1. EnCoins-Aiken-Core

Link: https://github.com/encryptedcoins/encoins-core-aiken

use aiken/builtin.{

add\_integer, serialise\_data, sha2\_256, snd\_pair, tail\_list,

verify\_ed25519\_signature,

}

use aiken/dict.{has\_key, to\_list}

use aiken/list.{any, filter, foldr, length, map}

use aiken/math.{abs}

use aiken/transaction.{InlineDatum, Mint, ScriptContext}

use aiken/transaction/credential.{Address}

use aiken/transaction/value.{

PolicyId, Value, ada\_asset\_name, ada\_policy\_id, add, flatten, from\_lovelace,

from\_minted\_value, lovelace\_of, merge, negate, policies, tokens, zero,

}

use encoins\_core\_aiken/value\_extra.{isNonnegativeValue}

const minAdaTxOutInLedger = 4000000

const minMaxAdaTxOutInLedger = 1000000000

// Beacon token and verifierPKH

type EncoinsPolicyParams =

(Value, ByteArray)

// Ledger address, change address, total fees

type TxParams =

(Address, Address, Int)

type EncoinsInputOnChain =

(Int, List<(ByteArray, Int)>)

type ProofHash =

ByteArray

type ProofSignature =

ByteArray

type EncoinsRedeemerOnChain =

(TxParams, EncoinsInputOnChain, ProofHash, ProofSignature)

fn hashRedeemer(red: EncoinsRedeemerOnChain) -> ByteArray {

let (params, input, proofHash, \_) = red

sha2\_256(serialise\_data((params, input, proofHash)))

}

fn checkLedgerOuputValue1(vals: List<Value>) {

when vals is {

[] -> True

[x, ..] ->

length(flatten(x)) <= 2 && lovelace\_of(x) >= minMaxAdaTxOutInLedger && checkLedgerOuputValue2(

tail\_list(vals),

)

}

}

fn checkLedgerOuputValue2(vals: List<Value>) {

when vals is {

[] -> True

[x, ..] ->

length(flatten(x)) == 2 && lovelace\_of(x) == minAdaTxOutInLedger && checkLedgerOuputValue2(

tail\_list(vals),

)

}

}

validator(par: EncoinsPolicyParams) {

fn encoinsPolicyCheck(

red: EncoinsRedeemerOnChain,

context: ScriptContext,

) -> Bool {

// Destructuring arguments

let info = context.transaction

let vMint = from\_minted\_value(info.mint)

expect Mint(encoinsSymb) = context.purpose

let (beacon, verifierPKH) = par

let ((ledgerAddr, changeAddr, feesWithSign), (v, inputs), \_, sig) = red

let fees = abs(feesWithSign)

let val = from\_lovelace(v \* 1000000)

let valFees = from\_lovelace(fees \* 1000000)

let deposits = foldr(map(inputs, snd\_pair), 0, add\_integer)

let valDeposits = from\_lovelace(deposits \* minAdaTxOutInLedger)

// Defining conditions

// Condition 0

let actual = to\_list(tokens(vMint, encoinsSymb))

let cond0 = actual == inputs && policies(vMint) == [encoinsSymb]

// Condition 1

let cond1 = verify\_ed25519\_signature(verifierPKH, hashRedeemer(red), sig)

// Condition 2

let refIns = map(info.reference\_inputs, fn(x) { x.output })

let cond2 =

any(

refIns,

fn(o) {

o.address == ledgerAddr && isNonnegativeValue(

merge(o.value, negate(beacon)),

)

},

)

// Conditions 3 and 4

let ins = map(info.inputs, fn(x) { x.output })

let ledgerSpentOuts =

filter(

ins,

fn(x) {

x.address == ledgerAddr && isNonnegativeValue(

add(x.value, ada\_policy\_id, ada\_asset\_name, -minAdaTxOutInLedger),

)

},

)

// vOuts are the values flowing out of the Ledger address

let vOuts = map(ledgerSpentOuts, fn(x) { x.value })

let vOut = foldr(vOuts, zero(), merge)

let ledgerProducedOuts =

filter(

info.outputs,

fn(x) { x.address == ledgerAddr && x.datum == InlineDatum(Void) },

)

// vIns are the values flowing into the Ledger address

let vIns = map(ledgerProducedOuts, fn(x) { x.value })

let vIn = foldr(vIns, zero(), merge)

// Wallet Mode

let cond3 = vIn == merge(vOut, val)

// Ledger Mode

let cond4 = vIn == merge(vOut, merge(vMint, merge(val, valDeposits)))

let cond5 = checkLedgerOuputValue1(vIns)

// Condition 6

let depositsOrZero =

if cond4 {

deposits

} else {

0

}

let valDepositsOrZero = from\_lovelace(depositsOrZero \* minAdaTxOutInLedger)

let valToProtocol = merge(val, merge(valFees, valDepositsOrZero))

let cond6 =

lovelace\_of(valToProtocol) >= 0 || any(

info.outputs,

fn(o) {

o.address == changeAddr && isNonnegativeValue(

merge(o.value, valToProtocol),

)

},

)

// Condition 7

let cond7 =

list.all(

value.flatten(vMint),

fn(asset) {

let (\_, \_, quantity) = asset

abs(quantity) == 1

},

)

// Imposing validation conditions

cond0 && cond1 && cond2 && ( cond3 || cond4 ) && cond5 && cond6 && cond7

}

}

type EncoinsLedgerValidatorParams {

policyId: PolicyId,

}

validator(encoinsSymb: EncoinsLedgerValidatorParams) {

fn ledgerValidatorCheck(

\_datum: Void,

\_redeemer: Void,

context: ScriptContext,

) -> Bool {

let policyId = encoinsSymb.policyId

let info = context.transaction

let rdmrs = info.redeemers

let purp = Mint(policyId)

has\_key(rdmrs, purp)

}

}