Zerocoin Light Node Protocol

Probabilistic Private Multi-Party Computation-Based Protocol

Matias Furszyfer
matiasfurszyfer@protonmail.com

November 5, 2018

Abstract

Zerocoin, a non-interactive zero-knowledge proof protocol[1] for UTXO-based blockchains, is one of the most vetted decentralized privacy protocols. At the time of writing, more than 500 projects have forked the open-source code and are using it around the globe.

The Zerocoin protocol breaks transaction traceability by using RSA accumulators, Pedersen commitments, and a predefined set of coin denominations to burn (mint) and create (spend) new, exactly equivalent value coins. These new coins are delinked from the transaction history graph, preventing graph analysis, all without the need to introduce trusted parties. However, although Zerocoin provides strong transaction anonymity, it was only designed for use by full nodes. This paper describes a Zerocoin implementation for light nodes, the Zerocoin Light Node protocol (ZLN).

We introduce a new extension to the PIVX peer-to-peer custom Zerocoin protocol to demonstrate that light nodes can privately spend their minted zPIV (Zerocoin-enabled PIVX coins) without needing to request or store the whole set of public commitments (aka public part of the "zerocoin"). The ZLN protocol reduces the amount of data and network usage required by light nodes, and allows users to define their desired privacy level by using a Bloom filter[2] on data derived from commitment values to obfuscate their goal.

This document will not describe the Zerocoin protocol or Bloom filters in detail. Further information can be obtained from their respective white papers[1][2].

Motivation

As usage of Zerocoin blockchains increases, the amount of bandwidth, data and computation needed to create a zerocoin spend transaction also increases.

At the same time, an increasing number of users are using cryptocurrencies on their mobile devices, which has increased the demand for light clients. Light clients only store block headers and the user’s own transactions, and are not always online or synchronized. Therefore, they cannot practically obtain the data needed to create a zerocoin spend transaction.

We seek to address both these trends by creating a protocol which allows light nodes to outsource part of the zerocoin calculation to full nodes, without unduly compromising privacy for the light client user or the network as a whole.

Since the goal is to integrate this with the existing zPIV system, our focus is on creating a protocol implementation for light nodes which is entirely compatible with the current Zerocoin base cryptography. Any change here would fundamentally alter the nature of the protocol and likely require a network consensus change, and is thus beyond the scope of this paper.

Introduction

In the Zerocoin protocol, a commitment scheme is used to bond and hide the so-called “zerocoins” that are added to a cryptographic accumulator as part of the minting process. One of the requirements for being able to spend these zerocoins is a proof of knowledge of a commitment value previously added to the accumulator. To demonstrate this knowledge without leaking information, a calculation must be performed on the entire set of commitment values ever added to the public accumulator, omitting the commitment value of the zerocoin to be spent.

Despite concessions to simplicity such as accumulator checkpoints, this calculation is extremely impractical for light nodes to attempt. Nevertheless, as the number of light nodes increases, there is significant pressure for privacy-oriented currencies such as PIVX to provide a way for these nodes to spend their zerocoins.

To solve this problem, we present the ZLN protocol and Probabilistic Private Multi-Party Computation protocol, which allows us to outsource the public parts of the witness calculation to full nodes, reducing the computational burden on light nodes and making it possible for them to perform zerocoin spend transactions.

Design rationale

To generate a valid witness for the accumulator proof of knowledge, it is necessary to add the commitment values for every zerocoin except the one being spent to the accumulator. Since full nodes generally have considerable computational, storage and bandwidth resources available, the standard approach has been to consider this as a single calculation and compute it locally, using accumulator checkpoints stored in block headers to reduce the burden somewhat. This is also the most secure and private approach, so generally there has been no advantage to thinking about this calculation step in any other manner.

However, this option is not available to light nodes, which have limited computational resources and access to data. Specifically, since light nodes only store the block headers and not the content of blocks, they have no information about the commitment values added after any zerocoin they want to spend, and are thus unable to locally compute the witness calculation.

But while light nodes have limited access to the data needed for the witness calculation, they do have some. Most importantly, light nodes have the secret data for the zerocoins they create themselves to generate a valid serial signature number of knowledge. They can also access the value of the regular accumulator checkpoint that occurs every certain number of blocks[1] and validate it against the stored checksum in every block header.

