PANTERCON©

START YOUR IDEA WITH THE PANTER

# Aubergine Paper

# Social Media



# Content Tech Paper
# Hydra Blockchain


**1. Planes of the Hydra hybrid blockchain 5**

1.1 General information 5
1.2 Mainchain 5
1.3 Sidechains 5
1.4 Parallelisation of verification processes. 6
1.5 ECHIDNA (master data of Hydra objects) 7
1.6 NYX (anonymous transactions) 8
1.7 PLUTOS (B2B transactions) 8
1.8 HERMES (HYDRA object marketplace) 8
1.9 DEMETER (ownership rights as commercial goods) 9
1.10 HERA (rights for chains, smart contracts) 9
1.10.1 Rights in general 9
1.10.2 Assignment of rights 10
1.11 HADES (Archive) 10
1.12 ARCHIMEDES (Hydra objects) 10
1.13 Test network 11
1.14 Tokens 11

**2. Role-based time-controlled user authorization concept 11**

2.1 General information 11
2.2 Components concept 12
2.2.1 Organizations 12
2.2.2 Organizational units 12
2.2.3 User login 13
2.2.4 Rights 13
2.2.5 Roles 13
2.2.6 Roles rights 13
2.2.7 User roles 13
2.3 Time limitation 14
2.3.1 Temporary rights 14
2.3.2 Access to data with a limited period of time 14

**3. Consensus algorithm 15**

**4. HERAKLES 15**

**5. PHOENIX 16**

**6. Really smart contracts 17**

6.1 General information 17
6.2 Automatisms 18
6.3 Standardization through modularization of smart contracts 18
6.4 Customizing of modules 19
6.5 Interaction with contracts during runtime 19
6.6 Termination in case of non-compliance 20

**7. The „time problem“ 20**

**8. Merkle Tree / Merkle-Proof 20**

# 1. Planes of the Hydra hybrid blockchain

## 1.1 General information

In order to guarantee high performance and low storage capacity, the Hydra consists of several layers. Similar to the plasma approach, MapReduce functions are used here, too, and the verification processes are parallelized into several forced childchains with Merkle Trees.

The use of sidechains (childchains) assigned to a main chain alone increases the scalability enormously.

## 1.2 Mainchain

This is the classic main variant and heart of the blockchain system. With the Hydra, this mainchain consists of many different levels, which are explained below.

## 1.3 Sidechains

The sidechains are separate blockchains that are attached to their parent blockchain by a two-way peg. This „Two-Way-Peg“ (Zerberus) regulates the interchangeability of objects between the superordinate blockchains and the sidechains. A fixed rate is defined for this. It is also possible to attach several sidechains to a sidechain. This means that entire hierarchies can also be mapped.

In order to name this hierarchical illustration always one speaks of a „Parent“-Chain and the subordinated

„Child"-Chain.

The technical procedure is as follows: Users must always send objects to a certain output address in the higher-level chain. There, the objects are locked so that they can no longer be used in this chain, so that users cannot duplicate them. After verification of this transaction, the objects are released in the sidechain. The user can now use them there. The reverse case occurs when returning from a sidechain to the main chain.

### 1.4 Parallelisation of verification processes.

If the current number of transactions for verification exceeds a specified value, a new verification process is automatically started in parallel.

This process can be repeated several times and will be limited to a certain number of parallel processes. Depending on the consensus algorithm used and the number of verification points, there may be a large deviation in performance in a direct comparison.

The number of transactions that trigger a new parallel process and the limit of the parallel processes are determined by tests during the conversion to determine the optimum.

Technically, the conversion takes place with Merkle trees and in forced sidechains.

Therefore, a transaction rate of 50,000 per second

should be achievable initially.

In the last expansion stage of Hydra, a multiple of this will be feasible.

### 1.5 ECHIDNA (master data of Hydra objects)

Contains the master data of all levels of all Child-, Sidechains and the Mainchain. Only the master data of the chains and their objects are verified in ECHIDNA. No classical transactions take place here. This level is used exclusively for the functionality of all other chains. The entire fee administration is also managed in ECHIDNA.

