Privacy when Everyone is Watching

An SOK on Anonymity on the Blockchain

Roy Rinberg †
Department of Computer Science
Columbia University
New York, USA
rr3426@columbia.edu

Nilaksh Agarwal †
Department of Computer Science
Columbia University
New York, USA
na2886@columbia.edu

Abstract

Blockchain technologies rely on a public ledger, where typically all transactions are pseudoanonymous and fully traceable. This poses a major flaw in its large scale adoption of cryptocurrencies, the primary application of blockchain technologies, as most individuals do not want to disclose their finances to the public. Motivated by the explosive growth in private-Blockchain research, this Statement-of-Knowledge (SOK) explores the ways to obtain privacy in this public ledger ecosystem. The authors first look at the underlying technology underling all zero-knowledge applications on the blockchain: zk-SNARKs (zero-knowledge Succinct Non-interactive ARguments of Knowledge). We then explore the two largest privacy coins as of today, ZCash and Monero, as well as TornadoCash, a popular Ethereum Tumbler solution. Finally, we look at the opposing incentives behind privacy solutions and de-anonymization techniques, and the future of privacy on the blockchain.

†These authors contributed equally to this work

We wish to thank Professor Tim Roughgarden for providing us with helpful comments and reviews on this work.

Contents

Introduction

The Blockchain is Intrinsically Not-Private

At its core, blockchain technologies remove the need for centralization during a transaction by broadcasting each transaction to all nodes on the blockchain. Cryptocurrencies like Bitcoin are often naively believed to be anonymous because one does not need to use their real name to make transactions. However, it is also not difficult to establish a link between a Bitcoin address and a real-world identity. As a toy example, if you buy a cookie from a cafe and pay with Bitcoin, while you would not need to reveal your real name, your physical identity gets tied to the Bitcoin transaction.

Deanonymization Techniques

One powerful and common privacy attack goal is to link UTXO-wallet addresses, by tracking how money is spent in shared ways. For instance, if two Bitcoin payments (inputs) are used to purchase a book (output), one can infer that the transactions are likely controlled by the same user. [Narayanan et al., 2016]

In "A Fistful of Bitcoins: Characterizing Payments Among Men with No Names" by Meiklejohn et al. (2013) [Meiklejohn et al., 2013], researchers traced a transaction graph analysis to de-anonymize Bitcoin transactions. They generated a graph of blockchain transactions, and then to reliably infer addresses as owned by particular identities, the researchers transacted with various service providers through purchasing items with BTC. By interacting with exchanges, gambling sites, and wallet services, the researchers are able to identify Bitcoin addresses that that merchants own. This in turn allows researchers to observe transactions with these merchants in real-time.

Deanonymization in itself could be a paper, and lots of work has been done on this (and continues to be); however, as our focus centers on privacy, we will only offer one more major form of deanonymization. Cryptocurrency transactions can be deanonymized through the network-layer. When a transaction is initiated, it connects to many nodes at once and broadcasts the transaction to peers in the network. Unless proper VPNs or Onion-Routing protocols are set up, the first agent (a light-client or full-node) to broadcast a transaction is the source of the transaction. The transaction could then be linked to the node's IP address, in which there exists numerous ways to unmask a person's identity behind an IP address, posing another threat to Bitcoin's privacy [Narayanan et al., 2016].

Deanonymization is becoming increasingly researched as it's dual, the anonymization techniques develop. For example, In September 2020, the United States Internal Revenue Service's criminal investigation division (IRS-CI), posted a $625,000 bounty for contractors who could develop tools to help trace Monero, other privacy-enhanced cryptocurrencies. With such high emphasis being given to deanonymization, we believe that there is a "forever war" between these two groups to shape the blockchain landscape.

Case Study: Colonial Pipeline

Deanonymization is not only a theoretical concept, it's been seen all over. One of the most public deanonymizations was the FBI identifying ransomware hackers who were paid in Bitcoin in the spring of 2021.

Colonial Pipeline is the largest fuel pipeline in the US, supplying approximately 45% of the East Coast's fuel, including gasoline, diesel, home heating oil, jet fuel, and military supplies. On May 7, 2021, Colonial Pipeline's billing system suffered a ransomware cyberattack. They paid 75 bitcoin in ransom (which at the time was 4.2 million USD).

Tuan Phan, founder of Zero Friction LLC publicly reconstructed the FBI's recovery process using only publicly available information and tools. Tuan started with a query to search the Bitcoin network for all addresses that partially match the partial address provided by the FBI. There was only one partial match; then, using a Bitcoin explorer, the author found that three transactions belonged to this address, with two being sent and one, the earliest, received. Following the transactions, the author was able to produce the transaction hash list. In fact, we tried our hand at following the money flow and finding the final retrieval of 63.7 BTC of the ransom payment and were successful. Specifically, the transaction hash:

0677781a5079eae8e5cbd5e6d9dcc5c02da45351a3638b85c88e5e3ecdc105a7

is the key, as of the 75 BTC sent as a ransom payment, the receiving address,

bc1qxu83k5qkj8kcqdqqenwzn7khcw4llfykeqwg45,

received only 63.7 BTC. Since DarkSide is a ransomware as a service organization, affiliates pay the service for the use of the ransom tools. Therefore the payment of 63.7 BTC is likely the fees to the affiliate, and the remaining balance is likely the share for the DarkSide developer. [Tuan Phan, 2021]

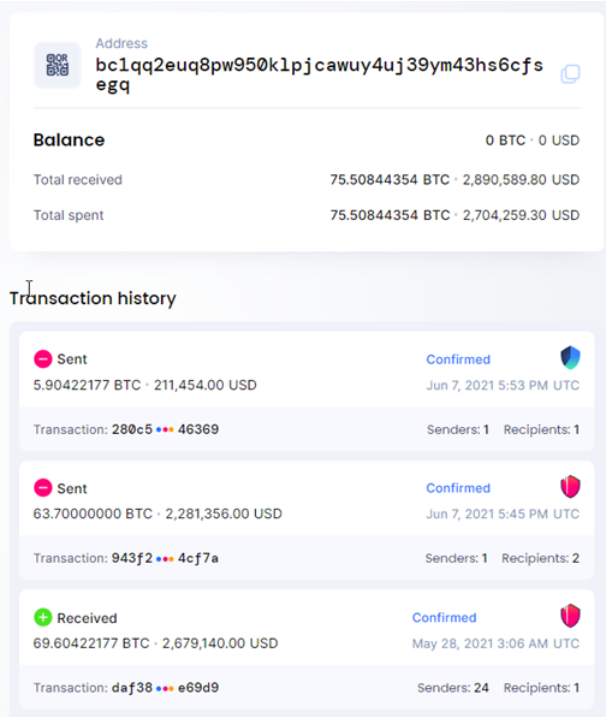


Figure 1: The original payment from Colonial Pipeline to DarkSide [Tuan Phan, 2021].

What is a Zero-Knowledge Proof?

As an introduction to a solution to some of the qualms of a public blockchain, we must introduce one of the great cryptographic wonders of the world: zero-knowledge proofs (ZKP). ZKP are proofs that an agent knows some particular piece of information, without revealing what you know. Imagine you are playing "Where's Waldo" with a friend and they claim that they have found Waldo. The simplest way to prove to you that they found him is to point him out directly. However, there are many reasons why they wouldn't want to do this; notably, this ruins the game for you. In order to really race, without ruining the game, you would need a trusted 3rd party, to whom both of you could point out where you believe Waldo is.

Why is this useful?

Finding Waldo is nice, but the application of ZKP extends to anything that involves asymmetric information. You imagine something as extreme as proving to a rival nation that you do indeed have a nuclear bomb (without having to set one off). Alternatively, on the blockchain, ZKP could be used to prove that someone does in fact have enough money to pay a transaction, without revealing the transaction. However, it's worth observing that the ability to prove something is true without revealing information about the nature of the truth is ultimately

only useful with regards to privacy and asymmetric information. If all parties are trusted and their values are aligned, there is no need for anyone to create a zero-knowledge proof, when they could simply create a proof.

Ali Baba's Cave

One of the most canonical examples of a ZKP is Ali Baba's Cave.

Imagine a cave with two entrances; at the back of the cave there is a wall that can only be opened with a special password. My goal is to convince you that I know the password. One way to convince a verifier that I know the password without sharing the password itself is as follows:

1. Prover enters the cave along one of the two tunnels (the verifier cannot see them enter)
2. Once the prover has entered, the verifier shouts to them which side (left or right) they wish the prover to exit from.
3. If the prover entered on that side, the prover simply walks out the way they came in. If the prover entered on the other side, they must use the passcode to exit on the verifier's stated side.
4. After one round, the verifier believes there is a 50% chance the prover knows the code; as either the prover knew the code, or they happened to be on the right side. The verifier and prover can repeat the previous steps as many times, each time halving the possibility that the prover doesn't know the code, and was just getting lucky.

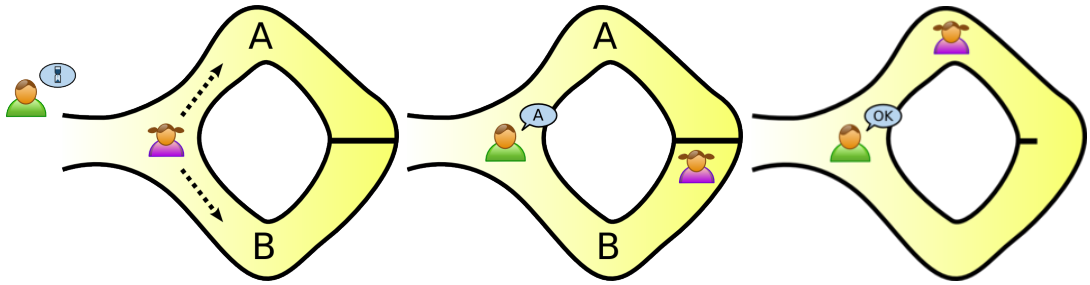


Figure 2: Ali Baba's Cave : Zero Knowledge Proof [Wikipedia, 2021b]

It should be noted that this is an interactive proof, and in Ali Baba's Cave you only prove to a single verifier that you in fact know the passcode, as to any onlooker, it is possible the verifier and prover colluded and agreed the commands ahead of time. Later in the paper we will discuss zk-SNARKs which are non-interactive zero-knowledge proofs, which eliminate this trust assumption.

So... Where is Waldo?

Returning to the game of "Where's Waldo", we present a simple algorithm for proving that you know where Waldo is.

1. Take a large piece of construction paper (at least 3 times the height and width of the page with Waldo in it), and cut a hole the exact shape and size of Waldo out.
2. Move the book behind the construction paper, so that your partner cannot see the book, and orient the book so that waldo directly falls behind the hole in the page.

At the end of this, your partner should be able to see Waldo, but not see the book or any of Waldo's surroundings. You have just proven that you know where Waldo is, without revealing information about where he is.

It should be noted that it is impossible (as far as we can tell) to create a zero-knowledge proof that is not probabilistic. The proof of knowing Waldo's location might seem like a *deterministic* proof, but it is actually probabilistic, as the prover could randomly orient the book behind the construction paper and happen to land on Waldo. Zero-knowledge proofs may be probabilistic, but they are typically structured such that it is incredibly improbable.

Privacy Coins

The rest of this report will be on tools that enable people to use the blockchain without sharing information with others about what they are doing. **Privacy coin** is a colloquial term for a cryptocurrencies that do not directly broadcast transactions, but rather cryptographic proofs to ensure that they have the capacity to pay for the transaction they wish. As an example, a regular transaction would look like this:

- Buyer sees: Price of a coffee
- Seller sees: Buyer's wallet/UTXO address (and from there, any linkable UTXO addresses).
- Everyone else sees: Buyer's wallet, or any linkable UTXO addresses.

A transaction over a privacy-coin blockchain looks like this :

- Buyer sees: Price of a coffee.
- Seller sees: Buyer has at least the price of the coffee in their account, but how much is unknown.
- Everyone else sees: A transaction occurs between two unknown parties, for an unknown sum of money.

Why Tumblers

Rather than using a completely off chain solution for privacy, sometimes users need to remain anonymous on non-private blockchains (for example, Bitcoin/Ethereum might be the only currencies accepted for a certain purchase, would still like anonymity).

For these solutions, Tumblers come into the picture, by pooling together transactions from various sources, and "mixing" them together, to hide their source and destination linkages. This still reveals transaction amounts, but no longer broadcasts the sender-receiver linkage.

zk-SNARKs: Zero-Knowledge to Hero-Knowledge

"A zk-SNARK is a collection of words that computer scientists arranged in order to spell 'zk-SNARK'" - Anonymous [Gabizon, 2018]

What is a zk-SNARK

ZK	**Z**ero-**K**nowledge
S	**S**uccinct
N	**N**on-interactive
AR	**AR**gument
K	of **K**nowledge

A zk-SNARK is one the basic building blocks of privacy on the blockchain. The acronym is a *Zero-Knowledge Succinct Non-interactive ARgument of Knowledge*. Rearranging the words, a zk-SNARK is an short proof that you know something that reveals no information about what you know, and does not require interactivity from a verifier. In this chapter we will dig deeply into the specifics of a zk-SNARK; and while ZK-proofs are very fashionable, we hope to impart that the truly impressive part of a zk-SNARK is succinct, and non-interactive.

Note: much of this chapter is drawn from "Why and How zk-SNARK Works: Definitive Explanation" [Petkus, 2019]; though the authors have shortened sections for brevity and clarity. The goal of this chapter is not to enable the reader to create a zk-SNARK from scratch, but for the reader to understand each part of a zk-SNARK construction well enough that with only an hour or two per section, the reader could. Much of the theoretical work for zk-SNARKs was introduced in "Short Pairing-based Non-interactive Zero-Knowledge Arguments" by Jens Groth in 2010 [Groth, 2010]. Though a lot of important improvements were added shortly after, in 2013, in both "Quadratic Span Programs and Succinct NIZKs without PCPs" [Gennaro et al., 2013] and in the first big zk-SNARK paper proposing a real implementation "Pinocchio: Nearly Practical Verifiable Computation" [Parno et al., 2013].

Why this particular acronym?

