CardMix: a Private Transactions Protocol on Cardano

Version 0.8

1 Introduction

Public blockchains are deigned to store the information in it forever. When a cryptocurrency wallet gets linked to the real-world identity of its owner, it enables several potential risk factors for the person and also may be considered as a violation of their privacy rights. This linking between the user identity and the wallet addresses can happen in a number of ways. Besides formal identification procedures, it may result from ordinary purchase of goods and services using cryptocurrency, or, for example, when an employer pays an employee in cryptocurrency.

This prompts the need for *forward privacy* on public blockchains. The tools that provide such privacy do not erase the history of the user's transactions per say but allow them to transfer funds to a fresh cryptocurrency wallet with no transaction history. The most popular type of tools in these category are called *mixers* or *tumblers*. The protocol called CardMix presented here belongs to this class as well.

Mixers often rely on zero-knowledge cryptography [4, 7, 6, 10] which has its uses in both private transactions systems and in scalability solutions [2, 3, 8, 9, 1, 5]. This work utilizes the zero-knowledge proving system called sigma protocols that has been previously used, for example, in ErgoMix. Our solution is different from ErgoMix in that in CardMix it is theoretically possible to create adequate privacy in less mixing rounds, i.e. with less interactivity between the user and the protocol. The more actions a user needs to take to anonymize the transaction, the more potential there is for an external observer to link the inflowing and outflowing coins (for example, by means of timing analysis).

This paper is split into several sections. In Section 2 we give a mathematical description of the proposed protocol. In Section 3 we present an overview of the CardMix dApp structure. In Section 4 we describe the protocol token that ensures normal functioning and continuous development of the project. Finally, in Appendix we give some details on the structure of CardMix's on-chain state.

2 Protocol design

In this section, we give an overview of the CardMix protocol that the CardMix dApp implements.

2.1 Basic 1-of-n Sigma Protocol

Let G be a cyclic multiplicative group of a prime order q. Any element $g \in G$ other than the identity is a generator of this group. Let us fix some generator g_1 and consider generators g_2 and g' such that the discrete logarithm relations between them and g_1 are not known.

Consider sets $D, W \subset G$. Let $d_1, \ldots, d_n \in D$, $A \in \mathbb{Z}_q \setminus \{0\}$, $d' \in G$, and $w \notin W$. The key element of CardMix's design is the sigma protocol for proving the following statement:

$$\text{I KNOW } x \in \mathbb{Z}_q, j \in \{1, \ldots, n\} \text{ SUCH THAT } g_1^x = d_j,\ (g')^x = d', g_2^{Ax} = w. \tag{1}$$

Following the ideas described in [4] and [1], we present the communication protocol for proving the knowledge of x and j:

1. First, the prover generates random numbers r_j, z_i, e_i for $i \in \{1, \ldots, n\} \setminus \{j\}$. They then compute $a_j = g_1^{r_j}$, $a_i = g_1^{z_i} d_i^{-e_i}$, $b_j = (g')^{r_j}$, $b_i = (g')^{z_i}(d')^{-e_i}$, $c_j = g_2^{Ar_j}$, $c_i = g_2^{Az_i} w^{-e_i}$. Finally, the prover sends commitment $(A, d', w, \{d_i\}, \{a_i\}, \{b_i\}, \{c_i\})$ to the verifier.
2. The verifier sends back a random number $s \in \mathbb{Z}_q$.

3. The prover calculates $e_j = s - \sum_{i \neq j} e_i$, $z_j = r_j + e_j x$ and returns $(\{e_i\}, \{z_i\})$ to the verifier.

4. To be sure that the prover knows x, the verifier must check the equalities $g_1^{z_i} = a_i d^{e_i}$, $(g')^{z_i} = b_i (d')^{e_i}$, $g_2^{Az_i} = c_i w^{e_i}$, $s = \sum e_i$.

We use the Fiat-Shamir transform to make this protocol non-interactive and compute s as a hash of the commitment data $(A, d', w, \{d_i\}, \{a_i\}, \{b_i\}, \{c_i\})$.

2.2 The CardMix Protocol

CardMix protocol consists of several *mixer instances*. Each mixer instance has its own pool of deposits associated with it. A mixer instance is parameterized by *token value* of the deposits and *mixing round*. Each mixer instance uses its own generators (g_1, g_2). All generators, except one, are chosen during the trusted setup ceremony (see below) so that the discrete logarithm relation between them is not known.

