

Stateful Smart Contracts On Cardano

Part 1.

Minting Unique Tokens With A Shared `PolicyId`

Lorenzo Fanton

887857@stud.unive.it

Problem

Our goals:

1. We want to distribute a smart contract's state over multiple UTxOs, *and...*
2. ... We need to ensure that it cannot be compromised by an attacker.

Otherwise, it would be pointless to use whatever abstractions we might come up with.

Problem

An informal statement of the problem might be the following.

“ A smart contract's state, distributed over a set of UTxOs, should be modified only by transactions validated by the smart contract itself.”

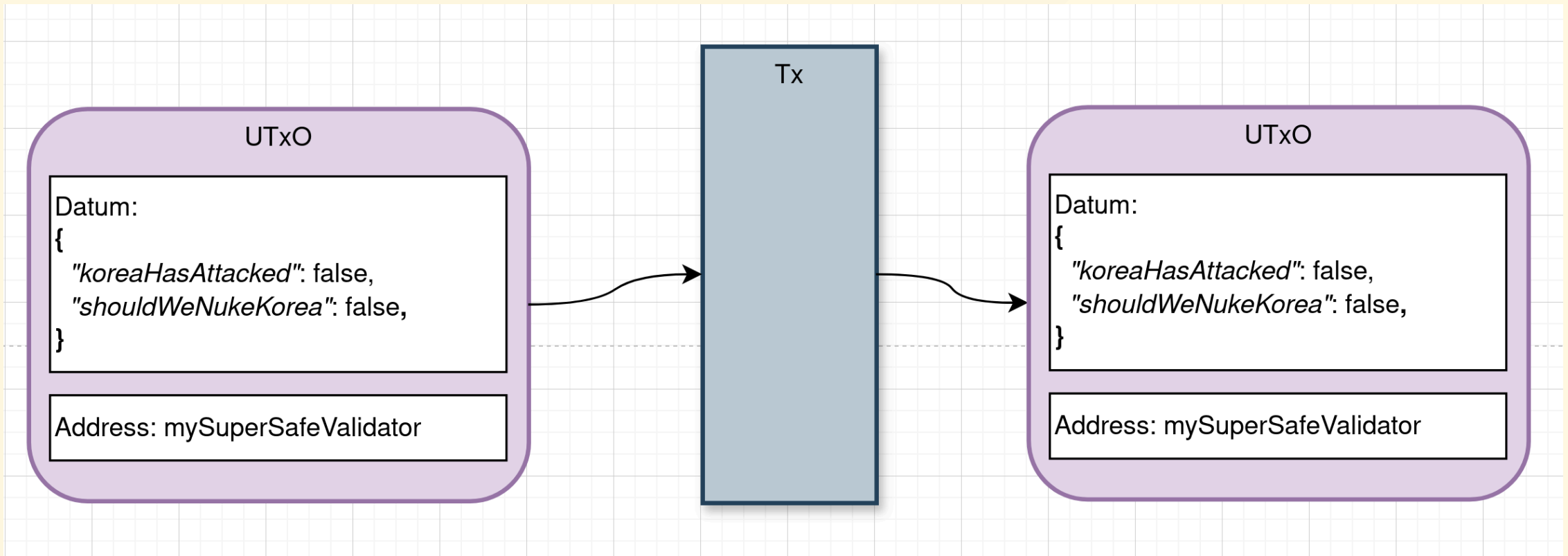
Problem

There are good news and bad news.