Acronym	What is stands for	What it is/Why it's useful
ARK	**AR**gument of **K**nowledge	A proof of something being true
S	**S**uccinct	So it can fit on the blockchain
N	**N**on-interactive	To make proofs that are verifiable forever, and avoid worries of collusion
ZK	**Z**ero-**K**nowledge	In order to prevent verifiers from learning what the prover knows

To define a zk-SNARK, you need to specify a function f, which will take some kind of (typically on-chain) data u as an argument. Let the output of f on u is $f(u) = c$. A zk-SNARK will be constructed from intermediate values of the compute f, and will reveal no information about the prover's (nor the verifier's) private information.

The goal of this chapter is to describe a zk-SNARK algorithm, which will take as its arguments f and input u, and return a proof z, such that the verifier will evaluate the proof z as valid IFF $f(u) = x$. With the one caveat being that : no verifier should learn any information about what the prover knows, except for the concrete proof itself. E.g. if the zk-SNARK seeks to prove that P has at least 100 ETH, it should be impossible to tell if P has 100.1 or 1 billion ETH.

What can a zk-SNARK be used for?

ZK-SNARKs are general purpose, so the specific problem doesn't matter. However, typically they are used for one of two things: 1. to prove that you have done an expensive computation, or 2. you wish to prove something about an encrypted value that is on-chain, which you have the secret key for.

Some examples of what you may want to prove are :

- Anonymous authorization:
  - Proving a user is on a whitelist (or blacklist), without revealing personally identifiable information.
  - Proving a user pays for a service without revealing their identity.
- Outsourcing computation:
  - Outsourcing an expensive computation and validating that the result is correct without redoing the execution.
- Proving statement on private data:
  - A user has more than X in their wallet
  - A bank did not transact with a particular user, in the last X amount of time
  - Accurately paying taxes without revealing one's earnings

[Petkus, 2019]

Outline of zk-SNARK structure

The high-level outline we have is:

- First we map the function f to a polynomial problem
  1. We first do this by breaking apart a function into a series of computations of the form $l(x) \times r(x) = o(x)$.
  2. Then, we observe that the roots of $t(x) = (l(x) \times r(x)) - o(x)$ will also be the solutions that satisfy the computation.
- Publicly share the "target polynomial" $t(x)$.
- As the prover does the computation, they map the steps of the function-computation to scalar-values. Then the prover will interpolate the scalars (with the x-value loosely as a function of time), to produce a polynomial $p(x)$.
- By construction, if the prover really did the computation, then their polynomial $p(x)$, should be able to compute a $h(x)$ such that $p(x) = t(x) \cdot h(x)$.
- The verifier sends the prover some encrypted points s, and the the prover evaluates $p(x)$ and $h(x)$ on s, and sends $p(s)$ and $h(s)$ back.
- The verifier(s) confirm that $p(s) = h(s) \cdot t(s)$ (in encrypted space). Noting that if the prover did not know $p(x) = h(x) \cdot t(x)$ the points $p(s)$ and $h(s) \cdot t(s)$ would almost certainly not be equal, because 2 polynomials of degree d will only intersect at most d points. If s is sampled over the reals, the probability of accidentally sampling one of the d points goes to 0 as d increases. † †

† We note that sharing the prover's polynomial directly is bad for 2 reasons 1. it reveals information about the prover's computation to everyone, and 2. it is not succinct (in theory, the degree d of the polynomial is not bounded. So instead.

† We also observe that the prover could simply fit the polynomial to the points given, if they are explicitly given; so we prevent the prover from being able to cheat, by making them must evaluate the polynomial over homomorphically encrypted points.

- We add ZK to the SNARK we evaluating the polynomial at random shifts from the homomorphically hidden points, which are not known to the verifier, but preserve the validity of the polynomial equation.

For most of this we will assume that we have access to random values, but, later we will describe where these come from in the section called Trusting-One-of-Many.

We are going to go a bit backwards, first we will show how to prove-knowledge-of-a-polynomial, succinctly (we call this SNARKOP) in a trusted setup. Then we will relax the trust requirements from "trusted third-party", to trusting one-of-many (a goal commonly used in cryptographic protocols, including TOR). Then add ZK, to create a zk-SNARKOP. Then we will show how to map solving a function to the derivation of a polynomial, thus by showing that if we know the polynomial, we must have done the computation. [Petkus, 2019][Groth, 2010]

SNARKOP

Our first (contrived) problem is for a a prover P to prove to a verifier V that they know a polynomial $p(x)$, has some set of roots $\{x_i\}$. We observe that having $\{x_i\}$ as roots is equivalent to there existing a polynomial $h(x)$ such that $p(x) = h(x) \cdot t(x)$, where $t(x) = \prod_{i=0}(x - x_i)$

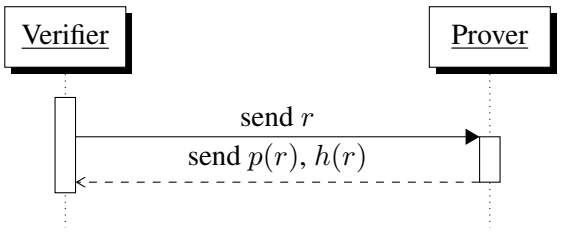
ARK-OP: ARgument of Knowledge Of a Polynomial

If $p(x) = h(x) \cdot t(x)$ then Prover P can derive $h(x) = \frac{p(x)}{t(x)}$. To verify that p(x) has the solutions to t(x), P can send V $h(x)$ and $p(x)$, and V can easily verify that $p(x) = t(x) \cdot h(x)$.

S_ARK-OP: Succinct ARgument of Knowledge Of a Polynomial.

Evaluation at a point rather than the entire polynomial.
One major problem is that the polynomials in question are typically very long, and sharing an entire polynomial would not be succinct. Additionally, jumping ahead, sharing the entire polynomial would prevent us from being able to apply zero-Knowledge, later. However, we make the important observation that if $a(x) = b(x) \cdot c(x)$, then we can evaluate $a(x)$ by evaluating its factors $b(x)$ and $c(x)$ then multiplying.

So, the first version of our algorithm will look like this:
Before the interaction: V selects a random r, computes $t(r)$. P computes $h(x) = \frac{p(x)}{t(x)}$



Verifier confirms $h(r) \cdot t(r) = p(r)$

To put this into words, and provide a concrete example: let's take $t(x) = (x-10)\cdot(x-11) = x^2-21x+110$, and $p(x) = ((x - 10) \cdot (x - 11)) \cdot ((x - 1) \cdot (x - 0)) = t(x) \cdot (x^2 - x)$

- Verifier V selects a random value r and sends to P. V also computes $t(r)$. As an example, let $r = 3$, so $t(r) = 7 \cdot 8 = 56$
- P has already computed $p(x)$ (this is what they are trying to prove they know). And now computes $h(x) = \frac{p(x)}{t(x)}$. Here, $h(x) = x^2 - x$
- P computes $h(r)$ and $p(r)$ and sends them back to V. Here, $h(r) = h(r) = 9-3 = 6$; $p(r) = 7\cdot8\cdot-2\cdot-3 = 336$
- V confirms that $h(r) \cdot t(r) = p(r)$. Here $56 \cdot 6 = 336$. (Which is true!).

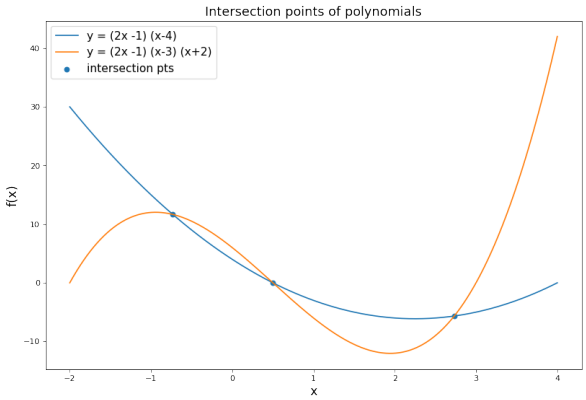
Soundness: It should be clear by construction, that the prover P would convinces verifier V that P knows $p(x)$ with shared roots of $t(x)$, specifcally in the case $p(x) = t(x) \cdot h(x)$

Completeness: (Prover cannot convince Verifier that it knows P when it does not).
In the case that prover P did NOT know a polynomial $p(x)$ with the roots x_0 and x_1, but instead knew $p'(x)$, then $h'(x) = \frac{p'(x)}{t(x)}$, which will produce a divisor and a remainder (e.g. of the form $h(x) = x + 7 + \frac{3x+1}{t(x)}$).

One way to exclude solutions that have a remainder is for Verifier V to require that $p(r)$ and $h(r)$ are integer values for all r, which eliminates solutions with remainders, for almost all values r. However, we note that this kind of constraint requires that $p(x)$ has only integer coefficients; so later, we will seamlessly replace this constraint, and instead, we will require the evaluation of $p(x)$ and $h(x)$ in an encrypted format, whose cryptographic primitives do not allow division.

There are 2 main things to observe with the above:

- $p(x)$ may be a d-dimensional vector, and so, very long. Instead $h(r)$ and $p(r)$ are just 2 scalars, and so very short. We have achieved **S**uccinctness!
- It is in fact possible that the polynomial $h(x) \cdot t(x)$ and $p(x)$, as can be seen by this very simple diagram :



However! for polynomials of degree d (or less), two polynomials intersect at most in d locations. Since points are sampled over the reals, which are infinite, the probability of randomly selecting one such point is $P = 0$.

Fixing SARK-OP

We observe 3 issues with the above SARK-OP algorithm. We overlooked (some of) them, by requiring that the prover be honest, but we wish to remove that now. *Note*: While this is its own section, we are not only adding Non-interactivity, we are also identifying and fixing a few issues.

1. if P knows t(x), P could pretend to have computed p(x), by taking t(x), and then multiply it by some trivial-to-compute polynomial h(x). (And in fact, P could do this even more simply by waiting to receive $t(x)$ and r, computing the scalar $t(r)$, then setting $h(x) = c$ to a random number, and $p(x) = c \cdot t(r)$
2. if P knows r, P could compute an infinite number of polynomials that intersect with $t(r) \cdot h(r)$
3. The prover claimed to know a polynomial of degree d, but we have no enforcement of degree of p(x) so far. There are actually an infinite number of polynomials $p(x)$ that intersect with $h(x) \cdot t(x)$, with higher than d degree.

Issues 1 and 2 arise because P has access to both $t(r)$ and r. We resolve this using a weaker form of homomorphic encryption : homomorphic hiding.

Homomorphic Encryption is a form of encryption, which describes an encryption function E, decryption function D, and a set of functions $\{f_i\}$, such that: $E(f_i(x)) = f_i(E(x))$ and $D(E(x)) = x$. In words this means

that if you apply a function on the a value and then encrypt it, this is equivalent to applying the function on the encrypted value.

Homomorphic Hiding (HH) is a weaker form of Homomorphic encryption that does not require a decryption function; it only requires $E(f_i(x)) = f_i(E(x))$.

We now can see that if prover P is not able to actually see the values of r and $t(r)$, but only use them in a homomorphically hidden fashion, prover P cannot compute p(x) by multiplying t(x) by a trivial-to-compute h(x). As this encryption does not have well-defined division, we have created a SNARKOP

Our encryption scheme is roughly: Choose a base number g and n (which is carefully chosen, as a function of two large primes), and then to encrypt value v, we exponentiate $g^v \text{mod} n$. We take it on faith that this in fact does encrypt something (and it is hard to decrypt).

We specifically choose an encryption scheme that fulfills properties of "**strong homomorphic encryption**". Such that the encryption is homomorphic under both multiplication and addition. As a toy example:

$$\begin{aligned} \text{encryption} &: 5^3 = 6 (\text{mod } 7) \\ \text{multiplication} &: 6^2 = (5^3)^2 = 5^6 = 1 (\text{mod } 7) \\ \text{addition} &: 5^3 \cdot 5^2 = 5^5 = 3 (\text{mod }) \end{aligned}$$

It is important to state that exponentiation is not possible under this kind of encryption. We can multiply an unencrypted value by an encrypted value. We are unable to multiply or divide two encrypted values (and subsequently, we cannot exponentiate an encrypted value).

So, we observe that to encrypt a polynomial $p(x) = x^2 + 3x - 3$, we get $g^{p(x)} = g^{x^2+3x-3} = g^{x^2} \cdot g^{3x} \cdot g^3$. † As if we are given $E(x)$, it is impossible to compute $E(x^2)$, thus the prover must be given the encrypted powers of $E(x)$ ahead of time (this will occur during setup, described later).

$$\begin{aligned} E(x^2 + 3x - 3) &= g^{x^2+3x-3} \\ &= g^{x^2} \cdot g^{3x} \cdot g^{-3} \\ &= E(x^2) \cdot E(x) \cdot 3 \cdot g^{-3} \end{aligned}$$

This way, our protocol changes slightly. P computes $h(x) = \frac{p(x)}{t(x)}$. V selects r, and computes powers of $E(r^i)$. V shares both r and its encrypted powers with P. V computes $h(r)$ and $p(r)$ using the encrypted values shared, and returns them to the verifier V. V confirms $E(p(r)) = E(h(r)) \cdot E(t(r))$.

Protocol X

1. Verifier:
   (a) samples a random value s
   (b) calculates powers $E(s^i) = g^{s^i}$. And sends them to the prover
   (c) evaluates unencrypted target polynomial on s
2. Prover:
   (a) calculates $h(x) = \frac{p(x)}{t(x)}$
   (b) Using $E(s^i) = g^{s^i}$, prover computes $h(s)$ and $p(s)$ in encrypted forms and sends them to verifier.
3. Verifier:
   (a) verifier checks $p(s) = t(s) \cdot h(s)$, in encrypted space.
   (b) in other words: $g^p = (g^h)^{t(s)}$. Which is $g^{p(s)} = g^{t(s) \cdot h(s)}$

† **Note**: for the rest of this paper we leave out the term mod p for all exponentations.

Restricting the Degree of the Polynomials

Now, we have shown that the prover is able to find a polynomial that has the roots $t(x)$. The only part to the proof-of-polynomial that remains is to prove the degree of a polynomial. We have tried to restrict that the prover only uses a selection of encrypted powers of s ($E(s^i)$). However, if we observe what the verifier is actually checking, it is: $g^{p(s)} = g^{t(s)\cdot h(s)}$. So, the prover could find any point $b_p = b_h^{t(s)}$, and submit that as a proof instead of $g^{p(s)}$, $g^{h(s)}$.

