DeFindex SDK: Fee Bump Transaction Examples

This example demonstrates how to implement **fee-bump transactions** for DeFindex vault operations, allowing a sponsor (such as a wallet or service provider) to pay transaction fees on behalf of users when depositing and withdrawing from DeFindex vaults.

Overview

A **fee-bump transaction** is a Stellar capability that enables one account to pay the transaction fees for an existing signed transaction without requiring the original transaction to be re-signed or re-created. This is particularly useful for wallet providers who want to offer "gasless" transactions to their users.
For more info read:
https://discord.com/channels/897514728459468821/1432786430739877929/1432786430739877929

How Fee-Bump Transactions Work

A fee-bump transaction consists of two parts:

1. **Inner Transaction**: The original transaction envelope with its signature(s) from the user
2. **Outer Transaction**: A fee-bump envelope containing the fee-bump transaction and the signature from the fee account (sponsor)

Transaction Flow

```
┌─────────────────────────────────────────────────────────────┐
│  1. Get unsigned transaction from DeFindex SDK              │
│     ├─ depositToVault() or withdrawShares()                │
│     └─ Returns transaction XDR (unsigned)                  │
└─────────────────────────────────────────────────────────────┘
                           ↓
┌─────────────────────────────────────────────────────────────┐
│  2. Sign inner transaction with user's keypair             │
│     ├─ transaction.sign(userKeypair)                       │
│     └─ Inner transaction is now signed and ready           │
└─────────────────────────────────────────────────────────────┘
                           ↓
┌─────────────────────────────────────────────────────────────┐
│  3. Create fee-bump transaction                            │
│     ├─ Use inner transaction fee (see Fee Rules below)     │
│     ├─ buildFeeBumpTransaction(sponsor, fee, innerTx)     │
│     └─ Wraps the signed inner transaction                  │
└─────────────────────────────────────────────────────────────┘
                           ↓
┌─────────────────────────────────────────────────────────────┐
│  4. Sign fee-bump transaction with sponsor's keypair        │
│     ├─ feeBumpTx.sign(sponsorKeypair)                      │
│     └─ Fee-bump transaction is now ready                   │
└─────────────────────────────────────────────────────────────┘
```

PROF

```
                              ↓
┌─────────────────────────────────────────────────────────────┐
│  5. Submit fee-bump transaction to Stellar network          │
│     ├─ Send signed fee-bump XDR                             │
└─────────────────────────────────────────────────────────────┘
```

How it Works (TL;DR)

- The DeFindex SDK returns an unsigned inner transaction with the necessary operations and fees (Resource + Inclusion).
- The user signs this inner transaction with their keypair.
- The sponsor wraps it in a fee-bump transaction and signs it. Set the fee-bump fee to the inner transaction fee (or higher if you need priority):

```
const innerTxFee = parseInt(transaction.fee);
const feeBumpTx = TransactionBuilder.buildFeeBumpTransaction(
  sponsorKeypair,
  innerTxFee.toString(),
  transaction,
  stellarNetwork
);
```

Project Structure

```
defindex-sdk-deposit-withdraw-fee-bump/
├── src/
│   ├── deposit-fee-bump.ts     # Deposit to vault with fee bump
│   ├── withdraw-fee-bump.ts    # Withdraw from vault with fee bump
│   └── rate-limiter.ts         # Rate limit utility with exponential
backoff
├── package.json
├── tsconfig.json
└── README.md
```

PROF

Installation

```
# Install dependencies
npm install
# or
pnpm install
```

Configuration

Create a `.env` file in the project root with the following variables:

```
# Network Configuration
NETWORK=testnet  # Options: "testnet" or "mainnet" (default: "testnet")

# API Configuration
DEFINDEX_API_KEY=your_api_key_here
DEFINDEX_API_URL=https://api.defindex.io

# Accounts (Secret Keys)
SPONSOR_SECRET=SDXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
CALLER_SECRET=SDYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY

# Vault Configuration
VAULT_ADDRESS=CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Switching Between Testnet and Mainnet

To switch between networks, set the `NETWORK` environment variable in your `.env` file:

- **Testnet (default)**: `NETWORK=testnet`
- **Mainnet**: `NETWORK=mainnet`

When using Mainnet, make sure:

- Your accounts have sufficient XLM balance to cover fees
- You're using mainnet-appropriate vault addresses
- Your API key has access to mainnet endpoints

For mainnet testing, you can use the USDC Soroswap Earn DeFindex Vault `CA2FIPJ7U6BG3N7EOZFI74XPJZOEOD4TYWXFVCIO5VDCHTVAGS6F4UKK`

Getting Testnet Keys

PROF

You can use Stellar Laboratory to generate testnet keypairs or use the Stellar CLI:

```
stellar keys generate --testnet
```

Usage

Running the Deposit Example

```
npm run deposit
# or
pnpm deposit
```

This will:

1. Fetch an unsigned deposit transaction from the DeFindex API
2. Sign it with the caller's keypair (inner transaction)
3. Create and sign a fee-bump transaction with the sponsor's keypair, using the inner transaction fee
4. Submit the fee-bump transaction to the Stellar network (testnet or mainnet based on `NETWORK` env variable)

Running the Withdraw Example

```
npm run withdraw
# or
pnpm withdraw
```

This follows the same flow but for withdrawing assets from a vault.

Current withdraw flow in this repo:

- It first queries the user's vault balance (`getVaultBalance`) to obtain `dfTokens` (shares) and `underlyingBalance` (assets).
- It withdraws by shares using `withdrawShares`, taking the first element of `dfTokens` as the number of shares to withdraw.
- The inner transaction is signed by the caller, then wrapped into a fee-bump signed by the sponsor, using the inner transaction fee for the outer fee.

Example Output (Deposit)

PROF

```
🚀 Starting fee bump deposit example...
📍 Vault Address: CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
👤 Caller: GDXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
💰 Sponsor: GAYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
💵 Amount: 50000000 stroops

📝 Getting unsigned deposit transaction...
✅ Received XDR from API

✍️  Signing inner transaction with caller...
✅ Inner transaction signed

🔄 Creating fee bump transaction...
✅ Using inner transaction fee for fee-bump
✅ Fee bump transaction created and signed by sponsor

📡 Submitting fee bump transaction to network...
✅ Transaction successful!
📊 Response: {
  "txHash": "...",
  "status": "SUCCESS"
}

🔍 View transaction on Stellar Expert:
   https://stellar.expert/explorer/testnet/tx/...
```

Fee-Bump Transaction Requirements

For a fee-bump transaction to be valid, it must meet these conditions:

Fee Requirements

- The outer fee must be ≥ the fee specified in the inner transaction
- The outer fee must be ≥ network minimum fee for total operations (inner + 1)
- For replace-by-fee, the outer fee should be significantly higher (commonly 10x)

Account Requirements

- Fee account (sponsor) must exist on the ledger
- Fee account must have sufficient XLM balance
- Fee account signature must be valid and meet low threshold

Signature Requirements

- Inner transaction signatures remain valid
- Fee-bump transaction requires sponsor signature
- Network passphrase must be part of the transaction hash

Rate Limiting

The example includes a rate limiter utility (`rate-limiter.ts`) that handles API rate limits with exponential backoff:

PROF

```
const response = await withRateLimit(() =>
  defindexSdk.depositToVault(vaultAddress, depositData,
SupportedNetworks.TESTNET)
);
```

Features:

- Automatic retry on 429 (Too Many Requests) errors
- Exponential backoff strategy
- Configurable max retries and initial delay

References

- Stellar Fee-Bump Transactions Documentation
- DeFindex SDK Documentation

- Stellar SDK Documentation