By choosing one of these checkpointed accumulator values, all that remains is to add the values of those zerocoins minted after the checkpoint (except the one being spent). These values are not secret, so it is feasible to outsource this part of the calculation to one or more full nodes, who by definition have direct access to the data required.

Of course, we cannot simply ask a remote full node to perform the full accumulator calculation, as this will reveal which coin we intend to spend. However, we can obfuscate this information by requesting the full node to perform most of the calculation we need without revealing precisely which coin(s) we plan to spend.

To understand why this is possible, we need to consider the accumulator witness calculation employed by the Zerocoin protocol.

In brief, the simplest way to find the accumulator witness, *w*, is to calculate:

$$w = ((((u{}^\wedge c_1 \bmod \mathrm{N}){}^\wedge c_2 \bmod \mathrm{N}){}^\wedge c_3 \bmod \mathrm{N}) \ldots{}^\wedge c_n \bmod \mathrm{N}), \quad \mathrm{c_i} \notin \mathrm{w}$$

Where u and N are public values, $c_{1,\, 2,\, 3\, \ldots\, j}$ is the complete set of zerocoin commitment values, and c_i is the commitment value for the zerocoin we want to spend, and which must be omitted from the calculation.

When performing this calculation locally, the most logical approach is to start at the earliest commitment value and proceed chronologically through the blocks, adding values to the accumulator as we find them.[2] But the calculation does not have to be performed in this manner. The accumulator witness calculation is quasi-commutative, meaning we can choose $\mathrm{c}_{1,2,3..j}$ in any order and always achieve the same result, provided each value is operated on exactly once.

Using this property, we can outsource large parts of the calculation to a full node, omitting arbitrarily many values for c_n, to generate a partial witness value. Once the calculation is

[1] The precise number of blocks between checkpoints varies by blockchain.

[2] This is also the method used to calculate the accumulator checkpoints

complete, the full node returns the accumulator value, along with all the omitted commitment values. These values can then be added later locally to complete the full witness calculation.

Crucially, the full node performing the partial witness calculation will have no way of knowing which of the omitted values is associated with the zerocoin we intend to spend.

To further obfuscate things, we can also split this partial witness calculation between arbitrarily many different nodes, provided no nodes duplicate any commitment values.

For example, imagine a simplified calculation involving only 8 zerocoins and 8 associated commitment values, $c_{1,2,3...8}$. Alice, a light node, wants to spend the zerocoin with associated commitment value c_2. To do so, Alice requests help from two full nodes, Bob and Carol.

Alice asks Bob to calculate:

$$(((u\text{^}c_1 \bmod \text{N})\text{^}c_3 \bmod \text{N})\text{^}c_4 \bmod \text{N}) = A_1$$

and return the value for c_2. Alice already knows this value, as it is the one associated with her zerocoin, but she does not want Bob to know this.

Alice then passes A_1 to Carol, who calculates:

$$(((w_1\text{^}c_5 \bmod \text{N})\text{^}c_6 \bmod \text{N})\text{^}c_8 \bmod \text{N}) = A_2$$

and returns the value for c_7.

Alice then calculates $(A_2\text{^}c_7 \bmod \text{N}) = w$, the final value she needs to spend her coin.

Even if Bob and Carol share information about their partial calculations, they cannot know whether Alice was trying to spend the coin associated with c_2 or c_7. In a real ZLN spend, the number of omitted values will be orders of magnitude larger, ensuring good privacy for the light node.

So far, we have assumed that the light node is able to specify in its requests precisely which commitment values to add to the accumulator and which to omit without breaking the zero knowledge nature of the protocol. But light nodes do not have this information, and thus do not know how many zerocoin mints, if any, are contained in a given block. Therefore, we need a way to request full nodes to omit a certain percentage of commitment values from a block, while definitely omitting the commitment value associated with the zerocoin we intend to spend, all without leaking any information about which value this is.

To do so, we employ a Bloom filter, which can filter values using a customizable false-positive rate. The light node requests the selected full node running the ZLN protocol to apply this filter to a range of blocks chosen by the light node, adding most of the coins to the accumulator while leaving the rest for the light node to add locally later.

