Transactions with improved anonymity

This is the draft of anonymous transactions design based on@vbuterin's proposal here. Criticism and discussions welcome.

Assets

Anonymous assets are Pedersen hashes of UTXO objects with the following structure:

struct utxo { uint240 balance; uint160 owner; uint160 secret; uint80 salt; }

State

The state is composed of an event-driven list of UTXOs and storage-driven list of Merkle root hashes for each block of 2^{16}

UTXOs, and hashmap(address=>hashmap(uint160=>bool))

of known secrets (may be replaced via zkSNARK-based accumulator).

Also, users can publish messages with UTXO's encrypted via the public key of the receiver.

When the user makes transfer or withdrawal, 2 last root hashes of not filled blocks and two any root hash must be sent inside the snark. In such case the transaction is current from 3 chunks, two has 65536 UTXOs and another one has from 65536 up to 131070 UTXOs.

Deposit

and transfer

transactions create 2 UTXO's inside the list (in deposit

case the 2nd UTXO is zero).

Relayer

As the solution for addresses initialization problem @vbuterin proposed to use relayers. The relayers may accept anonymous transactions data from Tor or I2P networks or somewhere else. In such case the user could hide his real IP.

The fee must be enough for the relayer to process the transaction on Ethereum blockchain and may be measured in any cryptocurrency presented on DEXes. The amount of the fee is not strictly determined at the protocol level and requested by the relayer.

Some UX properties

We need to make the transaction visible for the receiver. I think, one of the best cases is using events with encrypted messages via the receiver's public key.

That's why ethereum uint160 addresses will not working at UX level in general case. The sender needs to know the public key of the receiver. So, the application will use uint257 addresses, corresponding to the points at secp256k1.

The receiver can scan the encrypted messages and make check decrypt(encrypted_owner)==owner

to find his assets.

Deposits

Solidity

function deposit(uint lastHashUpdated, uint utxoHash, bytes proof) external payable returns(bool);

UTXO

Deposit function creates 2 UTXOs in last not filled block and updates the last not filled block's hash.

zkSNARK

public: field lastHash field lastHashUpdated field utxoHash uint240 balance
uint160 owner
private: uint160 secret uint80 salt field[15] merkleProoflastHashBefore field[15] merkleProoflastHashAfter field insertionIndex

Transfers

Solidity

Transfers are managed by relayer. The sender signs (secrets[2], fee)

. We do not need in nonce here, because for each owner the transaction with the same secret may be executed only once.

function transfer(uint[2] rootHashes, uint lastHashUpdated, uint[2] utxoNewHashes, uint160[2] secrets, uint fee, byte[65] signature, bytes proof, bytes ecryptedMessage) external returns(bool);

UTXO

All transactions have 2 inputs and 2 outputs. Following equation must be carried out for the transactions:

sum(unique(inputs)) === sum(outputs) + fee

When the transaction is executed we fixed the secret of spent UTXO's onchain. The user selects the UTXOs inside the blocks via zkSNARK private inputs.

zkSNARK

public: field[4] rootHashes // +2 last hashes from the storage field lastHashUpdated filed[2] utxoNewHashes uint160[2] secrets
uint160 owner field fee private: bit[2][4] selectors field[2][16] merkleProofs field[2] utxoIndexes

{uint240 balance, uint80 salt}[2] utxosOld uint640 utxosNew

field[15] merkleProoflastHashBefore field[15] merkleProoflastHashAfter field insertionIndex

Withdrawals

Solidity

Withdrawals are managed by relayer. The sender signs (secret, fee).

function withdrawal(uint rootHash, uint balance, uint160 secret, uint fee, byte[65] signature, bytes proof) external payable returns(bool);

zkSNARK

public: field[3] rootHashes // +2 last hashes from the storage field lastHashUpdated field utxoHash uint240 balance
uint160 owner
private: uint160 secret uint80 salt field[16] merkleProof filed utxoIndex

Further optimizations

1. Implement snark-based accumulator for known secrets. The design is described at"Shorter Merkle proofs for Snapps".
2. Implement truebit protocol for zkSNARKs verification and time auction for relays for collision resistance. (It will require complication of withdrawals, but the problem is solved).
3. Implement in-snark signatures. EDDSA or ECDSA
4. Usage of more improved zkSNARK friendly hash functions (Does anybody here know something better?)

With such optimizations, the performance and gas cost of transactions will be close to rollup.

Links

- @barryWhiteHat miximus
- @barryWhiteHat rollup

- @vbuterin “We need a first step toward more privacy”
- zDai-mixer