Make a Fee Relay Transfer

Fee relay transfer allows creating a transaction where fees are paid by another address. This can be useful when you split the wallets that contain fees and payment funds. This process requires to construct a custom transaction body and make several manual checks because of certain features of the OMG Network.

Implementation

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

Copy

JavaScript (ESNext)

2. Import dependencies

Make a fee relay transaction on the OMG Network involves using ChildChain `omg-js` object. Here's an example of how to instantiate it:

```
import BN from "bn.js";
import { ChildChain, OmgUtil } from "@omisego/omg-js";

const childChain = new ChildChain({ watcherUrl });
const ethUtil = require('ethereumjs-util');
const sigUtil = require('eth-sig-util');
```

Copy

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

There are several ways to send a transaction on the OMG Network. It's recommended to use the first method but you may want to choose another approach for your specific use case.

3. Define constants

```
// constants
const sender = {
  address: '0x8CB0DE6206f459812525F2BA043b14155C2230C0',
  privateKey: 'CD55F2A7C476306B27315C7986BC50BD81DB4130D4B5CFD49E3EAF9ED1EDE4F7'
}

const receiver = {
  address: '0xA9cc140410c2bfEB60A7260B3692dcF29665c254'
}

const feePayer = {
  address: '0x001ebfBd3C6D6855bF4EF1a2Ec7f2a779B1028C2',
  privateKey: '5444fec49a972a1c6264745610ef3c8107980540ff3d732af4c5ddb87c5f03c2'
}

const transactionToken = '0xd92e713d051c37ebb2561803a3b5fbabc4962431';
const plasmaContractAddress = plasmaContractAddress;
const feeCurrency = OmgUtil.transaction.ETH_CURRENCY;
const amount = "29";
```

Copy

- `plasmaContractAddress` - `CONTRACT_ADDRESS_PLASMA_FRAMEWORK` for defined environment.

4. Create helpers

Fee helper

The fee helper retrieves the fee amount you need to pay during a transfer.

```
// NOTE: bn.js no longer supports conversion from float numbers, only integers
const transferAmount = BN.isBN(amount)
  ? amount
  : new BN(amount.toString());

// get a fee amount required for payment
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 a transaction. Currently, the OMG Network has a limitation of only 4 inputs and 4 outputs, thus you can have a maximum of 3 payment inputs and reserve the last one for a fee input. For simplicity purposes, we select UTXOs that have an amount that is greater or equal to the amount requested in a transaction.

```
// get utxos that can be used for payment
const getUsableUtxos = async (address, currency, spendAmount) => {
  const utxos = await childChain.getUtxos(address);
  return 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 amount.gte(spendAmount);
    })
    .map(utxo => {
      const amount = BN.isBN(utxo.amount)
        ? utxo.amount
        : new BN(utxo.amount.toString());
      return {
        ...utxo, amount
      }
    });
}
```

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 signs and returns the inputs by a sender and fee payer.

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

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

4. Create a transaction body

To construct a relay 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,
  feePayer,
  feeAmount,
  feeUtxo,
  metadata
) => {

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

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

  // check payment and fee utxo for change
  checkUtxoForChange(sender, senderUtxo, transferAmount, transactionBody);
  checkUtxoForChange(feePayer, feeUtxo, feeAmount, transactionBody);
  return transactionBody;
}
```

Copy

5. Submit a relay transaction

The process of submitting a relay transaction is the same as submitting any other transaction, except you pass a custom transaction body created earlier.

```
// create a relay transaction
async function transactionRelay() {
  // retrieve fees for a given currency
  const feeAmount = await getFeeAmount(feeCurrency);

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

  // derive sender UTXOs usable for payment
  const senderUtxos = await getUsableUtxos(
    sender.address,
    transactionToken,
    transferAmount
  );

  // check how many inputs do you use to cover payment
  if (senderUtxos.length !== 1) {
    console.log("This transaction only supports one payment input. To proceed with transaction, please merge the selecte
  } else {
    // construct a transaction body
    const transactionBody = createTransactionBody(
      sender.address,
      senderUtxos[0],
      receiver.address,
      feePayer.address,
      feeAmount,
      feeUtxos[0],
      "relay test"
    );

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

    // collect signatures
    const signatures = getSignatures(
      typedData,
      sender,
      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);
  }
}
```

> NOTE, you can create a custom logic and include up to 3 payment inputs. This sample demonstrates the easiest option where you have only 1 payment input and 1 fee input.

Lifecycle

1. A user retrieves the fee amount for a defined fee currency.
2. A user selects fee payer UTXOs usable for payment.
3. A user selects sender UTXOs usable for payment.
4. If the transaction can't be covered with 1 payment input, a user should merge the inputs first.
5. A user constructs a custom transaction body for a fee relayer transfer.

6. A user sanitizes the transaction into the correct typedData format.
7. A user signs sender's and fee payer's inputs with corresponding private keys.
8. A user collects signatures for sender and fee payer inputs.
9. A user encodes and submits the transaction's data to the child chain and the Watcher for validation.
10. If the transaction is valid, the child chain server creates a transaction hash and adds the transaction to a pending block.
11. The child chain bundles the transactions in the block into a Merkle tree and submits its root hash to the `Plasma Framework` contract.
12. The Watcher receives a list of transactions from the child chain and recomputes the Merkle root to check for any inconsistency.