So, we need some way to ensure that the prover is actually using the values the verifier is providing them.

Because we are operating in a space that preserves multiplication, one intuitive solution is to provide another data point alongside s, which is scaled by some value α. In short, the verifier provides 1 point, and 1 point that is multiplicatively scaled from the first point (alongside, any necessary encrypted values to evaluate the polynomials on those points) ; the prover evaluates both points and returns their evaluations; and the verifier not only validates the accuracy of those points, but that the relationship to one another is the same α as the verifier specified originally.

This concept is "Knowledge-of-Exponent" and was introduced in 1991 by Ivan Damgård [Damgård, 1991]. We now present how the Knowledge-of-Exponent scheme works, and then will explain how to use it as a black-box.

Knowledge-of-Exponent Our goal is to confirm that another agent (Bob) exponentiated a value:

1. Alice chooses value a and a random value r. Alice sends Bob (a, a^r).
2. Bob chooses whatever value v they wish to exponentiate a by. And returns to Alice $(a^v, (a^{r^v}))$.
3. Alice exponentiates the first value, and verifies that $(a^v)^r = (a^{r^v})$.

Alice is confident that Bob exponentiated a by some value, but cannot compute v. Alice cannot learn v, and Bob cannot learn r, for the same reason : they are encrypted.

Our goal is to use this tool, so that the prover return $g^{p(s)}$, $g^{h(s)}$ and $g^{\alpha\cdot p(s)}$, $g^{\alpha\cdot h(s)}$ to the verifier, and the verifier is able to confirm both a statement about the relationship of p and h, and that the prover really did compute them only using the exponentials provided.

Note : Verifier does not provide $g^{(\alpha s)^i}$ values, but rather provides $g^{\alpha(s^i)}$ values.

Consider Protocol X (stated in Fixing SARK-OP) as a black box that we have.

1. Verifier chooses some α.
2. We initiate protocol X on randomly selected value s and on value αs (this requires sending encrypted s, s^i, and $\alpha(s)^i$.
3. At the end of protocol X, the verifier checks $g^{p(s)} = g^{t(s)\cdot h(s)}$, and now, also checks $g^{p(\alpha s)} = g^{t(s)\cdot \alpha h(s)}$.
4. Now the verifier also issues the check that $(g^{p(s)})^\alpha = (g^{\alpha p(s)})$. And $(g^{h(s)})^\alpha = (g^{\alpha h(s)})$

Now we have completed a large part of our zk-SNARK proof! The verifier is able to validate that the prover knows a polynomial $p(x)$ that contains the roots $t(x)$. No one can take this away from us.

ZK-SARK-OP : Zero-Knowledge

Zero knowledge is perhaps the easiest part of the zk-SNARK. We observe that using encrypted multiples of r)($E(r)$, $E(r^2)$, ...), we seek to satisfy :

$$\begin{aligned} g^{p(r)} &= \left(g^{h(r)}\right)^{t(r)} \text{ p(x) has roots shared with t(x)} \\ g^{p(r)^\alpha} &= g^{p'(r)} \text{ correct degree is used} \end{aligned} \tag{1}$$

So we observe that if both sides were multiplied by a random value q (in the exponent) that the prover selects, the equation still holds, while so long as the verifier doesn't learn the value q, they will be unable to learn anything about $p(x)$.

$$\begin{aligned} g^{p(r)\cdot q} &= \left(g^{h(r)\cdot q}\right)^{t(r)} \\ g^{p(r)\cdot q^\alpha} &= g^{p'(r)\cdot q} \end{aligned} \tag{2}$$

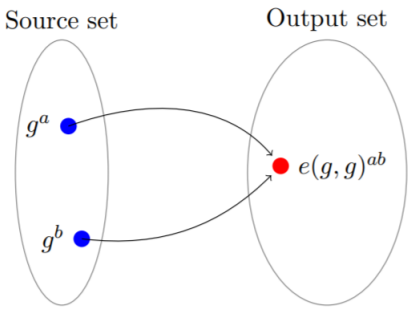


Figure 3: Relevent properties of a bilinear map [Petkus, 2019]

This is possible, because P computes $h(r)$ and $p(r)$, and so can simply multiply by value q.

Bilinear Maps and Elliptic Curves

So far we have been manually requiring the verifier to check things like $(g^p(s))^\alpha = g^{\alpha \cdot p(s)}$. But that requires the verifier knowing α. Ideally, we would have a tool that allows us to forget α, and only store encrypted-α (g^α); then to share encrypted-α with many verifiers.

With that in our minds, we take a short detour, to introduce a mathematical term "Elliptic Curve Encryption" (ECE). zk-SNARKs rely on ECE; explicitly, the following section on trusted-parties (and trusting-one-of-many) is why zk-SNARKs commonly rely on ECE.

We will ignore nearly all of the mathematics of ECE and just introduce a single concept : a **bilinear map**. A bilinear map deterministically maps 2 values in encrypted space, to 1 value in another encrypted space.

The core property we care about is :

$$e(g^a, g^b) = e(g^b, g^a) = e(g, g^b)^a = e(g^a, g)^b = e(g, g)^{ab} \tag{3}$$

We will not go into greater detail; rather we will trust that this bilinear map is a one-way function that preserves the above properties.

zk-SNARK-OP : Non-interactivity

Now we come back to our discussion of zk-SNARK-OP. So far we have had an important trust assumption that required that there was no collusion between the verifier and the prover. The ZK-SARK-OP we have so far enables a prover to prove *to a single verifier* that they know some polynomial. Why? Because, we have not removed the possibility of collusion between the prover and verifier (the verifier could share α and s with the prover through a side channel. This can be useful sometimes, but not on a blockchain.

So, now our goal is to create a means to remove this trust assumption of a single party. We will replace trust-in-one, with trust-in-one-of-many.

So far, we have been describing a 3 stage interaction : 1. verifier computes values and sends encrypted versions to prover (deleting the unecrypted versions), 2. prover uses encrypted values to compute $p(x)$ and $h(x)$, 3. verifier checks prover's response, without having to remember the unencrpyted values. The first step is actually entirely reusable; if different provers have different polynomial that have roots $t(x)$, they can use the same setup values.

For this reason, the first stage is usually called "Common Reference String" (CRS):

1. Proving Key : $(g^{s^i}, g^{\alpha s^i})$
2. Verifier values : $(g^{t(s)}, g^\alpha)$

Rather than the previous protocols which compare $g^{\alpha \cdot p(s)}$ and $(g^p(s))$, where the verifier computes $(g^p(s))^\alpha$, because they know α. With bilinear maps, the verifier can forget α, and only remember g^α.

Because, under bilinear map m, to check $p(s) = h(s) \cdot t(s)$ (in encrypted space):

- The verifier just confirms $m(g^{p(s)}, g) = m(g^{h(s)}, g^{t(s)})$ (to confirm $p = t \cdot h$ on encrpyted point s).

- And $m(g^{p(s)}, g^{\alpha}) = m(g^{\alpha \cdot p(s)}, g))$ (to confirm the polynomial restriction).

If we have a single trusted-party, we trust that they will compute the Proving Key, Verifier Key, and the delete the "toxic waste" s^i and αs^i.

Which leads us to our next section:

Trusting-One-of-Many

In order to destroy the need for a single trusted party, we wish to create a different way to generate s^i and αs^i, as a joint effort between many agents, and if **even one** agent destroys their secret values, the CRS is protected (this will be called a ceremony).

Each agent, 0, 1, 2, ... N (for N participants) will select a random value $alpha_i$ and s_i. Agent 0 will publish $(g^{s_0^i}, g^{\alpha_0 s_0^i})$ for i in 0...d (d, agreed-upon degree of polynomial). Agent 1, also generates their values s_1 and α_1. Then augments Agent 0's values:

$$((g^{(s_0)^i})_1^s, (g^{\alpha_0 (s_0)^i})^{\alpha_1 s_1}) = (g^{(s_0 \cdot s_1)^i}, g^{\alpha_0 \alpha_1 (s_0 \cdot s_1)^i}) \tag{4}$$

And this continues through all the agents.

At this point, the only risk of a prover being able to prove something incorrect is if **ALL** the agents in the CRS-generating ceremony collude.

However, there is one last catch that needs to be qualified : an agent j could insert random values for their $\alpha_i s_j^i$ and s_j^i for each value $i \in [d]$. This would make the CRS invalid and useless.

So, to confirm that each actor is behaving properly, we once again use the bilinear map:

Using bilinear map m, for each user, we confirm:

$$m\left(g^{s^i}, g^1)\right) = m\left(g^{s^{i-1}}, g^{s^1})\right) = ... \tag{5}$$

This is done for all powers, and so each participant shows that all their s_i's come from the same base value. And the same is done with αs_i

$$m\left(g^{\alpha s^i}, g^1)\right) = m\left(g^{s^{i-1}}, g^{\alpha s^1})\right) = ... \tag{6}$$

That is it. We have just created a zk-SNARK-OP. We are able to prove that an agent knows a polynomial $p(x)$, that has roots $t(x)$, without learning anything about $p(x)$, and only having to trust that at least one agent of those participating in a CRS-ceremony was trustworthy.

We now review the protocol of zk-SNARK-OP as a whole. [Petkus, 2019][Groth, 2010]

From Computations to Polynomials

We have made some pretty incredible progress on a goal that seems a bit contrived - knowledge of a polynomial. We now tackle the problem of taking an arbitrary function/computation, and mapping it onto the Knowledge-Of-Polynomial problem, which we can create a zk-SNARK-OP for.

Quadratic Arithmetic Programs (QAP)

*"**Qap**la"*[†] - Lieutenant Worf, Son of Mogh, Starship Enterprise

First we must introduce a problem in NP : Quadratic Span Programs (QSP), and a variant of QSPs: Quadratic Arithmetic Programs (QAP). At a high-level, a QSP consists of a set of polynomials and the goal is to a find a linear combination, which is a multiple of another given polynomial. Later we will describe how to assign a QAP to represent each computation in a program (in section Map Problem to R1CS).

[†] "Qapla" means "Success" in Klingon

An example computation to a polynomial

Let us start with a computation that is intuitive to convert to a polynomial.

Algorithm 1 Example computation 1

```
function F(x, a, b):
    if x == 1 then
        return a × b
    else if x == 0 then
        return a + b
    end if
end function
```

For $x \in [0, 1]$, f can be easily mapped to the polynomial $f(x, a, b) = x(a \times b) + (1 - x)(a + b)$.

At a high-level, we have a function which we can have some agent P compute, which a verifier (any verifier) can confirm that P accurately computed $f1$, on inputs (x, a, b) and returned value y, without the verifier having to actually compute f1 themselves!

Arbitrary computations to polynomial

Functions (in the computer science sense) are composed of multiple steps, so we need to show 2 things: 1. how to map an arbitrary computation to a polynomial, and 2. how to compose computations/polynomials together.

Arbitrary computation

While it is possible to specify an operator that takes any number of arguments, the concept of an operation can be reduced and simplified to the form :

(left-operand) **operator** (right-operand) = output

So, if the prover seeks to claim that they made such an operation, we can map this onto:

$$(l(x) \textbf{ operator } r(x) = o(x)$$
$$p(x) = (l(x) \textbf{ operator } r(x) - o(x) = 0$$

If value a solves the above equality, then $t(x) = x - a$ is a factor of the above polynomial ($p(x)$).

For example, let us consider $f(a) = 3 = 15$.

Observe: pay attention that we are not solving $f(a)$, we are verifying that $f(a) = 15$.

This can be expressed as $l(x) = 3x$, $r(x) = 5x$, $o(x) = 15x$, for $a = 1$.

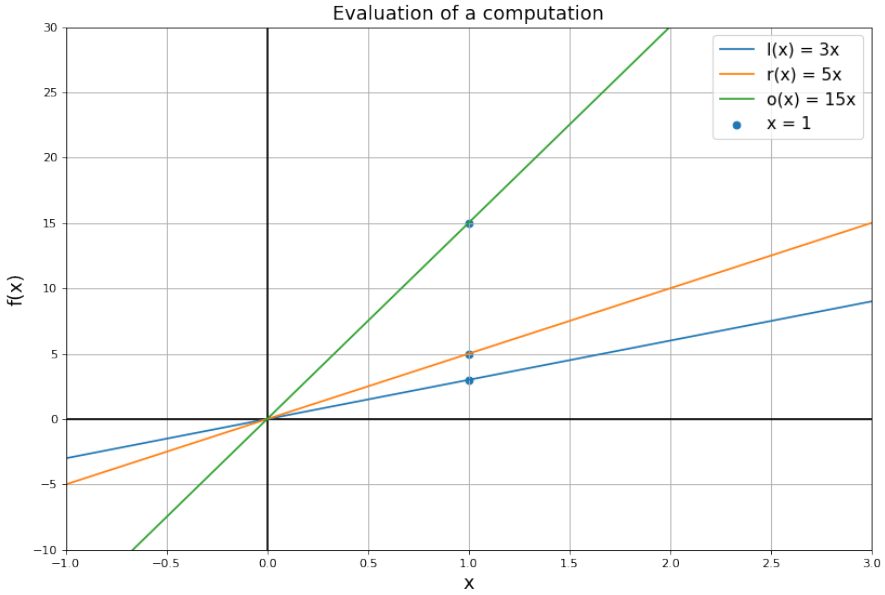


Figure 4: Arbitrary Polynomial Computation

This polynomial can rewritten as : $(3x \times 5x) - 15x = 0$, which is $15x^2 - 15x = 0$, which is $(15x)(x - 1)$

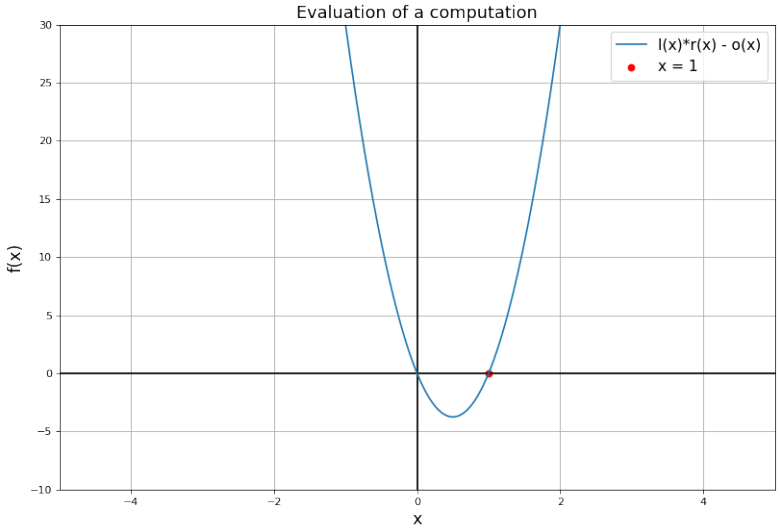


Figure 5: Arbitrary Polynomial Computation ($15x^2 - 15x = 0$)

However, this set of polynomials is entirely arbitrary! It can also be written as $l(x) = 2x^2 - 1.5$, $r(x) = 3x + 0.5$, $o(x) = 15$, for $a = 1.5$.