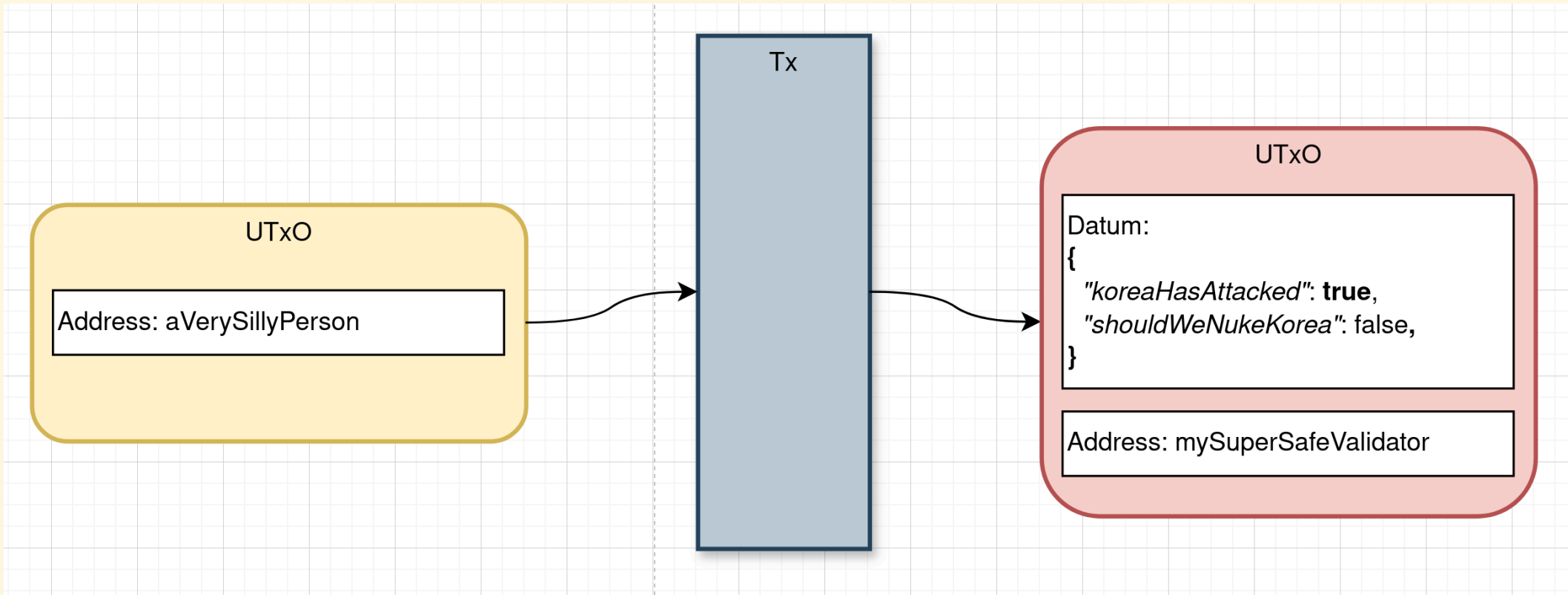
- **Good news:** Smart contracts (on-chain) can enforce the recipient(s) of the UTxOs being created, but...
- **Bad news:** ... They have **zero knowledge** about the sender of the UTxOs being spent.

Why is that a problem?

