Aaron Smith

Dr. Austin

CS 168

29 October 2020

Implementing zk-SNARKs in Spartan-Gold

A major problem with bitcoin is the lack of anonymity. Even if a wallet address is never used in more than one transaction, there are still ways to trace bitcoin back to individual users. Various solutions have been proposed to fix this, but the one that my project will focus on is the use of zk-SNARKs. A zk-SNARK is a Zero-Knowledge Succinct Non-Interactive Argument of Knowledge. This is a type of zero-knowledge proof that can be used to basically encrypt transactions on the blockchain while still providing a way to validate those transactions. The bitcoin fork Zcash [1], released in 2016 and based off of Zerocash [2], is the most prominent usage example of zk-SNARKs. In Zcash, there are two types of addresses: private (z-addresses) and transparent (t-addresses). Transactions between t-addresses work like they do in bitcoin, while transactions between z-addresses are encrypted in such a way that the addresses and transaction amount are hidden. Two more transaction types exist: from z-addresses to t-addresses and vice versa. In these transactions, only one of the addresses are encrypted. Zcash and its use of zk-SNARKs is the primary inspiration for my project.

The way zk-SNARKs work is actually quite complicated and difficult to understand [3], but I will try to explain in brief. The first step is to encode a transaction validity function into a mathematical representation (in the form of polynomials). In order to prove that a transaction is valid, a prover must prove that they know such a polynomial without revealing the polynomial itself (hence zero-knowledge). Homomorphic encryption and elliptic curve pairings are used to

evaluate a polynomial “blindly”, i.e. neither the prover nor the verifier know which point on the polynomial is being evaluated (which prevents cheating). This process is a one time job (non-interactive) so everyone on the network can easily verify (succinct) without extra interaction with the prover. In order to be non-interactive, encrypted public parameters (a common reference string that includes a prover and a verifier key) are created and used by everyone in the proving/verification process. The original parameters (sort of like a seed) used to create the reference string must be destroyed or else someone can use them to cheat the system (the creation of the reference string is in itself a complicated process). The final result is that a prover can prove a transaction is valid with extremely high probability while not revealing the contents of the transaction.

For my project, I will not attempt to create a zk-SNARK library myself (lest I fail because of how complicated the algorithm is). Instead, I will use an existing zk-SNARK JavaScript implementation (snarkjs) [4] that can be installed through npm. I will extend Spartan-Gold to use zk-SNARKs to hide transaction details. If that is not already too much work, then I will also attempt to implement Zcash’s two address types (z-addresses and t-addresses) and the four possible combinations of transactions between them. I have broken down the project into these steps:

- Create Spartan-Gold subclasses that override the default transaction behavior. Transaction details will have to be hidden and must be verified according to a proof generated by snarkjs.
    - At least transaction.js, block.js, client.js, and miner.js will have to be overridden, possibly others.

    - (Optional) Further extend this system so transactions work between z-addresses and t-addresses.
- Obtain/generate a reference string (snarkjs provides a method that uses the "powers of tau ceremony", from which the prover/verifier keys can be created).
- Create a circuit to verify a transaction (a "circuit" is a representation of polynomials from the discussion above, snarkjs uses a template to define this).
- Create a driver.js that will test functionality and make sure my implementation works correctly.

References

[1] Overview of Zcash. https://z.cash/technology/

[2] The original Zerocash paper that Zcash is based on.

http://zerocash-project.org/media/pdf/zerocash-extended-20140518.pdf

[3] Explanation of how zk-SNARKs work.

http://petkus.info/papers/WhyAndHowZkSnarkWorks.pdf

[4] Javascript zk-SNARK implementation repository. https://github.com/iden3/snarkjs