Make an Atomic Swap

Atomic Swap transaction represents an exchange between 2 different tokens on the OMG Network. A standard exchange rate is 1:1 but you can set your own rate as well. Atomic swaps can be useful during trading operations, claiming community tokens, etc.

1. Install `omg-js` , `ethereumjs-util` , `eth-sig-util` , `bn.js` , `bignumber.js`

To access network features from your application, use our official libraries:

Node | Browser | React Native

Requires Node >= 8.11.3 < 13.0.0

```
npm install @omisego/omg-js ethereumjs-util eth-sig-util bn.js bignumber.js
```

Copy

JavaScript (ESNext)

2. Import dependencies, define constants

```
import BN from "bn.js";
import BigNumber from "bignumber.js";
import { ChildChain, OmgUtil } from "@omisego/omg-js";
const ethUtil = require('ethereumjs-util');
const sigUtil = require('eth-sig-util');

const childChain = new ChildChain({
  watcherUrl: watcherUrl,
  watcherProxyUrl: '',
  plasmaContractAddress: plasmaContractAddress
});

// constants
const sender = {
  address: '0x8CB0DE6206f459812525F2BA043b14155C2230C0',
  privateKey: 'CD55F2A7C476306B27315C7986BC50BD81DB4130D4B5CFD49E3EAF9ED1EDE4F7',
  currency: OmgUtil.hexPrefix("0xd92e713d051c37ebb2561803a3b5fbabc4962431")
}

const receiver = {
  address: '0xA9cc140410c2bfEB60A7260B3692dcF29665c254',
  privateKey: 'E4F82A4822A2E6A28A6E8CE44490190B15000E58C7CBF62B4729A3FDC9515FD2',
  currency: OmgUtil.hexPrefix("0x2d453e2a14a00f4a26714a82abfc235c2b2094d5")
}

const feePayer = {
  address: '0x001ebfBd3C6D6855bF4EF1a2Ec7f2a779B1028C2',
  privateKey: '5444fec49a972a1c6264745610ef3c8107980540ff3d732af4c5ddb87c5f03c2',
  currency: OmgUtil.transaction.ETH_CURRENCY
}

const plasmaContractAddress = plasmaContractAddress;
```

Copy

- `watcherUrl` - the Watcher Info URL for defined environment (personal or from OMG Network).
- `plasmaContractAddress` - `CONTRACT_ADDRESS_PLASMA_FRAMEWORK` for defined environment.

The above constants are defined for the Rinkeby environment. If you want to work with the Mainnet, check the Environments page.

3. Create helpers

BigNumber helper

BigNumber helper converts a given amount into a BigNumber. It helps when you want to do an atomic swap between ERC20 tokens that have different decimals. Most of the tokens have 18 decimals by default, however, some tokens have 6 (e.g. USDT), or even 0 decimals.

```
// NOTE: bn.js no longer supports conversion from float numbers, only integers
// convert number into a BigNumber with defined decimals
const amountToBN = (amount, decimals = 18) => {
  const multiplier = new BigNumber(10).pow(decimals);
  const subunit = new BigNumber(amount).times(multiplier).toFixed();
  return new BN(subunit);
}
```

Fee helper

The fee helper retrieves the fee amount you need to pay during an atomic swap.

```
// get a fee amount required for atomic swap
const getFeeAmount = async (currency) => {
  const fees = await childChain.getFees();
  const selectedFee = fees['1'].find(fee => fee.currency === currency);
  return BN.isBN(selectedFee.amount)
    ? selectedFee.amount
    : new BN(selectedFee.amount.toString());
}
```