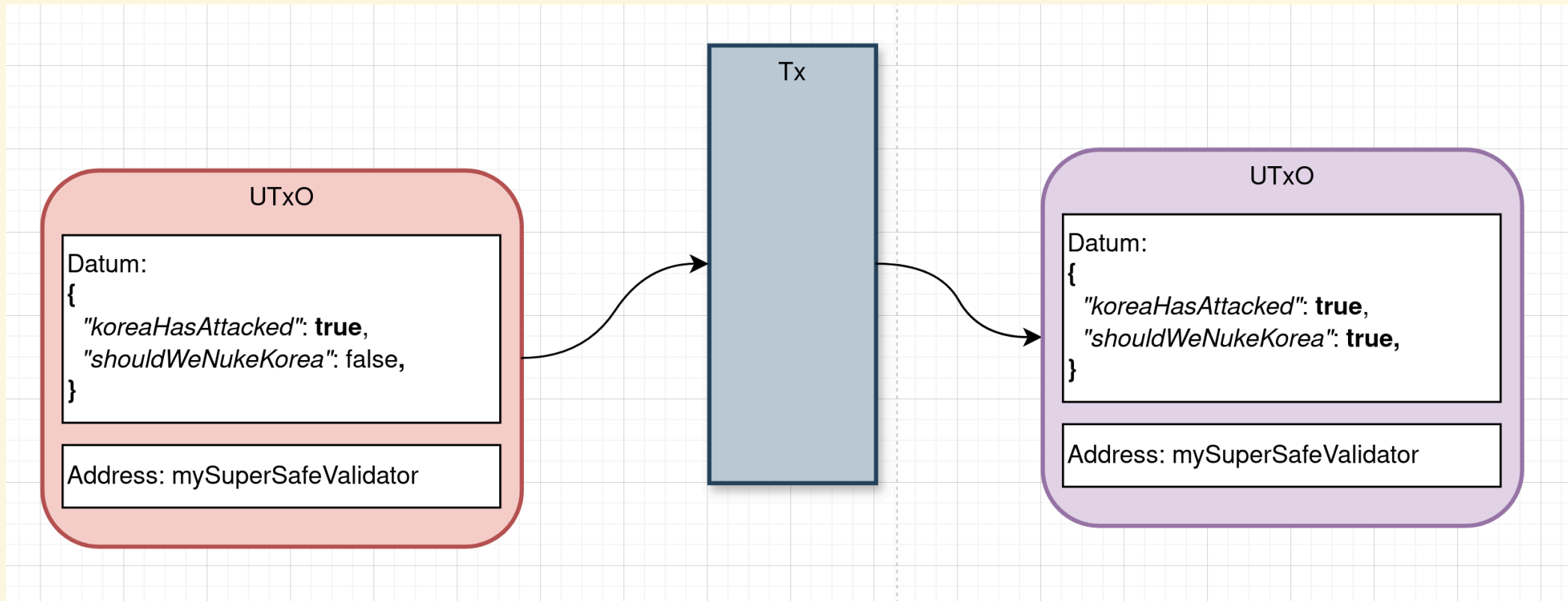
Example: Good State



Example: Attacker



Example: (Very) Bad State



Problem

Compromised state could be the source of other, more serious issues.

What could be worse than WW3? Well, for example:

“ *You* could be losing money!! ”

While total thermonuclear annihilation might be somewhat tolerable and even fun, losing money is not and we must avoid that at all costs.

Native Tokens

Native tokens, also called *assets*, are defined as:

- $\text{AssetId} := \text{PolicyId} \times \text{AssetName}$
- $\text{PolicyId} := (\mathbb{F}_{256})^{28}$ (hash of the minting validator script)
- $\text{AssetName} := (\mathbb{F}_{256})^{32}$ (arbitrary string)

Where $\mathbb{F}_q := \{0, \dots, q - 1\}$.

Spending Validators

We call V the type of a Spending Validator:

$$V := (\text{Datum} \times \text{Redeemer} \times \text{ScriptContext}) \rightarrow \mathbb{B}$$

Minting Validators

We call M the type of a Minting Validator:

$$M := (\text{Redeemer} \times \text{ScriptContext}) \rightarrow \mathbb{B}$$

NB: Minting Validators do not take a Datum as inputs, because they do not consume UTxOs.

Minting The NFTs

Let `OutputReference` represent an on-chain reference to an UTxO.

We declare the following functions that produce smart contracts:

- $new_m : (\text{OutputReference}) \rightarrow M$
- $add_m : M \rightarrow M$
- $add_v : (M \times M) \rightarrow V$

Minting The NFTs

newm takes an `OutputReference` `utxo` and returns a `Minting Validator` that produces a single token with `AssetName` `"Auth"`.

Such script is a **one-time** minting validator, because it ensures that `utxo` is consumed in the transaction (which can happen only once).

Minting The NFTs

In particular, let o an UTxO reference.

- $S_m = new_m(o)$
- $A_m = add_m(s_M)$
- $A_v = add_v(s_M, a_M)$

Code: *new_m*

```
8  validator(utxo: OutputReference) {
9      fn run(_redeemer: Data, context: ScriptContext) → Bool {
10         // 0. The UTXO is consumed in the inputs
11         expect Some(_input) =
12             context.transaction.inputs
13             ▷ list.find(fn(input) { input.output_reference == utxo })
14         // 1. The transaction is a minting transaction
15         expect Mint(policy_id) = context.purpose
16         // 2. There is a single NFT minted in this transaction, with unit amount
17         expect [(asset, 1)]: List<(AssetName, Int)> =
18             context.transaction.mint
19             ▷ value.from_minted_value()
20             ▷ value.tokens(policy_id)
21             ▷ dict.to_list()
22         // 3. The asset name of the NFT is "Auth"
23         expect asset == "Auth"
24         // 4. There is only one output with the NFT minted
25         expect [output] =
26             context.transaction.outputs
27             ▷ list.filter(
28                 fn(output: Output) {
29                     1 == quantity_of(output.value, policy_id, "Auth")
30                 },
31             )
32         // 5. The datum of the NFT output is a InlineDatum
33         expect InlineDatum(datum) = output.datum
34         // 6. The datum is an empty list of AssetName(s)
35         expect []: List<AssetName> = datum
36         True
37     }
38 }
39
```

