ChainBlender: An autonomous crypto-currency mixing and anonymizing mechanism.

Abstract

In this document I propose an autonomous decentralised crypto-currency mixing and privacy mechanism by use of deterministically elected relay nodes or by use of neighboring nodes.

Definitions

- cbjoin_t - A join message.
- cbstatus_t - A status message.
- cbbroadcast_t - A broadcast message.
- cbleave_t - A leave message.

Background

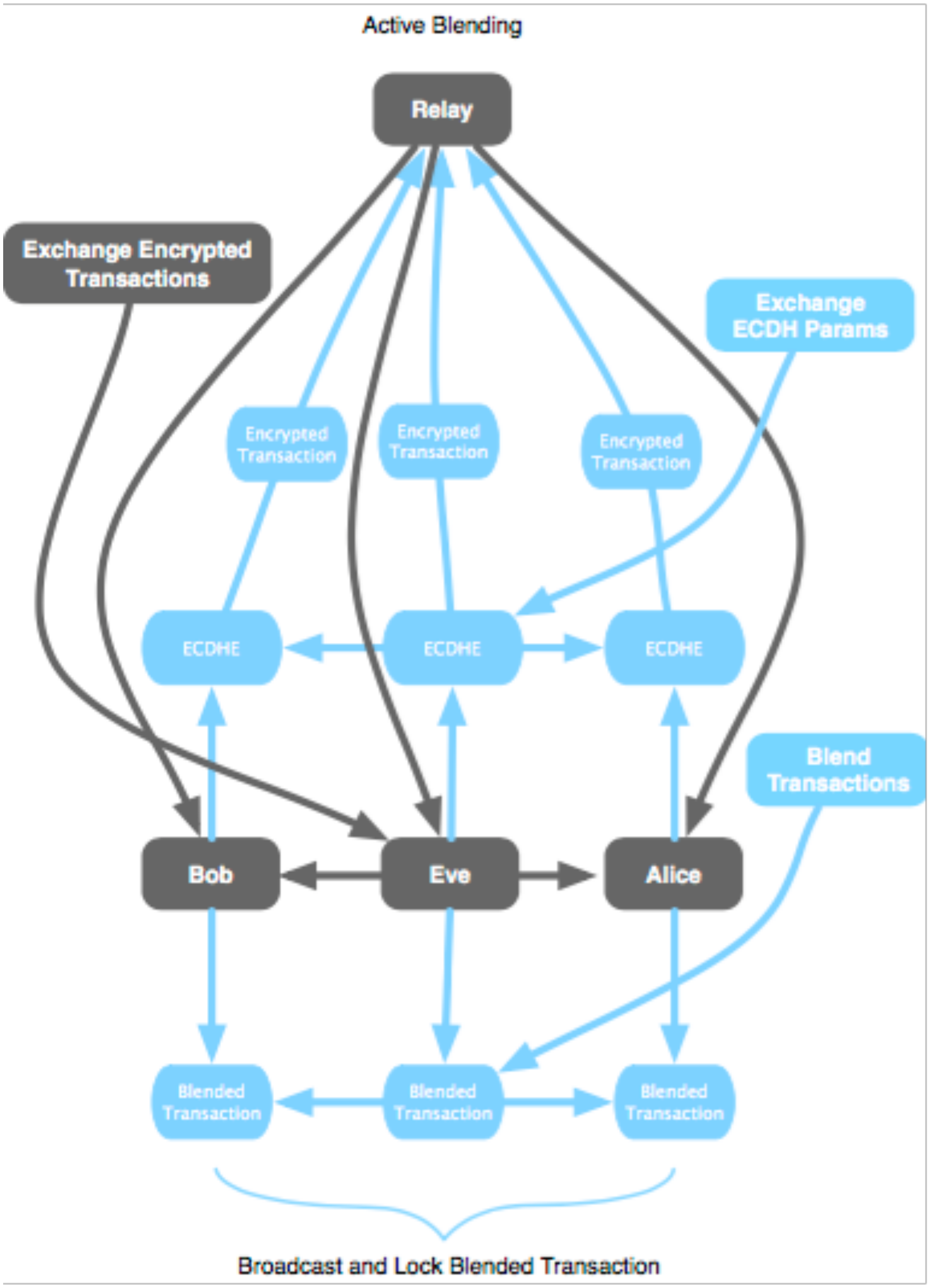
Bitcoin transactions are made up of inputs and outputs. The way these inputs and outputs are arranged makes them inherently traceable through blockchain analysis. Because of this Bitcoin also lacks the properties of fungibility.

