

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 (CAP-0015) 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.

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 withdrawFromVault()
└ 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
└ Calculate dynamic fee
└ 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

↓

5. Submit fee-bump transaction to Stellar network

- └ Send signed fee-bump XDR
- └ Network charges fee account (sponsor)

Key Concepts

1. Inner Transaction Generation

- The original transaction is created by the DeFindex SDK
- Contains the actual operations (deposit, withdraw, etc.)
- Must be signed by the user/caller

2. User Signature

- The user signs the inner transaction with their keypair
- This signature is preserved in the final fee-bump transaction
- No need to re-sign or modify the original transaction

3. Fee-Bump Wrapping

- A new transaction envelope wraps the signed inner transaction
- The wrapper includes the fee account (sponsor)
- Fee calculation must account for both inner and outer transactions

4. Sponsor Payment

- The sponsor's account pays the transaction fees
- The fee account balance must be sufficient to cover the calculated fees
- The original sequence number comes from the inner transaction's source account

5. Dynamic Fee Calculation

```
// Fee bump fee must be at least: innerTxFee + (baseFee * operations)
const minFeeBumpFee = innerTxFee + baseFee * operationCount;

// Can apply network multiplier for priority
const dynamicFee = Math.max(minFeeBumpFee, innerTxFee *
networkMultiplier);
```

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
```

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=SDYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY

# Vault Configuration
VAULT_ADDRESS=CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

PROF

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
[CA2FIPJ7U6BG3N7E0ZFI74XPJZ0E0D4TYWXFVCI05VDCHTVAGS6F4UKK](#)

Getting Testnet Keys

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
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.




Example Output


```
🚀 Starting fee bump deposit example...  
📌 Vault Address: CAXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
👤 Caller: GDXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX  
💰 Sponsor: GAYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY  
💵 Amount: 50000000 stroops  
  
📝 Getting unsigned deposit transaction...  
✅ Received XDR from API
```

```
 Signing inner transaction with caller...
 Inner transaction signed

 Calculating dynamic fee...
  Inner transaction fee: 100 stroops
  Base fee: 100
  Operations: 1
  Network multiplier: 10
  Calculated fee bump fee: 1100 stroops

 Creating fee bump transaction...
 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 fee must be \geq network minimum fee for total operations (inner + 1)
- The fee must be \geq the fee specified in the inner transaction
- For replace-by-fee, the fee must be 10x higher than the first transaction

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:

```
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](#)