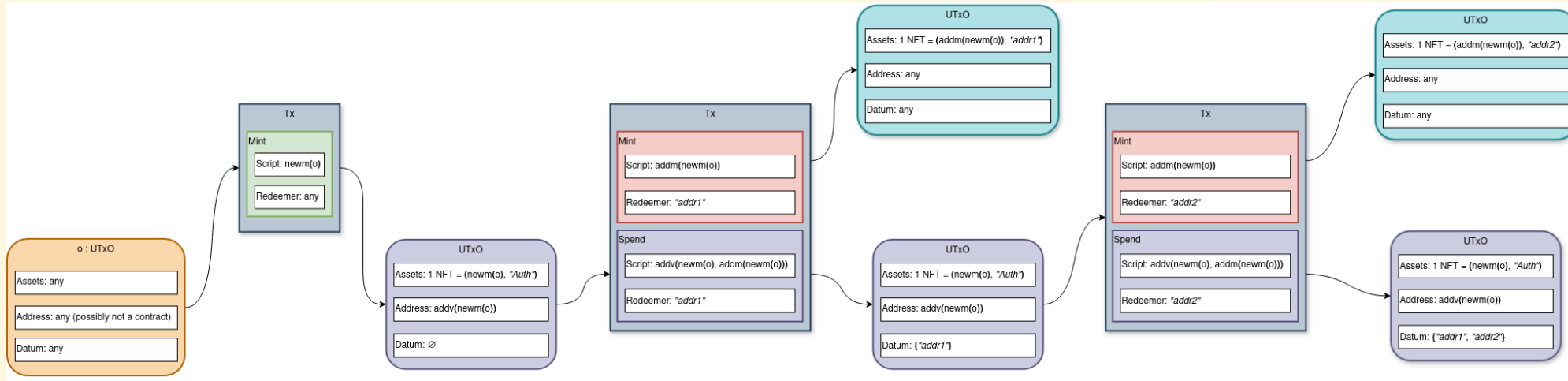
Code: *add_v*

```
11 // validator(newm_script: PolicyId, addm_script: PolicyId) {
12   fn run(
13     datum: (List<AssetName>, Hash<Blake2b_224, Script>),
14     redeemer: AssetName,
15     context: ScriptContext,
16   ) → Bool {
17     // 0. The ScriptPurpose is to Spend an UTxO
18     expect Spend(outref): ScriptPurpose = context.purpose
19     // 1. The addm_script is involved in the transaction
20     expect Some(mint_redeemer): Option<Redeemer> =
21       context.transaction.redeemers ▷ dict.get(Mint(addm_script))
22     // 2. The redeemer of the addm_script is the same as our redeemer
23     expect address: AssetName = mint_redeemer
24     expect address == redeemer
25     // 3. The transaction pays a single UTxO to this script
26     expect Some(input): Option<Input> =
27       context.transaction.inputs ▷ list.at(outref.output_index)
28     let self_address: Address = input.output.address
29     expect [output] =
30       context.transaction.outputs
31       ▷ list.filter(fn(output: Output) { output.address == self_address })
32     // 4. The output UTxO has the authorizing start NFT
33     expect 1 == quantity_of(output.value, newm_script, "Auth")
34     // 5. The datum of the output UTxO is the correct list of new_addm_assets
35     expect InlineDatum(inline_output_datum) = output.datum
36     expect output_datum: (List<AssetName>, Hash<Blake2b_224, Script>) =
37       inline_output_datum
38     let old_addm_assets: List<AssetName> = list.unique(datum.1st)
39     let new_addm_assets: List<AssetName> = list.unique(output_datum.1st)
40     expect 1 == list.length(new_addm_assets) - list.length(old_addm_assets)
41     expect [redeemer] == list.difference(new_addm_assets, old_addm_assets)
42     // 6. The address of the pay script is always the same
43     expect datum.2nd == output_datum.2nd
44     True
45   }
46 }
47
```