A user of the protocol has three commands available to them.

1. **Deposit**. Sends a fixed token value into the first mixing round.

2. **Mix**. Withdraw a deposit from one mixing round into the next one.

3. **Withdraw**. Withdraw a deposit from the protocol.

When sending the funds into the protocol, the depositor generates a random *secret key* $x \in \mathbb{Z}_q$ and communicates the element $d = g_1^x \in G$ called the *deposit key*.

When withdrawing funds, the recipient picks n deposit keys from the set D, including d_j for which they know the *secret key* x: $d_j = g_1^x$. Then they pick the address to withdraw into, transform it to the representation $A \in \mathbb{Z}_q$, and calculate $d' = (g')^x$, $w = g_2^{Ax}$. Here g' is the first generator of the next mixing round. The recipient then submits zero-knowledge proof (described above) that they know x and j. The element $w \in G$ called the *withdraw key* must not belong to the set of used withdraw keys W. A successful withdraw transaction updates W by adding w to it. Sets D, W are different for each mixer instance.

2.3 Properties

Here we summarize the main privacy and security properties of the protocol:

- At any time, the number of deposits in the mixer instance is always greater or equal than the number of withdrawals made from it;

- It is always possible to withdraw a deposit for which the user knows the secret key;

- A deposit can be withdrawn only into the address specified by the user with the knowledge of the secret key;

- When withdrawing after k rounds, the recipient's funds could correspond to up to n^k deposits. It is the potentially attainable maximum of uncertainty for this protocol.

2.4 The Setup Ceremony

As mentioned above, every mixer instance has its own pair of cyclic group generators. Note that each proof involves a triple of generators (g_1, g_2, g'). Therefore, we only need three generators in total: we may use (g_1, g_2, g_3) for even mixing rounds and (g_3, g_2, g_1) for odd mixing rounds.

The security of the protocol is based on the assumption that the discrete logarithm relations between those generators are not known. In order to achieve this, we propose to do the following trusted setup ceremony. Each participant generates a random number $y \in \mathbb{Z}_q$, computes $c = g_1^y$, and makes a contribution by sending c along with proof of knowledge of y. An *honest* contributor does not keep the record of y.

Given the list of valid contributions $\{c_1, \ldots, c_l\}$, generator g_2 is picked as the product of all contributions: $g_2 = \prod_{i=1}^{l} c_i = g_1^{\sum_{i=1}^{l} y_i}$. From the difficulty of the discrete logarithm problem, we get that solving this equation is also hard provided that there was at least one honest contributor. This process is repeated for g_3 as well.

3 The CardMix dApp

CardMix dApp infrastructure consists of the Web Frontend and a number of servers running the Relayer App, Cardano Node, Cardano Wallet Backend, and Chain Index App.

3.1 Web Frontend

In order to interact with the protocol, the user can connect to the frontend using a CIP-30 compatible browser wallet.

Deposit function comes with the following options. The user may choose the token to deposit and its amount. Cardano ledger rules state that every UTXO must contain a certain amount of ADA. For this reason, every native asset deposit (except for ADA deposits) is bundled with an additional 2 ADA payment. These 2 ADA are recovered when the user withdraws from the protocol. During the execution of the deposit command, a secret key x is generated on the frontend and is revealed to the user. The deposit transaction sends funds to the corresponding script address and attaches $d = g_1^x$ as datum (g_1 is the first group generator of the respective mixer instance).

Mix and Withdraw functions ask the user to enter the secret key x. The Web Frontend retrieves the on-chain state of CardMix from a randomly selected active relayer. It then randomly selects n deposits (adjustable parameter) from the same mixer instance. One of those deposits must be $d = g_1^x$. The frontend computes zero-knowledge proof of (1) and sends it to the relayer along with the public commitment data, withdrawal address (required for withdraw requests), and the mixer instance description.

3.2 Relayer app

Each relayer performs three main functions:

- It tracks the actual on-chain state of the whole CardMix protocol (all mixer instances mentioned in configuration files);
- It maintains the list of pending mix/withdrawal requests received from the protocol users;
- It automatically executes certain types of transactions that change the on-chain state of CardMix protocol.

