TxPipe SHOP

Audit Report

13th September 2024

Strike - Forwards

Contents

1 - Summary

This report offers a thorough audit of the Strike Forwards protocol, which allows the decentralized creation of a binding agreement between two parties to exchange specified assets at a future date for a price agreed upon at the contract's creation.

The audit is conducted without warranties or guarantees of the quality or security of the code. It's important to note that this report only covers identified issues, and we do not claim to have detected all potential vulnerabilities.

1.a - Overview

The Forwards Protocol enables traders to establish short positions between two assets, anticipating that the price will rise. This allows them to acquire assets below the future market price.

Users can create Forwards by depositing collateral tokens and possibly Strike tokens. Then, another user can enter these Forwards by depositing their collateral tokens and some strike tokens. Afterward, for each Forward that isn't expired, the two involved parties can retire their collateral by depositing the tokens that they want to exchange. Otherwise, for the expired Forwards the remaining party can get the total amount of locked collaterals. In this last action, the Strike token locked in each Forward will be burnt.

1.b - Process

Our audit process involved a thorough examination of Strike Forwards validators. Areas vulnerable to potential security threats were closely scrutinized, including those where attackers could exploit the validator's functions to disrupt the platform and its users. This included evaluating potential risks such as the creation of a forward with a past date, or the liquidation with an upcoming date, amoung others. This also included the common vulnerabilities such as double satisfaction and minting policy vulnerabilities.

Findings and feedback from the audit were communicated regularly to the Strike team through Discord. Diagrams illustrating the necessary transaction structure for proper interaction with Strike are attached as part of this report. The Strike team addressed these issues in an efficient and timely manner, enhancing the overall security of the platform.

2 - Specification

2.a - UTxOs

2.a.a - Forward UTxO

One UTxO for each created Forward. Where the forward created locks collateral tokens, possible strike tokens and a forward control token.

- Address: Hash of validators/forwards:spend parameterized on the `collateral` validator validators/collateral, the asset name of the forward control token and the strike staking credential.

All Forwards in the protocol have the same address.

- Value:
  - Min ADA
  - Collateral tokens
  - 1 Forward token:
    - PolicyId: validators/forwards:mint
    - TokenName: forwards validator's parameter
- Datum: lib/types:ForwardsDatum

- Ref Script: None

2.a.b - Collateral UTxO

One UTxO for each accepted Forward. Where the collateral created locks both collaterals and possible strike tokens.

- Address: Hash of validators/collateral:spend parameterized on the `agreeement` validator validators/agreement, the `liquidate` validator validators/always_fail, a fix strike address and the strike staking credential.

All Collaterals in the protocol have the same address.

- Value:
  - Min ADA
  - Collateral tokens
  - 2 Forward token:
    - PolicyId: validators/forwards:mint
    - TokenName: forwards validator's parameter
- Datum: lib/types:CollateralDatum
- Ref Script: None

2.a.c - Agreement UTxO

Two UTxO for each completed Forward. Where each agreement UTxO will have the deposited tokens and a forward control token.

- Address: Hash of validators/agreement:spend.

All Agreements in the protocol have the same address.
- Value:
  - Min ADA
  - Deposited tokens
  - 1 Forward token:
    - PolicyId: validators/forwards:mint
    - TokenName: forwards validator's parameter
- Datum: lib/types:AgreementDatum
- Ref Script: None

2.a.d - Liquidate UTxO

One UTxO for each liquidated Forward. Where each agreement UTxO will have strike tokens.

- Address: Hash of validators/always_fail:spend.

All Liquidates in the protocol have the same address.
- Value:
  - Min ADA
  - Strike tokens
- Datum: None
- Ref Script: None

2.b - Scripts

2.b.a - Forwards script
- **Parameters**: AssetName, Collateral script, Strike Stake Credential
- **Purpose**: Spend

  - **Redeemer**: lib/types:ForwardsRedeemer
  - **Datum**: lib/types:ForwardsDatum
- **Purpose**: Mint
  - **Redeemer**: lib/types:MintRedeemer

2.b.b - Collateral script

- **Parameters**: Agreement script, Liquidate script, Strike Address, Strike Stake Credential
- **Purpose**: Spend
  - **Redeemer**: lib/types:CollateralRedeemerAction
  - **Datum**: lib/types:CollateralDatum

2.b.c - Agreement script

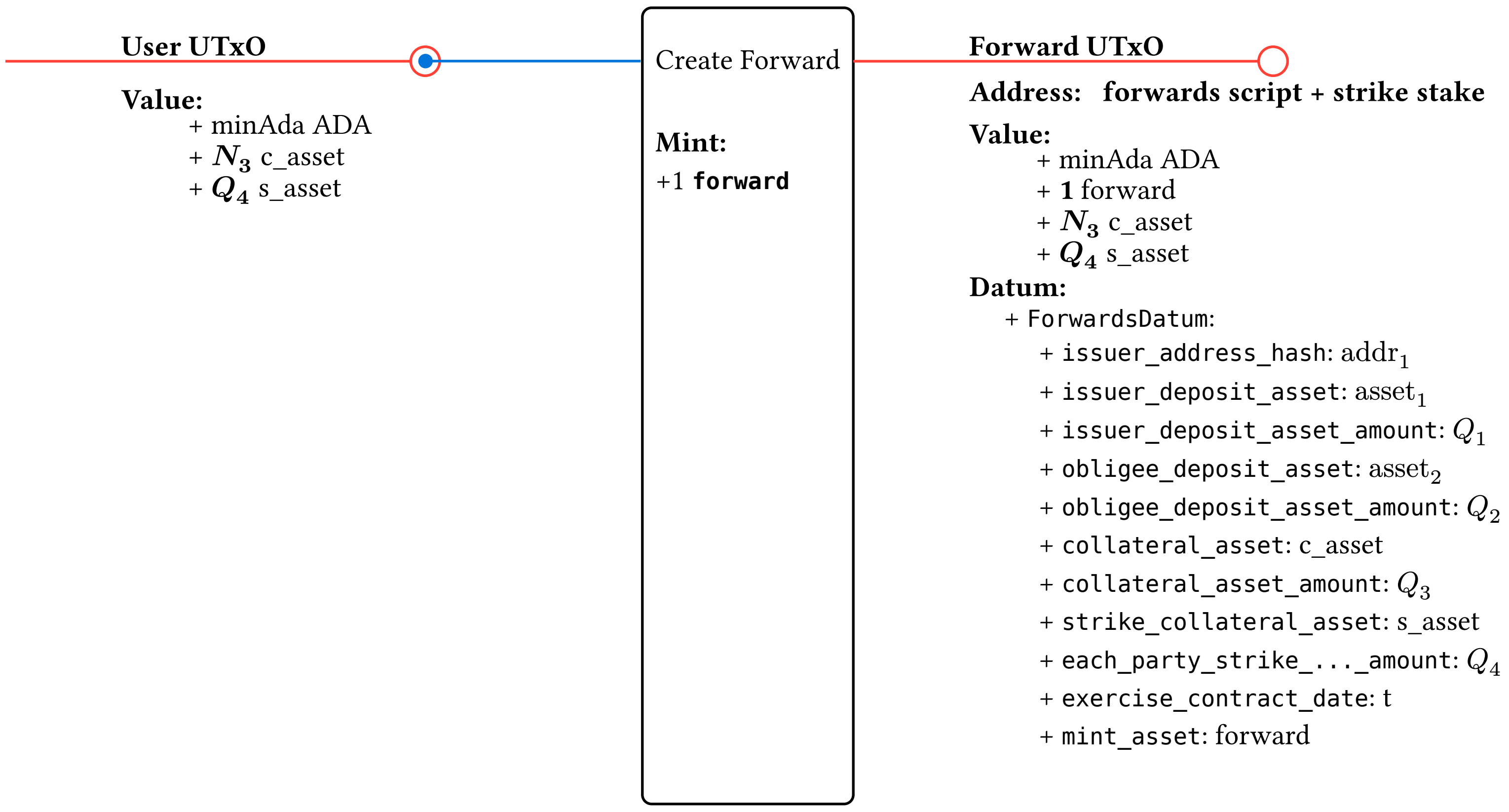
- **Parameters**: None
- **Purpose**: Spend
  - **Redeemer**: Int (Not used)
  - **Datum**: lib/types:AgreementDatum

2.b.d - Liquidate script

- **Parameters**: None
- **Purpose**: Spend
  - **Redeemer**: Int (Not used)
  - **Datum**: Data