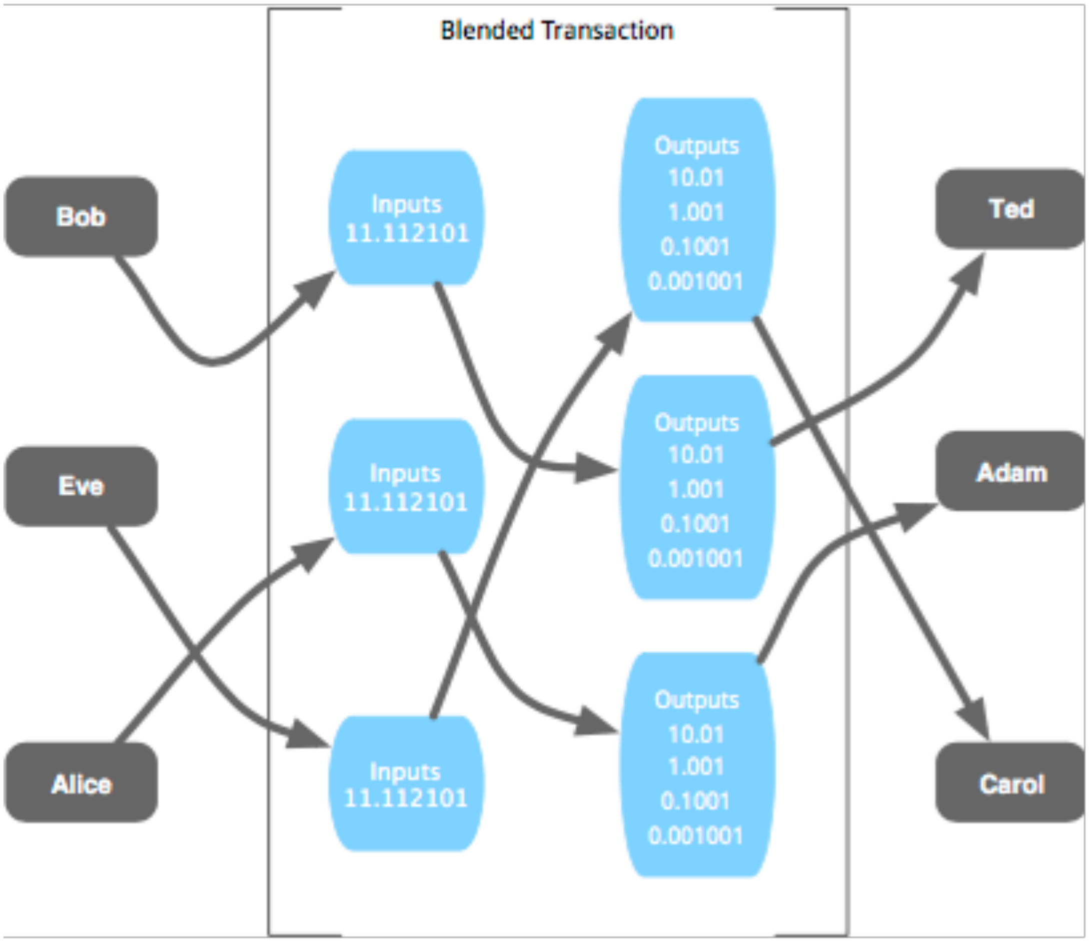
Fungibility is the property of a good or a commodity whose individual units are capable of mutual substitution. That is, it is the property of essences or goods which are capable of being substituted in place of one another[1].

By lacking the properties of fungibility certain Bitcoin may be selectively blocked or rejected due to the previous transaction history associated with it.

General Overview

By using common denominations and merging multiple similar inputs and outputs in a random order we are able to form a single transaction from multiple parties by relaying these operations in an encrypted manner by use of Elliptic Curve Diffie–Hellman Exchange (ECDHE)[4] between participants and blending them on the client-side. In this way the middle-man is reduced to a blind relay and the participants would be unable to reconstruct the original inputs from the blended transaction. Because of this it makes for a truly untraceable system offering a high degree of anonymity without counter-party risk.


Active Blending
Relay
Exchange Encrypted Transactions
Exchange ECDH Params
Encrypted Transaction
Encrypted Transaction
Encrypted Transaction
ECDHE
ECDHE
ECDHE
Blend Transactions
Bob
Eve
Alice
Blended Transaction
Blended Transaction
Blended Transaction
Broadcast and Lock Blended Transaction



Message Structures

```
typedef struct cbjoin_s
{
    sha256 hash_session_id;
} cbjoin_t;
```

```
typedef struct cbstatus_s
{
    sha256 hash_session_id;
    uint8_t code;
    uint8_t participants;
    uint8_t reserved2;
    uint8_t reserved3;
} cbstatus_t;
```

```
typedef struct cbbroadcast_s
{
    sha256 hash_session_id;
    uint16_t type;
    uint16_t length;
    char * value;
} cbbroadcast_t;
```

```
typedef struct cbleave_s
{
    sha256 hash_session_id;
} cbleave_t;
```

