ERTP ⅄ Orchestration

Enable offer safety for orchestrated assets

https://github.com/Agoric/agoric-sdk/pull/10504

Friday 20 December 2024

Common Scenario

Three realizations

Alice “starts” with 300 bucks.
Alice pays Bob 100 bucks.
Alice burns 2 bucks.

1. (mock) Orchestration API
2. (mock) ERTP on fresh Orch purse/acct & denom/mint
3. (mock) ERTP window on existing Orch acct & denom

Orchestration API

Alice “starts” with 300 bucks.

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

Orchestration API

+ orchAdmin for minting orch assets

Alice “starts” with 300 bucks.

```
const { orchestrator, admin: orchAdmin } = mockOrchestratorKit();
const aChain = await orchestrator.getChain('a');
const aliceAcct = await aChain.makeAccount();
const bChain = await orchestrator.getChain('b');
const bobAcct = await bChain.makeAccount();

const bucksDenomMint = await orchAdmin.makeDenom('bucks', aChain);
await bucksDenomMint.mintTo(aliceAcct.getAddress(), {
  denom: 'bucks',
  value: 300n,
});
```

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

Orchestration API

```
//…
await bucksDenomMint.mintTo(aliceAcct.getAddress(), {
  denom: 'bucks',
  value: 300n,
});
```

Alice pays Bob 100 bucks.

```
await aliceAcct.transfer(bobAcct.getAddress(), {
  denom: 'bucks',
  value: 100n,
});
```

Alice burns 2 bucks.

```
await aliceAcct.transfer(ZERO_ADDR, {
  denom: 'bucks',
  value: 2n,
});
```

ERTP on fresh Orch

Alice “starts” with 300 bucks.

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

ERTP on fresh Orch

Alice “starts” with 300 bucks.

```
const { orchestrator, admin: orchAdmin } = mockOrchestratorKit();
const ertpOrch = mockERTPOrchestrator(orchestrator, orchAdmin);
const aChain = await orchestrator.getChain('a');
const bucksKit = ertpOrch.provideIssuerKitForDenom('bucks', ‘a’);
```

```
const aliceBucksPurse = await bucksIssuer.makeEmptyPurse();
const bobBucksPurse = await bucksIssuer.makeEmptyPurse();

const initBucksPayment = await bucksMint.mintPayment(bucks(300n));
await aliceBucksPurse.deposit(initBucksPayment);
```

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

ERTP on fresh Orch

```
//…
const initBucksPayment = await bucksMint.mintPayment(bucks(300n));
await aliceBucksPurse.deposit(initBucksPayment);
```

Alice pays Bob 100 bucks.

```
const alicePaysBob = await aliceBucksPurse.withdraw(bucks(100n));
await bobBucksPurse.deposit(alicePaysBob)
```

Alice burns 2 bucks.

```
const aliceFirewood = await aliceBucksPurse.withdraw(bucks(2n));
await bucksIssuer.burn(aliceFirewood);
```

ERTP window on existing Orch

Alice “starts” with 300 bucks.

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

ERTP window on existing Orch

Alice “starts” with 300 bucks.

```
const { orchestrator, admin: orchAdmin } = mockOrchestratorKit();
const aChain = await orchestrator.getChain('a');
const aliceAcct = await aChain.makeAccount();
const bChain = await orchestrator.getChain('b');
const bobAcct = await bChain.makeAccount();

const bucksDenomMint = await orchAdmin.makeDenom('bucks', aChain);
await bucksDenomMint.mintTo(aliceAcct.getAddress(), {
  denom: 'bucks',
  value: 300n,
});
```

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

ERTP window on existing Orch

Alice “starts” with 300 bucks.

```
//…
await bucksDenomMint.mintTo(aliceAcct.getAddress(), {
  denom: 'bucks',
  value: 300n,
});

const ertpOrch = mockERTPOrchestrator(orchestrator, ~~orchAdmin~~);
```

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

ERTP window on existing Orch

Alice “starts” with 300 bucks.

```
//…
await bucksDenomMint.mintTo(aliceAcct.getAddress(), {
  denom: 'bucks',
  value: 300n,
});

const ertpOrch = mockERTPOrchestrator(orchestrator);
const bucksKit = ertpOrch.provideIssuerKitForDenom('bucks', ‘a’);

const aliceBucksPurse = await bucksAdmin.makePurse(aliceAcct);
const bobBucksPurse = await bucksAdmin.makePurse(bobAcct);
```

Alice pays Bob 100 bucks.