2.c - Transaction

2.c.a - Create Forward



User UTxO

Value:
+ minAda ADA
+ N_3 c_asset
+ Q_4 s_asset

Create Forward

Mint:
+1 **`forward`**

Forward UTxO

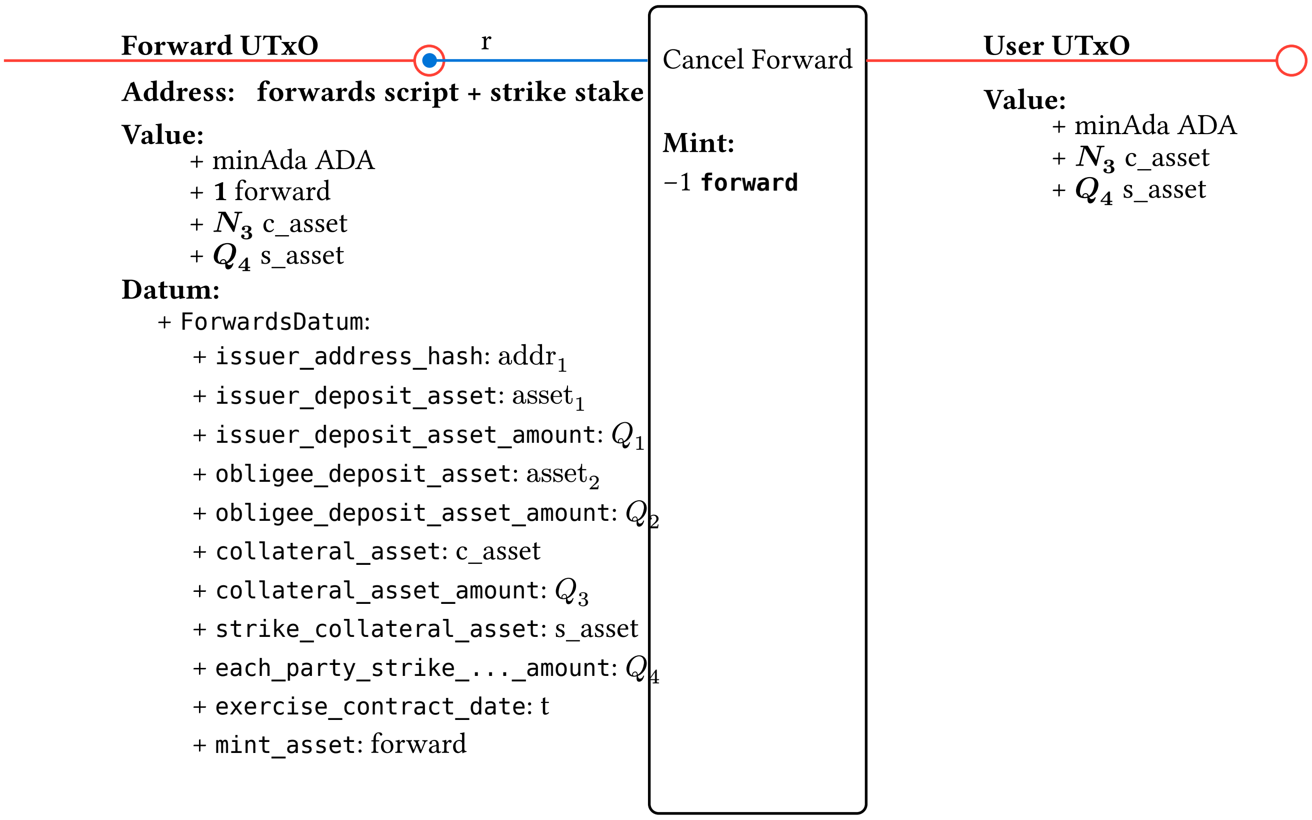
Address: forwards script + strike stake

Value:
+ minAda ADA
+ **1** forward
+ N_3 c_asset
+ Q_4 s_asset

Datum:
+ `ForwardsDatum`:
  + `issuer_address_hash`: addr_1
  + `issuer_deposit_asset`: asset_1
  + `issuer_deposit_asset_amount`: Q_1
  + `obligee_deposit_asset`: asset_2
  + `obligee_deposit_asset_amount`: Q_2
  + `collateral_asset`: c_asset
  + `collateral_asset_amount`: Q_3
  + `strike_collateral_asset`: s_asset
  + `each_party_strike_..._amount`: Q_4
  + `exercise_contract_date`: t
  + `mint_asset`: forward

Note:
$N_3 \geq Q_3$
`forward` minting redeemer: `CreateForwardMint`

2.c.b - Cancel Forward


Forward UTxO
r
Cancel Forward
User UTxO
Address: forwards script + strike stake
Value:
+ minAda ADA
+ **1** forward
+ $\boldsymbol{N_3}$ c_asset
+ $\boldsymbol{Q_4}$ s_asset
Datum:
+ `ForwardsDatum`:
+ `issuer_address_hash`: addr_1
+ `issuer_deposit_asset`: asset_1
+ `issuer_deposit_asset_amount`: Q_1
+ `obligee_deposit_asset`: asset_2
+ `obligee_deposit_asset_amount`: Q_2
+ `collateral_asset`: c_asset
+ `collateral_asset_amount`: Q_3
+ `strike_collateral_asset`: s_asset
+ `each_party_strike_..._amount`: Q_4
+ `exercise_contract_date`: t
+ `mint_asset`: forward
Mint:
−1 **`forward`**
Value:
+ minAda ADA
+ $\boldsymbol{N_3}$ c_asset
+ $\boldsymbol{Q_4}$ s_asset

Note:

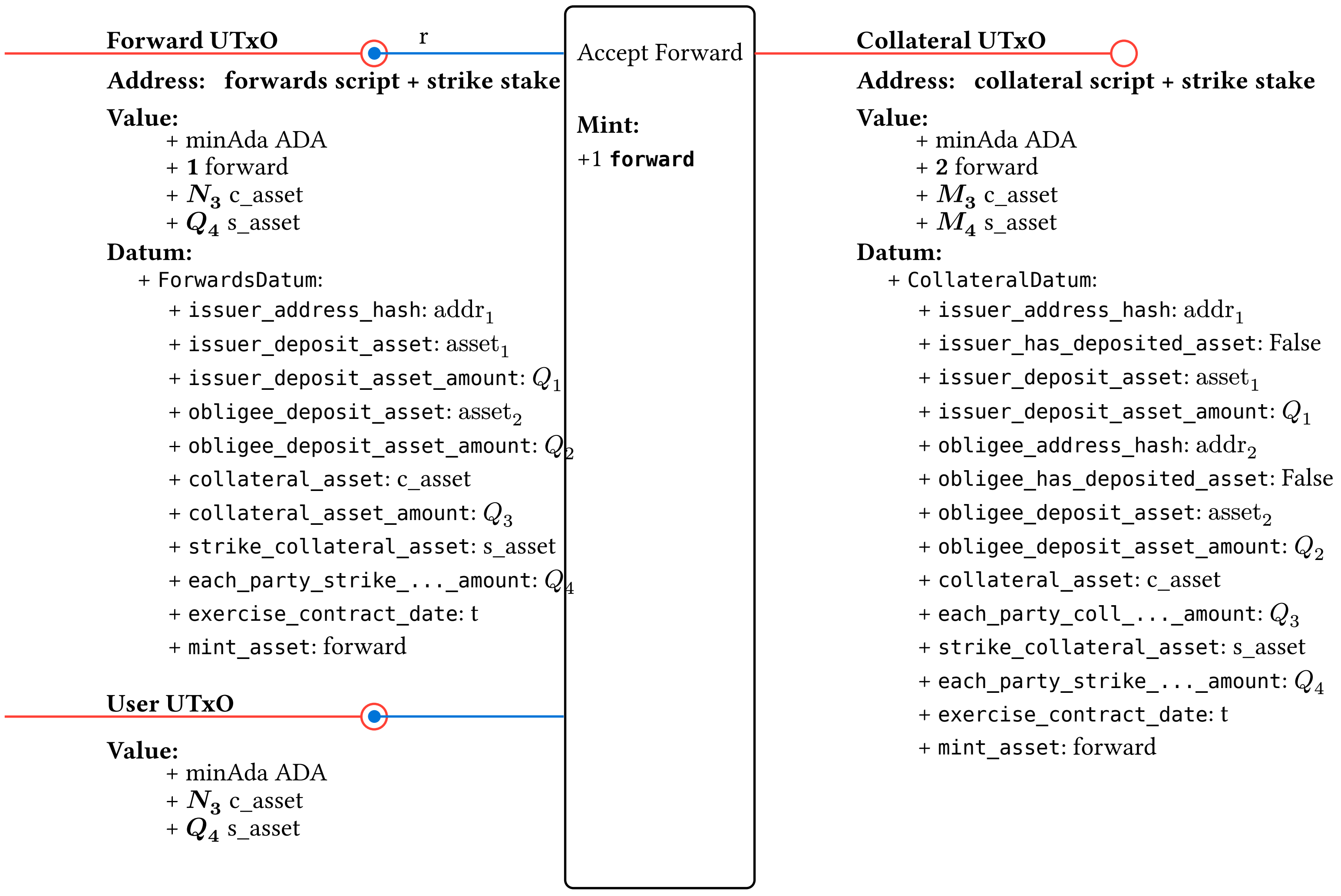
$N_3 \geq Q_3$

r = CancelForwardsContract

The transaction must be signed by `issuer_address_hash`

`forward` burning redeemer: `SingleBurn`

2.c.c - Accept Forward



Forward UTxO

Address: forwards script + strike stake

Value:
+ minAda ADA
+ **1** forward
+ $\boldsymbol{N_3}$ c_asset
+ $\boldsymbol{Q_4}$ s_asset

Datum:
+ ForwardsDatum:
  + issuer_address_hash: addr_1
  + issuer_deposit_asset: asset_1
  + issuer_deposit_asset_amount: Q_1
  + obligee_deposit_asset: asset_2
  + obligee_deposit_asset_amount: Q_2
  + collateral_asset: c_asset
  + collateral_asset_amount: Q_3
  + strike_collateral_asset: s_asset
  + each_party_strike_..._amount: Q_4
  + exercise_contract_date: t
  + mint_asset: forward