Message Codes

```
typedef enum cbstatus_code_s
{
    cbstatus_code_none = 0,
    cbstatus_code_accepted = 1,
    cbstatus_code_declined = 2,
    cbstatus_code_status = 3,
    cbstatus_code_error = 0xfe,
} cbstatus_code_t;
```

Broadcast Types

```
typedef enum cbbroadcast_type_s
{
    cbbroadcast_type_none = 0,
    cbbroadcast_type_ecdhe = 1,
    cbbroadcast_type_tx = 2,
} cbbroadcast_type_t;
```

Broadcast Structures

```
typedef struct cb_ecdhe_s
{
    key_public public_key;
} cb_ecdhe_t;
```

```
typedef struct cb_tx_s
{
    varint count;
    vector< pair<varint, vector<uint8_t> > > cb_txs_encrypted;
} cb_tx_t;
```

Active Mixing

```
enum { K = 8 };
enum { N = 2 };
```

1. Connect to one of the K closest elected blender relay nodes.
2. Send a cbjoin_t with a null hash_session_id.
3. If the blender node responds with a cbstatus_code_t with a type of cbstatus_code_accepted and a valid hash_session_id continue to step 4. otherwise start over at step 1.
4. Wait for a cbstatus_code_t with a type of cbstatus_code_status with a participants count of at least N.
5. Send a cbbroadcast_type_t with a type of cbbroadcast_type_ecdhe.
6. Wait for all session participants to broadcast their cbbroadcast_type_ecdhe

message.

7. Derive the ECDHE shared secrets for all session participants.
8. Send a cbbroadcast_type_t with a type of cbbroadcast_type_tx containing an encrypted transaction for each session participant.
9. When the transactions are received from all session participants they may be locally blended.
10. Once the transactions have been blended the resulting transaction SHOULD be broadcast to the network and a ZeroTime[2] lock should be placed on it's inputs.
11. Send a cbleave_t and disconnect from the blender relay node.

Client-Side Procedures

Each participant generates the blended transaction as follows:

```
transaction tx_blended;

for (auto & i : unblended_txs)
{
    for (auto & j : i.outs())
    {
        tx_blended.outs().push_back(j);
    }

    for (auto & j : i.ins())
    {
        tx_blended.ins().push_back(j);
    }
}
```

Each participant orders the blended transactions as follows:

```
random_device rd;
mt19937 g(rd());

random_shuffle(tx_blended.ins().begin(), tx_blended.ins().end(),
random_shuffle(tx_blended.outs().begin(), tx_blended.outs().end(
```

Each participant sends a cbleave_t and disconnects from the blender relay node broadcasting both the transaction and a ZeroTime[2] lock on it's inputs. The process MAY repeat for unblended coins.

Passive Mixing

Passive mixing is identical to active mixing with the only difference being that the relay node is one of the N nodes currently connected to each other. Instead of sending the cbjoin_t message to an elected blender relay node it is instead sent to each currently connected node one at a time until a cbstatus_code_accepted is received.

Blended Transactions

The participant to first broadcast and lock their blended transaction will have it respected by the network, all other transactions will be rejected (double spend detected) by the ZeroTime[2] lock.

Handling Failures

If at any point the session participants count drops due to a cbleave_t all clients MUST send a cbleave_t. If at any point during the network procedures should a client-side error occur a cbleave_t MUST be sent. A node may OPTIONALLY start over if a failed session occurs.

Liquidity

If there is a lack of users participating in the system liquidity could become a problem. Having a system in place such as a node incentives[3] can solve a liquidity problem should one arise in a live system. An incentivised node could be rewarded for providing liquidity to the network for blending operations. With passive mixing a liquidity problem is less likely to arise.

Security Considerations

None

Conclusion

With our proposal we have satisfied the requirements which are essential making a fungible and untraceable crypto-currency with a high degree of anonymity with no counter-party risk involved.

Author

John Connor

Public Key:

```
047d3cdc290f94d80ae88fe7457f80090622d064757
9e487a9ad97f77d1c3b3a9e8b596796ebeb23a78214
fc0a95b6a093b3f1d5e2205bd32168ac003f50f4f557
```

Contact:

```
BM-NC49AxAjcqVcF5jNPu85Rb8MJ2d9JqZt
```

References

1. https://en.wikipedia.org/wiki/Fungibility

2. https://raw.githubusercontent.com/john-connor/papers/master/zerotime.pdf
3. https://raw.githubusercontent.com/john-connor/papers/master/node_incentives.pdf
4. https://en.wikipedia.org/wiki/Elliptic_curve_Diffie–Hellman

Draft Revision 01