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.

TOKENS

In the Chrysalis protocol, the state is composed of **tokens**. Tokens are created by **mints** as an asymmetric key pair and represents a contract between the mint and the owner to the exchange of some asset on token **purchase** or **settlement**. Each token, including its constituent contract, is cryptographically signed by the mint and may be legally voidable. Ownership of a specific token is defined as the possession of its private key.

Tokens assume two forms: **hashed** and **complete**.

Complete tokens are used to assert ownership of a token. Complete tokens are only used by the mint to prove **settlement**, as they contain its full history and are consequently space inefficient. Complete tokens contains the following fields (Figure 1):

- The **public key**
- The **proof**, the new public key signed with the current private key
- The **signature**, created by the mint signing the public key to attest token validity
- The **parent**, the direct predecessor of the token. It contains a reference to its parent, thereby tracing the full history of the token.
- The **data**, a field for read-only arbitrary data that was created with the token.
- The **value**, a positive integer representing the value of the token. For indivisible tokens this value is always `1`.

```
"token": {
    "pub":"KJV5Uz/bApcUAUHzAdSwG1oflFsbVG0RD3LEu+QB/PE=",

"proof":"L0qxENuCuuonVMz+VP7xTDki4J1QLTNhR0xFblkgDhxFhOIaPnHLuUZfzyQSkNlPs
iMXmco+BX7TpQbFYjszBA==",
    "signature":
"/uzySnOkg2q5agHFfFGw+t7/c26o2eEygQA1M1/ovY9U5KnbjZ/m6x1AVFiKFPOjTMOTdrI4Q
nzcmaaHSFZXDw==",
    "data": "",
    "value": 1,
    "parent": {
        "pub": "uBuhQqW30CYDjD3ke6KPz/oJ2tTjgzfaakXDg1Hjs1M=",
        "signature":
"LOjnTH9oSnkjYs6mrwaLFqdc079x+B2W+uqczxkn4Mui03ZupyLDulEm/YN6RXJ3AbdDpg9WO
HAAYj7joxXFAQ==",
        "proof":
"578LgCS7mlRJ2GByA2Za/gu7b/9HNQGYpgtgWrDR6PsnbSTomhrw/ggMoRau+hibwq/cJGgfk
lhoCZOR8vk4BQ==",
        "data": "",
        "value": 2,
        "parent": null
    },
}
```

Figure 1: token structure

Hashed tokens are used to verify the validity of a token in metamorphosis. Their space requirements do not scale with history. Hashed tokens contain the following fields (Figure 2):

- The **public key**
- The **proof**
- The **signature**
- The **ancestor key**, the initial public key as created by the mint
- The **history**, a recursive hash representing all metamorphoses enacted on the token. Computed as `history=Hash(history ⧺ pub)`, where `⧺` denotes concatenation.
- The **height**, a positive integer equal to the number of metamorphoses enacted on the token.
- 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 `1`.

```
"token": {
    "pub":"KJV5Uz/bApcUAUHzAdSwG1oflFsbVG0RD3LEu+QB/PE=",

"proof":"L0qxENuCuuonVMz+VP7xTDki4JlQLTNhR0xFblkgDhxFhOIaPnHLuUZfzyQSkNlPs
iMXmco+BX7TpQbFYjszBA==",
    "signature":
"/uzySnOkg2q5agHFfFGw+t7/c26o2eEygQA1M1/ovY9U5KnbjZ/m6x1AVFiKFPOjTMOTdrI4Q
nzcmaaHSFZXDw==",
    "data": "",
    "value": 1.5,
    "history":
"982d9e3eb996f559e633f4d194def3761d909f5a3b647d1a851fead67c32c9d1",
    "height": 19
}
```

TRANSACTIONS

METAMORPHOSIS

The basis of ownership being defined as the possession of the token's private key allows all parties with this key to share ownership of the token. This is expected behavior and enables fund sharing between trusted parties. Token owners can execute a **metamorphosis** operation to re-establish sole ownership of a token.

Metamorphosis executes a key change on a token. The operation requires a signature of the current canonical token(s) as input and returns a signature of the new token as output. When multiple metamorphoses are broadcasted involving the same parent token, only one will be approved. Consequently, there is only one canonical lineage tree for each token.

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 possible 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 nearly the same time as the legitimate recipient.

RE-ISSUING

When a metamorphosis operation is executed on a token with an excessively long history, the mint may settle the token and return a newly created token with the same contract and value at its discretion. This re-issuing allows the mint to truncate history to preserve space.

SETTLEMENT

The settlement transaction is an exchange of a valid token and its private key signature with some asset stipulated in the token's constituent contract. The mint is required to make this transfer to any party presenting a token bearing its signature, unless it can disprove that the token is not the canonical descendant or the token's lineage has already been settled.

To prove the token is not the canonical descendant, the token is required in its complete form. To prove prior settlement, the complete token must be timestamped from a time stamp authority as specified in *RFC 3161*, or be included in a prior **settlement block**.

A settlement block is a collection of tokens that have been settled. The block is presented as a Merkle tree where leaf nodes are tokens. A complete block features complete tokens as leaf nodes, whereas hash blocks features hash tokens. The block is signed by the mint. Settlement block broadcasts must adhere to the following constraints:

1. Blocks must be signed by the mint
2. Complete blocks must be broadcasted for a interval specified in the contract, usually around a month
3. Hash blocks must be available upon request until the contract ceases to be valid
4. For each broadcast interval, there must be at most one settlement block

CONSENSUS

The Chrysalis protocol operates under a proof of fraud model; mints are trusted by default, but malicious behavior will always be discovered by a subsequent transaction and are cryptographically indisputable. Furthermore, in-network penalties proportional to the infraction will manifest as an effect of the token contracts. Infractions can be shared between transactors and will heavily damage the reputation of the mints.

FRAUD CONDITIONS

Chrysalis protocol requires the mint's signature on all transactions, which cryptographically proves the mint's approval of the transaction. Therefore, the mint is not able to defraud the network by subtracting transactions. The only viable attack would be to add invalid metamorphosis and settlement transactions. Chrysalis invalidates these attacks by placing the burden of proof on the mint. The token contracts specify that the mint must accept all settlement transactions unless it can prove the token has already been settled or the token is no longer valid. If the mint approves multiple metamorphoses on the same token, all created tokens will be valid. Consequently, the mint will have to payout during settlement. Likewise, if the mint creates multiple settlement broadcasts in the same interval, all offending broadcasts will be invalidated. The tokens in the broadcast will not be settled and the mint will have to execute another payout.

PROOFS

Multiple metamorphoses does not require a proof; the mint will silently approve the duplicate settlements, as it is unable to provide a proof of prior settlement or token invalidity. To prove duplicate settlement broadcasts, the transactor needs to knowledge of all past hashed settlement broadcasts.

NETWORK

The Chrysalis protocol specifies that each mint runs its own endpoints. These endpoints would compose a federated network which utilizes the same protocol but are not interconnected. Cross-mint transactions occur as simple exchanges of key pairs without the need of bridges. To prevent network congestion, mints can require a proof of work on all requests made to the endpoints. For divisible tokens, each output token can include its own proof of work to discourage frivolous token splitting. Transactors can also connect with each other to form a distributed hash table to share fraud proofs and prior settlement broadcasts.