Code: add_m

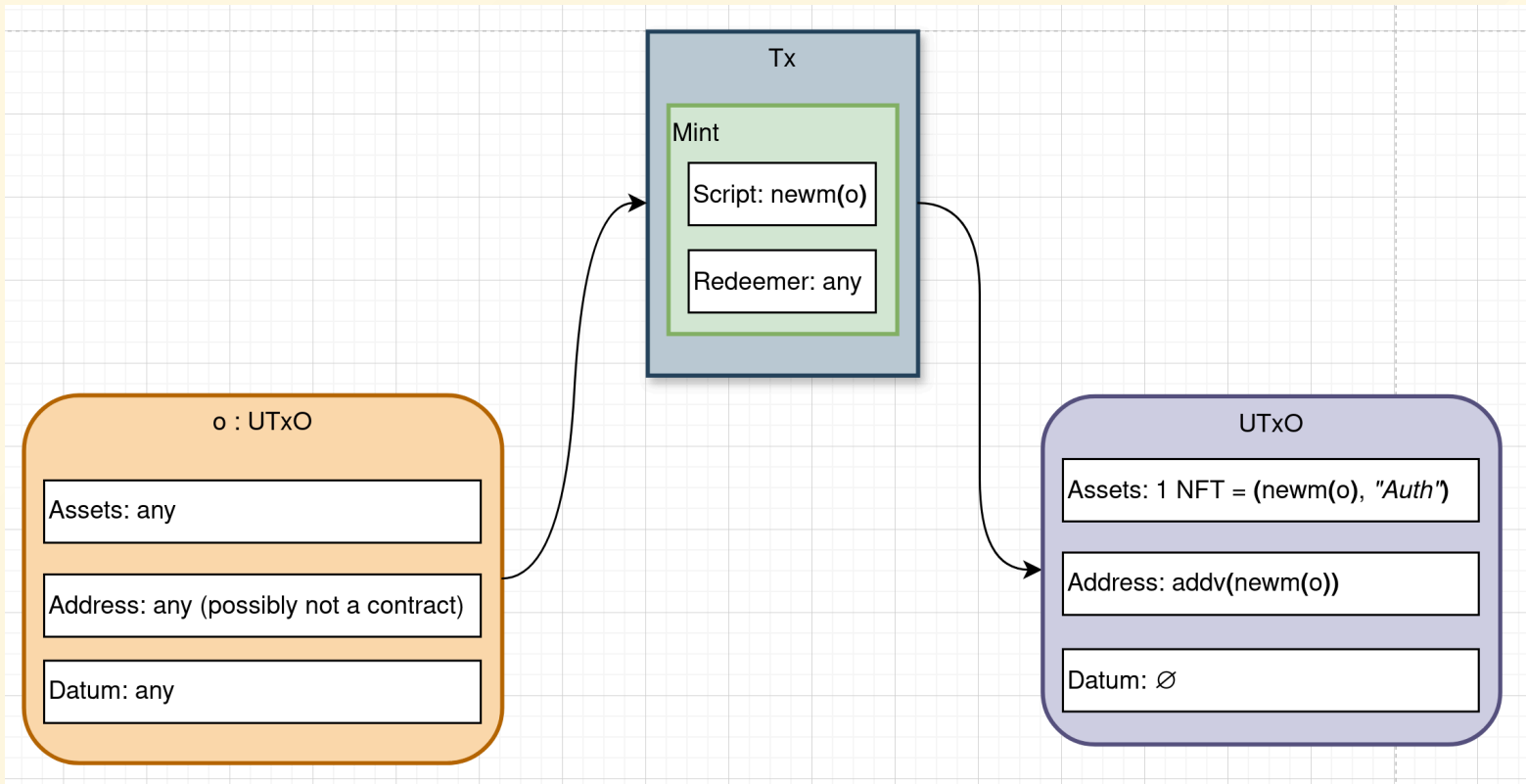
```
6  validator(newm_script: PolicyId) {
7    fn run(redeemer: AssetName, context: ScriptContext) → Bool {
8      // 0. The ScriptPurpose is to Mint a new Token
9      expect Mint(policy_id) = context.purpose
10     // 1. There is a single minted asset
11     expect [(asset, amount)]: List<AssetName, Int> =
12       context.transaction.mint
13       ▷ value.from_minted_value()
14       ▷ value.tokens(policy_id)
15       ▷ dict.to_list()
16     // 2. The amount of the minted asset is 1
17     expect 1 = amount
18     // 3. The name of the new asset is the redeemer
19     expect asset = redeemer
20     // 4. There is one output UTxO with the redeemer
21     expect [_] =
22       context.transaction.outputs
23       ▷ list.filter(
24         fn(output: Output) {
25           1 = quantity_of(output.value, policy_id, redeemer)
26         },
27       )
28     // 5. There is one output UTxO with the authorizing start NFT
29     // This makes sure that the addv script is called, since the Auth NFT is locked in the addv script
30     expect [_] =
31       context.transaction.outputs
32       ▷ list.filter(
33         fn(output: Output) {
34           1 = quantity_of(output.value, newm_script, "Auth")
35         },
36       )
37     True
38   }
39 } ✨
40
```

