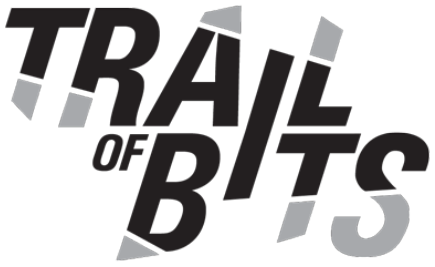

Zcash

Security Assessment

12/4/2019

Prepared For:
Benjamin Winston | *Electric Coin Co.*
bambam@electriccoin.co

Prepared By:
Benjamin Perez | *Trail of Bits*
benjamin.perez@trailofbits.com

Will Song | *Trail of Bits*
will.song@trailofbits.com

James Miller | *Trail of Bits*
james.miller@trailofbits.com

Paul Kehrer | *Trail of Bits*
paul.kehrer@trailofbits.com

Executive Summary

From November 4th to November 27th, the Electric Coin Co. engaged Trail of Bits to review two Zcash Improvement Protocols (ZIPs). Specifically, the assessment specified a review of ZIP 213: Shielded Coinbase (commit hash `90ad18ff`) and ZIP 221: FlyClient Zcash SPV (commit hash `b6be3770`). The first two calendar weeks of the assessment were dedicated to studying and increasing understanding of the Zcash Sapling specification as required to assess the two ZIPs and how they interact with the system. The remainder of the assessment was dedicated to analyzing the two ZIPs themselves.

To understand the Zcash Sapling protocol, Trail of Bits read through the specification, relevant blog posts, and additional resources such as conference presentations. We also engaged in multiple discussions with members of the Zcash team. After studying the Sapling protocol, Trail of Bits began analyzing ZIP 213 and ZIP 221. This analysis took the form of manual review of the protocols, reviewing related reading materials, and engaging in discussions internally and with Zcash team members.

Trail of Bits does not report any significant security findings in the Sapling specification or the two ZIPs. We found that the ZIPs do have some impact on privacy in the Sapling protocol, and we have documented our findings on those matters in this report. In addition, the authors of the ZIPs mention components of their protocols that they're either concerned about or would like more feedback on; Trail of Bits has provided feedback on some of these as well.

While reviewing the Sapling protocol, we found that the only relevant materials were the specification itself (a very dense, low-level document), and related blog posts, which are mostly high-level overviews. Therefore, Trail of Bits concluded that it would be valuable to produce a white paper that contains more detail than most blog posts but is less dense than the specification itself. Members of the Electric Coin Co. agreed with this idea, so Trail of Bits will deliver this white paper at the conclusion of this assessment.

Engagement Goals and Coverage

The engagement was scoped to provide a security assessment of ZIP 213: Shielded Coinbase and ZIP 221: FlyClient Zcash SPV. In Zcash Sapling, all coinbase transactions must go into the transparent pool. From there, these transactions are put into the shielded pool; however, this is typically done in bulk, is easy to detect, and leaks privacy. ZIP 213 attempts to address this by shielding coinbase transactions immediately. This does not completely solve the problem, as many miners of Zcash use exchanges that only employ transparent addresses, so the transactions are eventually unshielded and also leak privacy. However, immediate shielding is a step towards solving these privacy issues.

ZIP 221 works towards integrating the FlyClient protocol into Zcash Sapling. The FlyClient protocol allows computationally limited devices to verify the blockchain (among other things) without the burden of storing the entire blockchain locally. This ZIP proposes altering the Zcash Sapling specification by requiring miners to store the blockchain in a Merkle Mountain Range (MMR) data structure and adding some additional information to the block headers.

Both ZIPs modify the current behavior of Zcash Sapling. Therefore, the goal for this assessment was to analyze how these ZIPs fit in with the Sapling specification. Specifically, we analyzed whether either ZIP presented new vulnerabilities or security concerns; however, neither ZIP significantly affected security. On the other hand, both of these ZIPs did affect privacy in noticeable ways, so this was one of our larger analysis goals. Specifically, we sought to answer the following questions:

- Do either of the ZIPs have any security vulnerabilities?
- How do each of the ZIPs affect privacy of users and nodes (and FlyClients)?
- Are any of the concerns raised by the authors of ZIP 221 cause for adjusting the protocol?