In principle, it is possible to charge other fees or additional fees in sidechains, which are then transferred to an assigned organization. This depends on the consensus algorithm and whether the sidechains are „public" or „private".

In order to obtain a simplification for the end user, fees of the transaction in the mainchain are divided into 3 priority levels (high, medium, low). High priority is associated with high fees, but guarantees an immediate transfer, etc.

In the final stage, users will also be able to create new fees themselves.

### 1.6 NYX (anonymous transactions)

The NYX sub-segment is intended to facilitate anonymous transactions.

However, there is still a need for legal clarification.

### 1.7 PLUTOS (B2B transactions)

The B2B transactions are stored here. Similar to other public chains, these addresses can theoretically be viewed. Through the user authorization it will be possible for the owners of a wallet to transfer time-controlled rights to another location. Example: A company grants reading rights of an account for the accountant of the tax consultant with a time restriction (example: only the last 3 months).

### 1.8 HERMES (HYDRA object marketplace)

The transactions of goods objects are documented here. In this area, the focus is on the one hand on traceability in the movement of goods and on the other hand on data provision. The focus here is on the following points in particular:

- Serial numbers
- Batches
- Logistics tokens

In the area of data provision, the aim is to provide data for B2B processes for other users. Possible use cases for this are that suppliers provide articles with all

attributes (specification of dimensions, text descriptions, availability and also prices). Via the assignment of rights it is possible to make these only available to predefined partners. These in turn can then work with these data and fill their ERP systems.

## 1.9 DEMETER (ownership rights as commercial goods)

This area offers a very high potential of use cases. With the transfer of ownership to one or more other users, rights are automatically assigned as well. These may be final or subject to time restrictions.

## 1.10 HERA (rights for chains, smart contracts)

### 1.10.1 Rights in general

In this area, the rights of the individual users, their roles and thus the authorizations for the respective chains are documented. The authorization system will be discussed in detail. The most important areas here are that two types of time control are bound to the rights.

- The right itself is only valid for a certain period of time (e.g.: holiday replacement, then a right can be created for an employee for 14 days).
- The right is subject to a time limit (e.g. employee may only view the transactions of the current quarter).

### 1.10.2 Assignment of rights

The rights are assigned according to a two-stage principle. The rights can be created and are automatically saved. However, these only become active after verification and the fee is incurred.

## 1.11 HADES (Archive)

The HADES area is used for archiving data. The Phoenix method is used to move data from the main chain to the HADESChain. This function can also be used for sidechain in the final development stage. The Phoenix method is explained in more detail below.

## 1.12 ARCHIMEDES (Hydra objects)

This is where the details of the Hydra objects are stored. In general, the term object is used for all kinds of tokens and newly generated things.

Furthermore, it is also possible to generate completely new types of objects that are not comparable to the „classic“ tokens. Here the limit is the imagination.

- Example: ToDoLists for project management,
- Management of property boundaries,
- Calorie counter for Weightwatchers
- Wandering needle administration etc.

### 1.13 Test network

The test network has all the features of the productive system. The differences here are that the transaction data (transactions) are deleted every quarter, but not the smart contracts. In addition, KRONOS allows you to simulate time. This means that time-controlled contracts can be tested at 10 times the speed as an example. This would mean that a contract that runs for 30 days in real terms would have run in 3 days. KRONOS not only has the ability to accelerate time, but also to stop at certain points or consciously go to certain points in time to test from that point.

### 1.14 Tokens

There will be a number of predefined tokens that already have certain properties. Such classic tokens become Security Token, Utility Token, Membership Token or Access Token. For each token, however, it will also be possible to freely program functions for it.

## 2. Role-based time-controlled user authorization concept

### 2.1 General information

This concept combines the classic role-based user authorization with a time-controlled assignment of rights and a time-controlled access to transaction data.

## 2.2 Components concept

It is possible to assign one or more roles (admin, clerk, etc.) to a user. These roles have special rights. A user may execute an action if he or she has been assigned the right on the basis of one (or more) of his or her roles. A user may execute an action if he or she has been assigned the right based on one (or more) of his or her roles. Here it will be possible that these roles have a time limit. Example: Transferring a role to another employee for vacation replacement.

In the final development stage, it will be possible to further individualize rights at the user level independently of roles.