Alice burns 2 bucks.

ERTP window on existing Orch

```
//…
const ertpOrch = mockERTPOrchestrator(orchestrator);
const bucksKit = ertpOrch.provideIssuerKitForDenom('bucks', ‘a’);

const aliceBucksPurse = await bucksAdmin.makePurse(aliceAcct);
const bobBucksPurse = await bucksAdmin.makePurse(bobAcct);
```

Alice pays Bob 100 bucks.

```
const alicePaysBob = await aliceBucksPurse.withdraw(bucks(100n));
await bobBucksPurse.deposit(alicePaysBob)
```

Alice burns 2 bucks.

```
const aliceFirewood = await aliceBucksPurse.withdraw(bucks(2n));
await bucksIssuer.burn(aliceFirewood);
```

Lazy bijection

```
getDenomForBrand(brand) {
  const { brand2Denom } = this.state;
  return brand2Denom.get(brand);
},
async provideIssuerKitForDenom(//…
```

```
async provideIssuerKitForDenom(
  denom,
  defaultChainName = 'defaultChainName',
) {
  const { orchestrator, denom2IssuerKit, brand2Denom, optOrchAdmin } =
    this.state;
  await null;
  if (!denom2IssuerKit.has(denom)) {
    let issuerKitPlus;
    try {
      // TODO should be a better way than try/catch to determine
      // if a denom already exists
      const { chain } = await E(orchestrator).getDenomInfo(denom);
      issuerKitPlus = mockIssuerKit(denom, chain);
    } catch (err) {
      if (optOrchAdmin === undefined) {
        throw Fail`Cannot mint without orchestrator admin facet`;
      }
      const chain = await E(orchestrator).getChain(defaultChainName);
      const denomMint = await E(optOrchAdmin).makeDenom(denom, chain);
      issuerKitPlus = mockIssuerKit(denom, chain, denomMint);
    }
    denom2IssuerKit.init(denom, issuerKitPlus);
    brand2Denom.init(issuerKitPlus.brand, denom);
  }
  return denom2IssuerKit.get(denom);
},
```

```
async provideIssuerKitForDenom(
  denom,
  defaultChainName = 'defaultChainName',
) {
  const { orchestrator, denom2IssuerKit, brand2Denom, optOrchAdmin } =
    this.state;
  await null;
  if (!denom2IssuerKit.has(denom)) {
    let issuerKitPlus;
    try {
      // TODO should be a better way than try/catch to determine
      // if a denom already exists
      const { chain } = await E(orchestrator).getDenomInfo(denom);
      issuerKitPlus = mockIssuerKit(denom, chain);
    } catch (err) {
      if (optOrchAdmin === undefined) {
        throw Fail`Cannot mint without orchestrator admin facet`;
      }
      const chain = await E(orchestrator).getChain(defaultChainName);
      const denomMint = await E(optOrchAdmin).makeDenom(denom, chain);
      issuerKitPlus = mockIssuerKit(denom, chain, denomMint);
    }
    denom2IssuerKit.init(denom, issuerKitPlus);
    brand2Denom.init(issuerKitPlus.brand, denom);
  }
  return denom2IssuerKit.get(denom);
},
```

```
async provideIssuerKitForDenom(
  denom,
  defaultChainName = 'defaultChainName',
) {
  const { orchestrator, denom2IssuerKit, brand2Denom, optOrchAdmin } =
    this.state;
  await null;
  if (!denom2IssuerKit.has(denom)) {
    let issuerKitPlus;
    try {
      // TODO should be a better way than try/catch to determine
      // if a denom already exists
      const { chain } = await E(orchestrator).getDenomInfo(denom);
      issuerKitPlus = mockIssuerKit(denom, chain);
    } catch (err) {
      if (optOrchAdmin === undefined) {
        throw Fail`Cannot mint without orchestrator admin facet`;
      }
      const chain = await E(orchestrator).getChain(defaultChainName);
      const denomMint = await E(optOrchAdmin).makeDenom(denom, chain);
      issuerKitPlus = mockIssuerKit(denom, chain, denomMint);
    }
    denom2IssuerKit.init(denom, issuerKitPlus);
    brand2Denom.init(issuerKitPlus.brand, denom);
  }
  return denom2IssuerKit.get(denom);
},
```