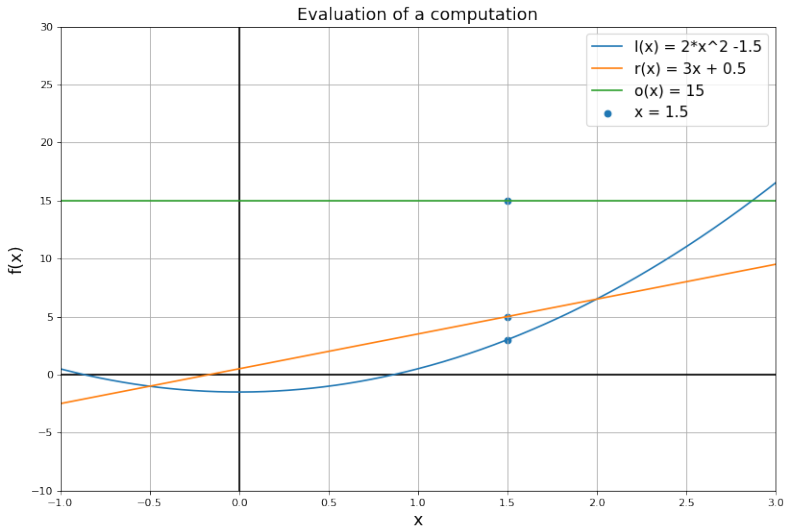


Figure 6: Arbitrary Polynomial Computation

This would be rewritten as : $l(x) \cdot r(x) - o(x) = 0$.
$6x^3 + x^2 - 4.5x - 15.75 = 0$

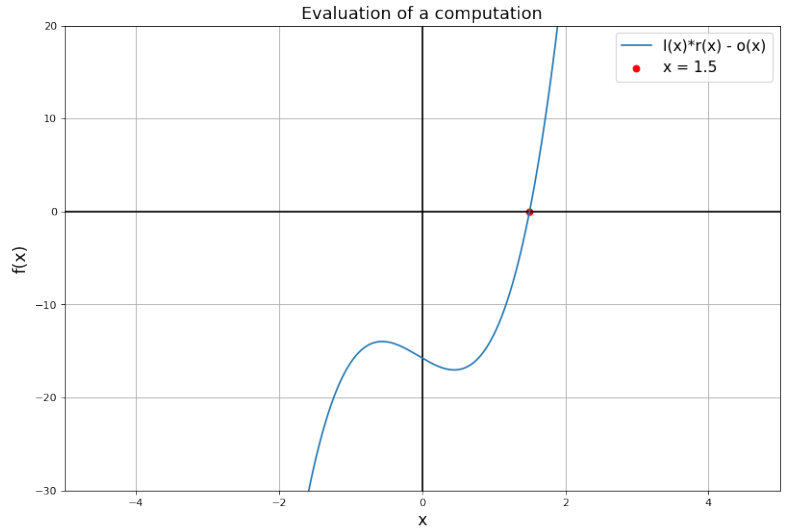


Figure 7: Arbitrary Polynomial Computation ($6x^3 + x^2 - 4.5x - 15.75$)

In both these polynomials, it is clear (both algebraically and visually) that the to solution to the function is a root of the polynomial (x = 1 and x = 1.5, respectively), for the two polynomial setups.

This means that if we were to specify $l(x), r(x), o(x)$ ahead of time, we could cleanly map the zk-SNARKOP scheme, to confirm that a prover does in fact know a solution to the equation. The only change we make to the zk-SNARKOP protocol is that $p(x) = l(x) \cdot r(x) - o(x)$. However, since we are operating in encrypted space, and subtraction is more difficult to compute than addition, we rewrite it is

$$\begin{aligned} l(x)r(x) - o(x) &= h(x) \cdot t(x) \quad \text{original polynomial proof} \\ l(x)r(x) &= h(x) \cdot t(x) + o(x) \quad \text{removing the need for subtraction} \end{aligned} \tag{7}$$

In encrypted space, the verifier checks:

- $m\left(g^{l(s)}, g^{r(s)}\right) = m\left(g^{h(s)}, g^{t(s)}\right) \cdot m\left(g^{o(s)}, g\right)$
- $m\left(g, g\right)^{l(s)r(s)} = m\left(g, g\right)^{t(s)h(s)+o(s)}$

We have continued to keep doing it! Given a operation ($3 \times 5 = 15$), and 3 polynomials that represent it, we have now been able to prove its validity.

All we did was :

1. select a value a
2. select 3 polynomials that preserve the validity of our operation at point a, for $l(a) \cdot r(a) = o(a)$.
3. Have the prover show that they know a polynomial with root a.

Multiple Operations

The operation of mapping a function into a polynomial seems pretty clear, but it's easy to gloss over where the details are. It's more obvious when you attempt to combine multiple operations together. For example, consider the equation $y = a \times b \times c$. Using binary operators, this looks like

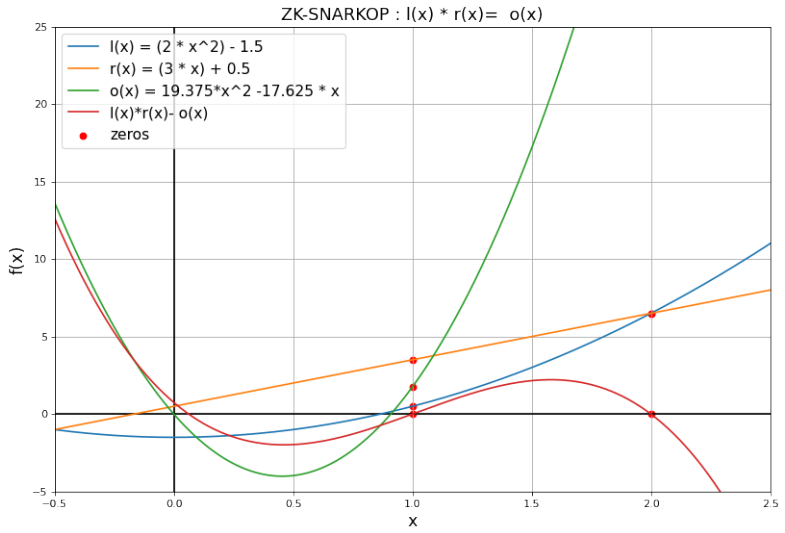
$$\begin{aligned} a \times b &= p \\ p \times c &= y \end{aligned} \tag{8}$$

We observe that we can actually pick polynomials $l(x)$, $r(x)$, and $o(x)$ that can encode both the first and the second equations that represent $y = a \times b \times c$.

We first select x_0; and then $l(x_0) = a$, $r(x_0) = b$, $o(x_0) = p$; then we select x_1; and then $l(x_1) = p$, $r(x_1) = c$, $o(x_1) = y$.

Let us pick $x_0 = 1$, $x_1 = 2$, as well as our previous $l(x)$, $r(x)$; $l(x) = 2x^2 - 1.5$, $r(x) = 3x + 0.5$. There are an infinite number of $o(x)$ we can pick, but for this first exercise, let us derive by hand one solution that works for $x_0 = 1$, $x_1 = 2$. $l(1) = 0.5$, $l(2) = 6.5$, $r(1) = 3.5$, $r(2) = 6.5$. So $o(x)$ must travel between points $(1, 1.75)$ and $(2, 42.25)$.

We pick $o(x) = 19.375x^2 - 17.625$



You may be wondering : if we are able to derive $l(x)$, $r(x)$, and $o(x)$ what do we need the prover for. Good question. This leads us to our next point, which we only implicitly mentioned so far: **polynomial interpolation**.

Polynomial Interpolation

Given d points (such that no points have the same x-value), there exists a unique d-degree polynomial that passes through each of those points. There are plenty of techniques to solve for this, the simplest to understand being simply a set of equations with unknowns, but also one can use Fast Fourier Transforms, Newton polynomials, or Lagrange polynomials.

During set up, a verifier specifies polynomials $l(x)$, $r(x)$, and d unencrypted points x. The prover will evaluate $l(x)$ and $r(x)$ on the unencrypted points x. Then once the d points are evaluated, the prover will

interpolate a polynomial $o(x)$. Once prover has $o(x)$ they can evaluate $o(x)$, along with $l(x)$, $r(x)$, $t(x)$, and $h(x)$ on the homomorphically hidden points s, as before.

At first glance, interpolation would only make sense for polynomials $l(x)r(x) - o(x)$ of a single variable. However one can define $p_i(x)$ for each variable i, and then combine them into a single polynomial $p(x)$ by evaluating each $p_i(x)$ on a distinct set of points, then interpolating between them.

Note: here $l(x)$ and $r(x)$ represent the actual computation that the prover is doing, so this should require minimal overhead to compute these evaluations.

QAP expressivity

We have described how to do polynomial interpolation for representing multiplicative computations, but the same is possible for addition, subtraction, and division.

Previously, in An example computation to a polynomial we informally showed that you can create conditional statements.

Addition and **subtraction** can be expressed by a simple operand

$$(a + b) \times 1 = o(x)$$
$$(a + (-1 \cdot b)) \times 1 = o'(x)$$

We have already shown how to express multiplication, and so **division** falls out naturally. If we wish to prove that $\frac{a}{b} = x$, this can be expressed as $b \times x = a$, and verified as a multiplication.

Using QAPs it's also possible to define arbitrary constant constraints; for example $(x - 2) \times 1 = 0$, requires that x is equal 2; $x \times x = x$ ensures that x is a binary number (1 or 0).

Informally we have just shown that zk-SNARKs can verify any Turing complete computation.

Map Problem to R1CS

Each operation is also called a "constraint," because the operation does not represent the computation, but rather only verifies that the prover already knows the results. In other words the prover is required to provide values that are consistent with the constraints [Petkus, 2019].

This collection of constraints is specified as a **Rank-1 Constraint System (R1CS)**. And mapping a series of computation in a function to these polynomials constraints is the job of a zk-SNARK compiler.

Note: we must ensure that any variable can only have a single value across every operation it is used in (otherwise, you create a free variable, and the prover can modify it, without using the intended l(x) and r(x)).

Zero-Knowledge R1CS

We previously added Zero-Knowledge to a polynomial expression $p(x) = h(x) \cdot t(x)$, by allowing the prover choose a random q to shift the polynomials by. In this new QAP constructions $l(x) \times r(x) = o(x)$, we again attain Zero-Knowledge by shifting by a random values q_l, q_r, q_o. To maintain the equality, the prover computes a value:

$$\Delta = f(q_l, q_r, q_o) = \frac{q_l \cdot l(s) \times q_r \cdot r(s) - q_o \cdot o(s)}{t(s)h(s)} \tag{9}$$

And then instead sends to the verifier the points $(l(s) + q_l)$, $(r(s) + q_r)$, $(o(s) + q_o)$, which correspond with:

$$(l(s) + q_l) \cdot (r(s) + q_r) - (o(s) + q_o) = t(s) \cdot (\Delta \times h(s)) \tag{10}$$

[Petkus, 2019][Groth, 2010]

zk-SNARK Protocol Summary

We now summarize the entirety of the generation of a zk-SNARK.

- Setup (done publicly through a trust-ceremony)
  1. select cryptographic bilinear map m. Select encryption generator g.
  2. For a given function f, compile the function into R1CS polynomial (degree d) form: $\big(\{(l_i(x), r_i(x), o_i(x)\}_{i\in[n]}, t(x)\big)$.
  3. Generate a CRS proving key and verification key:
     (a) Sample s, $\{\alpha_l, \alpha_r, \alpha_o\}$
     (b) Set the proving key :

$$\begin{pmatrix} \{g^{s^k}\}_{k\in[d]}, \\ \{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\}_{i\in[n]}, \\ \{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\}^{\alpha}_{i\in[n]}, \\ \{g_l^{t(s)}, g_r^{t(s)}, g_o^{t(s)}\}, \\ \{g_l^{\alpha t(s)}, g_r^{\alpha t(s)}, g_o^{\alpha t(s)}\} \end{pmatrix}$$

     (c) Set the verifying key

$$\begin{pmatrix} g, g_o^{t(s)}, \\ \{g_l^{l_i(s)}, g_r^{r_i(s)}, g_o^{o_i(s)}\}_{i\in[m]}, \\ g^{\alpha_l}, g^{\alpha_r}, g^{\alpha_o} \end{pmatrix}$$

- Proving
  1. Prover executes computation f(u), storing the intermediary values v_i as they execute the computation.
  2. Interpolate a unencrypted polynomials for $L(x)$, $R(x)$, $O(x)$; following the form: $L(x) = l_0(x) + \sum_{i=1}^{n} v_i \cdot l_i(x)$
  3. sample random values q_l, q_r, q_o.
  4. Compute $h(x) = \frac{L(x)R(x)-O(x)}{t(x)} + (q_l L(x) + q_r R(x) + q_l q_r t(x) - q_o)$
  5. Compute $h(s)$, $p(x)$ using the encrypted values provided in the CRS (shifted both by a random q for zero-knowledge, and by α in order to restrict the polynomial).
  6. Send the verifier(s) $\left(g_l^{L_p(s)}, g_r^{R_p(s)}, g_o^{O_p(s)}, g_l^{L_p'(s)}, g_r^{R_p'(s)}, g_o^{O_p'(s)}, g^{h(s)}\right)$
- Verifying
  1. Compute the polynomials ($g_l^{L_v(s)}$, $g_r^{R_v(s)}$, $g_o^{O_v(s)}$)from the values sent by the prover; following the form $g_l^{L_v(s)} = \prod_{i=0}^{m} \left(g_l^{l_i(s)}\right)^{v_i}$.
  2. issue checks on the polynomials; notably:
     (a) $e\left(g_l^{L_p} \cdot g_l^{L_v(s)}, g_r^{R_p} \cdot g_r^{R_v(s)}\right) = e\left(g_o^{t(s)}, g^h\right) \cdot e\left(g_o^{O_p} \cdot g_o^{O_v(s)}, g\right)$
     (b) and the polynomial degree restriction check for $L(x)$, $R(x)$, $O(x)$, of the form: $e\left(g_l^{L_p}, g^{\alpha_l}\right) = e\left(g_l^{L_p'}, g\right)$

[Petkus, 2019][Groth, 2010][Gennaro et al., 2013]

zk-SNARK Conclusions

How zk-SNARKs are Applied to Create a Shielded Transaction

Looking at Z-Cash, the zk-SNARKs are applied in the following fashion: If you have sent an encrypted coin $c = h(v)$ to the chain. You know the value v coin, which represents an encrypted coin present in the list of encrypted coins available. In ZCash, there is a single zk-SNARK setup required, which computes the proof which ensures that the withdrawer of the coins knows the nullifier and the randomness needed to compute the coin value. For more details refer to section Construction.