The number of false positives returned by the Bloom filter and the range of blocks to apply the filter to are both fully customizable, depending on the desired privacy level and transaction speed. To further obfuscate things, the process can be divided across several full nodes, and the light node can modify the filter with every request.

The process can also be spread over time, pre-computing partial witnesses on top of each already computed value, working on a chosen range of blocks before pausing and moving to a different peer at a later time to continue the calculation. In fact, since it is reasonable to assume that zerocoins are minted with the intention of being spent, the pre-computation can begin almost immediately after minting, and continued whenever the light node comes back online and resynchronizes with the network. This prevents the need for impractically long calculations when spending older zerocoins.

It is important that the witness calculation request should never be performed using the height of a checkpointed accumulator value, or the full node receiving the request will gain significant information about the coin the light node intends to spend. For this reason, the light node should download several commitment values and add them to the chosen checkpoint value before making the first partial witness calculation request.

Once the light node receives a response from every full node involved in the calculation, all that remains is to finish the calculation locally with the values for the chosen zerocoin and any remaining unaccumulated values. Then the light node can broadcast the completed signed transaction to any node.

Implementation

Specifically, we split the process into three phases: 1) data selection and filter setup; 2) probabilistic private outsource computation protocol; and 3) delta witness computation and transaction finalization.

Steps 2 and 3 comprise a probabilistic private multi-party computation protocol that can be pre-computed over time and distributed across different random peers, using new filters and decoys each time.

1) Data Selection and Filter Setup.

   In this first phase, the client selects the zerocoin/s that the user wants to spend and generates the request messages. These include a chosen privacy level based on the start height and a pre-loaded Bloom filter that can index one or more elements acting as decoys.

2) Probabilistically Private Outsourced Computation

The second phase involves a probabilistically private outsourced calculation of the partial accumulator witness.

Any full node on the network that implements the ZLN protocol will allow computation of a partial witness value without directly knowing which data must not be included in the exponentiation. The light node receives a reduced set of public coins that were not included in the calculation from the entire set of public zerocoins that were part of the process.

To be more precise, after receiving a chain height and loading a preset filter, the full node calculates a partial accumulator witness, without adding any members of the set that return positive on the Bloom filter's membership test. As the Bloom filter has a predefined false-positive rate, the membership test will return more than the added value/s. As a result, the node performing the calculation never knows which commitment the client intends to spend.

Because Bloom filters are probabilistic, light nodes can make a trade-off between precision and privacy by varying the false-positive rate chosen by the client. A light node with access to lots of bandwidth may choose to have a high false-positive rate, meaning the remote peer cannot accurately know which commitment values belong to the client and which do not. A light node with very little bandwidth may choose to use a very accurate filter, meaning they only get the witness without only the relevant commitment value.

Bloom filters are compact and testing membership in them is fast. This allows satisfactory performance with minimal potential for DoS attacks.

3) Delta Witness Computation and Transaction Finalization.

In this last phase, the light node takes the most recent partial witness value received from a full node and uses the set of unaccumulated commitment values to calculate the remaining delta on its own. If the calculation is still not finished, the process returns to step two.

Once the calculation process is completed, the light node finishes generating the transaction. This includes the serial signature of knowledge, using the light node's secret data and the associated private key to sign the hash of the transaction, among other needed data.

After all three steps have finished, the light node is ready to broadcast the transaction to the network.

Privacy, Accuracy, and Security

For the ZLN protocol to be useful, it needs to replicate or at least strongly approximate the process of privately calculating the public witness on a full node without compromising the essential features of Zerocoin. Specifically, we need to be confident that the zerocoin spend remains anonymous, and that outsourcing part of the spend calculation has not introduced the need for a trusted third-party.

To see why the ZLN protocol achieves this, we proceed by way of example:

Alice is a light node planning to spend a zerocoin. She outsources parts of the accumulator witness calculation to Bob, Charlie and Dave. Although these are each a third party to the calculation, Alice does not need to trust them. First, returning an inaccurate partial witness calculation result provides no meaningful advantage for Bob, Charlie or Dave – it achieves nothing beyond mischief. Second, Alice can always verify the results by checking them herself, accumulating the omitted commitments into the returned witness value and comparing the checksum against the stored checksum on each block header. She can also pass the same calculation to another full node to double-check.