We spent the first two calendar weeks of the assessment learning the protocol by reading the Zcash Sapling specification and relevant blogs. Once we had a strong understanding of the Sapling protocol, we were able to analyze both ZIP 213 and ZIP 221, and determine how they impact Sapling. Our analysis of both ZIPs took the form of manual review of the ZIPs themselves and relevant material, such as the FlyClient paper. In addition, we engaged in various discussions internally and with members of the Zcash team. Lastly, we produced a white paper concisely detailing the Zcash Sapling protocol and the two ZIPs, which we will deliver along with this document.

Zcash Sapling

In addition to other cryptographic primitives, Zcash uses encryption schemes and pseudorandom functions (PRFs). In the Sapling specification, Zcash introduces the idea of a "key privacy" encryption scheme. These encryption schemes have the same security properties as symmetric encryption schemes, with the additional property of not revealing the secret encryption key. PRFs are used throughout Sapling for generating keys from the secret spending key and generating nullifiers. The security of the protocol relies on a particular security property of these PRFs: namely that they should be able to generate other keys without revealing the secret spending key. In other words, the security of the protocol relies on these PRFs having an analogous "key privacy" property. The Zcash Sapling specification currently does not explicitly state this property as a requirement. Trail of Bits discussed this with members of the Zcash team and they pointed out that, unlike encryption schemes, a secure PRF already includes this "key privacy" property. Therefore, it is not necessary to explicitly state this property as a requirement for PRFs in the Sapling specification.

ZIP 213: Shielded Coinbase

ZIP 213 modifies the consensus rules of Zcash to allow coinbase transactions to contain shielded outputs. Previously, all coinbase transactions were required to have only transparent outputs and then subsequently enter the shielded pool. In particular, the following changes are made to the Sapling consensus rules:

1. The consensus rule preventing coinbase transactions from containing shielded outputs is no longer active, and coinbase transactions MAY contain Sapling outputs.
2. The consensus rules applied to `valueBalance`, `vShieldedOutput`, and `bindingSig` in non-coinbase transactions MUST also be applied to coinbase transactions.
3. The existing consensus rule requiring transactions that spend coinbase outputs to have an empty `vout` is amended to only apply to transactions that spend transparent coinbase outputs.
4. The existing consensus rule requiring coinbase outputs to have 100 confirmations before they may be spent (coinbase maturity) is amended to apply only to transparent coinbase outputs.
5. All Sapling outputs in coinbase transactions MUST have valid note commitments when recovered using a 32-byte array of zeros as the outgoing viewing key.

Trail of Bits did not find any security or privacy concerns in these modifications to the Sapling specification. However, we believe that users may misinterpret this ZIP to mean that coinbase transactions can now be made untraceable, or that it is now more difficult to analyze money flowing through the shielded pool. Before explaining why this is not the case, we first look at prior work on de-anonymizing shielded transactions.

In *An Empirical Analysis of Anonymity in Zcash*, Kappos et al. shrink the anonymity set of Zcash substantially by identifying shielding and deshielding transactions that contain miner and founder rewards. In the case of `t-z` transactions, this task is fairly straightforward since ZEC from a coinbase transaction must go into the shielded pool. For `z-t` transactions, several heuristics can be used to determine if they are for miner rewards, such as timing information between shielding and deshielding transactions. Mining pools can be easily detected since they often result in `z-t` transactions with many (> 100) outputs. Miner and founder rewards account for an overwhelming majority of `t-z` and `z-t` transactions—nearly 70%. Once these transactions have been identified, the remaining 30% are easier to analyze, especially considering that in total only around 10% of Zcash transactions are shielded.

Ideally, allowing coinbase transactions to have shielded outputs would prevent this type of analysis, or at least reduce its efficacy. However, ZIP 213 mandates that all shielded coinbase transactions have an outgoing viewing key of 0, so any blockchain observer can identify these commitments and their recipient (though the recipient will be a diversified

address). This is unfortunately necessary, since otherwise nodes would be unable to determine the validity of coinbase transactions. Since anyone can identify shielded coinbase transactions, the techniques discussed above are still feasible. While it may no longer be possible to directly correlate transparent addresses entering miner rewards into the shielded pool with those receiving them through a deshielding transaction, it remains easy to detect which deshielding operations stem from coinbase transactions, especially in the case of mining pools where prior techniques are still applicable. Therefore, it will still be possible to reduce the anonymity set by an amount similar to the one in the Kappos paper.