### 2.2.1 Organizations

Organizations can best be declared as legal entities, companies, NGOs, etc.

They can run sidechains or work with the mainchain. For the objects created by you (tokens, sidechains etc.) you have the possibility to assign authorizations.

### 2.2.2 Organizational units

These are the units that make up an organization and represent the departments as in the classic organization chart. These organizational units can then be assigned roles that are passed on to the users.

### 2.2.3 User login

In general, a user must log in to Mainnet in order to move to the other levels. When logging in, the permissions of the different levels are loaded and allow navigation in the individual areas.

### 2.2.4 Rights

Rights are defined as the release or prevention of an action. Example: Reading transaction data, creating objects.

### 2.2.5 Roles

Roles are defined as functions of persons who must have several rights in order to exercise them. These roles can be for example: administrator, clerk etc.

### 2.2.6 Roles rights

Authorizations are then assigned to these roles. As an example, an administrator may read Sales and Transactions for the last quarter, but may not make any bank transfers.

### 2.2.7 User roles

Here the individual users are assigned one or more roles. This means that it is theoretically possible that role rights contradict each other. The „most favorable“ right

then applies here. Example A user is assigned to the role „sales clerk“ and is therefore not allowed to make transfers. However, he is also assigned to the role of „Management“. This role does indeed have the right to make transfers, so this right is granted.

## 2.3 Time limitation

### 2.3.1 Temporary rights

As already mentioned at the beginning, you can assign rights temporarily. These can be assigned for a certain period of time or with an expiration date. In the final version, it will be possible to have these event-controlled via Smart Contracts. Example: After receipt of a payment, a right is extended by one month.

### 2.3.2 Access to data with a limited period of time

Another important point is the possibility of making transaction data with limitations available. This can be a static period or a dynamic period.

So it would be conceivable that certain users may see only the transactions since beginning of the year and others always only all transactions, which are not older than 3 months.

## 3. Consensus algorithm

In the mainchain, an algorithm similar to the Proof of Importance will be used. An importance value is assigned to each individual point, which is made up of various factors such as the number of PANXs, holding time, height and number of transactions, and so on. This method helps to ensure that all computers on the network match. Users with high „importance“ can „harvest“ and earn rewards.

At Hydra, HXP can be specifically harvested through this process. These, in turn, are needed to carry out actions, or can simply be sold.

For self created Hydraobjects and especially in the sidechains different consensus algorithms can be used.

In the final stage the most common consensus algorithms will be provided and it will be possible to program own consensus methods.

## 4. HERAKLES

Herakles is the GUI for creating new side- and childchain, rights management and the whole object management. This formed the core part for the entire management system.

Special attention is paid to intuitive usability. Furthermore, there will be interfaces for interacting with the individual areas of the blockchain or individual smartcontracts.

# 5. PHOENIX

Phoenix allows the burning of tokens and their simultaneous rebirth not only within a chain. This functionality can also be used for customize tokens or is used in the Hydra Main Chains. The simplest comparison is with a closing and opening balance sheet. The balances for all booked accounts are created and transferred to the new posting period. This usually happens at the end of the year, but can happen at Hydra in freely selectable periods.

Here a kind of balancing takes place in the Phoenix protocol. The mainchain becomes a Hades archivechain and a new mainchain is created. From the Phoenix protocol, the balanced values are now transferred to the new mainchain. The transactions have not been lost, but can still be viewed via the Hades ArchiveChain.

For the user it will be in such a way that it works as if these data were still in the main chain.

By this methodology the blockchain is kept small again and again and archives can be created. Theoretically it would also be possible to destroy such an archive.

Since this methodology is also available in sidechains, there can be also points of contact with the DSGVO here depending on the function mode and the data that are stored. In order to satisfy these legal aspects, the Phoenix method can be used here.

# 6. Really smart contracts

## 6.1 General information

The programming language for creating Really Smart Contracts will be C#. The goal is to be able to design contracts so „intelligently“ that they run automatically and cover all important components of a conventional contract.

As with these classic contracts, terms, notice periods, time points, etc. can be defined.

However, depending on the type of contract, there will be either an „abort“ or a „rollback“ function. The „Abort“ function cancels the contract and overrides it, for example: standing order for an employee‘s salary if the employee has resigned. The „rollback“ function reverses a contract. Example: An ICO has not reached the soft cap and the investors receive their deposit back.

In the contracts, a separate process chain can be defined for the „abort“ or „rollback“ function, e.g.: Supplier cancels the contract and has to pay the customer a penalty in a predefined amount. Ideal applications for this are quantity and call-off contracts. Further application areas could be commission payments with external companies, subscriptions with end customers as well as consignment contracts.

The contracts must have a term.

The activation of a contract is the responsibility of the order creator, but the orders can be assigned authorizations that force the approval of the other contract partners.

### 6.2 Automatisms

In the case of contracts, special attention is paid to automatisms. For this two core elements are considered: event-controlled and time-controlled execution of functions. An event can be defined as the receipt of a transaction, which then triggers an entire process chain. Time-controlled would be if the amounts were automatically transferred from a branch to the head office at midnight every day.

### 6.3 Standardization through modularization of smart contracts

We provide standardized modules that can be combined by the user. Each module provides input and output parameters and has the ability to be executed in loops.  As an example a module can be called „time-controlled pricing".

So it would be possible to deposit a table here, which gives a new price for each day.

By combining modules, event-driven process chains can also be mapped here.

### 6.4 Customizing of modules

Nevertheless, it will be possible to write and store code for special applications instead of the module code itself. For each module it will also be possible to enter a user-defined code instead of the suggested code. Thus, the module time-controlled pricing could be reprogrammed in such a way that the price automatically increases by 10% after the first 100 sales per day. For this a new input parameter for the number of daily sales would have to be stored.

In general, a self-written code is temporarily stored in a database. The code is only written into the blockchain and becomes valid when the smart contract is activated.

### 6.5 Interaction with contracts during runtime

Before the contract is activated, parameters that allow interaction must be defined and assigned authorizations. An example of this is a contract extension. Depending on the authorization, the contract owner can enforce this on his or her own, or approval must be obtained from the other contract partners. In general, there will be a GUI for interacting with Smart Contracts. Depending on authorizations, users will be able to access the functions provided by the Smart Contract owner. There will also be an interface for batch processing. Data can then be uploaded in a contract by the contract owner. Example: Several addresses,

which are all to be placed on a whitelist, or payments to several addresses with different amounts (e.g.: Airdrop or Bounty programs).

### 6.6 Termination in case of non-compliance

If a contract is not fulfilled, termination events are defined. Example: Payment was not made. This releases the „exit“ scenario for the disadvantaged party. In most cases this is the „Abort“ function, but can also lead to a „Rollback“ function, if so defined and possible by the contract type. The disadvantaged party must complete these functions and activate them manually.

## 7. The „time problem“

To ensure a uniform time for block creation (acquisition time), unix time is generally used. In parallel, the time of the verification point is stored for each block (Time editor). Here a check with a central time unit takes place in advance. The editor time is checked for deviation and if this does not exceed 13 seconds, it is executed.

## 8. Merkle Tree / Merkle-Proof

Verification is done with Merkle Trees.

A Merkle tree is in the broadest sense a kind of compilation of data blocks, which is based on representing the blocks in a kind of tree structure. Each branch contains only a few blocks, which are combined, hashed together and then lead to another branch. Each of

these branches now goes through the same process, which is repeated until the total number of remaining hashes is only one value: the „root hash".

A Merkle proof consists of the „root hash" of the tree and that of the „branch", which consists of all hashes that go from the branch to the root.

Thus it can be checked whether the hashing at least for this branch was carried out consistently up to the end of the tree and whether the position of the branch in the tree is actually at this point in the tree.

The application is simple: there is a large database and the entire content of the database is stored in a Merkle tree, in which the root of the Merkle tree is publicly known and trustworthy (e.g. it has been digitally signed by enough trustworthy parties, or there is a lot of evidence for the work on it). A user can request a Merkle proof and after receiving the proof check whether it is correct and whether the received value is actually at the checked position in the database with that particular root.

In the course of the implementation, not only binary Merkle trees are used, but also a more complex variant similar to the Patricia tree principle.