Beyond zk-SNARKs

zk-SNARKs rely on a trusted setup (i.e. a public CRS for the prover and the verifier). If this reference string is leaked, an adversary can create undetectable fake proofs (Toxic Waste) - though multi-party ceremony reduces the risk of this. Furthermore, this string is tied to a circuit and a implementation, that means that the zk-SNARK produced cannot scale for arbitrary computation and moreover, if any modifications or error corrections need to be done in the code, it will require a new trusted setup.

In recent times, there have been efforts to develop universal zk-SNARKs, that is arbitrary code like smart contracts can be run as a zk-SNARK. Furthermore, with the ever increasing threat of Quantum-Computing's ability to break our current encryption, research efforts have been focused on building quantum-resistant hash functions and zk-SNARK alternatives to this paradigm.

	Common Reference String (CRS)		Structured Reference String (SRS)		
	Transparent arguments		Universal		Circuit Specific
Verification time			Updatable	Static	
Linear	Ligero, Aurora	Bulletproofs, Halo		AuroraLight	
Fast for all circuits	Fractal	Spartan, SuperSonic-CG	Sonic, Marlin, SuperSonic-RSA, Plonk		Groth16, BTCV14
Fast for optimized circuits	zk-STARK, Succinct Aurora	Hyrax		Libra	

Figure 8: Classification of zk-SNARKs based on the type of reference string [Mannak, 2019]

Recent approaches in this field have been through zk-STARKs [Ben-Sasson et al., 2018], Fractal [Chiesa et al., 2020b], Halo [Bowe et al., 2019], SuperSonic [Bünz et al., 2020], Marlin [Chiesa et al., 2020a] and Plonk [Gabizon et al., 2019]. However, a lot of these suffer from large proof sizes, sizeable proof/verification times, and in a lot of these implementations, the verification time is no longer constant.

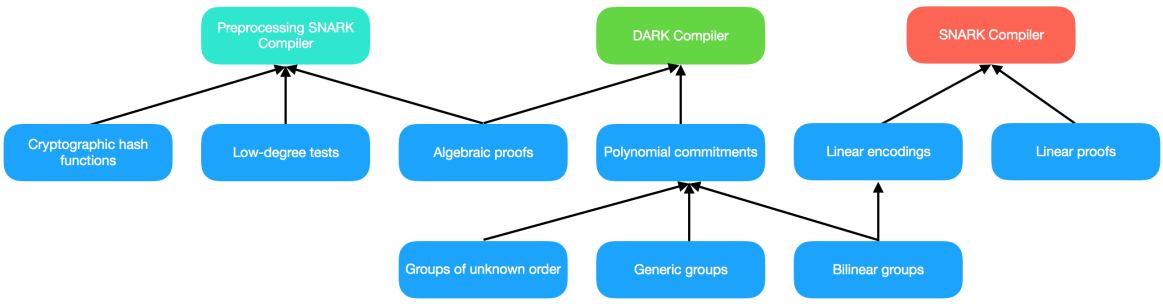


Figure 9: The three types of zk-SNARK compilers (colors match the above table). [Mannak, 2019]

The reason why these Universal zk-SNARKs are better is due to the non-dependence on a trusted setup (using publicly verifiable randomness to create trustless verifiable computation systems). Furthermore, they are resistant to attacks from quantum-computers. Furthermore, as with some L2 zk-SNARK implementations, moving these universal zk-SNARKs off-chain would result in significant scalibility of computation and size when moved off chain. `https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/zk-starks/`

Conclusion to ZK-snarks

We have now shown, at a decently low-level degree, how a zk-SNARK can be constructed, given an arbitrary computation. We now recall the properties that we wished to have in our ZK-SNARK. The last ≈ 12 pages have

ZK	**Z**ero-**K**nowledge
S	**S**uccinct
N	**N**on-interactive
ARK	**AR**gument of **K**nowledge

ideally shown how to: take a function, deconvolve it into a set of computations, translate those computations into polynomials, evaluate those polynomials as scalars; then to interpolate those scalars into a single polynomial, and then to evaluate that polynomial as a series of homomorphically hidden points. Ensuring that no matter the length of the function, the verification would be a constant (short) length, non-interactive, and zero-knowledge.

Privacy Solutions

In blockchain privacy is particularly tricky to achieve due to its design that all transactions are transparent and the supply of coins is verifiable. Hence, privacy solutions have to ensure these mechanisms are preserved while protecting privacy.

However, privacy coins handle two different aspects; anonymity and untraceability. Anonymity hides the identity behind a transaction, while untraceability makes it virtually impossible for third-parties to follow the trail of transactions using services such as blockchain analysis.

The two major ways blockchain solutions implement privacy is using Tumblers (Layer-2 solutions) or a creating a separate blockchain itself.

In the next sections we do a deep-dive into ZeroCash (the protocol beind ZCash) and Monero, two of the most popular privacy coins as well as Tornado Cash, a Ethereum L2 tumbler.

ZCash/ZeroCash

ZCash is based on the ZeroCash protocol [Sasson et al., 2014] with certain improvements to their protocol [More information on the z.cash website]. In the following section, we will describe the ZeroCash protocol, and the privacy guarantees they achieve.

The main feature of ZeroCash was Decentralized Anonymous Payment Schemes (DAP schemes), which hides the payments origin, destination and amount.

Construction

The construction of this scheme follows 6 incremental steps:

Step 1: User Anonymity with Fixed-Value Coins

To start, the authors assume a simple construction, where all transactions have the same value (e.g. 1 Z-Coin). This step showcases how to hide the sender's information. In addition to zk-SNARKs, they also utilize a commitment scheme (defined below)

If $\mathbf{COMM}$ is a statistically hiding non-interactive commitment scheme, that is, given randomness r and message m, the commitment o is

$$o := \mathbf{COMM}_r(m)$$

Also, o is opened by revealing r and m, and one can verify that $\mathbf{COMM}_r(m)$ equals o
In this construction, a new coin $\mathbf{c}$ is minted as follows:

1. The user u samples a random serial number sn
2. The user u also samples a trapdoor randomness r (a random number generated using a trapdoor function)
3. The user u computes coin commitment $cm := \mathbf{COMM}_r(sn)$
4. The coin is set to $\mathbf{c} := (r, sn, cm)$
5. A minting transaction tx_{Mint} containing only cm is sent to the blockchain ledger
6. tx_{Mint} is appended to the ledger only if user u has deposited 1 Z-Coin into a escrow pool (maintained by chain). This payment proof can be encoded into the memo/plaintext information of tx_{Mint}
7. Let $CMList$ be a list of all coins (like tx_{Mint}) committed onto the blockchain
8. The user u may spend the coin $\mathbf{c}$ by executing a spending transaction tx_{Spend}
9. tx_{Spend} contains the coin's serial number sn as well as a zk-SNARK proof π of that statement *I know r such that* $\mathbf{COMM}_r(sn)$ *exists in* $CMList$.
10. As long as sn belongs to a UTXO (doesn't appear in any previous tx_{Spend} transactions), the user u can get back the deposited 1 Z-Coin from the escrow pool (which can be kept, transferred or deposited again)

Firstly, we note that Mint transactions are thus certificates of deposit, deriving their value from the backing pool. Secondly, if sn already appeared on the ledger, this is considered double spending, and the transaction is discarded.

Finally, this construction guarantees anonymity of user u as the proof π is zero-knowledge, i.e., even though sn is revealed, no information about r is given, so finding which of the numerous transactions tx_{Mint} in $CMList$ corresponding to inverting the commitment scheme $f(x) := \mathbf{COMM}_x(sn)$, which is assumed to be infeasible and hence, the payment sender is anonymous.

Step 2: Compressing the List of Coin Commitments

In the zk-SNARK proof statement in Step 1, $CMList$ is defined as a list of coin commitments. However, this limits scalability as this time and space complexity of searching, verifying etc. would grow linearly [O(n)] with the size of this list. Moreover, even if we wanted to drop spent coins from this list, we wouldn't be able to do that since they cannot be identified (due to the Zero Knowledge property that provides anonymity)
Similar to other blockchains (Bitcoin/Ethereum) they use a collision resistant hash function (CRH) to create a merkle tree of these transactions. This datastructure is append-only and the time and space for verification/insertion algorithms is now proportional to the depth of the tree [O(log n)]. So, the zk-SNARK proof π statement is updated as *I know r such that* $\mathbf{COMM}_r(sn)$ *exists as a leaf in a merkle tree with a root* r_t. Specifically, ZeroCash uses merkle trees of height 64, so they can support 2^{64} coins. (For reference ZCash only has 21 million coins totally planned. Which means each coin can be used on average for $\sim$ 1 Billion shielded transactions)

Step 3: Extending Coins for Direct Anonymous Payments

In our current model. the commitment of a coin $\mathbf{c}$ is the commitment of it's serial number sn. However this creates three problems.

Firstly, when transferring the coin to another user $u_A \xrightarrow{\mathbf{c}} u_B$, since the sender u_A knows sn the reciever u_B cannot spend this coin anonymously, (since u_A knows sn and hence can track the spendings of coin $\mathbf{c}$, also the user u_A can spend the coin himself. Hence, u_B must immediately spend $\mathbf{c}$ and mint a new coin $\mathbf{c}'$ to protect their anonymity.

Secondly, if u_A wants to transfer 100 BTC to u_B they must use 100 seperate transfers which is impractical as well as non-anonymous (since now the amount is leaked).

Thirdly, any denominations of fractional BTC are not supported.

To combat this, the ZeroCash authors modify the derivation of a coin commitment, and use pseudorandom functions to target payments and to derive serial numbers. They create three psudorandom functions (derived from a single one). For a seed x, they are denoted by $PRF_x^{addr}(\cdot)$, $PRF_x^{sn}(\cdot)$ and $PRF_x^{pk}(\cdot)$, with the assumption that $PRF_x^{sn}(\cdot)$ is collision resistant.

Firstly, to provide targets for payments, addresses are used: each user u generates an address key pair (a_{pk}, a_{sk}), the address public key and address private key respectively. The coins of u contain the value a_{pk} and can be spent only with knowledge of a_{sk}. A key pair (a_{pk}, a_{sk}) is sampled by selecting a random seed a_{sk} and setting $a_{pk} := PRF_{a_{sk}}^{addr}(0)$. A user can generate and use any number of address key pairs.

Next, the minting policy is redesigned to increase functionality. To mint a coin $\mathbf{c}$ with a value v, the user u first samples ρ, a secret value to calculate the coin's serial number as $sn := PRF_{a_{sk}}^{sn}(\rho)$. Then u commits to the tuple (a_{pk}, v, ρ). This happens in two phases:

1. u computes $k := COMM_r(a_{pk}||\rho)$ for a random r
2. u computes $cm := COMM_s(v||k)$ for a random s

This results in a coin $c = (a_{pk}, v, \rho, r, s, cm)$ and minting transaction $tx_{Mint} := (v, k, s, cm)$. The major advantage of this method is, due to nested nested commitment, anyone can verify that cm in tx_{Mint} is a coin commitment of a coin of value v (by checking that $COMM_s(v||k)$ equals cm) but cannot discern the owner (by learning the address key a_{pk}) or serial number (derived from ρ) because these are hidden in k

To describe coin spending, they define a $pour$ operation as follows:

1. User u with address key pair $(a_{pk}^{old}, a_{sk}^{old})$ wishes to spend his coin $c^{old} = (a_{pk}^{old}, v^{old}, \rho^{old}, r^{old}, s^{old}, cm^{old})$ and produce two new coins c_1^{new} and c_2^{new} with total value $v_1^{\text{new}} + v_2^{\text{new}} = v^{\text{old}}$, respectively targeted at address public keys $a_{\text{pk},1}^{\text{new}}$ and $a_{\text{pk},2}^{\text{new}}$. (The addresses $a_{\text{pk},1}^{\text{new}}$ and $a_{\text{pk},2}^{\text{new}}$ may belong to u or to some other user.)

2. The user u, for each $i \in \{1, 2\}$, proceeds as follows:

   (i) u samples serial number randomness ρ_i^{new}

   (ii) u computes $k_i^{\text{new}} := \text{COMM}_{r_i^{\text{new}}} \left(a_{\text{pk},i}^{\text{new}} \| \rho_i^{\text{new}} \right)$ for a random r_i^{new}

   (iii) u computes $\text{cm}_i^{\text{new}} := \text{COMM}_{s_i^{\text{new}}} \left(v_i^{\text{new}} \| k_i^{\text{new}} \right)$ for a random s_i^{new}.