The above discussion shouldn't be misconstrued as a critique of ZIP 213. Rather, we're highlighting the security properties that users may mistakenly believe ZIP 213 possesses. In general, discouraging the use of transparent addresses in the Zcash ecosystem is a boon to everyone's privacy, and allowing shielded coinbase transactions is a major step in that direction.

ZIP 221: FlyClient Zcash SPV

ZIP 221 proposes to add a new data structure—a Merkle Mountain Range (MMR)—into the Zcash Sapling protocol. To accomplish this, each block in the blockchain is stored in this MMR, and the root of the resulting tree is added into the block headers. Inclusion of the MMR allows the FlyClient protocol to be run on Zcash Sapling. This protocol allows for computationally limited devices to verify:

- The validity of a blockchain received from a node (without downloading the entire chain or even every block header)
- The inclusion of a block in the blockchain as well as certain block metadata

The authors of the ZIP mention that the addition of the MMR will result in an increased validation cost, as validators now have to maintain the MMR for every block added, and are concerned this could worsen existing denial-of-service (DoS) attacks. Specifically, they mention an attack scenario in which an adversary maintains two different blockchains of approximately equal length. The adversary can alternately send these chains to a node, causing them to repeatedly reorganize their MMR. However, as the authors mention, the cost to add a block to the MMR will be at worst `O(log(n))` for `append` and `delete` operations (where `n` is the number of blocks). Both of these operations use `Blake2b`, which is very fast. Also, unless this adversary has significantly better mining performance than the rest of the nodes, the probability of successfully maintaining a separate fork of length `k` is exponentially small in terms of the size of `k`. A successful attack would require on the order of `O(k*log(n))` operations for the validator (at most `O(log(n))` for each block).

In order for this attack to be effective, an attacker needs a large enough `k` to cause a DoS on the client, but a `k` value of that size makes the probability of success negligible. Therefore, this should only be a concern if this adversary has significantly better mining performance, which would break one of the fundamental security assumptions of the system.

The authors of the ZIP also expressed an interest in including commitments to nullifier vectors in the node metadata; they decided against it, but have also asked for comments on the idea. As the FlyClients are limited in resources, we assume they will not be storing the entire nullifier set locally. Presumably, the idea would be for the nodes to maintain a nullifier MMR (or similar data type), so that FlyClients can verify the nullifier set or verify that a particular nullifier is in the set, all without having to store the entire set themselves. However, in order for a FlyClient to verify whether a nullifier is in the set, they would have to query the FlyClient server for a proof for that particular nullifier. This would be a privacy concern because the server can clearly identify particular commitments that each FlyClient

is interested in (and presumably involved with). Therefore, we support the decision not to include this.

While this ZIP only covers the addition of MMR commitments in the block header, we believe it is worth noting the privacy risks involved with using FlyClient within the Zcash ecosystem. For transparent addresses, FlyClient poses no risk since these are simply Bitcoin addresses, and transactions between them are public. However, using FlyClients in a privacy-preserving manner when sending or receiving funds from shielded addresses becomes difficult. For instance, if a user sends someone money from a shielded address and wants to verify the transaction has actually been committed to the blockchain, they will need to query a full node to get the MMR proof that the transaction was included in a block. Performing this query would signal to the proving node that the client is interested in the transaction in question. Therefore, shielded addresses running FlyClient cannot make queries about specific transactions without losing privacy.

Currently, the only solution for this issue is having FlyClients affiliated with shielded addresses scan every transaction committed to the chain. While this imposes a somewhat substantial computational burden on the client, it doesn't require any information storage. The Zcash team is currently discussing ways to make this scanning process more efficient. Despite the fact that shielded addresses need to perform this scanning operation, the FlyClient protocol is useful in reducing storage requirements for users.

Finally, there are a few small errors in the ZIP we wanted to point out. The top portion of Figure 2 seems to have swapped the position of `left_child.nEarliestHeight` and `right_child.nLatestHeight`. Also, in the pseudocode function `get_peaks`, it is possible for the input node to be a leaf node (or the only node in a one node MMR). This would result in a node with a `null` `left_child` and `right_child`. If this is the input, the current code would result in an error. A quick fix would be to verify the children are not `null`.