ENCOINS V2 PROTOCOL SPECIFICATION

VLADIMIR SINYAKOV

1. Introduction

Cardano is a proof-of-stake smart contract blockchain that utilizes the UTxO ledger model and has *fully deterministic* transactions. The latter means that transactions are fully constructed on the user machine or some backend before they are gossiped to the network.

On Cardano, smart contracts are implemented through scripts. A script, in essence, is a *specification* imposed on transactions. The ledger rules demand that nodes execute a particular script on the transaction data (i.e., check the specification) in a number of cases. These cases include minting or burning of a particular token or spending an output at a particular address.

ENCOINS v2 protocol is a smart contract on the Cardano blockchain that lets users create non-fungible tokens (NFTs) that can be redeemed for a predetermined value. In the following, to distinguish these NFTs from other assets on Cardano, we will call them *coins*. At the time of minting, the user must lock the corresponding value in the protocol so that it can be later redeemed. This way, the value locked in the protocol and available for redemption is always greater or equal to the sum of redemption values of all currently existing coins.

The main function of these coins is to add a privacy layer to the Cardano blockchain. A typical ENCOINS protocol transaction mints and burns several coins at once, with the difference in value being deposited into the protocol or withdrawn from it. An outside party can only observe this value difference but cannot learn the redemption values of individual coins. This is achieved by using a cryptographic primitive called *zero knowledge proofs*.

2. Preliminaries

Let H be a hash function. Let `tx` be the transaction data structure available to scripts.

3. Coin structure

Every coin is completely determined by a triplet (V, S, γ), which we refer to as the *secret key* of the coin. Each coin also has a *public name*, which is given by the equation

$$T = H(V, S, \gamma).$$

In the above,

- V is the multi-asset value that the coin can be redeemed for;
- S is a script that must succeed on a transaction that burns the coin;
- γ is a *nonce* that prevents adversaries from guessing V and S from publicly available T.

In the following, let S_0 denote an "always true" script and let $T_0 = H(0, S_0, 0)$.

4. Value update sub-protocol

Let us consider a tuple

$$\tau = (T, A', T', D, A, v, s).$$

We say that this tuple is a *value update* if there exist triplets (V, S, γ) and (V', S', γ') such that the following conditions hold:

$$\begin{aligned} T &= H(V, S, \gamma) \\ T' &= H(V', S', \gamma') \\ V' &= V + D \\ s &= \begin{cases} 1 \text{ if } S \neq S_0 \\ 0 \text{ otherwise} \end{cases} \end{aligned}$$

Let $\mathcal{U}(\tau)$ be the statement that τ is a value update. The *value update sub-protocol* is a zkSNARK protocol in which the validity of $\mathcal{U}(\tau)$ is proved.

It is important to note that the zkSNARK must be *non-malleable* for the security of this protocol.

5. Burn validation sub-protocol

Consider a pair $\beta = (\texttt{tx}, T)$. We say that this pair is a *valid burn* if there exist a triplet (V, S, γ) and a proof π such that the following conditions hold:

$$T = H(V, S, \gamma)$$
$$\pi \quad \text{proves} \quad S(\texttt{tx})$$

Let $\mathcal{B}(\beta)$ be the statement that β is a valid burn. The *burn validation sub-protocol* is a zkSNARK protocol in which the validity of $\mathcal{B}(\beta)$ is proved.

6. Specification: coin minting policy

In this section, we define the *coin minting policy*. It is a statement about transaction `tx` that must be true whenever a coin is minted or burned.

Let *ENCOINS transaction input* be the following tuple

$$\iota = \left(n, A_a, A_c, \mathcal{S}, \{\tau_i\}_{i=1}^n, \{\pi_i\}_{i=1}^n, \{\pi'_i\}_{i=1}^n\right).$$

Here $\tau_i = (T_i, A'_i, T_{i'}, D_i, A_i, v_i, s_i)$. Let $V(\texttt{tx})$ be the total value of all transaction inputs at address A_a and $V'(\texttt{tx})$ be the total value of all transaction outputs at address A_a.

Now, we introduce the transaction validity conditions.

<u>The beacon data is valid:</u>

1. The transaction input with the beacon token must be referenced. It must have datum $(A_a, A_c, \mathcal{S})$.

<u>The script outputs are valid:</u>

2. There's at most one transaction output at address A_a. It must have datum () and contain at most k different tokens.
3. Every transaction output at address A_c must have datum () and contain only *ada* and coins. The coins must have the currency symbol $\mathcal{S}$.

<u>Only the owner can spend UTxOs at A_c:</u>

4. Every coin from the transaction inputs at address A_c must be burned in the transaction.

<u>The coin mint/burn data is valid:</u>

5. For all $i \in [n]$, $T_i = T_0$ OR the coin with the public name T_i is burned.
6. For all $i \in [n]$, $T'_i = T_0$ OR the coin with the public name T'_i is minted.
7. For all $i \in [n]$ such that T'_i is minted, there must be an output at address A'_i. It must have datum () and contain only *ada* and the coin T'_i.
8. No other coins are minted or burned.

<u>The redemptions are honored:</u>

9. For every $i \in [n]$, if $v_i > 0$ then there must be an output at address A_i with datum () and value v_i.

<u>Value updates succeed:</u>

10. π_i proves $\mathcal{U}(\tau_i)$, $i \in [n]$.

<u>Burn validation succeeds:</u>

11. $s_i = 0$ OR π'_i proves $\mathcal{B}(\texttt{tx}, T_i)$, $i \in [n]$.

<u>Value balance is preserved:</u>

12. $V'(\texttt{tx}) = V(\texttt{tx}) + \sum_{i=1}^n D_i$.

Let us say that `tx` is an *ENCOINS v2 transaction* if there exists an ENCOINS transaction input ι such that conditions 1-12 hold.

7. Specification: other scripts

In this section, we consider other scripts involved in the protocol. Unlike the minting policy, these scripts are simple Plutus V3 scripts.

The *beacon token*'s minting policy has two conditions:

1. It requires a specific UTxO to be consumed.
2. It must be placed in an output UTxO at an “always false” validator script address with datum $(A_a, A_c, \mathcal{S})$.

The *asset lock* and the *coin ledger* validators, i.e., the validators with addresses A_a and A_c, respectively, return “True” if and only if the minting policy is invoked in the transaction.

Address A_a has the staking part. The staking credential corresponds to the staking validator that returns “True” if there exists an address A_w such that the following conditions hold:

1. The value of transaction outputs at address A_w is greater or equal to the value of transaction inputs at that address plus the value of ada being withdrawn.
2. The input with the *withdrawal authorization token* is referenced. Its address is also A_w.

Here, the withdrawal authorization token is a unique NFT that allows the party controlling it to make a withdrawal transaction.