User UTxO

Value:
+ minAda ADA
+ $\boldsymbol{N_3}$ c_asset
+ $\boldsymbol{Q_4}$ s_asset

Accept Forward

Mint:
+1 **forward**

Collateral UTxO

Address: collateral script + strike stake

Value:
+ minAda ADA
+ **2** forward
+ $\boldsymbol{M_3}$ c_asset
+ $\boldsymbol{M_4}$ s_asset

Datum:
+ CollateralDatum:
  + issuer_address_hash: addr_1
  + issuer_has_deposited_asset: False
  + issuer_deposit_asset: asset_1
  + issuer_deposit_asset_amount: Q_1
  + obligee_address_hash: addr_2
  + obligee_has_deposited_asset: False
  + obligee_deposit_asset: asset_2
  + obligee_deposit_asset_amount: Q_2
  + collateral_asset: c_asset
  + each_party_coll_..._amount: Q_3
  + strike_collateral_asset: s_asset
  + each_party_strike_..._amount: Q_4
  + exercise_contract_date: t
  + mint_asset: forward

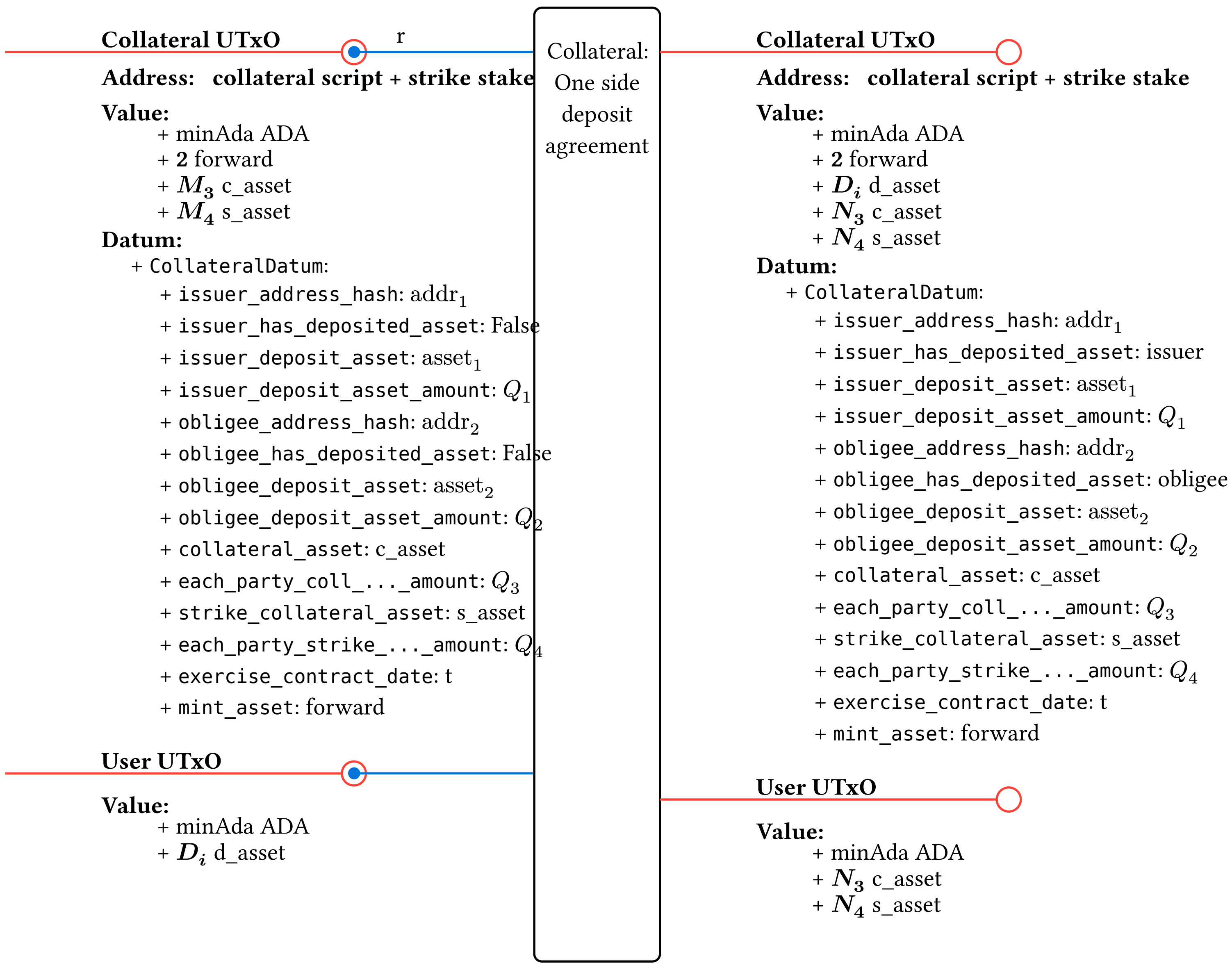
Note:

$M_i \geq 2 * Q_i$

r = AcceptForwardsContract(addr_2)

forward minting redeemer: EnterForwardMint

2.c.d - Collateral: One side deposit agreement


Collateral UTxO
r
Address: collateral script + strike stake
Value:
+ minAda ADA
+ 2 forward
+ M_3 c_asset
+ M_4 s_asset
Datum:
+ CollateralDatum:
+ issuer_address_hash: addr_1
+ issuer_has_deposited_asset: False
+ issuer_deposit_asset: asset_1
+ issuer_deposit_asset_amount: Q_1
+ obligee_address_hash: addr_2
+ obligee_has_deposited_asset: False
+ obligee_deposit_asset: asset_2
+ obligee_deposit_asset_amount: Q_2
+ collateral_asset: c_asset
+ each_party_coll_..._amount: Q_3
+ strike_collateral_asset: s_asset
+ each_party_strike_..._amount: Q_4
+ exercise_contract_date: t
+ mint_asset: forward
User UTxO
Value:
+ minAda ADA
+ D_i d_asset
Collateral: One side deposit agreement
Collateral UTxO
Address: collateral script + strike stake
Value:
+ minAda ADA
+ 2 forward
+ D_i d_asset
+ N_3 c_asset
+ N_4 s_asset
Datum:
+ CollateralDatum:
+ issuer_address_hash: addr_1
+ issuer_has_deposited_asset: issuer
+ issuer_deposit_asset: asset_1
+ issuer_deposit_asset_amount: Q_1
+ obligee_address_hash: addr_2
+ obligee_has_deposited_asset: obligee
+ obligee_deposit_asset: asset_2
+ obligee_deposit_asset_amount: Q_2
+ collateral_asset: c_asset
+ each_party_coll_..._amount: Q_3
+ strike_collateral_asset: s_asset
+ each_party_strike_..._amount: Q_4
+ exercise_contract_date: t
+ mint_asset: forward
User UTxO
Value:
+ minAda ADA
+ N_3 c_asset
+ N_4 s_asset

Note:

$$M_i \geq 2 * N_i$$

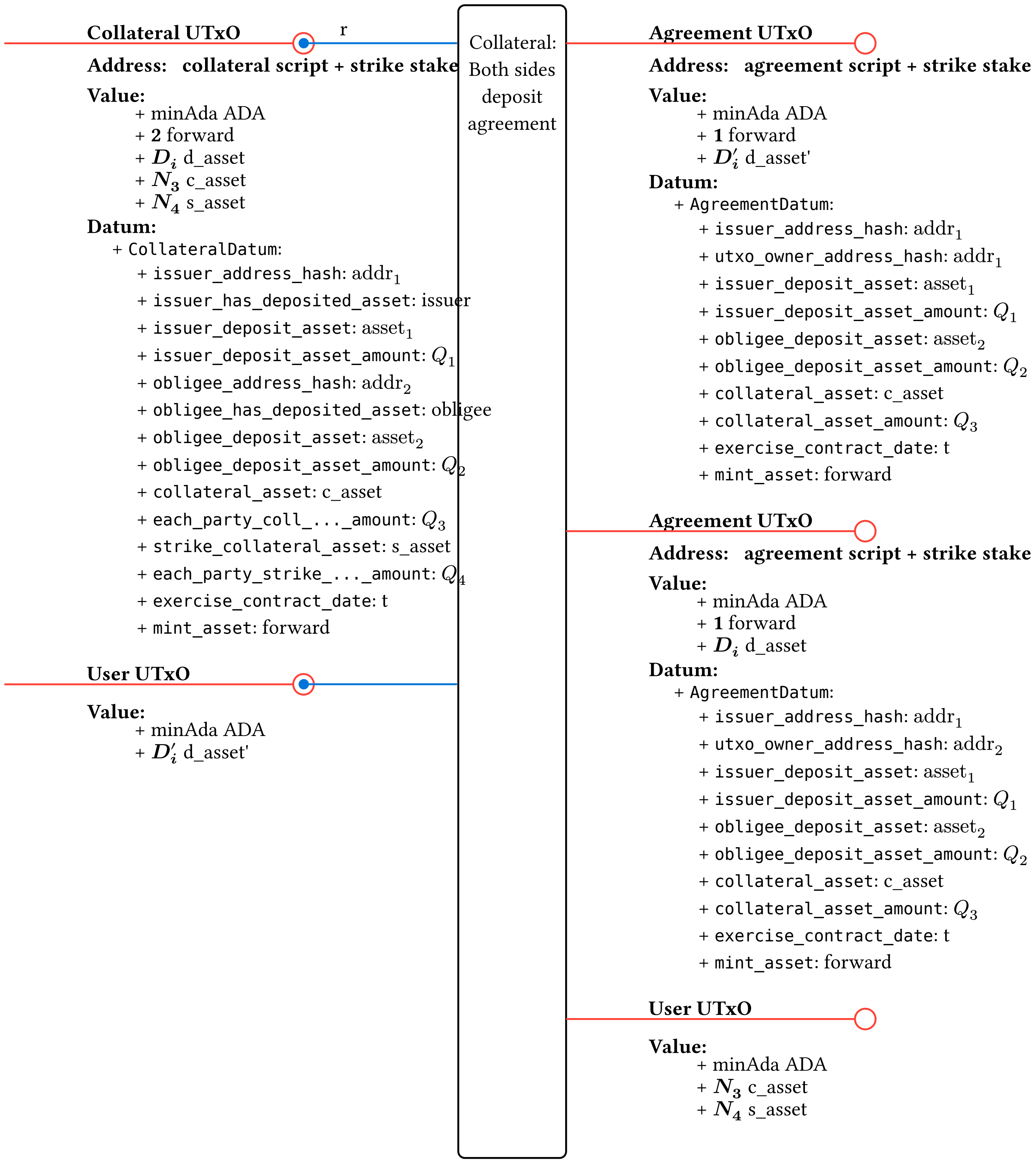
`r = OneSideDepositAgreement(n)`

`(issuer, obligee) = if n == 0 then (True,False) else (False,True)`

$D_i \geq$ (`if n == 0 then` Q_1 `else` Q_2)

`d_asset = if n == 0 then` asset_1 `else` asset_2

2.c.e - Collateral: Both sides deposit agreement


Collateral UTxO
r
Address: **collateral script + strike stake**
Value:
+ minAda ADA
+ **2** forward
+ D_i d_asset
+ N_3 c_asset
+ N_4 s_asset
Datum:
+ CollateralDatum:
+ issuer_address_hash: addr_1
+ issuer_has_deposited_asset: issuer
+ issuer_deposit_asset: asset_1
+ issuer_deposit_asset_amount: Q_1
+ obligee_address_hash: addr_2
+ obligee_has_deposited_asset: obligee
+ obligee_deposit_asset: asset_2
+ obligee_deposit_asset_amount: Q_2
+ collateral_asset: c_asset
+ each_party_coll_..._amount: Q_3
+ strike_collateral_asset: s_asset
+ each_party_strike_..._amount: Q_4
+ exercise_contract_date: t
+ mint_asset: forward
User UTxO
Value:
+ minAda ADA
+ D_i' d_asset'
Collateral: Both sides deposit agreement
Agreement UTxO
Address: **agreement script + strike stake**
Value:
+ minAda ADA
+ **1** forward
+ D_i' d_asset'
Datum:
+ AgreementDatum:
+ issuer_address_hash: addr_1
+ utxo_owner_address_hash: addr_1
+ issuer_deposit_asset: asset_1
+ issuer_deposit_asset_amount: Q_1
+ obligee_deposit_asset: asset_2
+ obligee_deposit_asset_amount: Q_2
+ collateral_asset: c_asset
+ collateral_asset_amount: Q_3
+ exercise_contract_date: t
+ mint_asset: forward
Agreement UTxO
Address: **agreement script + strike stake**
Value:
+ minAda ADA
+ **1** forward
+ D_i d_asset
Datum:
+ AgreementDatum:
+ issuer_address_hash: addr_1
+ utxo_owner_address_hash: addr_2
+ issuer_deposit_asset: asset_1
+ issuer_deposit_asset_amount: Q_1
+ obligee_deposit_asset: asset_2
+ obligee_deposit_asset_amount: Q_2
+ collateral_asset: c_asset
+ collateral_asset_amount: Q_3
+ exercise_contract_date: t
+ mint_asset: forward
User UTxO
Value:
+ minAda ADA
+ N_3 c_asset
+ N_4 s_asset

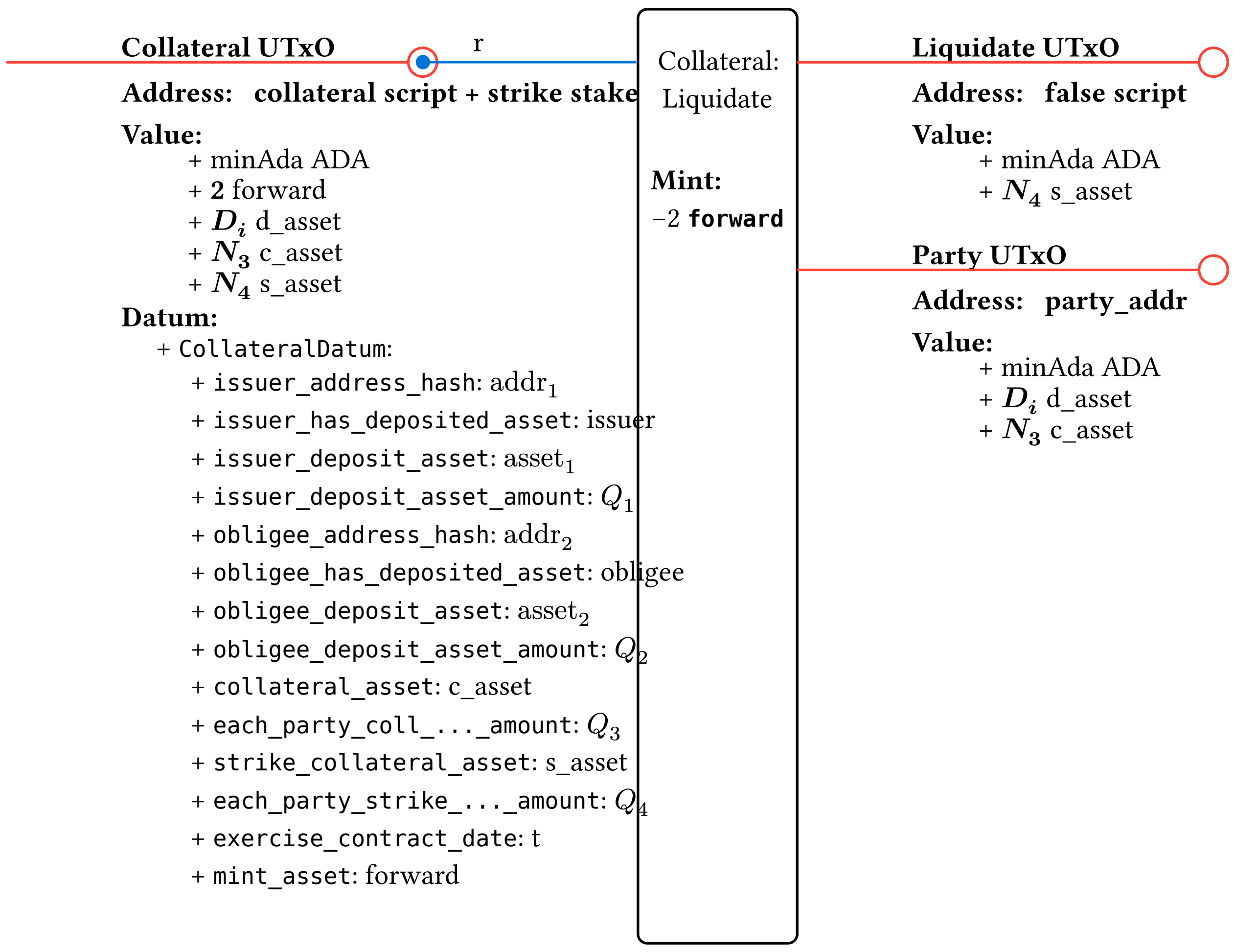
Note:

r = `BothSidesDepositAgreement`

$(D_i, D_i') \geq$ `if issuer then` (Q_2, Q_1) `else` (Q_1, Q_2)

`(d_asset, d_asset') = if issuer then` $(\text{asset}_1, \text{asset}_2)$ `else` $(\text{asset}_2, \text{asset}_1)$

2.c.f - Collateral: Liquidate



Collateral UTxO

r

Address: collateral script + strike stake

Value:

+ minAda ADA
+ **2** forward
+ $\boldsymbol{D_i}$ d_asset
+ $\boldsymbol{N_3}$ c_asset
+ $\boldsymbol{N_4}$ s_asset

Datum:

+ `CollateralDatum`:
  + `issuer_address_hash`: addr_1
  + `issuer_has_deposited_asset`: issuer
  + `issuer_deposit_asset`: asset_1
  + `issuer_deposit_asset_amount`: Q_1
  + `obligee_address_hash`: addr_2
  + `obligee_has_deposited_asset`: obligee
  + `obligee_deposit_asset`: asset_2
  + `obligee_deposit_asset_amount`: Q_2
  + `collateral_asset`: c_asset
  + `each_party_coll_..._amount`: Q_3
  + `strike_collateral_asset`: s_asset
  + `each_party_strike_..._amount`: Q_4
  + `exercise_contract_date`: t
  + `mint_asset`: forward