UTXO helper

UTXO helper filters UTXOs that can be used during an atomic swap. Currently, the OMG Network has a limitation of only 4 inputs and 4 outputs, thus you can have a maximum of 3 payment inputs (sender, receiver, fee payer) and 1 fee input. For simplicity purposes, we select UTXOs that have an amount that is equal to the amount requested in a swap operation.

```
// get utxos that can be used for payment
const getUsableUtxos = async (address, currency, spendAmount, filterEqual) => {
  const utxos = await childChain.getUtxos(address);
  const filteredUtxos = utxos
    .filter(utxo => {
      return utxo.currency.toLowerCase() === currency.toLowerCase();
    })
    .filter(utxo => {
      const amount = BN.isBN(utxo.amount)
        ? utxo.amount
        : new BN(utxo.amount.toString());
      return filterEqual ? amount.eq(spendAmount) : amount.gte(spendAmount);
    })
    .map(utxo => {
      const amount = BN.isBN(utxo.amount)
        ? utxo.amount
        : new BN(utxo.amount.toString());
      return {
        ...utxo, amount
      }
    });
  if (!filteredUtxos.length) {
    console.log(`There are no utxos that can cover a payment for ${spendAmount} '${currency}'`);
  }
  return filteredUtxos;
}
```

UTXO change helper

UTXO change helper checks if the provided UTXO (payment or fee) needs a change. If the change is needed, the helper creates and pushes an additional output to the existing array of transaction outputs.

```
// check utxo for change
const checkUtxoForChange = (address, utxo, amount, transactionBody) => {
  if (!utxo || utxo.length === 0) {
    throw new Error(`No UTXO provided for ${address}`);
  }

  if (transactionBody.outputs.length > 4) {
    throw new Error(`The provided transaction body has 4 outputs. You need to have at least 1 spare output to proceed.`)
  }

  if (utxo.amount.gt(amount)) {
    const changeAmount = utxo.amount.sub(amount);
    const changeOutput = {
      outputType: 1,
      outputGuard: address,
      currency: utxo.currency,
      amount: changeAmount
    }
    transactionBody.outputs.push(changeOutput);
  }
}
```

Signature helper

Signature helper signs and returns the inputs by the sender, receiver, and fee payer.

```
// retrieve signatures by sender and fee payer
const getSignatures = (typedData, sender, receiver, feePayer) => {
  const toSign = OmgUtil.transaction.getToSignHash(typedData);
  const senderSignature = getSignature(toSign, sender);
  const receiverSignature = getSignature(toSign, receiver);
  const feePayerSignature = getSignature(toSign, feePayer);
  return [senderSignature, receiverSignature, feePayerSignature];
}

// retrieve a signature to sign a defined type of input
const getSignature = (toSign, signer) => {
  const signature = ethUtil.ecsign(
    toSign,
    Buffer(signer.privateKey.replace('0x', ''), 'hex')
  );
  return sigUtil.concatSig(signature.v, signature.r, signature.s);
}
```

3. Create a transaction body

To construct an atomic swap transaction, you need to create a custom transaction body. It should contain details about the sender, receiver, fee, fee payer, additional metadata.

```
// create transaction body to make a payment
const createTransactionBody = (
  sender,
  senderUtxo,
  receiver,
  receiverUtxo,
  feePayer,
  feeAmount,
  feeUtxo,
  metadata
) => {

  // encode metadata
  const encodedMetadata = metadata
    ? OmgUtil.transaction.encodeMetadata(metadata)
    : OmgUtil.transaction.NULL_METADATA;

  // construct transaction body
  const transactionBody = {
    inputs: [senderUtxo, receiverUtxo, feeUtxo],
    outputs: [
      {
        outputType: 1,
        outputGuard: receiver.address,
        currency: sender.currency,
        amount: senderUtxo.amount
      },
      {
        outputType: 1,
        outputGuard: sender.address,
        currency: receiver.currency,
        amount: receiverUtxo.amount
      }
    ],
    metadata: encodedMetadata
  }

  // check fee utxo for change
  checkUtxoForChange(feePayer.address, feeUtxo, feeAmount, transactionBody);
  return transactionBody;
}
```

