CS291D Final Report: a Basic Zcash Implementation

Gwyneth Allwright, Karl Wang, Dewei Zeng

December 8, 2020

Abstract

In this project, we attempt a basic implementation of Zerocash [1] in Python. Zerocash is a ledger-based digital currency that makes use of zero-knowledge Succinct Non-Interactive Arguments of Knowledge (zk-SNARKs) to provide stronger privacy guarantees than currencies such as Bitcoin [2] and Zerocoin [3]. This functionality is provided through a decentralized anonymous payment (DAP) scheme that hides a transaction's origin, destination and amount. We follow [1] to implement the following core functions: `Setup`, `CreateAddress`, `Receive`, `Mint`, `VerifyTransaction`, `Pour`, `KeyGen`, `Prove` and `Verify`, which form the foundations of Zerocash.

Contents

1 Introduction

Data on blockchains such as Bitcoin is public, including the sender, receiver and the amount of money transferred in a payment. While Bitcoin users often utilize different identities to disguise their transactions, it is possible to gain access to both the structure of the transaction graph and the values and dates of transactions. Zerocoin, a cryptographic extension to Bitcoin, aims to introduce better privacy guarantees without requiring new trusted parties, but it still cannot hide the location that money is sent to, as well as the amount of money that is transferred [3]. In addition, it lacks some features of fully-fledged cryptocurrencies, such as payments of exact values.

In order to solve these problems with Bitcoin and Zerocoin, a new digital currency known as Zerocash was devised. Zerocash makes the sender, receiver and amount of money transferred in a payment anonymous, while also improving on the efficiency of Zerocoin [1]. These outcomes are achieved with the help of zk-SNARKs, which are efficient variants of zero-knowledge proofs of knowledge. Zero-knowledge proofs allow the prover of a certain statement to demonstrate that the statement in question is true without revealing additional information about the statement that could result in a compromise of privacy.

One of the primary objectives of this project is to explore zk-SNARKs and their potential applications in the world of blockchains and cryptocurrencies. To achieve this, we use existing zk-SNARK tooling to implement a minimal version of Zerocash in Python with the purpose of gaining a better understanding of Zerocash's theoretical underpinnings. This would be a first step towards demonstrating that zk-SNARKs are a feasible method of enhancing the privacy and performance of transactions on a simple blockchain. Next steps would include benchmarking and comparisons to a blockchain with similar functionality that does not make use of zk-SNARKs.

2 Problem Definition

We wish to understand how to incorporate zk-SNARKs into a basic blockchain in order to improve the blockchain's privacy guarantees. The setup of our scheme must not require any trust beyond a one-time trusted setup of public parameters. The implementation needs to support the minting, merging and splitting of coins without exposing the identities of the users who perform the transactions and the amounts of the currency involved.

3 Solution

The above objectives can be achieved through the combination of zk-SNARKs and a decentralized anonymous payment (DAP) scheme. As part of the Zerocash DAP scheme, we implement the following core functions: `Setup`, `CreateAddress`, `Receive`, `Mint`, `VerifyTransaction` and `Pour` [1]. For the zk-SNARK, we require the additional functions (`KeyGen`, `Prove`, `Verify`) [1]. In the sections that follow, we provide an overview of the DAP scheme, zk-SNARK and the associated functions just mentioned.

3.1 DAP Scheme

3.1.1 Basecoin

The Zerocash protocol is applied on top of a ledger-based currency (e.g. Bitcoin). This ledger-based currency is referred to as the *basecoin*. All basecoin transactions are recorded in an append-only ledger, which can be accessed by all Zerocash users at all times.

In addition to the basecoin transactions, Zerocash includes two new kinds of transactions — minting and pouring — which will be described later. Mint and pour transactions are also recorded in the basecoin ledger.

3.1.2 Public Parameters

In addition to the basecoin ledger, users have access to a set of public parameters. These are part of the one-time trusted setup that takes place before other functions are allowed to execute.

3.1.3 Address Key Pairs

Users may generate as many public and private address key pairs as they desire. The public address keys are published with the purpose of allowing users to make payments among themselves. The secret keys are used for receiving payments.

3.1.4 Coins

Coins are data structures that encapsulate the following information:

- A coin commitment, which is a string that we append to the basecoin ledger once the coin is minted.
- A coin value (between 0 and some parameter $\mathrm{v}_{\max}$) that specifies the coin denomination in basecoin units.
- A coin serial number, which is a string that uniquely identifies the coin and is used to prevent double-spending.
- A coin address — the public address key of the user who owns the coin.