Alice must accept a trade-off in privacy depending on the parameters chosen for the Bloom filter. However, this trade-off is known to Alice, and there is no way for Bob, Charlie, or Dave to increase Alice's risk here, even if they pool the information they have received.

If we posit a fourth node, Eve, with malicious intent, she can gain no useful information about Alice's intended coin spend, even if she spies on Alice's interactions with Bob, Charlie, and Dave, or indeed if she herself is one of the nodes co-opted to provide a partial witness calculation.

Timing attacks are also impractical, as this would require the malicious node to be part of every partial calculation outsourced from Carol to various different full nodes, each of which will contain decoys and false positives deliberately added to the calculation process.

Specification

New messages

We start by adding two messages to the protocol:

- `genwit`, which is a request for the partial accumulator witness generation.

- `pubcoins`, a response to `genwit` that contains the witness value and the commitment values not added into the calculation.
- `accvalue`, which requests the checkpoint accumulator value from a specific block height.

The `genwit` command is defined as follows:

Field Size	Description	Data type	Comments
?	`filter`	uint8_t[]	The filter itself is simply a bit field of arbitrary byte-aligned size. The maximum size is 36,000 bytes.
4	`nHashFuncs`	uint32_t	The number of hash functions to use in this filter. The maximum permitted value is 50.
4	`nTweak`	uint32_t	A random value to add to the seed value in the hash function used by the Bloom filter.
1	`nFlags`	uint8_t	A set of flags that control how matched items are added to the filter.
8	`startCheckpoint`	uint32_t	The checksum of the checkpoint accumulator to start accumulating the coins.
8	`requestNum`	uint32_t	The request number.
32	`witValue`	CBignum	Partial witness accumulator value to continue accumulating commitments on top of it.

The `pubcoins` command is defined as follows:

Field Size	Description	Data type	Comments
8	`requestNum`	uint32_t	`genwit` request number.
8	`AccValue`	uint32_t	The accumulator value.
8	`AccWitnessValue`	uint32_t	The partial accumulator witness value.
8	`nSize`	uint32_t	Number of commitment values not added.

?	`values`	list<uint 32_t>	The commitment values that were not added.

Conclusion And Future Work

The proposed ZLN protocol is a good solution to the current lack of a Zerocoin protocol for light nodes. It addresses all currently open concerns about privacy, storage, network usage and performance when spending zerocoins with limited hardware.

The ZLN protocol has been integrated into the PIVX blockchain as a public implementation of the entire protocol.

Several open questions remain, particularly with regards to scalability, as the current Zerocoin implementation was not conceived with light nodes in mind. As more light nodes join the network and the number of zerocoins increases, the burden on full nodes computing partial witness calculations will increase. We will continue research in this area in an attempt to find a solution, which will likely require new base cryptography for the Zerocoin protocol.

The ZLN protocol also makes no specific provision for incentives, which will be important as it requires remote peers to shoulder a large portion of the spend calculation for coins which they do not control. As such, they have no reason to agree to perform these calculations, beyond the general well-being of the network. However, the best way to address this will issue will depend on the implementation specifics of the network where the ZLN protocol is deployed, which falls beyond the scope of this work.

References

[1] Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin: *Zerocoin: Anonymous Distributed E-Cash from Bitcoin*. http://zerocoin.org/media/pdf/ZerocoinOakland.pdf

[2] Burton H. Bloom: *Space/time trade-offs in hash coding with allowable errors*. Communications of the ACM, 1970.

[3] *Zcoin*. https://zcoin.io

[4] *PIVX*. https://pivx.org

[5] Tim Ruffing, Sri Aravinda Thyagarajan, Viktoria Ronge, Dominique Schröder: *Burning Zerocoins for Fun and for Profit. A Cryptographic Denial-of-Spending Attack on the Zerocoin Protocol*. https://www.chaac.tf.fau.de/files/2018/04/attack-cryptocur.pdf

[6] Bitcoin Improvement Proposal 37 (BIP37) https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki