CHRYSALIS TOKEN WHITEPAPER

INTRODUCTION

BACKGROUND

The advent of Bitcoin in 2009 has ushered in an era of digital currency development. As of January 2021, there are now more than 4000 cryptocurrencies, each one boasting greater capabilities than the last. There are cryptocurrencies promising to secure online voting, to platform arbitrary Turing-complete smart contracts, to completely anonymize blockchain transactions. As of crypto's real world use as a payment method, around 2,300 out 32.5 million US businesses accept any sort of cryptocurrency at the time of writing, representing 0.0001% of the total. Meanwhile, crypto is on the verge of being outlawed in several countries due to excessive energy use and speculation, while a bitcoin fork featuring a certain dog reached a market cap of $54 Billion dollars, despite being created as a joke. It is clear that cryptocurrency has deviated significantly from its original intent of acting as a viable financial system that is secured through decentralization to being just mere speculative digital assets.

This outcome results from the insufficient performance of current cryptocurrencies at scale. Visa processes an average of 1700 transactions per second with a theoretical peak of 24,000 at no cost to the end user. Meanwhile, Bitcoin is limited to 7 transactions per second, a full node requires 300 GB of space, and average transactions cost $2+, making

it entirely unfeasible as a real payment method. Other solutions attempt to improve performance and cost characteristics through sidechain aggregation and proof of stake, but these designs compromise the security and decentralization that is the raison d'etre of DeFi solutions in the first place. **Chrysalis** will provide a simple payment protocol for retail adoption, leveraging trusted bases and a token specific directed acyclic graph architecture to provide off-network transactions with zero storage requirements, zero costs, and zero latency before confirmation.

ABSTRACT

The purpose of the Chrysalis token is to create a protocol for representing arbitrary tokens with the lowest possible overhead while obscuring the identity of parties involved in any transactions. This is accomplished by localizing the history of state to each cryptographically secure token while excluding any information about the addresses involved. Owners of Chrysalis would only need to store the private keys and histories of tokens they currently hold instead of a history of every transaction in the network. Transacting with Chrysalis is simply a matter of transferring private keys to the receiver and verifying against the token mint or any other persistent aggregators in the network. The Chrysalis protocol also makes it possible for anyone to minting tokens and back them with any asset requiring only a single lightweight network validation node.

LEDGER

TOKENS

In the Chrysalis protocol, the state is composed of **tokens**. Tokens are created by mints in collections called **mintings**. A **minting** represents a contract between the mint and the purchaser of the token where the mint guarantees each token is backed by an amount of some external asset. Note that the legal status of these guarantees is not governed by the protocol and may depend on jurisdiction. Only buy from trusted mints.

Issues as broadcasted by the mint contains the following fields (Figure 1):

- The **mint**, the public key of the mint
- The **root hash**, the top hash of the Merkle tree constructed from all tokens in the minting
- The **contract hash**, the hash of the backing guarantee for tokens in the minting. The owner of the token should keep a copy of the guarantee.
- The **signature**, the content of the minting signed by the mint
- The **divisible** flag, which specifies if the tokens in the minting can be split and joined with each other. Set to false for unique assets such as NFTs.

```
"minting": {
    "mint":
"MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEpbjW9ZMr4mJzqgDldbmbU7VoKKrjC0T6NB+4lrHrj
2k44IyIyKDzbzRrGboLy6qg/whS/vwFjHKFal2CAUn+oQ==",
    "root_hash":
"70176124abd98d92abaaeeb88c51ff0d7822982f53d35fd231198391078acabb",
    "contract_hash":
"50e721e49c013f00c62cf59f2163542a9d8df02464efeb615d31051b0fddc326",
    "signature":
"MEUCIBPbT8LCUqKo39WMU66SCXfcalQ7zj25n77CAq/eZHiIAiEAkN+rq+AW4k7J3eE1oeWTt
ZEZ2DB1z5I2esAe8RMkoXg=",
    "divisible": true
}
```

Figure 1: Issue structure

Tokens contain the following fields (Figure 2):

- The **public key**
- The **ancestor key**, the initial public key as created by the mint
- The **history**, a recursive hash representing all transformations enacted on the token. Computed as `history=Hash(history+pub)`
- The **data**, a field for read-only arbitrary data.
- The **value**, a positive integer representing the value of the token. For indivisible tokens this value is always implicitly `1`.

```
"token": {
    "pub":
"MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEW0C5GSsEnYS/QQknIlNZxb/zGE0Rgteud0JSrpa7L
yi7DPqKuSdMWEwTx+z8LjGhVpFKZHWJo/q1w6svXOt2qg==",
    "ancestor":
"MFYwEAYHKoZIzj0CAQYFK4EEAAoDQgAEpbjW9ZMr4mJzqgDldbmbU7VoKKrjC0T6NB+4lrHrj
2k44IyIyKDzbzRrGboLy6qg/whS/vwFjHKFal2CAUn+oQ==",
    "history":
"1a48baa4a49846e8cbc26978ae3bcce43373c422d906a7ab70bf9a70b7ce0489",
    "data": {},
    "value": 100,
}
```

Figure 2: token structure

Ownership of a specific token is defined as possessing its private key. A token can be determined to belong to a minting by obtaining the minting's Merkle tree from the seller or other source and verifying it against the root hash provided by the mint. This data structure guarantees that the size of the token is constant regardless of how many times it has been transacted.

TRANSACTIONS

The basis of ownership being defined as the possession of the token's private key allows means that all parties with this key share ownership of the token. This is expected behavior and enables fund sharing between trusted parties. For operations that are transactional or when a private key is suspected to be exposed, token owners can execute a **metamorphosis** operation to reestablish sole ownership of a token.

METAMORPHOSIS

Metamorphosis executes a key change on a token. The operation requires a signature of the current canonical token(s) as input. When multiple metamorphoses are broadcasted involving the same parent token, the first to reach the network is considered canonical.

INDIVISIBLE TOKENS

Metamorphosis for indivisible tokens consists of the following steps:

1. Generate a new public-private key pair
2. Acquire the private key for a current token as **input**
3. Construct a new token
    1. Set the public key to the new value
    2. Set the history as `history=Hash(history+pub)`
    3. Set the ancestor to that of the current token
4. Broadcast the new token into the network along with the key signature of the **input** token

DIVISIBLE TOKENS

Metamorphosis for divisible tokens consists of the following steps:

1. Generate an amount of public-private key pairs of your choosing
2. Gather the private keys for tokens as **input**
    1. Determine the **total value**
        1. Ensure all keys share the same `ancestor`
3. Create a token from each key pair
    1. Set `pub` to the new public key

    2. Set the history as `history=Hash(history+all_pub)`, where `all_pub` is the concatenation of all public keys in the input
    3. Set the value so that the sum of values is equal to the **total value**
    4. Set the ancestor to that of the input
4. Broadcast the new tokens into the network along with each key signature of the **input** tokens

The metamorphosis will then be either rejected or confirmed by the network. If the operation is confirmed, then the party that executed it will have sole ownership and the new token(s) will be recognized as the canonical descendant of its ancestor key. If the operation is rejected, then the token has already been transacted. For this reason, only consider a transaction verified if a metamorphosis on all tokens involved succeeds.

INCOMPLETE TRANSACTIONS

When transactions involve multiple tokens, it is for metamorphosis to be successful for some but not all tokens. If atomicity is a priority, the recipient could employ the following procedure to make multi-spend attacks much less likely to succeed.

1. Precompute metamorphoses
2. Wait a pseudorandom amount of time after private keys are received
3. Check that the provided tokens are still valid
4. Broadcast prepared metamorphoses

This procedure ensures that the attacker must broadcast at the exact same network time block as the legitimate recipient.

NETWORK

The Chrysalis network uses an unstructured protocol to guarantee network transparency and defend against eclipse attacks. This also allows auditor nodes to ensure the mint node is behaving correctly.

NODES

There are 3 types of nodes on the Chrysalis network.

- Transaction nodes. These short lived nodes broadcast metamorphoses and wait for confirmation.
- Auditor nodes. These nodes aggregate received metamorphoses and participate in the confirmation process.
- Issuer nodes. These mint-operated nodes aggregate received metamorphoses and lead the confirmation process. For every minting there must be exactly one mint node.

BLOCKS

The Chrysalis network processes transactions by the block. Blocks are intervals of approximately constant time encoded as the output of a **VDF** which references the previous block. During a given block, the auditor and mints aggregate received requests into a candidate list and signs it with the block hash. Simultaneously, the mint merges all candidate lists from the previous block given by auditors with its own, then computes and broadcasts the list of all valid descendants. The auditor nodes compare this output with their own merged lists to check for discrepancy. Any malicious behavior from the mint such as attempts to exclude valid operations can be cryptographically proven using its last broadcast.

CONFIRMATION

The validity of a given metamorphosis is given by the following algorithm, which is executed by auditors and mints:

1. Check that the ancestor token belongs to this minting
2. Check that the history is well-formed
3. Check that there is not another metamorphosis with the same ancestor token in this block
4. If there is a proof of work requirement for descendants of this ancestor, verify it.

PROOF OF WORK

Every single metamorphosis broadcast must include a proof of work that is derived from the history of its constituent tokens. This proof of work is used to prevent network congestion and defend against DDoS-class attacks, and should take around 5-10 seconds for a typical device. For divisible tokens, each output token must include its own proof of work to discourage frivolous token splitting.

BLOCK STATE BROADCAST OBJECT

The state broadcast at the end of the each block consists of the following:

- The **Block**, a hash of the VDF output representing the current block
- The **Tokens**, a list of the canonical descendants for each token in the minting
- The **Proof of work target**, a table of proof of work targets for ancestor nodes. Nodes are represented by their hashes.

```
"state": {
    "block":
"75d527c368f2efe848ecf6b073a36767800805e9eef2b1857d5f984f036eb6df891d75f72
d9b154518c1cd58835286d1da9a38deba3de98b5a53e5ed78a84976",
    "tokens": [],
    "pow_targets": {
        "4538aacc6ccae167eb462bd2d6ced3537edf6f8d88af709be7b130c0": 14000,
        "ed79d646f40c12ee61f863054409241d26dd803ee0300aadda2e2d25": 200000
    }
}
```

Figure 3: state structure

Unlike a typical blockchain implementation, only the last state object is needed to compute the current state. As the tokens are of constant size, the state for any particular minting has a constant space requirement, which makes auditor nodes runnable on almost any device. Therefore, there would be sufficient auditors to secure the network from malicious mint activities even without financial incentives.