Collateral: Liquidate

Mint:

−2 **`forward`**

Liquidate UTxO

Address: false script

Value:

+ minAda ADA
+ $\boldsymbol{N_4}$ s_asset

Party UTxO

Address: party_addr

Value:

+ minAda ADA
+ $\boldsymbol{D_i}$ d_asset
+ $\boldsymbol{N_3}$ c_asset

Note:

r = `LiquidateCollateral(n)`

`party_addr = if n == 0 then issuer_address_hash else obligee_address_hash`

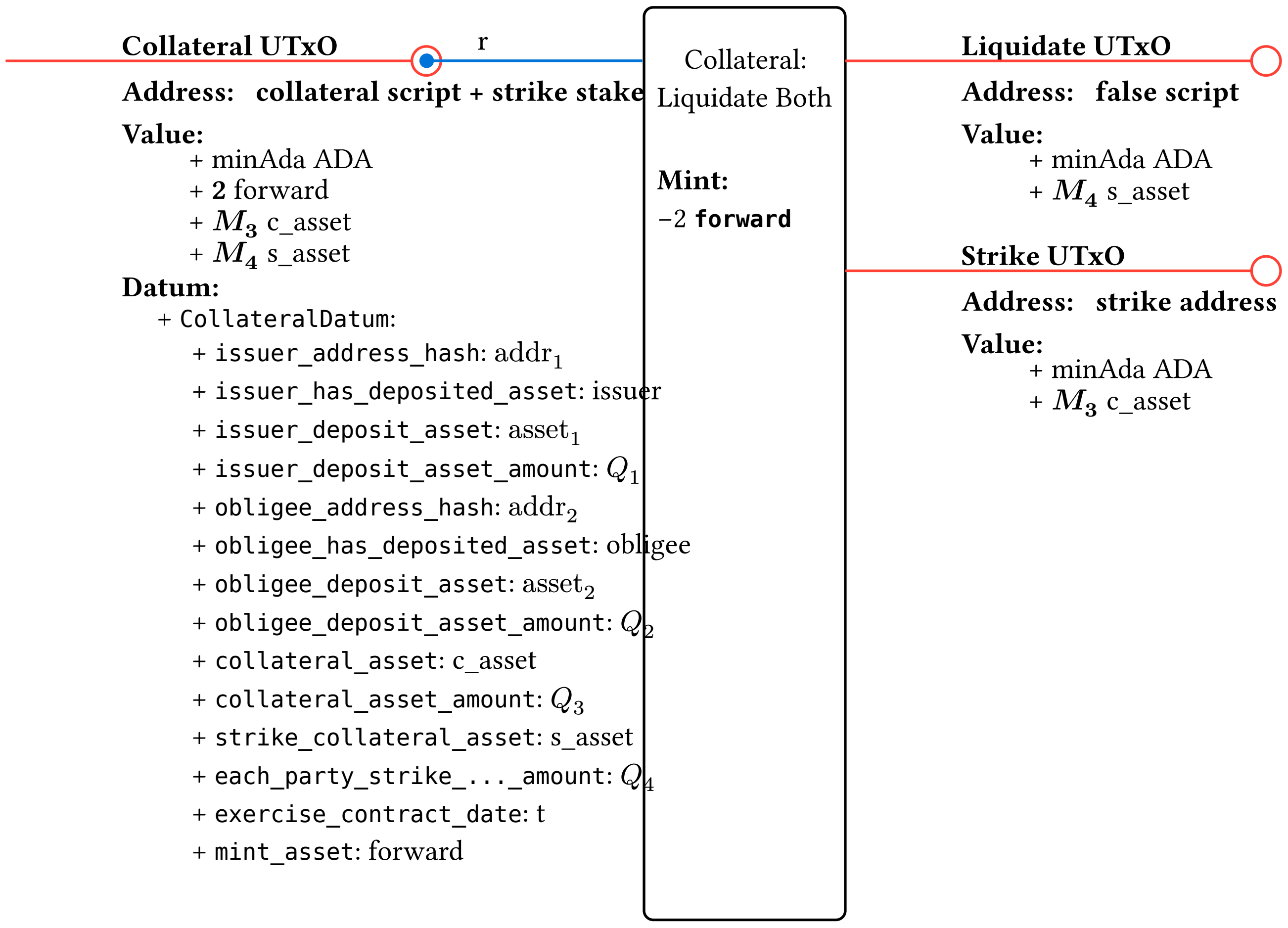
$D_i \geq$ `if n == 0 then` Q_1 `else` Q_2

The transaction must be signed by `party_addr`

t < transaction validaty range lower bound

`forward` minting redeemer: `LiquidateBurn`

2.c.g - Collateral: Liquidate Both


Collateral UTxO
r
Address: collateral script + strike stake
Value:
+ minAda ADA
+ 2 forward
+ M_3 c_asset
+ M_4 s_asset
Datum:
+ CollateralDatum:
+ issuer_address_hash: $addr_1$
+ issuer_has_deposited_asset: issuer
+ issuer_deposit_asset: $asset_1$
+ issuer_deposit_asset_amount: Q_1
+ obligee_address_hash: $addr_2$
+ obligee_has_deposited_asset: obligee
+ obligee_deposit_asset: $asset_2$
+ obligee_deposit_asset_amount: Q_2
+ collateral_asset: c_asset
+ collateral_asset_amount: Q_3
+ strike_collateral_asset: s_asset
+ each_party_strike_..._amount: Q_4
+ exercise_contract_date: t
+ mint_asset: forward
Collateral: Liquidate Both
Mint:
−2 forward
Liquidate UTxO
Address: false script
Value:
+ minAda ADA
+ M_4 s_asset
Strike UTxO
Address: strike address
Value:
+ minAda ADA
+ M_3 c_asset

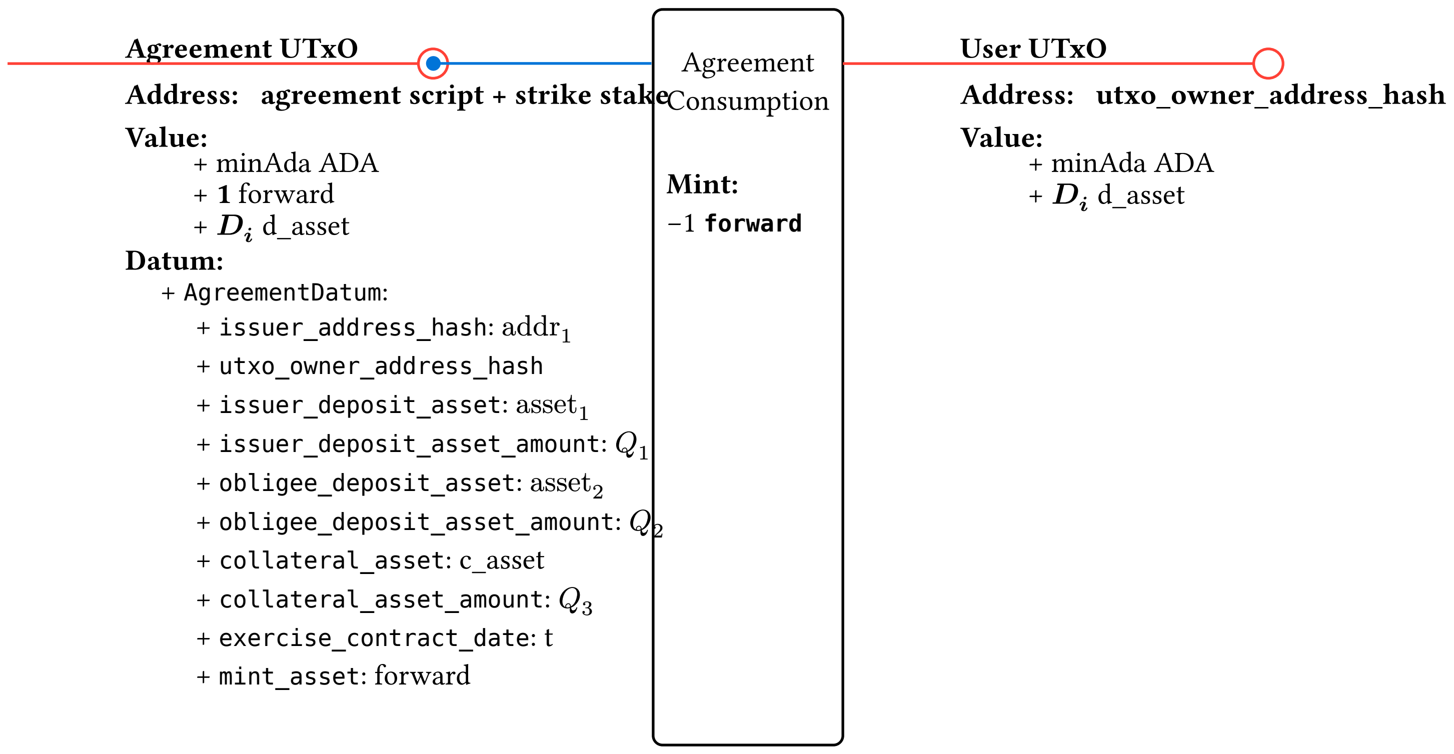
Note:

r = `LiquidateBothParties`

t < transaction validaty range lower bound

`forward` minting redeemer: `LiquidateBurn`

2.c.h - Agreement Consumption



Agreement UTxO

Address: agreement script + strike stake

Value:

+ minAda ADA
+ **1** forward
+ $\boldsymbol{D_i}$ d_asset

Datum:

+ `AgreementDatum`:
  + `issuer_address_hash`: addr_1
  + `utxo_owner_address_hash`
  + `issuer_deposit_asset`: asset_1
  + `issuer_deposit_asset_amount`: Q_1
  + `obligee_deposit_asset`: asset_2
  + `obligee_deposit_asset_amount`: Q_2
  + `collateral_asset`: c_asset
  + `collateral_asset_amount`: Q_3
  + `exercise_contract_date`: t
  + `mint_asset`: forward

Agreement Consumption

Mint:

−1 **`forward`**

User UTxO

Address: utxo_owner_address_hash

Value:

+ minAda ADA
+ $\boldsymbol{D_i}$ d_asset

Note:
The transaction must be signed by `utxo_owner_address_hash`
`forward` minting redeemer: `SingleBurn`

2.c.i - Audited Files

Below is a list of all audited files in this report. Any files **not** listed here were **not** audited. The final state of the files for the purposes of this report is considered to be commit `18c2d8539ca6351a33f1d1f21d373356424e0461`.

Filename
./lib/constants.ak
./lib/types.ak
./lib/utils.ak
./validators/collateral.ak
./validators/agreement.ak
./validators/always_fail.ak
./validators/forwards.ak

3 - Findings

ID	Title	Severity	Status
STF-001	UTxO address not validated in Create Forward operation	Critical	Resolved
STF-002	Potential loss of collateral if neither party deposits the asset	Critical	Resolved
STF-003	Double satisfaction in operations that require token burning	Critical	Resolved
STF-004	Missing validations in Accept Forward operation	Critical	Resolved
STF-005	Double counting of tokens in values	Critical	Resolved
STF-006	One Side Deposit can be performed multiple times	Critical	Resolved
STF-007	Missing datum fields validation in Create Forward	Critical	Resolved
STF-101	Users could deposit assets after the exercise date has passed	Major	Resolved
STF-201	Prevent inclusion of reference scripts	Minor	Resolved
STF-202	One Side Deposit can be bypassed	Minor	Resolved
STF-203	Party identity can be forged	Minor	Resolved
STF-301	Do Datum comparissons in Data	Info	Resolved

STF-302	Clean up output lookup in Both Sides Deposit	Info	Resolved
STF-303	Standarize the output lookups	Info	Resolved
STF-304	Cleanup output lookup in Accept Forwards	Info	Resolved
STF-305	Various recommendations for the Types module	Info	Resolved

4 - STF-001 UTxO address not validated in Create Forward operation

Category	Commit	Severity	Status
Bug	`589b55f9a53c363ad4e636e037cc31e354700e86`	Critical	Resolved

4.a - Description

During the Forward Create operation, a token is minted. The minting policy of this token validates certain aspects of the datum and value of the UTxO where it is paid. But there's a missing validation for the address of the UTxO. This token should always be paid to the Forwards contract address, but with the current code an attacker could send it to any arbitrary address, given the datum and value checks still pass. This would give full control of the token to the attacker, which could create an UTxO at the forwards address now breaking the minting policy preconditions.

4.b - Recommendation

We recommend adding a check that the payment credential of the address of the `output_to_forward_contract` UTxO is the same as the Forwards script. Given that the minting policy is a multivalidator of the forwards script, the address can be calculated using the `policy_id`

4.c - Resolution

Resolved in commit `a36a503bc7f3e84e3cca4e039d790be71d693a35`

5 - STF-002 Potential loss of collateral if neither party deposits the asset

Category	Commit	Severity	Status
Bug	`589b55f9a53c363ad4e636e037cc31e354700e86`	Critical	Resolved

5.a - Description

Once a Forward has been accepted, a new UTxO is created at the Collateral script address. This UTxO holds the collateral tokens for both the issuer and the obligee. Before the exercise date, parties can deposit their assets to this UTxO, unlocking their collateral. If when the exercise date comes, only one party has deposit their assets, they can use the Liquidate Collateral operation to recover their assets and the collateral of the other party. However, if by the exercise date, neither party has deposited the assets, both `issuer_has_deposited_asset` and `obligee_has_deposited_asset` will be false, so the Collateral UTxO won't be able to be consumed, locking the collaterals and ADAs forever.

Although there's always an incentive for at least one party to deposit the asset, not having an operation in case neither party does can create an unspendable UTxO, essentially burning ADAs and the collateral tokens, which should always be disincentivized.

5.b - Recommendation

We recommend adding an operation that would allow collaterals to be unlocked if this situation arises. The new operation should burn the `forward` tokens and send each party their collateral tokens.

5.c - Resolution

Resolved in commit `cc11f977bfdec7d960151ae472e2a40cd539470b`

6 - STF-003 Double satisfaction in operations that require token burning

Category	Commit	Severity	Status
Bug	`b10b0f5f44cf01d3f507938141754135e823ec50`	Critical	Resolved

6.a - Description

Multiple actions in the protocol require tokens to be minted or burned. In the validators, this is checked using the `validate_token_mint` function, where one of the parameters is the expected mint for that operation. For example, in `CancelForwardsContract`, one token is expected to be burned, so the mint field of the transaction is validated to contain exactly one token being burned.

The problem arises in certain operations, where multiple similar actions can be performed in the same transaction, but all of them expecting the same number of burns. In these cases, a double satisfaction attack can be performed and the attacker would take control of already minted tokens.

The affected operations are: `CancelForwardsContract`, `LiquidateCollateral` and `Agreement`.

6.b - Recommendation

We recommend adding to all three operations mentioned in the finding description the `only_one_validator_input` validation that is seen in operations like `LiquidateBothParties` or `AcceptForwardsContract`.

6.c - Resolution

Resolved in commit `f53a3605c053d44b9fe6b42e0ac41f9798cb631e`

7 - STF-004 Missing validations in Accept Forward operation

Category	Commit	Severity	Status
Bug	`f53a3605c053d44b9fe6b42e0ac41f9798cb631e`	Critical	Resolved

7.a - Description

In the Accept Forward operation, a new UTxO is created at the collateral script address. That UTxO must contain two "forward tokens". One of them will be gathered from the consumed Forward UTxO, and the other one has to be minted.

The forwards minting policy contains the `enter_forward_mint_valid` operation, but the forwards validator never makes sure that this is actually called. So it can be ignored, and the token stolen.

Another missing validation is regarding the `exercise_contract_date`. With the current implementation, a user can accept a forward that has the exercise date in the past. This would result in both users losing their collateral.

7.b - Recommendation

We recommend adding three validations to the Accept Forward operation:

- Check that one and only one token is being minted
- Check that the collateral UTxO contains two tokens
- Check that the `exercise_contract_date` is in the future.

7.c - Resolution

Resolved in commit `fc4a6d5cc690b3ece300ec2a9088d91633a85d49`

8 - STF-005 Double counting of tokens in values

Category	Commit	Severity	Status
Bug	`f53a3605c053d44b9fe6b42e0ac41f9798cb631e`	Critical	Resolved

8.a - Description

In the Strike Forwards protocol, all validations regarding UTxO values are done using `quantity_of`. This can create a couple of issues. The first one is that it allows for other, unrelated tokens, to be added to all values. An attacker could then fill a UTxO with a collection of "trash" tokens, increasing the min lovelace required, fees and execution units of the contract.

Another, more important issue that arises is a double counting of assets in the case that the collateral_asset is equal to the issuer or obliguee assets. For example, let's use the following scenario:

```
obligee_asset: USDM
obligee_asset_amount: 4.000

issuer_asset: ADA
issuer_asset_amount: 8.000

collateral_asset: ADA
collateral_asset_amount: 5.000
```

Let's imagine that the forward has been accepted and now it's the turn of someone to deposit their assets. In this case, if the issuer were to deposit first, it could use the fact that 5.000 ADAs are already in the contract to only deposit 3.000 ADAs themselves. This would pass all validations, because each check is done independently. Then, when the obligee goes to deposit their assets, they would need to cover the other 5.000 ADAs to send to the agreement validator, or they could choose not to deposit their asset and loose their collateral. In both situations the obligee lost 5.000 ADAs.

8.b - Recommendation

We recommend refactoring how the value validations are done in the protocol, changing the multiple calls to `quantity_of` to a single call to `match` for each value, constructing the expected value before hand. In the above case, when building the expected value, both the 5.000 ADAs of collateral and the 8.000 ADAs of the issuer should be added together (Using the `value.add` function), mitigating the issue. Also, with this approach, no other tokens could be added.

8.c - Resolution

Resolved in commit `fc4a6d5cc690b3ece300ec2a9088d91633a85d49`

9 - STF-006 One Side Deposit can be performed multiple times

Category	Commit	Severity	Status
Bug	`fc4a6d5cc690b3ece300ec2a9088d91633a85d49`	Critical	Resolved

9.a - Description

The One Side Deposit Operation is meant to be done once, by the first party that deposits funds into the collateral UTxO. For this purpose, there's a couple of boolean fields in the `CollateralDatum`, `issuer_has_deposited_asset` and `obligee_has_deposited_asset`. Once a party deposits their assets, the corresponding boolean is set to True, and it is also validated that the other boolean stays False.