3. This yields the coins $\mathbf{c}_1^{\text{new}} := \left(a_{\text{pk},1}^{\text{new}}, v_1^{\text{new}}, \rho_1^{\text{new}}, r_1^{\text{new}}, s_1^{\text{new}}, \text{cm}_1^{\text{new}} \right)$ and $\text{c}_2^{\text{new}} := (a_{\text{pk},2}^{\text{new}}, v_2^{\text{new}}, \rho_2^{\text{new}}, r_2^{\text{new}}, s_2^{\text{new}}, \text{cm}_2^{\text{new}})$. Next, u produces a zk-SNARK proof π_{Pour} for the following statement, which we call POUR:

   "Given the Merkle-tree root rt, serial number sn^{old}, and coin commitments $\text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}$, I know coins $\mathbf{c}^{\text{old}}, \mathbf{c}_1^{\text{new}}, \mathbf{c}_2^{\text{new}}$, and address secret key $a_{\text{sk}}^{\text{old}}$ such that:

   - The coins are well-formed: for c^{old} it holds that $k^{\text{old}} = \text{COMM}_{r^{\text{old}}} \left(a_{\text{pk}}^{\text{old}} \| \rho^{\text{old}} \right)$ and $\text{cm}^{\text{old}} = \text{COMM}_{s^{\text{old}}} \left(v^{\text{old}} \| k^{\text{old}} \right)$; and similarly for $\mathbf{c}_1^{\text{new}}$ and $\mathbf{c}_2^{\text{new}}$.
   - The address secret key matches the public key: $a_{\text{pk}}^{\text{old}} = \text{PRF}_{a_{8d}^{\text{old}}}^{\text{addr}}(0)$.
   - The serial number is computed correctly: $\text{sn}^{\text{old}} := \text{PRF}_{a_{sk}^{\text{old}}}^{\text{sn}} \left(\rho^{\text{old}} \right)$.
   - The coin commitment cm^{old} appears as a leaf of a Merkle-tree with root rt.
   - The values add up: $v_1^{\text{new}} + v_2^{\text{new}} = v^{\text{old.}}$." A resulting pour transaction $tx_{pour} := (\text{rt}, \text{sn}^{\text{old}}, \text{cm}_1^{\text{new}}, \text{cm}_2^{\text{new}}, \pi_{\text{POUR}})$ is appended to the ledger (As before, the transaction is rejected if the serial number sn appears in a previous transaction.)

Now suppose that u does not know, say, the address secret key $a_{\text{sk},1}^{\text{new}}$ that is associated with the public key $a_{\text{pk},1}^{\text{new}}$. Then, u cannot spend $\mathbf{c}_1^{\text{new}}$ because he cannot provide $a_{\text{sk},1}^{\text{new}}$ as part of the witness of a subsequent pour operation. Furthermore, when a user who knows $a_{\mathbf{sk},1}^{\text{new}}$ does spend $\mathbf{c}_1^{\text{new}}$, the user u cannot track it, because he knows no information about its revealed serial number, which is $\text{sn}_1^{\text{new}} := \text{PRF}_{a_{sk,1}^{n}}^{\text{sn}^{n+1}} (\rho_1^{\text{new}})$.

Also observe that tx_{pour} reveals no information about how the value of the consumed coin was divided among the two new fresh coins, nor which coin commitment corresponds to the consumed coin, nor the address public keys to which the two new fresh coins are targeted. The payment was conducted in full anonymity.

More generally, a user may pour $N^{\text{old}} \geq 0$ coins into $N^{\text{new}} \geq 0$ coins. For simplicity they consider the case $N^{\text{old}} = N^{\text{new}} = 2$, without loss of generality. Indeed, for $N^{\text{old}} < 2$, the user can mint a coin with value 0 and then provide it as a "null" input, and for $N^{\text{new}} < 2$, the user can create (and discard) a new coin with value 0. For $N^{\text{old}} > 2$ or $N^{\text{new}} > 2$, the user can compose $\log N^{\text{old}} + \log N^{\text{new}}$ of the 2-input/2-output pours.

Step 4: Sending Coins

Suppose that $a_{\text{pk},1}^{\text{new}}$ is the address public key of u_1. In order to allow u_1 to actually spend the new coin $\mathbf{c}_1^{\text{new}}$ produced above, u must somehow send the secret values in c_1^{new} to u_1. One way is for u to send u_1 a private message, but the requisite private communication channel necessitates additional infrastructure or assumptions. They build this capabilities directly into the blockchain as follows:

1. They modify the structure of an address key pair.

2. Each user now has a key pair (addr $_{pk}$, addr $_{sk}$), where $addr_{pk} = (a_{pk}, pk_{\text{enc}})$ and $addr_{sk} = (a_{sk}, \text{sk}_{\text{enc}})$.

3. The values $(a_{\text{pk}}, a_{\text{sk}})$ are generated as before.

4. In addition, $(\text{pk}_{\text{enc}}, sk_{enc})$ is a key pair for a key-private encryption scheme.

5. Then, u computes the ciphertext $\mathbf{C}_1$ that is the encryption of the plaintext $(v_1^{\text{new}}, \rho_1^{\text{new}}, r_1^{\text{new}}, s_1^{\text{new}})$, under $\text{pk}_{\text{enc}\,1}^{\text{new}}$ (which is part of u_1 's address public key $\text{addr}_{\text{sk}}^{\text{new}}$), and includes $\mathbf{C}_1$ in the pour transaction tx_{pour}.

6. The user u_1 can then find and decrypt this message (using his $sk_{enc,1}^{new}$) by scanning the pour transactions on the blockchain.

7. Note that adding $\mathbf{C}_1$ to tx_{pour} leaks neither paid amounts, nor target addresses due to the key-private property of the encryption scheme (The user u does the same with $\mathbf{c}_2^{\text{new}}$ and includes a corresponding ciphertext $\mathbf{C}_2$ in tx_{pour}.)

Step 5: Public Outputs

The construction so far allows users to mint, merge, and split coins But how can a user redeem one of his coins, i.e., convert it back to the base currency (Bitcoin)? For this, they modify the pour operation to include a public output. When spending a coin, the user u also specifies a nonnegative v_{pub} and a transaction string info $\in \{0,1\}^*$. The balance equation in the NP statement POUR is changed accordingly: " $v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v^{\text{old}}$" . Thus, of the input value v^{old} , a part v_{pub} is publicly declared, and its target is specified, somehow, by the string info The string info can be used to specify the destination of these redeemed funds (e.g., a Bitcoin wallet public key). [5] Both v_{pub} and info are now included in the resulting pour transaction tx_{pour}. (The public output is optional, as the user u can set $v_{\text{pub}} = 0$.)

Step 6: Non-Malleability

To prevent malleability attacks on a pour transaction tx_{pour} (e.g., embezzlement by re-targeting the public output of the pour by modifying info), they further modify the NP statement POUR and use digital signatures. Specifically, during the pour operation, the user u

1. Samples a key pair $\left(\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}}\right)$ for a one-time signature scheme

2. Computes $h_{\text{Sig}} := \text{CRH}\left(\text{pk}_{\text{sig}}\right)$

3. Computes the two values $h_1 := \text{PRF}^{\text{pk}}_{a^{\text{ol}}_{\text{sk},1}}(h_{\text{Sig}})$ and $h_2 := \text{PRF}^{\text{pk}}_{a^{\text{old}}_{\text{sk},2}}(h_{\text{Sig}})$, which act as MACs to "tie" h_{Sig} to both address secret keys

4. Modifies POUR to include the three values h_{Sig}, h_1, h_2 and prove that the latter two are computed correctly

5. Uses sk_{sig} to sign every value associated with the POUR operation, thus obtaining a signature σ, which is included, along with pk_{sig}, in tx_{Pour}. Since the $a^{old}_{sk,i}$ are secret, and with high probability h_{Sig} changes for each pour transaction, the values h_1, h_2 are unpredictable

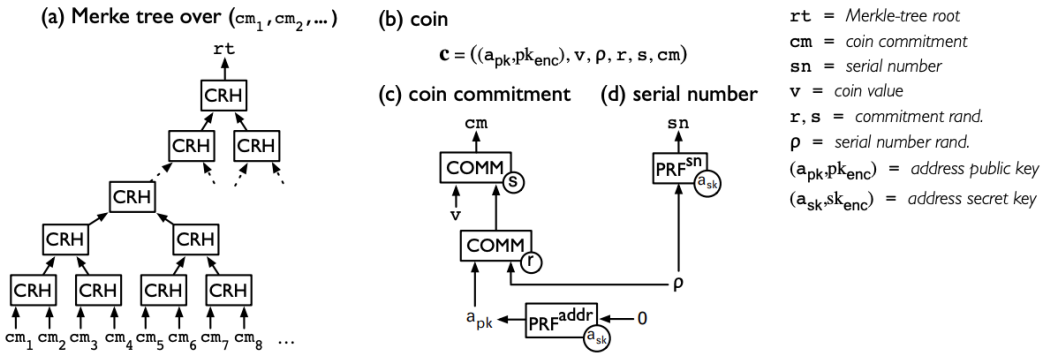


Figure 11: (a) Illustration of the CRH-based Merkle tree over the list CMList of coin commitments. (b) A coin c. (c) Illustration of the structure of a coin commitment cm. (d) Illustration of the structure of a coin serial number sn. Source:[Sasson et al., 2014]

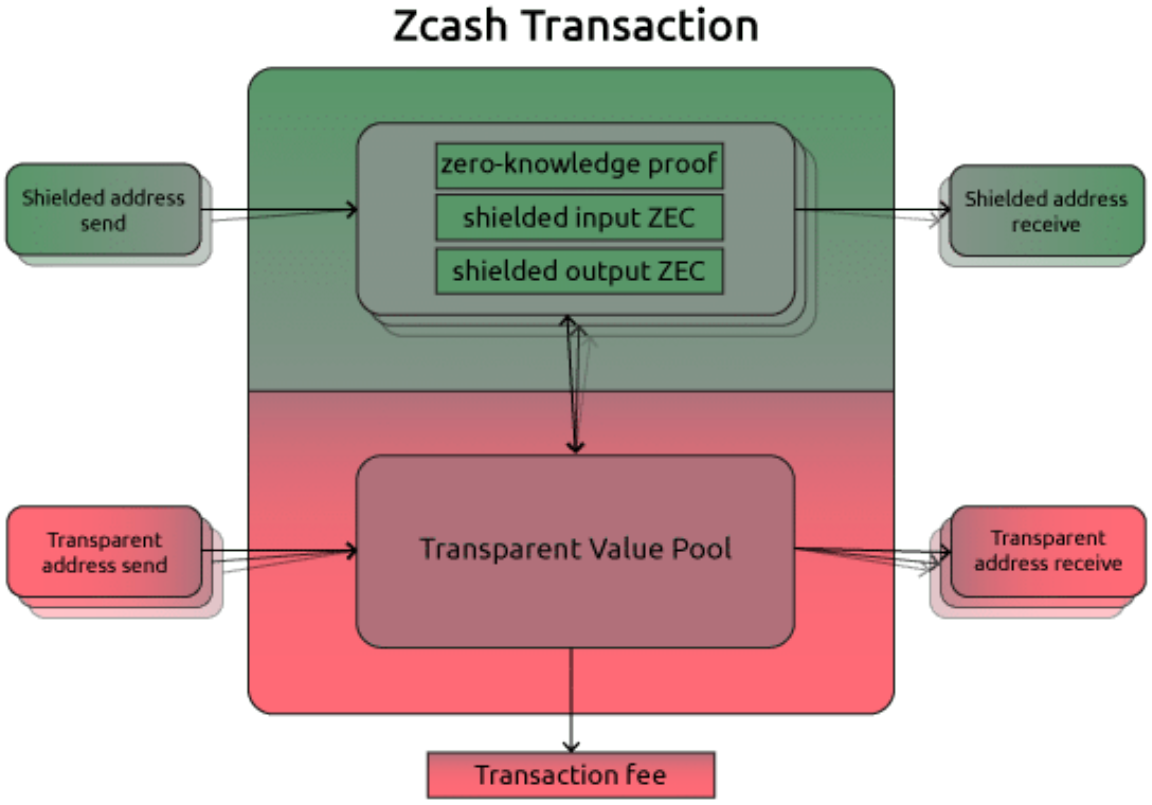


Figure 10: Anatomy of a ZCash Transaction : `https://electriccoin.co/blog/anatomy-of-zcash/`

Security

Their protocol construction satisfies three security guarantees:

- Ledger Indistinguishability: This property captures the requirement that the blockchain reveals no new information to the adversary beyond the publicly-revealed information, even when the adversary can induce honest parties to perform the Anonymous Payments of their choice. The public information available on the blockchin is of the kind values of minted coins, public values, information strings, total number of transactions, etc.
- Transaction non-malleability. This property requires that no bounded adversary A can alter any of the data stored within a (valid) pour transaction tx_{Pour}. This implies that malicious attackers cannot modify the contents of a POUR transaction before it is added to the blockchain
- Balance. This property requires that no bounded adversary A can own more money than what they minted or received via payments from others.

These three conditions are guaranteed by zeroCash, using relatively few assumptions (non-invertibility of cryptographic hash functions, existance of a trusted setup, etc). Furthermore, even if a trusted setup fails to generate unknown public parameters, the Ledger Indistinguishability is still maintained, but the other two properties fail. The new ZCash update - Sapling will target this vulnerability through using a new elliptical curve called JubJub.

Even then, there are a few considerations to keep in mind to guarantee anonymity. Firstly, Network traffic which is used to interact with the Blockchain (announce transactions, retrieve blocks, contact merchants etc) leaks information (IP Address, Geo-Timezone etc). A fix for this is using Tor [Dingledine et al., 2004] or Mix [Chaum, 1981] Networks. Moreover, a powerful adversary can simulate an additional block purely for a targeted

user, resulting in the loss of anonymity. But again, to combat this a user can wait a fixed number of blocks before interacting with recent blocks on the chain, or verify the existance of new blocks with trusted parties.

Privacy Concerns

An issue with z-cash is, it allows both shielded and unshielded transactions. However regulatory agencies (Government) disincentivize using the shielded transactions. According to CoinTelegraph (reference link) only about 6% of transactions are completely shielded and only about 15% are shielded in some form. These numbers indicate the due to multiple reasons (regulations to showcase origin of funds at exchanges, most wallets not supporting anonymous addresses etc) there is a long way to go before privacy solutions become mainstream.

