$  chance to process the next block. The miner's processing cost is  $kC$ . Validators will include transactions in which  $kR/N > kC$  or  $R > NC$ .  $R$  refers to the sender's per-operation fee. This is a lower limit on the benefit the sender derives through the transaction.  $N$  represents the total cost for the entire network of processing an operation. Validators are motivated to include transactions that have a greater total utilitarian value than the cost

There are, however, important deviations from these assumptions that can be found in real life:

Because the additional verification time delays block propagation and increases the likelihood that the block will become stale, the Validator has to pay a higher processing cost than other verifying nodes.


Another factor disincentivizing large Bitcoin block sizes is that large blocks take longer to propagate and have a higher chance of becoming stales. High-gas-consuming blocks in D-Chain can take longer to propagate. This is due to the fact that they require more time to process transaction state transitions and validate transactions. This delay disincentive is a major consideration in Bitcoin but less in D-Chain due to the GHOST protocol. Therefore, relying upon regulated block limits provides a stable baseline.

## TURING-COMPLETENESS AND COMPUTATION



Important note: the D-Chain virtual machine can encode any computation that can be carried out. DVM code permits looping in two different ways. There is the JUMP instruction which allows the program jump to a previously accessed spot in the code. A JUMPI instruction will enable for conditional jumping. This allows

statements such as `while * 27: x =`. Contracts can also call other contracts, allowing for looping through recursion. This creates a problem. Can malicious users shut down Validators and full nodes by forcing them into an endless loop? This problem arises from a problem in computer science called the halting issue. It is impossible to predict whether or not a program will ever stop.



Our solution, as described in the state transformation section, requires transactions to specify a maximum number computation steps they are allowed to take. If the execution takes longer, fees will still be paid. The same applies to messages. These examples will demonstrate the motivation behind our solution:

An attacker creates an infinite loop contract and sends the transaction activating it to the miner. After processing the transaction, the Validator will wait for the infinite loop to finish. Even though execution stops halfway the transaction will still be valid and the attacker will still pay the Validator for each computational step.

The attacker will create an infinite loop that is very long and will force the Validator into calculating for so long that the last blocks are out. It will be impossible for the miner, to claim the fee, to include the transaction. The attacker will have to submit a value `START GAS` that limits the number of computation steps that can be executed. This will ensure that the Validator is aware that the computation will take too many steps.

An attacker sees a contract with code of some form like `send(A,contract.storage[A]); contract.storage[A] = 0`, and sends a

transaction with just enough gas to run the first step but not the second (i.e., Making a withdrawal but not letting it go down. Because execution can stop at any point during execution, the changes are reverted to the contract author.

To minimize risk, a financial contract takes the median of nine proprietary feeds. An attacker takes over one of the data feeds, designed to be modifiable via the variable-address-call mechanism described in the section on DAOs, and converts it to run an infinite loop, thereby attempting to force any attempts to claim funds from the financial contract to run out of gas. To prevent this from happening, however, the financial contract can place a gas limit on the message.

The alternative to Turing completeness is Turing-incompleteness, where JUMP and JUMP-PI do not exist and only one copy of each contract is allowed to live, in the call stack at any given time. This system would eliminate the need for the fee system and uncertainties surrounding the effectiveness of our solution. The cost of executing a contract is limited by size, so the fees described might not be necessary. Additionally, Turing-incompleteness is not even that big a limitation; out of all the contract examples we have conceived internally, only one required a loop, and even that loop could be removed by making 26 repetitions of a one-line piece of code. Turing completeness has serious consequences. The limited benefit is not worth it. Why not just have a Turing-incompleteness language? In reality, however, Turing incompleteness is far from a neat solution to the problem. Take a look at the following contracts to see why.

```
C0: CALL(C1); CALL(C1);
```

```
C1: CALL(C2); CALL(C2);
```

```
C2: CALL(C3); CALL(C3);
```

```
...
```

```
C49: CALL(C50); CALL(C50);
```

```
C50: (RUN ONE STEP OF A PROGRAM AND RECORD THE CHANGE IN STORAGE)
```



Send a transaction to A. In 51 transactions, this contract takes up 2 50 computing steps. Validators could attempt to detect logic bombs before they happen by keeping a value beside each contract that specifies the maximum number and how many computational steps it can take. This would calculate the maximum number of contracts that call other contracts recursively. However, this would require Validators not to allow contracts to create other contracts. Since the creation and execution of 26 of the 26 contracts could be easily rolled into one contract, the address field in a message can be variable, so it is impossible to predict which contracts a particular contract will call. We have come to a surprising conclusion. Turing completeness can be managed easily, while the absence of Turing completeness can be challenging to manage without the same controls. But, in such cases, why not let the protocol be Turing-comprehensive?

## CURRENCY

The D-Chain network has its currency, Decos. This serves two purposes: it provides a primary liquidity layer that allows for the efficient exchange of various digital assets, and payment of transaction fees. The denominations will be pre-labeled for convenience and avoidance of future arguments (see the current Bitcoin mBTC/uBTC/satoshi discussion).