Example: Full Example

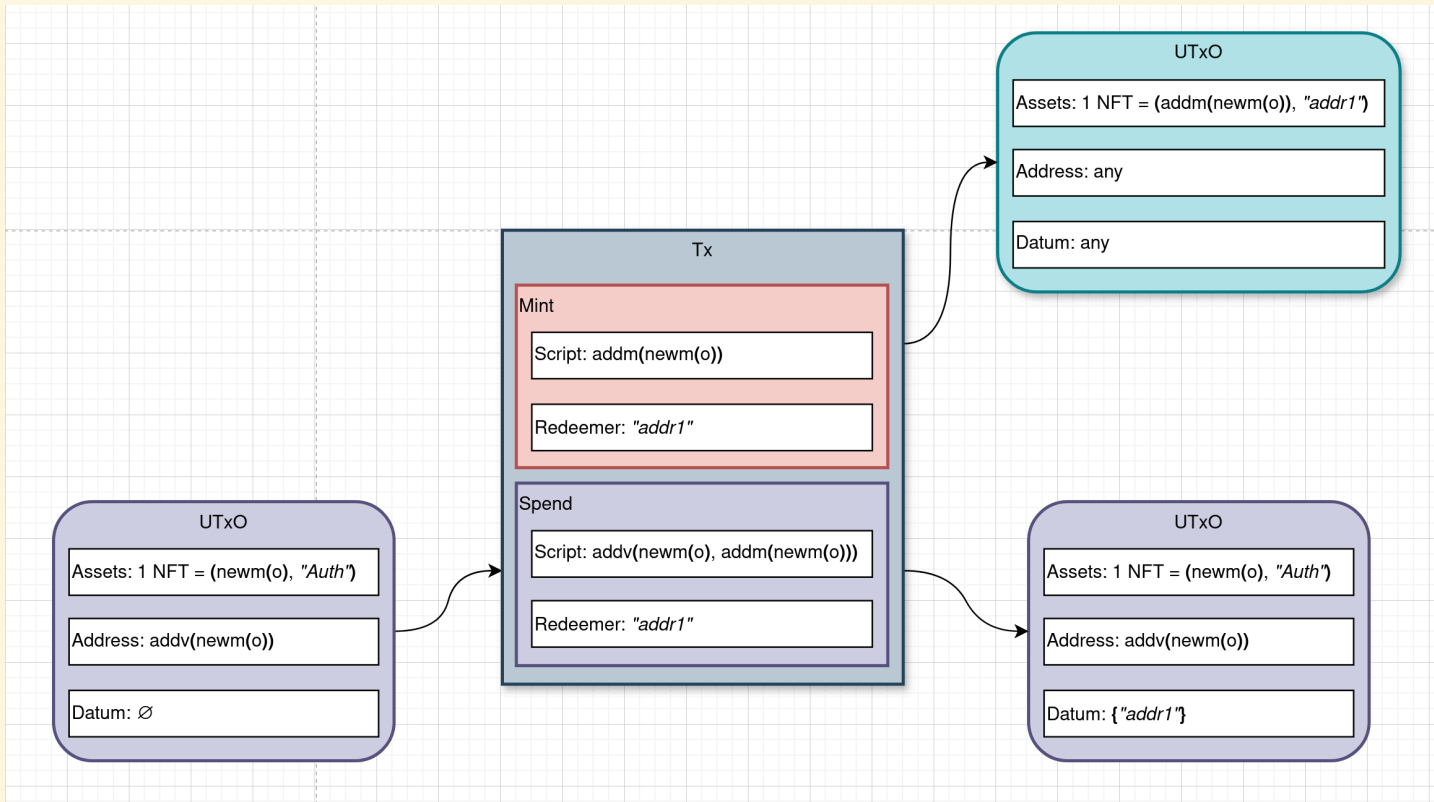


NB Cyan UTXOs contain the minted NFTs that we will use to authenticate our smart contract's state.

Example: Start Minting



Example: Minting The 1st NFT



Example: Minting The 2nd NFT