But these values are not checked in the datum of the UTxO being consumed. This means that a party could run the One Side Deposit operation even if they or the other party has already deposited.

An attacker (for the sake of the example let's say the have the role of issuer, but the attack works with either role) could wait until the obligee deposits their asset, then, run the One Side deposit operation, depositing their asset, reseting the `obligee_has_deposited_asset` to False and stealing the assets the obligee deposited.

9.b - Recommendation

We recommend adding a check that both `*_has_deposited_asset` are set to False.

9.c - Resolution

Resolved in commit `3eeaf82c9babfc83386a4e2b6ca8c1867257f9a7`

10 - STF-007 Missing datum fields validation in Create Forward

Category	Commit	Severity	Status
Bug	`fc4a6d5cc690b3ece300ec2a9088d91633a85d49`	Critical	Resolved

10.a - Description

During the Create Forward operation, the fields of the output datum are not validated, this could cause invalid or unreasonable Forwards to be created.

10.b - Recommendation

These are the list of validations that we recommend adding:

- `issuer_deposit_asset` must not be the same that `obligee_deposit_asset`
- `issuer_deposit_asset_amount` and `obligee_deposit_asset_amount` should not be negative
- `exercise_contract_date` should be in the future
- `mint_asset` must correspond with the asset being minted

We also recommend removing the strike_collateral_asset from the datum and adding it as a parameter, given that the policy will not change at any point in the future and it is already known.

10.c - Resolution

Resolved in commit `eb2eeadcea73d82ff7528f98755696eacb11e50a`

11 - STF-101 Users could deposit assets after the exercise date has passed

Category	Commit	Severity	Status
Bug	b10b0f5f44cf01d3f507938141754135e823ec50	Major	Resolved

11.a - Description

In the Strike Forwards protocol, most actions have to happen either before or after the `exercise_contract_date`. To validate this, the validators use the transaction validity range, and compare it against the timestamp stored in the datum.

But all checks are done against the lower bound of the validity range. Given that the range of the transaction can be arbitrarily configured, this could cause issues. In particular, the `OneSideDepositAgreement` and `BothSidesDepositAgreement` actions need to check that the excercise date is in the future. With the current validations, an attacker could set the lower bound of the transaction to an arbitrary point in the past, and even if the exercise date has passed, the validation would return True.

11.b - Recommendation

We recommend changing the validations in the `OneSideDepositAgreement` and `BothSidesDepositAgreement` actions to compare against the upper bound of the transaction, this way the validator can make sure that the transaction is submitted before the exercise date. The modified code would look like this:

```
let deadline_not_passed: Bool =
    datum.exercise_contract_date > get_upper_bound(transaction.validity_range)
```

11.c - Resolution

Resolved in commit `f53a3605c053d44b9fe6b42e0ac41f9798cb631e`

12 - STF-201 Prevent inclusion of reference scripts

Category	Commit	Severity	Status
Improve-ment	`fc4a6d5cc690b3ece300ec2a9088d91633a85d49`	Minor	Resolved

12.a - Description

With the addition of the `minFeeRefScriptsCoinsPerByte` protocol parameter in the upcoming Voltaire era, including a reference script in any input (whether it's a reference or not) will impact the transaction fees, regardless of whether the script is executed.

Given that the reference script field is not validated in any output of the protocol, there's an attack vector where a malicious party includes a huge reference script in every output of a transaction, costing more fees to the next party interacting with those UTxOs.

12.b - Recommendation

We recommend ensuring that any UTxO belonging to the protocol does not include a reference script.

12.c - Resolution

Resolved in commit `5283d644e579ac85406896303de5455282eb78b0`

13 - STF-202 One Side Deposit can be bypassed

Category	Commit	Severity	Status
Bug	`fc4a6d5cc690b3ece300ec2a9088d91633a85d49`	Minor	Resolved

13.a - Description

The Both Sides Deposit operation is meant to be run when one party has already deposited their assets. In this operation the other party provides their assets and creates two UTxOs at the agreement script address. But there's no check that assures that a party has deposited before.

So a situation can happen where a user runs the Both Sides Deposit operation, when neither party has deposited. This results in the user having to provide both sets of assets and taking control of both sets of collaterals (collateral tokens and strike).

13.b - Recommendation

We recommend adding a parameter to the `both_sides_deposit_agreement` function that indicated which party is doing the deposit. Then, make sure that the other party has already deposited.

13.c - Resolution

Resolved in commit `69a11cea9228c345df4b6da38e5d45a2c98c798a`

14 - STF-203 Party identity can be forged

Category	Commit	Severity	Status
Bug	`5283d644e579ac85406896303de5455282eb78b0`	Minor	Resolved

14.a - Description

During the One Side Deposit and Both Sides Deposit operations, the party executing them is specified in the redeemer. But this party is never validated, so any user can submit the transaction. This could allow a third party to participate in the protocol, or for one party to deposit the other's asset, incurring unwanted costs.

14.b - Recommendation

We recommend adding some validation to make sure the party that is specified in the redeemer is signing the transaction. The implementation should follow how `must_be_signed_by_owner` is implemented in the Liquidate Collateral operation.

14.c - Resolution

Resolved in commit `18c2d8539ca6351a33f1d1f21d373356424e0461`

15 - STF-301 Do Datum comparissons in Data

Category	Commit	Severity	Status
Optimization	`fc4a6d5cc690b3ece300ec2a9088d91633a85d49`	Info	Resolved

15.a - Description

At multiple points of the protocol, the datum field of an output is compared to the expected datum value to make sure all fields have valid values. For this comparissons, the expected datum is built using the corresponding type, then, the datum field of the output (That is given as `Data`) is casted to the corresponding type and compared to the expected value. This operation can be costly if the type is complex and has many fields, as it is the case in the Strike protocol.

This process can be optimized if, instead of casting from Data to each datum's type, the expected datum is casted to Data, as the downcasting is faster and cheaper

15.b - Recommendation

We recommend refactoring all places where output datums are compared to an expected datum to use Data.

15.c - Resolution

Resolved in commit `18c2d8539ca6351a33f1d1f21d373356424e0461`

16 - STF-302 Clean up output lookup in Both Sides Deposit

Category	Commit	Severity	Status
Optimiza-tion	fc4a6d5cc690b3ece300ec2a9088d91633a85d49	Info	Resolved

16.a - Description

In the Both Sides Deposit operation, two outputs must be created at the agreement validator address. The current implementation has a filter to lookup the outputs at the specific address, and a check to make sure the resulting list has only 2 elements. But to identify each one, a complicated setup involving two more `list.filter` calls and the expect datums is used.

16.b - Recommendation

We recommend refactoring the current implementation to something like the following:

```
expect    [expected_issuer_utxo,    expected_obligee_utxo]    =
find_script_outputs(transaction.outputs, agreement_validator)
```

Then, having all the usual checks for those UTxOs, including checking that the datum matches. This solution required that the issuer output UTxO is always the first output to the agreement address and the obligee UTxO is always the second output, but results in a more simple and optimized implementation.

16.c - Resolution

Resolved in commit `18c2d8539ca6351a33f1d1f21d373356424e0461`

17 - STF-303 Standarize the output lookups

Category	Commit	Severity	Status
Code Style	`5283d644e579ac85406896303de5455282eb78b0`	Info	Resolved

17.a - Description

Most operations in the protocol involve looking at the outputs that are paid to the script address. In most cases, this lookup is done by filtering outputs by script hash. But there are two operations where this pattern is broken. The first one is One Side deposit, where the full address is used to filter, the second one is Create Forward Mint check where the token is used to filter.

17.b - Recommendation

We recommend standarizing the lookup to always use the script hash. Then, for the Create Forward operation, a new check should be added to make sure the value contains the token, and the `output_is_to_forward_validator` can be removed.

17.c - Resolution

Resolved in commit `18c2d8539ca6351a33f1d1f21d373356424e0461`

18 - STF-304 Cleanup output lookup in Accept Forwards

Category	Commit	Severity	Status
Code Style	`5283d644e579ac85406896303de5455282eb78b0`	Info	Resolved

18.a - Description

Similar to STF-302, the Accept Forwards operation implements a roundabout way to get the output_to_collateral_utxo.

18.b - Recommendation

We recommend removing the find implementation and using expect to make sure that outputs_to_collateral_validator has only one element

18.c - Resolution

Resolved in commit `18c2d8539ca6351a33f1d1f21d373356424e0461`

19 - STF-305 Various recommendations for the Types module

Category	Commit	Severity	Status
Code Style	`5283d644e579ac85406896303de5455282eb78b0`	Info	Resolved

19.a - Description

The Types module defines the types used in the rest of the code. We identified a list of improvements that can be made to cleanup and simplify the types and the code that uses them. They are:

- Define a new type with two constructors instead of using Int to differentiate between Issuer and Obligee.
- Replace AddressHash and ScriptHash with types from stdlib: ScriptHash and VerificationKeyHash
- Refactor the CollateralDatum to include the ForwardsDatum inside. The proposed type looks like this:

  ```
  pub type CollateralDatum {
    issuer_has_deposited_asset: Bool,
    obligee_address_hash: AddressHash,
    obligee_has_deposited_asset: Bool,
    associated_forwards_datum: ForwardsDatum
  }
  ```

All data would still be available and it would simplify code when creating and updating the expected collateral datum

- Remove unused fields from AgreementDatum. The only fields that are used are: `utxo_owner_address_hash`, `mint_aset` and `exercise_contract_date`
- Rename ObligeeInfo to ObligeeAddress to make it clearer what info it represents.