### UNIT OF D-COIN (DCX) AND ITS DENOMINATIONS WITH COMMON AND SI NAME

Value(In Dei)	Exponent	Common name	SI Name
1	1	Dei	Dei
1,000	$10^3$	KDei	KiloDei
1,000,000	$10^6$	MDei	MegaDei
1,000,000,000	$10^9$	GDei	GigaDei
1,000,000,000,000	$10^{12}$	$\mu$ Dei	MicroDecos
1,000,000,000,000,000	$10^{15}$	MiDei	MilliDecos
1,000,000,000,000,000,000	$10^{18}$	Decos	Decos or D-Coin(DCX)
1,000,000,000,000,000,000,000	$10^{21}$	KDecos	KiloDecos
1,000,000,000,000,000,000,000,000	$10^{24}$	MDecos	MegaDecos
1,000,000,000,000,000,000,000,000,000	$10^{27}$	GDecos	GigaDecos
1,000,000,000,000,000,000,000,000,000,000	$10^{30}$	$\mu$ Decos	Micro Decos

This should be viewed as an extended version of the concept of "dollars"/cents or "BTC", and "satoshi." We expect "Decos" will be used for regular transactions, whereas Tera, Peta, and Dei will be used for fees and protocol implementation in the near future. The remaining denominations might become useful later on and should not be included as clients


# SCALABILITY

The issue of scaling is a common concern with D-Chain. D-Chain is similar to Bitcoin in that each transaction must be processed by every node within the network. The current Bitcoin blockchain is approximately 15 GB and grows by around 1 MB per hour. If the Bitcoin network could process Visa's 2000 transactions per minute, it would grow 1 MB per 3 seconds (1 GB/hour, 8 TB/year). The growth rate of the D-Chain will likely be similar. This is due to the fact that D-Chain full nodes must store only the state and not the entire blockchain history.

Centralization risk is a problem when you have a large blockchain. The risk of centralization is high when the blockchain size reaches 100 TB. This would mean that very few large businesses would have full nodes, and all other users would use light SPV nodes. There is a possibility that full nodes could come together and agree to cheat profitably (e.g., Change the block reward and give yourself BTC. This would be impossible for light nodes to detect immediately. There would be at least one honest full-node, and information about fraud would start to trickle out via channels like Reddit. However, it would soon be too late. Users would have to organize an effort to blacklist the blocks. This would create a vast and likely impossible coordination problem, similar to that used to pull off a 51% attack. This is a problem in Bitcoin. However, Peter Todd has suggested a solution.

D-Chain will employ two other strategies to address this issue in the short term. First, the blockchain-based validating algorithms will force every validator to be a full node. This will reduce the limit on the number of full nodes. We will also include an intermediate state root in the blockchain to process each transaction. A verification protocol can circumvent the centralization problem even if block validation has to be done centrally. As long as there is one honest verifying node, it will not matter if the block validation process is centralized. A Validator publishing an invalid block means that the block is either poorly formatted or that the state is incorrect. is considered to be correct. Therefore, it must have a first state that is incorrect in the place is correct. The index would be provided by the verifying node, as well as a "proof-of-invalidity" that includes the subset Patricia tree nodes required to process  $S[i-1], TX[i] \rightarrow S[i]$ . The  $S[i]$  generated by nodes will not match the  $S[i]$ .





A more sophisticated attack involves malicious validators publishing incomplete blocks. This means the the information needed to determine whether blocks are valid is unavailable This problem can be solved using a challenge-response protocol. — verification.nodes issue "challenges," which are target transaction indices. After receiving a node, a light node treats it as untrusted until another, regardless of whether the validator is another verifier, provides a subset of Patricia nodes as proof of validity.

## CONCLUSION

The D-Chain protocol is designed to be an enhanced version of cryptocurrency. It provides advanced features like on-blockchain withdrawal limits, on-blockchain escrow, and financial contracts. Although the D-Chain protocol does not support specific applications, the Turing-complete programming language allows for creating arbitrarily designed contracts for any transaction or application. The most exciting thing about D-Chain is its ability to move beyond currency. Protocols based on decentralized file storage, decentralized computation, and decentralized prediction markets, among many other concepts, have the potential for significant efficiency improvements in the computational industry. They also provide an enormous boost to the peer-to-peer protocol by adding an economic layer. There are also many applications that do not involve money. The D-Chain protocol's implementation of an arbitrary state function provides a unique platform. Instead of being a closed-ended protocol that can only be used for one purpose, such as data storage, gambling, or finance, D-Chain is designed to be open-ended and will serve as a foundation layer for many other financial and non-financial protocols over the coming years.

**Legal Disclaimer:** This paper is for informational purposes only. This is not a recommendation to trade a particular digital asset or to employ a particular investment strategy. D-Chain does not represent the suitability of the information provided or any particular asset. By wrapping or holding DCX, you agree that you have read, understand, and accept all D-Chain terms and conditions (DCX).