https://pastebin.mozilla.org/k03WgtEQ

TypoScript Expires in: 6 days, 23 hours | Delete Now Raw Slim Copy Snippet Edit Snippet Wordwrap

```
import {
  hash256,
  importAuthenticationTemplate,
  authenticationTemplateToCompilerBCH,
  addressContentsToLockingBytecode,
  lockingBytecodeToCashAddress,
  binToHex,
  hexToBin,
  Input,
  Output,
  cashAddressToLockingBytecode,
  encodeLockingBytecodeP2sh32,
  sha256,
  utf8ToBin,
  generateTransaction,
  decodePrivateKeyWif,
  encodeTransaction,
  encodeCashAddress,
  secp256k1,
  hash160,
  compilerOperationAddressData,
  generateScenarioBCH,
  // generatePrivateKey
} from "@bitauth/libauth";
import type {
  CompilationData,
  AuthenticationTemplate,
  TransactionCommon,
  WalletImportFormatType,
} from "@bitauth/libauth";
```

```
import {
  p2pkhTemplate,
  localCryptosTemplate,
  // localCryptos,
} from "./templates.js";
const log = console.log;

const arbiter = decodePrivateKeyWif(
  "cSKRmn9YTe9fay59xhqcQeiuPprV3JQVpFstoJmtHkQfAX8W35TP"
) as {
  privateKey: Uint8Array;
  type: WalletImportFormatType;
};
const seller = decodePrivateKeyWif(
  "cUSZ2kZnkwfNwJFBGUsKHjoJLHxjzi4Wkw9PGEbTZoGE4ctbnFgf"
) as {
  privateKey: Uint8Array; // const localCryptosRedeem =
  type: WalletImportFormatType;
};
const buyer = decodePrivateKeyWif(
  "cP3MMLh52HfgFLsw5ffy6LgBsmqt5bJTAt3j1zwWCfiZivU8rD6e"
) as {
  privateKey: Uint8Array;
  type: WalletImportFormatType;
};


// const pubkey = secp256k1.derivePublicKeyCompressed(arbiter.privateKey);

const template = importAuthenticationTemplate(localCryptosTemplate);
// const template = importAuthenticationTemplate(p2pkhTemplate);
const compiler = authenticationTemplateToCompilerBCH(
  template as AuthenticationTemplate
);

const escrowLockingBytecode = compiler.generateBytecode({
```

```
  scriptId: "escrow_output_template",
  data: {
    keys: {
      privateKeys: {
        arbiter_key: arbiter.privateKey,
        seller_key: seller.privateKey,
        buyer_key: buyer.privateKey,
      },
    },
  },
});

/* const p2pkhLockingBytecode = compiler.generateBytecode({
  scriptId: "lock",
  data: {
    keys: {
      privateKeys: {
        arbiter_key: arbiter.privateKey,
      },
    },
  },
});
 */
/* log(
  escrowLockingBytecode.success
    ? lockingBytecodeToCashAddress(escrowLockingBytecode.bytecode,'bchtest')
    // ? binToHex(escrowLockingBytecode.bytecode)
    : escrowLockingBytecode.errors
); */

const someInput: Input = {
  outpointIndex: 0,
  outpointTransactionHash: hexToBin(
    "2bd28dc59857659d03bb1cd90493ec776e31ae3ff874b5d24a568a41bb323b7c"
  ),
  sequenceNumber: 0xffffffff,
```

```
  unlockingBytecode: Uint8Array.from([]),
};
const satsAvailable: bigint = 8800n;

const someOutput: Output = {
  // lockingBytecode: p2sh32Output,
  lockingBytecode: hexToBin(
    // "a91438977640c921680dbf9b7bfd51eb61b43091bb0187"
    "76a914621ef47d56a16a4806c7a2120fa5db45fd466fde88ac"
  ),
  valueSatoshis: satsAvailable - 600n,
  // valueSatoshis: 10000n,
};
const changeOut: Output = {
  lockingBytecode: hexToBin(
    "76a914621ef47d56a16a4806c7a2120fa5db45fd466fde88ac"
  ),
  // valueSatoshis:satsAvailable - (someOutput.valueSatoshis + 1000n)
  valueSatoshis: satsAvailable - (someOutput.valueSatoshis + /* fee */ 600n),
};

/* const p2pkhInput = {
  outpointIndex: someInput.outpointIndex,
  outpointTransactionHash: someInput.outpointTransactionHash,
  sequenceNumber: 0,
  unlockingBytecode: {
    compiler,
    data: {
      keys: { privateKeys: { key: arbiter.privateKey } },
    },
    valueSatoshis: BigInt(satsAvailable),
    script: "unlock",
    // token: libAuthToken,
  },
};
 */
```

```
const escrowInput = {
  outpointIndex: someInput.outpointIndex,
  outpointTransactionHash: someInput.outpointTransactionHash,
  sequenceNumber: 0,
  unlockingBytecode: {
    compiler,
    data: {
      keys: {
        privateKeys: {
          arbiter_key: arbiter.privateKey,
          seller_key: seller.privateKey,
          buyer_key: buyer.privateKey,
        },
      },
    },
    valueSatoshis: BigInt(satsAvailable),
    script: "buyer_return",
    // token: libAuthToken,
  },
};


const inputWithScript = {
  outpointIndex: someInput.outpointIndex,
  outpointTransactionHash: someInput.outpointTransactionHash,
  sequenceNumber: 0xffffffff,
  unlockingBytecode: escrowInput.unlockingBytecode,
};
const transaction = generateTransaction({
  inputs: [inputWithScript],
  locktime: 0,
  outputs: [someOutput /* changeOut */],
  version: 2,
});

// log(lockingBytecodeToCashAddress(changeOut.lockingBytecode, "bchtest"));
```

```
if (transaction.success) {
  log("\n", binToHex(encodeTransaction(transaction.transaction)));
} else {
  log(transaction.errors[0]);
}
log(transaction /* .success */);
```