The first type of transactions become available when a user makes a new deposit. The relayer moves the funds from the *deposit script* into the *mixer script* and mints a *deposit token*. This deposit token represents the proof that a deposit with a certain deposit key d has been made at some point. Thanks to input referencing, this proof can be efficiently supplied to the withdraw transaction where it is needed.

Upon minting, deposit tokens are sent into a special script that does not allow non-ADA withdrawals. This achieves two goals:

1. Almost all deposit tokens are constantly available for reference in mix/withdraw transactions;
2. The protocol achieves the maximal information storage efficiency allowed by the ledger rules as many deposit tokens can be stored in a single UTXO thus reducing the network fees paid by the user.

An ordering on G is introduced to efficiently keep track of the set W of used withdraw keys. This set is represented on-chain by *withdraw tokens*. Each such token contains the information about the current withdraw key w and the next one w'. Thus, a collection of all withdraw tokens forms an ordered list of unique elements. These tokens are locked upon minting by the similarly defined script that does not allow non-ADA withdrawals. This on-chain representation of W enables us to easily verify a statement of the form $w \notin W$. Indeed, it is sufficient to present a token with data (w', w'') such that $w' < w$ and $w < w''$.

The second type of transactions become available when a relayer receives a mix/withdraw request from a user. Assume that before the transaction we had a withdraw token with data (w_{j-1}, w_{j+1}). The transaction burns this token and mints two instead with data (w_{j-1}, w_j) and (w_j, w_{j+1}), respectively. This allows to maintain the on-chain representation of W.

3.3 Fees

For both deposit and mix/withdraw transaction, a relayer is allowed to take a fee (from the user's deposit) that covers the Cardano network fee for the respective transaction. In addition, for every mix/withdraw transaction the relayer may charge 0.05% of the deposit. Thus, every transaction submitted by a relayer results in net profit for them.

4 Protocol utility token

4.1 MIX token

To enable the sustainable development of the protocol, we propose to introduce a utility token (MIX token). The token is used for two purposes.

First, it facilitates voting on protocol improvement proposals and usage of the project's treasury. Once the governance mechanisms are implemented, token holders will control the parameters of the protocol, vote on improvement proposals, and determine the direction of the project's future development. They will also vote on the allocation of treasury funds as they are unlocked. Governance is a post-launch feature.

Second, MIX tokens serve as collateral that relayers must stake to be added to the list of relayers. For a mix/withdraw transaction, the Web Frontend will connect the user one of the relayers from this list. The stake amount we propose to use is 100 000 MIX.

It should be noted that, as an open source software, the Relayer App can be run by anyone (with some technical knowledge). By modifying the Web Frontend files, an experienced user will be able to connect to a custom relayer. However, this is not generally advised as choosing a relayer at random provides better privacy guarantees.

4.2 Initial token distribution

The MIX token has a fixed supply of 100 000 000 tokens, minted at the protocol launch. The initial distribution of the token is summarized in the following pie chart below.

Below we give the description of each category.

- 15 % of tokens are sent to the early supporters of the project. Of those, 10 % are given to the ISPO participants, 2 % are reserved for partnerships, and 3 % for bug bounty rewards.
- 15 % of the tokens are sent to the participants of the trusted setup ceremony.
- 30 % of the tokens are sent to the early users of the protocol. Every depositor gets some MIX to (partially) offset the fees until the funds run out.
- 25 % of the tokens will constitute the project's treasury. Treasury tokens will start unlocking 6 months after the protocol launch at the rate of 1% per month.
- The final 15 % will be given to the original development team of the CardMix dApp. The team tokens will be unlocked in 3 vesting tranches: 5% at launch, 5% after 6 months, and the last 5% after 12 months.