Monero

Unlike Z-Cash, which has selective shielding, Monero [Alonso et al., 2020] is a cryptocurrency where every user is anonymous by default. This is done through the use of three important techniques:

Stealth Addresses

Stealth addresses are an important part of Monero's inherent privacy. They allow and require the sender to create random one-time addresses for every transaction on behalf of the recipient. The recipient can publish just one address, yet have all of his/her incoming payments go to unique addresses on the blockchain, where they cannot be linked back to either the recipient's published address or any other transactions' addresses. By using stealth addresses, only the sender and receiver can determine where a payment was sent.

When you create a Monero account you'll have a private view key, a private spend key, and a Public Address. The spend key is used to send payments, the view key is used to display incoming transactions destined for your account, and the Public Address is for receiving payments. Both the spend key and view key are used to build your Monero address. Moreover, you can decide who can see your Monero balance by sharing your view key. Monero is private by default and optionally disclosure compliant.

Ring Signatures

In cryptography, a ring signature is a type of digital signature that can be performed by any member of a group of users that each have keys. Therefore, a message signed with a ring signature is endorsed by someone in a particular group of people. One of the security properties of a ring signature is that it should be computationally infeasible to determine which of the group members' keys was used to produce the signature.

In Monero, a ring signature makes use of account keys and a number of public keys (also known as outputs) pulled from the blockchain using a triangular distribution method. Over the course of time, past outputs could be used multiple times to form possible signer participants. In a "ring" of possible signers, all ring members are equal and valid. There is no way an outside observer can tell which of the possible signers in a signature group belongs to your account. So, ring signatures ensure that transaction outputs are untraceable. Moreover, there are no fungibility issues with Monero given that every transaction output has plausible deniability (e.g. the network can not tell which outputs are spent or unspent).

Ring Confidential Transactions

RingCT [Noether, 2015], short for Ring Confidential Transactions, is how transaction amounts are hidden in Monero.

Ring CT was implemented in block #1220516 in January 2017. After September 2017, this feature became mandatory for all transactions on the network.

RingCT introduces an improved version of ring signatures called "A Multi-layered Linkable Spontaneous Anonymous Group signature", which allows for hidden amounts, origins and destinations of transactions with reasonable efficiency and verifiable, trustless coin generation.

Privacy Concerns

The internet traffic to Monero nodes is not hidden, so anyone monitoring your internet can see that you are using Monero and think that you have something to hide. This is scheduled to be fixed by implementing Invisible Internet Project (I2P). The project, called Kovri, protects users at the protocol level.

Another interesting result was that for transactions between 2014-2016, about 62% have been successfully linked [Miller et al., 2017]

Comparisons Between Coins

As we notice, not just the two privacy coins we covered, but many others like Dash, Verge, Horizen etc. are also forks of Bitcoin. In contrast, there exist very few privacy coin forks for Ethereum (Zether is one of the only ones).
There are multiple reasons for this, the primary being UTXO based cryptocurrencies (bitcoin) already have some level of privacy in-built compared to Account based blockchains (Ethereum) since in a UTXO based chain, the receiver of the transaction is unknown until the coins are spent. Hence, already there is a basic level of privacy. Moreover, in Ethereum, it is harder to implement Layer-1 solutions for privacy (in forks) since specifying a receiver account is essential by the design of the blockchain. Furthermore, Layer-2 solutions are popular and scalable on Ethereum, and hence more developers are incentivized to develop the same for Ethereum based cryptocurrencies.

A major difference between ZCash and Bitcoin (apart from the privacy features) are the distribution of mining rewards. While miners receive 80% of a block reward, 20% is given to the "Zcash development fund": 8% to Zcash Open Major Grants, 7% to Electric Coin Co., and 5% to The Zcash Foundation (`https://z.cash/zcash-development-and-governance/`).

Monero on the other hand, differs from bitcoin in two major (non-privacy) aspects. Monero is ASIC resistant, which leaves CPU mining as the only option. Furthermore, Monero incorporates a variable block size to handle high transaction volume periods.

The major difference between ZCash and Monero is the optional shielding available in ZCash whereas Monero shields every transaction. We can see the impact this has, since only about 10% of transactions on the ZCash network are shielded in any way. When we look at the daily numbers, this comes to 500 transactions per day being shielded. In contrast, Monero had about 12k transactions everyday, all of which were shielded `https://bitinfocharts.com/`.

Tornado Cash

Tornado Cash [Pertsev et al., 2019] is a coin mixer / tumbler, which aims to hide transactions on a blockchain by mixing them with multiple other transactions, such that each transaction's links are untraceable. Tornado Cash is a non-custodial solution which allows users to make their Ethereum of other ERC-20 tokens private by sending them through this smart contract.

After sending a deposit to the Tornado Cash smart contract the coins can be withdrawn to a new Ethereum address. This process ensures that the withdrawn funds can't be linked to the deposit source, thus ensuring the privacy and anonymity of the assets. Unlike some other tools meant to create privacy the Tornado Cash protocol is fully owned and governed by its community. This was accomplished in May 2020, when the development team relinquished control over the protocol's multi-signature wallet in a Trusted Setup Ceremony. Now it can be considered to be fully decentralized, assuming atleast one honest user participated in the Trusted Setup.

Moreover, there is a governance token associated with the project. The TORN token is an ERC-20 token with a fixed supply of 10 million tokens. Holding the TORN token gives a user the ability to submit proposals and to vote on protocol changes. In addition, the users of Tornado Cash accrue Anonymity Points as they interact with the protocol. These are deposited to a shielded account, and once enough are accumulated they can be converted to TORN tokens in a unique process known as Anonymity Mining.

Deposit

To deposit a coin, a user proceeds as follows:

1. Generate two random numbers $k, r \in \mathbb{B}^{248}$ and computes $C = H_1(k \| r)$

   Here H_1 is the Pedersen Hash function which maps a sequence of bits to a compressed point on an elliptic curve. [Ref: Iden3 Documentation]

   We call k the nullifier and r the randomness.

2. Send Ethereum transaction with N ETH to contract $\mathcal{C}$ with data C interpreted as an unsigned 256 -bit integer. If the tree is not full, the contract accepts the transaction and adds C to the tree as a new non-zero leaf.

Withdrawal

To withdraw a coin (k, r) with position l in the tree a user proceeds as follows:

1. Select a recipient address A and fee value $f \leq N$;
2. Select a root R among the stored ones in the contract and compute opening $O(l)$ that ends with R.

   Here $O(l)$ is the value value of sister nodes on the way from leaf l to the root in Merkle Tree.

3. Compute nullifier hash $h = H_1(k)$.

   Here H_1 is again the Pedersen Hash function.

4. Compute proof P by calling Prove on d_p.

   Here, the statemnt of Knowledge is the follows :

$$\mathcal{S}[R, h, A, f, t] = \{\text{I KNOW } k, r, l, O \text{ SUCH THAT } h = H_1(k) \\ \text{AND } O \text{ is the opening of } H_2(k \| r) \text{ at position } l \text{ to } R\}$$

   Here H_2 is the MiMC hash function [Albrecht et al., 2016]. We can construct a zk-SNARK for the above statement of Knowledge which will result in (d_p, d_v) which are the Proof Constructor and the Proof Verifier.

5. Perform the withdrawal in one of the following ways:

   - Send an Ethereum transaction to contract $\mathcal{C}$ supplying R, h, A, f, t, P in transaction data.
   - Send a request to Relayer supplying transaction data R, h, A, f, t, P. The Relayer is then supposed to make a transaction to contract $\mathcal{C}$ with supplied data.

The contract verifies the proof and uniqueness of the nullifier hash. In the successful case it sends $(N - f)$ to A and f to the Relayer t and adds h to the list of nullifier hashes.

Smart Contracts

Let $\mathcal{C}$ be the smart contract that has the following functionality:

- It stores the last $n = 100$ root values in the history array. For the latest Merkle tree $\mathcal{T}$ it also stores the values of nodes on the path from the last added leaf to the root that are necessary to compute the next root.
- It accepts payments for N ETH with data C The value C is added to the Merkle tree, the path from the last added value and the latest root is recalculated. The previous root is added to the history array.
- It verifies the alleged proof P against the submitted public values (R, h, A, f, t). If verification succeeds, the contract releases $(N - f)$ ETH to address A and fee f ETH to the Relayer address t.
- It verifies that the coin has not been withdrawn before by checking that the nullifier hash from the proof has not appeared before and if so, adds it to the list of nullifier hashes.

Privacy Considerations

Tornado cash is only able to create anonymity sets for 4 discrete deposit amounts (0.1, 1, 10, 100) ETH. And the anonymity sets for these are in the order of 20,000 Each. However, each denomination has a separate set, hence a certain lull of traffic on one particular denomination can lead to a loss/reduction of privacy.

Tornado cash also requires a trusted setup assumption, which assumes atleast one party which helped generate the randomness is honest. Even though this is not an extreme assumption, it leaves the protocol vulnerable to attack.

Tornado cash allows the ETH to be withdrawn into an account directly (as long as the acceptor has enough ETH to cover gas fees) or a new ETH wallet as well (though relayer nodes). This ensures an additional level of privacy.

As per the creators of Tornado cash, the users need to wait between a deposit and the withdrawal to ensure anonymity (as it increases the anonymity set). However, this poses potential concerns when we are looking at instantaneous applications of privacy, which is required for many scaling scenarios of cryptocurencies.

The Current Cost of Privacy

Average Transaction Fee

Before we start, some information about the ZCash and Monero Transactions. The transaction size for ZCash : unshielded, 500 bytes; shielded, 2,000 bytes [Ref] . In contrast a Monero Transaction is 2380 bytes [Ref] . In contrast the Bitcoin Transaction with 1 input and 2 outputs (one to self one to sender) is approximately 250 Bytes [ref]

The average fees per Bitcoin transaction is 20 USD [ref] . For ZCash, the per transaction fee is roughly 0.00001 ZEC which is roughly 0.0015 USD. Similarly for Monero, the fees is about 0.05 USD.

Hence, if a Monero or ZCash transaction was to occur in Bitcoin (being 8-9 times as large as a Bitcoin Transaction) it's fees would be 150-180 USD. (Which is 10^5 as high as ZCash and 3600 as high as Monero's.

Here we see that despite having significantly more privacy considerations, both ZCash and Monero are cheaper than their alternative (Bitcoin).

In tornado cash, being part of the EVM, we will showcase the fees in terms of gas [Tornado forum] A deposit transaction takes about 1m gas (0.05 ETH at 50 GWEI, 0.1 ETH at 100 GWEI). A withdrawal transaction takes about 400k gas ((0.02 ETH at 50 GWEI and 0.04 ETH at 100 GWEI). On top of this, there is a withdrawal relayer fee of 0.05% - 0.2%. On top of this there is the gas that the relayer uses for their transaction. All in all, assuming the smallest possible gas fee, and no relayer charges, we get a cost of about 270 USD. This is significant compared to the cost of a simple Ethereum transaction (which requires 21,000 gas - 0.001 ETH at 50 GWEI, 0.002 ETH at 100 GWEI) which is about 4-8 USD per transaction. Hence, we see a 3000% increase in the cost of privacy for Tornado Cash.

Block Visualization

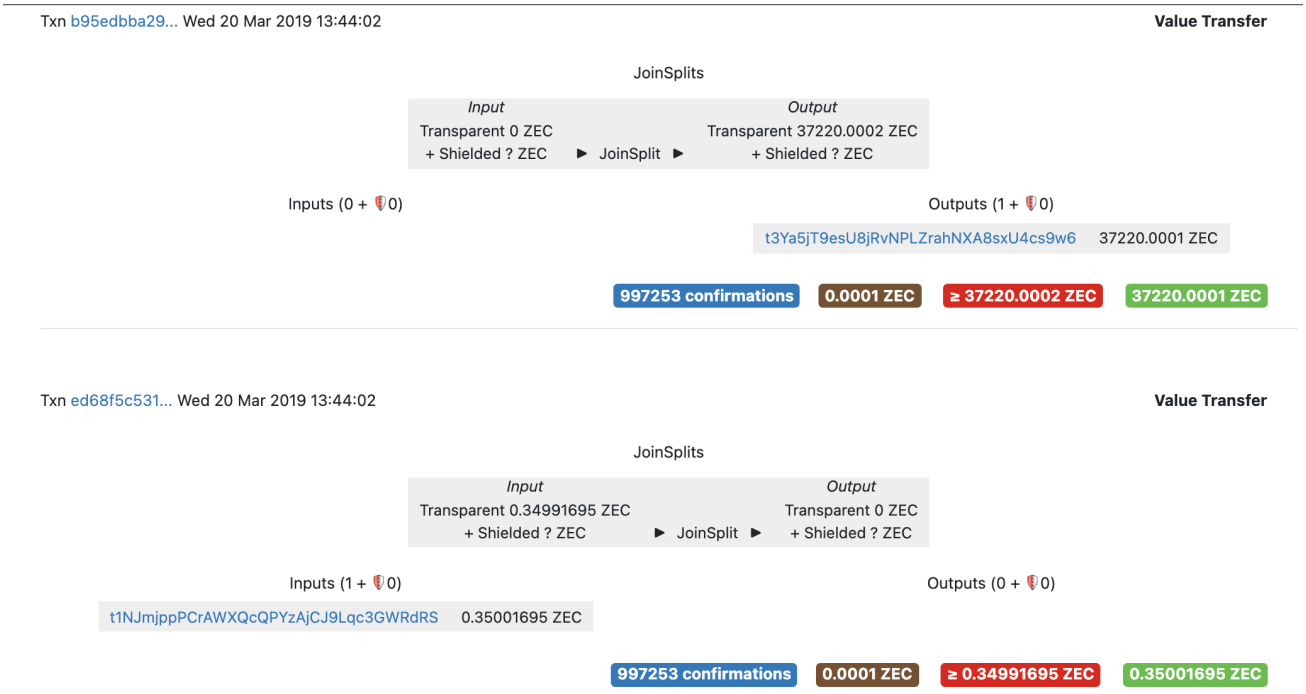


Figure 12: ZCash Explorer : https://explorer.zcha.in/

Here we can see that in any ZCash explorer, any shielded transaction showcases two values (Transparent/Shielded) for both the input and output of a transaction. Only the Transparent components have their values displayed.