Copy

4. Submit an atomic swap

The process of submitting an atomic swap transaction is the same as submitting any other transaction, except you pass a custom transaction body created earlier. The transaction described below creates an atomic swap of 400 TUSDT (with 6 decimals) into 100 OMG (18 decimals) with an exchange rate of 4:1.

```
// create an atomic swap transaction
async function atomicSwap() {
  // retrieve fees for a given currency
  const feeAmount = await getFeeAmount(feePayer.currency);

  // convert 400 into a BigNumber with 6 decimals: 400000000
  const amount = "400";
  const swapAmountSender = amountToBN(amount, 6);

  // derive fee payer UTXOs usable for payment
  const feeUtxo = await getUsableUtxos(
    feePayer.address,
    feePayer.currency,
    feeAmount,
    false
  );

  // derive sender UTXOs usable for payment
  const senderUtxo = await getUsableUtxos(
    sender.address,
    sender.currency,
    swapAmountSender,
    true
  );

  // get exchange rate for atomic swap operation
  // the exchange rate is 4:1 with 6 decimals => 4000000
  const exchangeRate = amountToBN(4, 6);

  // define swap amount for receiver => 100
  const tempSwapAmountReceiver = swapAmountSender.div(exchangeRate);

  // convert 100 into BigNum with 18 decimals => 100000000000000000000
  const swapAmountReceiver = amountToBN(tempSwapAmountReceiver, 18);

  // derive receiver UTXOs usable for payment
  const receiverUtxo = await getUsableUtxos(
    receiver.address,
    receiver.currency,
    swapAmountReceiver,
    true
  );

  // check how many inputs do you use to cover payment
  if (senderUtxo.length > 1) {
    console.log("You can have only 1 sender UTXO to cover the atomic swap. To proceed with a transaction, please merge t
  }
  if (receiverUtxo.length > 1) {
    console.log("You can have only 1 receiver UTXO to cover the atomic swap. To proceed with a transaction, please merge
  }
  if (feeUtxo.length > 2) {
    console.log("You can have only 2 fees UTXO to cover the atomic swap fee. To proceed with a transaction, please merge
  }
  else {
    // construct a transaction body
    const atomicSwapBody = createTransactionBody(
      sender,
      senderUtxo[0],
      receiver,
      receiverUtxo[0],
      feePayer,
      feeAmount,
      feeUtxo[0],
      "atomic swap"
    );

    // sanitize transaction into the correct typedData format
    const typedData = OmgUtil.transaction.getTypedData(
      atomicSwapBody,
      plasmaContractAddress
    );

    // collect signatures
    const signatures = getSignatures(
```

```
        typedData,
        sender,
        receiver,
        feePayer
      );

      // return encoded and signed transaction ready to be submitted
      const signedTx = childChain.buildSignedTransaction(
        typedData,
        signatures
      );

      // submit a signed transaction to the child chain
      const receipt = await childChain.submitTransaction(signedTx);
      console.log(receipt);
    }
  }
```

Lifecycle

1. A user retrieves the fee amount for a defined fee currency.
2. A user converts the sender's amount to swap into a BigNumber with corresponding decimal numbers.
3. A user selects fee payer UTXOs usable for payment.
4. A user selects sender UTXOs usable for payment.
5. A user retrieves a swappable exchange rate for two tokens (e.g. 1:1, 4:1, etc).
6. A user converts the receiver's amount to swap into a BigNumber with corresponding decimal numbers.
7. A user selects receiver UTXOs usable for payment.
8. If the transaction can't be covered with a defined number of inputs for the sender, receiver, and fee payer, a user should merge the corresponding inputs first.
9. A user constructs a custom transaction body for an atomic swap.
10. A user sanitizes the transaction into the correct typedData format.
11. A user signs sender's, receiver's, and fee payer's inputs with corresponding private keys.
12. A user collects signatures for the sender, receiver, and fee payer inputs.
13. A user encodes and submits the transaction's data to the child chain and the Watcher for validation.
14. If the transaction is valid, the child chain server creates a transaction hash and adds the transaction to a pending block.
15. The child chain bundles the transactions in the block into a Merkle tree and submits its root hash to the `Plasma Framework` contract.
16. The Watcher receives a list of transactions from the child chain and recomputes the Merkle root to check for any inconsistency.