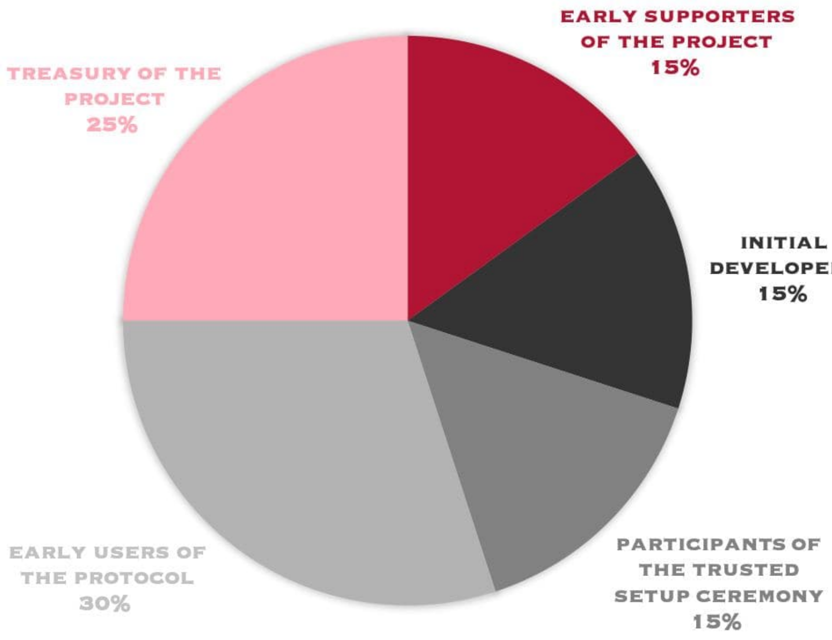
References

[1] Amitabh Saxena Alexander Chepurnoy. Zerojoin: Combining zerocoin and coinjoin, 2020.

[2] Kurt M. Alonso and Jordi Herrera Joancomartí. Monero: Privacy in the blockchain, 2018. `https://eprint.iacr.org/2018/535.pdf`.

[3] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin, 2014. `http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf`.

[4] Ivan Damgård. On σ-protocols. CPT 2010, v.2, 2010.



[5] Daniel Firth, Lukasz Gołębiewski, Mason Mackaman, Ryan Matovu, Pawel Szulc, and Morgan Thomas. Orbis 1.0: A general-purpose layer 2 zero-knowledge rollup protocol. White paper, 2022. `https://papers.orbisprotocol.com/whitepaper.pdf`.

[6] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[7] Jens Groth and Markulf Kohlweiss. One-out-of-many proofs: Or how to leak a secret and spend a coin. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 253–280, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[8] Aram Jivanyan. Lelantus: A new design for anonymous and confidential cryptocurrencies. Cryptology ePrint Archive, Report 2019/373, 2019. `https://ia.cr/2019/373`.

[9] Alexey Pertsev, Roman Semenov, and Roman Storm. Tornado cash privacy solution, 2019. `https://tornado.cash/Tornado.cash_whitepaper_v1.4.pdf`.

[10] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 704–737, Cham, 2020. Springer International Publishing.

5 Appendix: On-Chain Scripts

Below we present (in textual form) every on-chain script used in the protocol as a set of conditions that all must be satisfied in order for the script to run successfully.

5.1 Validators

In this subsection we describe the validators.

5.1.1 ADAWithdraw

Redeemer: None.
Datum: None.
Condition: The sum of non-ADA outputs at this script address is greater than the sum of non-ADA inputs.

5.1.2 MixerDeposit

Redeemer: None.
Datum: deposit key d.
Condition: A deposit token is minted in this transaction.

5.1.3 Mixer

Redeemer: A pair of withdraw tokens (w, w'').
Datum: None.
Condition: A withdraw token is minted in this transaction.

5.2 Minting policies

In this subsection we describe the minting policies.

DepositToken. Conditions:

1. Exactly one token with this minting policy is minted. Its name is a byte representation of the deposit key d.
2. The transaction produces an UTXO at Mixer address with value greater or equal than *mixerValueAfterDeposit*.
3. The transaction produces an UTXO at ADAWithdraw address with value greater or equal than the deposit token.

WithdrawToken. Conditions:

1. A correct proof is supplied for input data $(A, w, d', \{d_i\})$.
2. Exactly two tokens with this minting policy are minted and one token is burned. Their names are byte representations of withdraw key pairs. The minted tokens correspond to (w', w) and (w, w''), the burned token corresponds to (w', w'').
3. The following relations hold: $w' < w$ and $w < w''$.
4. The transaction produces an UTXO at withdrawal address with value greater or equal than *mixerValueAfterWithdraw* and the datum be the byte representation of d'.
5. The transaction produces an UTXO at ADAWithdraw address with value greater or equal than the sum of two minted withdraw tokens.
6. A is a representation of withdraw address in G.
7. For all i, a deposit token with the deposit key d_i is in the referenced UTXOs.