Transactions (56)

Hash	Output total	Fee	Bytes
ed27dc60cb44f3f925b61a30d67db8ebd38315466eaf3a75db289bada4ee0469	confidential	0.000339520000	1455
99e10d2ba26b69588e7b5ba6d1946f01bab359b104fc570e819e35c0d6eb63c1	confidential	0.000249850000	1970
9b1e0e07b88c321d8ef534cc7542d2f8d73269ff88b75142ccd630ed77f3c9de	confidential	0.000058140000	1449
f69880cfb3439b16ebeffca7bac02db0edc7fab04b722cae1c3caea88c58ecc5	confidential	0.001045815000	27730
24a4b85a5ed2d737ab79ba76e912a5c543c2bece2f51882debd758cea09ab866	confidential	0.000038760000	1451
a715929080591d2fb0ea9ae467270ca75b6d6f6115f0282cf82190eae9dd5004	confidential	0.000038760000	1454
92c5caf42143adeb482bf17bc76de590d1c850b476c6ead081677ec16007f562	confidential	0.000050230000	1984
46d89c181424b70f2f7bcf0e62201ca346d1ffa15d0ea3db1dc5b21500376d2d	confidential	0.000013590000	1452
31cd20119f848d92092743500addfb7f3bd9ab9f8d729d452cb2056b1c8e6a1e	confidential	0.000013590000	1457
df7d0e6b3d701c0a6d5664b14ea29b563ebb3e623cc62bb20dca624f8c87f437	confidential	0.000007380000	1453
66124d8aa343ad05e4ef11df40ff85f5f2b6b91def96e293d8af183ca2f89433	confidential	0.000007380000	1453

Figure 13: Monero Explorer : https://localmonero.co/

In Monero's explorer, things are even more hidden with only the transaction fees being displayed. Moreover, each transaction has a value of MixIns which are the number of users who have signed the RingCT. (From a quick check, it seems to be around 10 for most recent blocks)

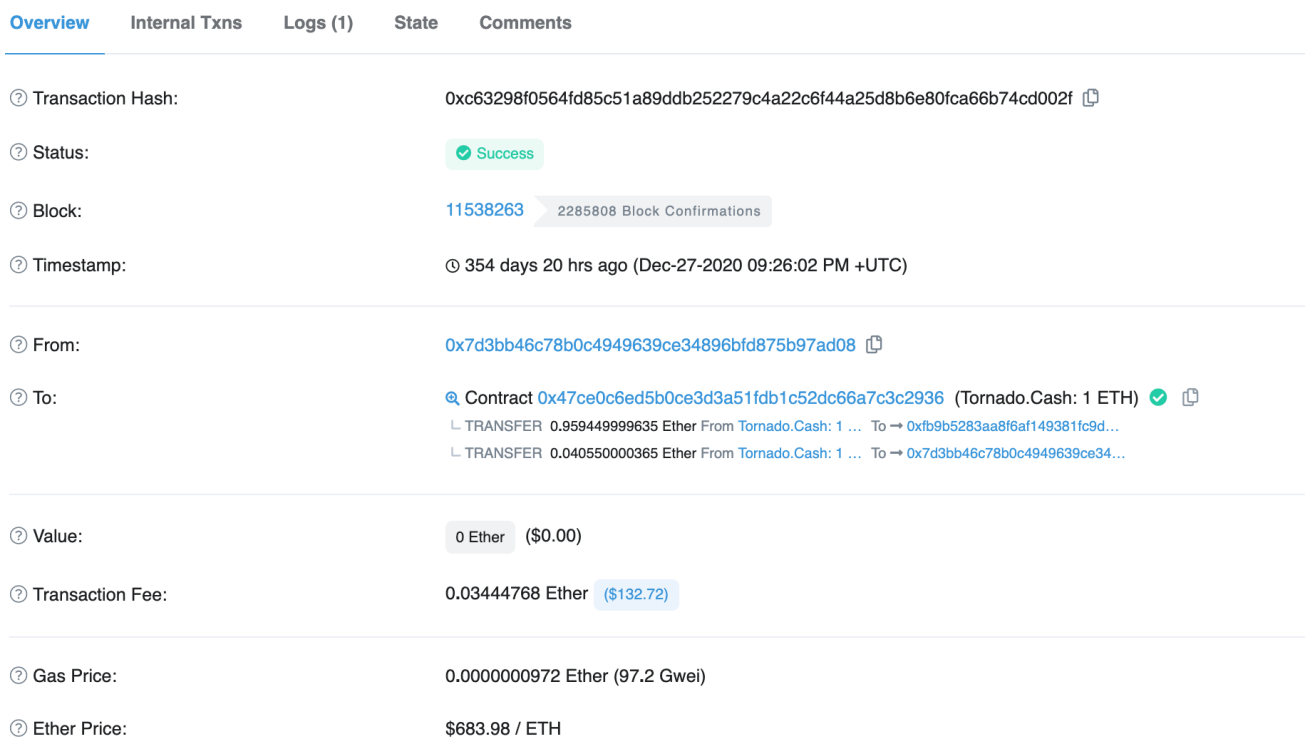


Figure 14: Tornado Cash Explorer : https://etherscan.io/

In Tornado cash, there are 4 separate smartcontracts, one for each denomination the tumbler supports. For every transaction, the only visible components are a user sending money to this contract, and the contract sending money to another user. As shown, through a relayer node, the gas fee and other components are also handled by them.

The Forever War

CryptoWars

Law enforcement has always wanted a secret key to unlock doors, both digital and physical. With every increase in capabilities for privacy and security, we see an outcry that this will only increase criminal activity; in the same breath, they say "if you have nothing to hide, you should feel comfortable revealing your information." In the digital world, since the 1970s, law enforcement has been fighting to have increased centralized power to audit and monitor its people in order to "catch the bad guys," as cryptographers have fought to increase privacy and security. These legal, technological, and moral debates have been lovingly termed the CryptoWars [Wikipedia, 2021a].

KYC laws

As the development of blockchain technologies emerges, governments seek to create processes for tracking assets and catching criminals, as new markets emerge. One of the biggest tools in the financial quadrants of governments is Know-Your-Customer laws (KYC). KYC laws are ethical requirements within the investment and financial services industry, to ensure that a customer is who they claims to be. Typically, KYC laws require the collection of social security numbers, date-of-birth, address, and sometimes ID documents like a passport [Chen, 2021].

Selective Disclosure

KYC laws are an important part of a regulated marketplace, and if blockchains are to expand, they need to be able to comply with laws. Further drawing inspiration from financial markets, zk-SNARK-based privacy coins can *selectively disclose* information to regulators and auditors. At a high-level, this simply means sharing secret information with auditors and regulators that remove the *Zero-Knowledge* component of zk-SNARKs. If we recall how Zero Knowledge is implemented in zk-SNARKs, it is through a prover selecting a random value (q, though sometimes denoted by δ), and not sharing it with anyone; the random value does not change anything about the correctness of the polynomial equality, but it does prevent the verifiers from learning the actual coefficients of the polynomials. And so, a prover that generated a zk-SNARK is selectively disclose information to regulators/auditors by sharing the random value with them, and removing ZK from their zk-SNARKs.

The Future

If every time Apple paid a VR company any amount of money, the whole world knew, corporate secrets would scarcely exist. As money makes the world go round, it seems highly unlikely that cryptocurrencies will remain a public transaction. We do not expect that either the blockchain or the demand for privacy on the blockchain will diminish. As cryptographers and computer scientists we sit in a unique position to develop mathematical tools and levers for ordinary citizens to protect themselves for exploitation. And as blockchain experts, we sit in a unique position to empower the financial means of its users in an equitable and trustless fashion.

Conclusion

In this report we will not make a judgement call about which side you should fall on in the CryptoWars or the PrivacyWars. But from the duration and vigor that the CryptoWars have continued with, it seems clear that both 1. reasonable people can disagree, 2. there will never be a perfect solution. However, we do remark that exploitation and abuse-of-power by powerful, central bodies is always possible. We also note that without perfect knowledge of each person, it is impossible to only let the "right" people have privacy and security. And so, while developing tools and levers to protect the privacy and security of people will never be without nuance, it will always be a tool that makes it possible for ordinary people to stand up to such abuses.

We conclude this report with quote from Dr. Eran Tromer (Co-founder of ZCash), who said in an interview with us, "I think that a world devoid of financial privacy is scary, deplorable and unlikely. Cryptocurrencies and decentralized finance will not fulfill their promise until you can buy a book without broadcasting your transaction on-chain. Therefore, for the DeFi vision to happen in earnest and with widespread adoption, privacy solutions must become ubiquitous. Our job, as cryptographers, is to help ensure it's the safe and secure privacy solutions that get used. As for Bitcoin, specifically: for the aforementioned reason, it would have to either evolve to provide privacy (the recent Taproot protocol upgrade is a tiny step in that direction), or more likely: become a reserve asset that people hesitate to directly transact in because who's crazy enough to "tweet" their account balances and transaction, but that can be bridged, tokenized and securitized to other platforms that do provide privacy." [Tromer, 2021]

Appendix

Here is a list of the resources that we found particularly helpful and relevant:

Theory, Primers, and Explainers

- Why and How zk-SNARK Works by Maksym Petkus
- zkSNARKs in a nutshell — Ethereum Foundation Blog
- A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography : https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/
- Deep Dive in to Bellman Library
- Trilogy Tutorial by Vitalik Buterin. (All linked from `https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6`.)
  - Quadratic Arithmetic Programs: from Zero to Hero
  - Exploring Elliptic Curve Pairings
  - ZK SNARKS : Under the Hood

Code resources

- Libsnark: `https://github.com/scipr-lab/libsnark`
- Bellman: `https://github.com/zkcrypto/bellman`
- WASM SNARK: `https://github.com/iden3/wasmsnark`
- SNARKjs: `https://github.com/iden3/snarkjs`

Bibliography

[Albrecht et al., 2016] Albrecht, M., Grassi, L., Rechberger, C., Roy, A., and Tiessen, T. (2016). Mimc: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 191–219. Springer.

[Alonso et al., 2020] Alonso, K. M. et al. (2020). Zero to monero.

[Ben-Sasson et al., 2018] Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. (2018). Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. ePrint Arch.*, 2018:46.

[Bowe et al., 2019] Bowe, S., Grigg, J., and Hopwood, D. (2019). Recursive proof composition without a trusted setup. *Cryptol. ePrint Arch., Tech. Rep*, 1021:2019.

[Bünz et al., 2020] Bünz, B., Fisch, B., and Szepieniec, A. (2020). Transparent snarks from dark compilers. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 677–706. Springer.

[Chaum, 1981] Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90.

[Chen, 2021] Chen, J. (2021). Know your client (kyc).

[Chiesa et al., 2020a] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., and Ward, N. (2020a). Marlin: Preprocessing zksnarks with universal and updatable srs. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 738–768. Springer, Cham.

[Chiesa et al., 2020b] Chiesa, A., Ojha, D., and Spooner, N. (2020b). Fractal: Post-quantum and transparent recursive proofs from holography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 769–793. Springer.

[Damgård, 1991] Damgård, I. (1991). Towards practical public key systems secure against chosen ciphertext attacks. In *Annual International Cryptology Conference*, pages 445–456. Springer.

[Dingledine et al., 2004] Dingledine, R., Mathewson, N., and Syverson, P. (2004). Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC.

[Gabizon, 2018] Gabizon, A. (2018). Zk0x02 - an intro to zcash and zksnarks - ariel gabizon (zcash).

[Gabizon et al., 2019] Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *IACR Cryptol. ePrint Arch.*, 2019:953.

[Gennaro et al., 2013] Gennaro, R., Gentry, C., Parno, B., and Raykova, M. (2013). Quadratic span programs and succinct nizks without pcps. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 626–645. Springer.

[Groth, 2010] Groth, J. (2010). Short pairing-based non-interactive zero-knowledge arguments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 321–340. Springer.

[Mannak, 2019] Mannak, R. (2019). Comparing general purpose zk-snarks. *Medium*.

[Meiklejohn et al., 2013] Meiklejohn, S., Pomarole, M., Jordan, G., Levchenko, K., McCoy, D., Voelker, G. M., and Savage, S. (2013). A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140.

[Miller et al., 2017] Miller, A., Möser, M., Lee, K., and Narayanan, A. (2017). An empirical analysis of linkability in the monero blockchain. *arXiv preprint arXiv:1704.04299*.

[Narayanan et al., 2016] Narayanan, A., Bonneau, J., Felten, E., Miller, A., and Goldfeder, S. (2016). *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press.

[Noether, 2015] Noether, S. (2015). Ring signature confidential transactions for monero.

[Pankova, 2013] Pankova, A. (2013). Succinct non-interactive arguments from quadratic arithmetic programs. Technical report, Technical report, University of Tartu, Cybernetica AS.

[Parno et al., 2013] Parno, B., Howell, J., Gentry, C., and Raykova, M. (2013). Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy*, pages 238–252. IEEE.

[Pertsev et al., 2019] Pertsev, A., Semenov, R., and Storm, R. (2019). Tornado cash privacy solution version 1.4.

[Petkus, 2019] Petkus, M. (2019). Why and how zk-snark works.

[Sasson et al., 2014] Sasson, E. B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., and Virza, M. (2014). Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE.

[Silfversten et al., 2020] Silfversten, E., Favaro, M., Slapakova, L., Ishikawa, S., Liu, J., and Salas, A. (2020). *Exploring the use of Zcash cryptocurrency for illicit or criminal purposes*. RAND Corporation, Santa Monica, CA.

[Tromer, 2021] Tromer, E. (2021). Email interview with eran tromer. *Roy Rinberg*.

[Tuan Phan, 2021] Tuan Phan, Founder, Z. F. L. (2021). Did the fbi hack bitcoin? deconstructing the colonial pipeline ransom.

[Wikipedia, 2021a] Wikipedia (2021a). Crypto wars.

[Wikipedia, 2021b] Wikipedia (2021b). Zero-knowledge proof.