3.1.5 Coin-Related Data Structures

The Zerocash protocol requires us to maintain the following coin-related information:

- A Merkle tree over coin commitments.
- A list of coin commitments that appear in mint and pour transactions.
- A list of all coin serial numbers that appear in pour transactions.

For efficiency reasons, it is useful to store the latter two lists (which could also be obtained from the basecoin ledger) separately.

3.1.6 New Transaction 1: Mint

Mint transactions are used to create coins. At its most basic, a mint transaction can be described as a tuple (cm, v), where cm is the commitment of the minted coin and v is its value. Whenever a coin is minted, this tuple is placed on the basecoin ledger.

3.1.7 New Transaction 2: Pour

Pour transactions record the pouring of two input coins into two new output coins (thereby spending the two initial coins). At its most basic, a pour transaction can be described as a tuple (rt, $\mathrm{sn}_1^{\mathrm{old}}$, $\mathrm{sn}_2^{\mathrm{old}}$, $\mathrm{cm}_1^{\mathrm{new}}$, $\mathrm{cm}_2^{\mathrm{new}}$, v_{pub}, info), where rt is the root of the Merkle tree over coin commitments, the sn are the serial numbers of the old coins, the cm are the commitments of the new coins, v_{pub} is a coin value and info is an arbitrary string. Pour transactions may also include implementation-specific information.

3.2 zk-SNARK

3.2.1 Arithmetic Circuits

For a given field $\mathbb{F}$, an arithmetic circuit C takes as input n field elements $\in \mathbb{F}$ and returns m field elements $\in \mathbb{F}$. We can therefore think of C as a map $\mathbb{F}^n \longrightarrow \mathbb{F}^m$.

In the Zerocash construction, we decompose the circuit input that lives in $\mathbb{F}^n$ into a main input and auxiliary input, where the latter is known as the *witness*. If the dimensions of these two subinputs are u and v respectively, then we can write C: $\mathbb{F}^u \times \mathbb{F}^v \longrightarrow \mathbb{F}^m$.

3.2.2 Circuit Satisfiability

zk-SNARKs can be described in terms of arithmetic circuit satisfiability. The key relationship involved is the following:

$$\text{For a given } \; X \in \mathbb{F}^u, \;\; \exists A \in \mathbb{F}^v \; \text{ such that } \; C(X, A) = 0^m. \tag{1}$$

The set of all X that satisfy Equation 1 form the set $\mathbb{L}_C$. The statement that a prover would want to demonstrate is that for a given X, we have $X \in \mathbb{L}_C$.

3.2.3 Properties

- Completeness:
- Succinctness:
- Zero knowledge:

3.3 Summary

The DAP scheme described above is implemented by means of a tuple of polynomial-time algorithms (`Setup`, `CreateAddress`, `Mint`, `Pour`, `VerifyTransaction`, `Receive`). The zk-SNARK consists of a tuple of polynomial-time functions (`KeyGen`, `Prove`, `Verify`). In the following subsection, we describe the arguments, outputs and interrelation of these functions.

3.4 Important Functions

3.4.1 `Setup`

The purpose of `Setup` is to perform the one-time trusted setup of public parameters. It takes as input a security parameter and produces the following list of public parameters as output:

- $(\mathrm{pk}_{\mathrm{POUR}}, \mathrm{vk}_{\mathrm{POUR}})$: a proving and verification key pair for the zk-SNARK.
- $\mathrm{pp}_{\mathrm{enc}}$: parameters for the encryption scheme.
- $\mathrm{pp}_{\mathrm{sig}}$: parameters for the digital signature scheme.

All three of the above are functions of the provided security parameter.

3.4.2 `CreateAddress`

The purpose of `CreateAddress` is to generate public-private address key pairs for users. It takes as input the public parameters generated by `Setup` and produces a key pair as output.

3.4.3 `Mint`

3.4.4 `KeyGen`

3.4.5 `Prove`

3.4.6 `Verify`

3.4.7 `Pour`

3.4.8 `Receive`

3.4.9 `VerifyTransaction`

4 Related Work

5 Evaluation

References

[1] Eli Ben-Sasson et al. “Zerocash: Decentralized Anonymous Payments from Bitcoin”. In: *IEEE Symposium on Security and Privacy* (2014), pp. 459–474.

[2] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: `https://bitcoin.org/bitcoin.pdf`.

[3] Ian Miers et al. “Zerocoin: Anonymous Distributed E-Cash from Bitcoin”. In: *IEEE Symposium on Security and Privacy* (2013), pp. 397–411.