19.b - Recommendation

We recommend applying all mentioned improvements.

19.c - Resolution

Resolved in commit `18c2d8539ca6351a33f1d1f21d373356424e0461`

20 - Style Recommendations

This section summarizes mostly style recommendations and some possible optimizations. None of the following suggestions are critical aspects that the Strike team must address. They are focused on improving readability and, in some cases, making minor performance improvements. The applied suggestions are marked in bold and the no longer relevant are marked with strikethroughs.

File names and line numbers mentioned were collected in commit `fc4a6d5cc690b3ece300ec2a9088d91633a85d49`, they might not be accurate in other commits.

20.a - General Recommendations

- **Standarize the `only_one_validator_input` check**
- **Prefer using patternmatching instead of the dot operator if accessing multiple fields on the same variable to improve readability and optimization**
- Prefer using `and {...}` expressions instead of multiple && operators to increase readability
- **Avoid variable name shadowing, for example, in the `let Some(datum) = datum` cases**
- **Move the `validator` code blocks to the top of the module to increase readability of the files**
- Consider adding code comments and documentation to each function
- **Standarize the use or no use of the `strike_is_used_as_collateral` variable**

20.b - Naming

- **Standarize the use of the `_valid` suffix in validator functions.**

20.c - Comments

- **collateral.ak, line 29. Remove double comment mark and trailing space**

20.d - Code Style

- **collateral.ak, lines 33-35, 41. Refactor to `expect [output_to_validator] = get_address_outputs(transaction, input_from_validator.output.address)`**
- **collateral.ak, lines 119-131, 154-166. Refactor `expected_obligee_datum` to be built from `expected_issuer_datum` and just modifying the `utxo_owner_address_hash` to make it clearer that most fields are shared.**
- **utils.ak, lines 101-123. Merge `get_asset_locked_based_on_party` and `get_asset_amount_locked_based_on_party` into a single function**
- forwards.ak, lines 217-221. Refactor to `expect [input] = get_validators_inputs(transaction)`
- **forwards.ak, lines 239-241, 251-253. Remove valid_mint variable, leave just the `validate_token_mint` call as the last expression**
- **forwards.ak, lines 232-254. Consider merging single_burn_valid and liquidate_burn_valid into a single function with an extra “amount” parameter indicating how many tokens to burn.**
- ~~utils.ak, lines 55-99. Refactor to creating a base `CollateralDatum` with both `*_has_deposited` set to false and just modifying the nessesary fields inside each branch of the if expression.~~
- **utils.ak, lines 149-196. Remove unused `get_asset_to_address` and `get_asset_to_address_valid` functions**
- **utils.ak, lines 198-215. Refactor using the `assets.tokens` function and an expect expression.**

20.e - Optimization

- **collateral.ak, 307-308. Move the `outputs_to_liquidate_validator` lookup inside the strike collateral branch of the if expression below.**

21 - Appendix

21.a - Terms and Conditions of the Commercial Agreement

21.a.a - Confidentiality

Both parties agree, within a framework of trust, to discretion and confidentiality in handling the business. This report cannot be shared, referred to, altered, or relied upon by any third party without Txpipe LLC, 651 N Broad St, Suite 201, Middletown registered at the county of New Castle, written consent.

The violation of the aforementioned, as stated supra, shall empower TxPipe to pursue all of its rights and claims in accordance with the provisions outlined in Title 6, Subtitle 2, Chapter 20 of the Delaware Code titled "Trade Secrets,", and to also invoke any other applicable law that protects or upholds these rights.

Therefore, in the event of any harm inflicted upon the company's reputation or resulting from the misappropriation of trade secrets, the company hereby reserves the right to initiate legal action against the contractor for the actual losses incurred due to misappropriation, as well as for any unjust enrichment resulting from misappropriation that has not been accounted for in the calculation of actual losses.

21.a.b - Service Extension and Details

This report does not endorse or disapprove any specific project, team, code, technology, asset or similar. It provides no warranty or guarantee about the quality or nature of the technology/code analyzed.

This agreement does not authorize the client Strike to make use of the logo, name, or any other unauthorized reference to Txpipe LLC, except upon express authorization from the company.

TxPipe LLC shall not be liable for any use or damages suffered by the client or third-party agents, nor for any damages caused by them to third parties. The sole purpose of this commercial agreement is the delivery of what has been agreed upon. The company shall be exempt from any matters not expressly covered within the contract, with the client bearing sole responsibility for any uses or damages that may arise.

Any claims against the company under the aforementioned terms shall be dismissed, and the client may be held accountable for damages to reputation or costs resulting from non-compliance with the aforementioned provisions. **This report provides general information and is not intended to constitute financial, investment, tax, legal, regulatory, or any other form of advice.**

Any conflict or controversy arising under this commercial agreement or subsequent agreements shall be resolved in good faith between the parties. If such negotiations do not result in a conventional agreement, the parties agree to submit disputes to the courts of Delaware and to the laws of that jurisdiction under the powers conferred by the Delaware Code, TITLE 6, SUBTITLE I, ARTICLE 1, Part 3 § 1-301. and Title 6, SUBTITLE II, chapter 27 §2708.

21.a.c - Disclaimer

The audit constitutes a comprehensive examination and assessment as of the date of report submission. The company expressly disclaims any certification or endorsement regarding the subsequent performance, effectiveness, or efficiency of the contracted entity, post-report delivery, whether resulting from modification, alteration, malfeasance, or negligence by any third party external to the company.

The company explicitly disclaims any responsibility for reviewing or certifying transactions occurring between the client and third parties, including the purchase or sale of products and services.

This report is strictly provided for ***informational purposes*** and reflects solely the due diligence conducted on the following files and their corresponding hashes using sha256 algorithm:

Filename: ./lib/constants.ak **Hash**: 145313a974aa463a98300ff41b4c8fc745b19935dddee24c264b6d666d595814
Filename: ./lib/types.ak **Hash**: 19e1fa059e299bb230b40d54abc2474da4450d26649dff421dad0cbb88903749
Filename: ./lib/utils.ak **Hash**: 59e484260c988c2496d3cc65b37b7bd8fd423a8fe31aabd285f463343bde190e
Filename: ./validators/collateral.ak **Hash**: 6190ac72c89a1a3a74ae7794fd87370de40e0b6d8090359c70a43cdaff13f884
Filename: ./validators/agreement.ak **Hash**: 0546c18ceda4386c6a4a081328659b8d003b61aae1fbc0b80497023e33d0bdc6
Filename: ./validators/always_fail.ak **Hash**: ad29861920e98f60d7234553e7b169d7bc8ebb3988e464a6d0164f521e2a36ca
Filename: ./validators/forwards.ak **Hash**: 9bceec287a9fb4d5e937f4eae44f2d5cd0a965c75f3373b0033850d4d94d6081

TxPipe advocates for the implementation of multiple independent audits, a publicly accessible bug bounty program, and continuous security auditing and monitoring. Despite the diligent manual review processes, the potential for errors exists. TxPipe strongly advises seeking multiple independent opinions on critical matters. It is the firm belief of TxPipe that every entity and individual is responsible for conducting their own due diligence and maintaining ongoing security measures.

21.b - Issue Guide

21.b.a - Severity

Severity	Description
Critical	Critical issues highlight exploits, bugs, loss of funds, or other vulnerabilities that prevent the dApp from working as intended. These issues have no workaround.
Major	Major issues highlight exploits, bugs, or other vulnerabilities that cause unexpected transaction failures or may be used to trick general users of the dApp. dApps with Major issues may still be functional.
Minor	Minor issues highlight edge cases where a user can purposefully use the dApp in a non-incentivized way and often lead to a disadvantage for the user.
Info	Info are not issues. These are just pieces of information that are beneficial to the dApp creator. These are not necessarily acted on or have a resolution, they are logged for the completeness of the audit.

21.b.b - Status

Status	Description
Resolved	Issues that have been **fixed** by the **project** team.
Acknowledged	Issues that have been **acknowledged** or **partially fixed** by the **project** team. Projects can decide to not **fix** issues for whatever reason.
Identified	Issues that have been **identified** by the **audit** team. These are waiting for a response from the **project** team.

21.c - Revisions

This report was created using a git based workflow. All changes are tracked in a github repo and the report is produced using typst. The report source is available here. All versions with downloadable PDFs can be found on the releases page.

21.d - About Us

TxPipe is a blockchain technology company responsible for many projects that are now a critical part of the Cardano ecosystem. Our team built Oura, Scrolls, Pallas, Demeter, and we're the original home of Aiken. We're passionate about making tools that make it easier to build on Cardano. We believe that blockchain adoption can be accelerated by improving developer experience. We develop blockchain tools, leveraging the open-source community and its methodologies.

21.d.a - Links

- Website
- Email
- Twitter