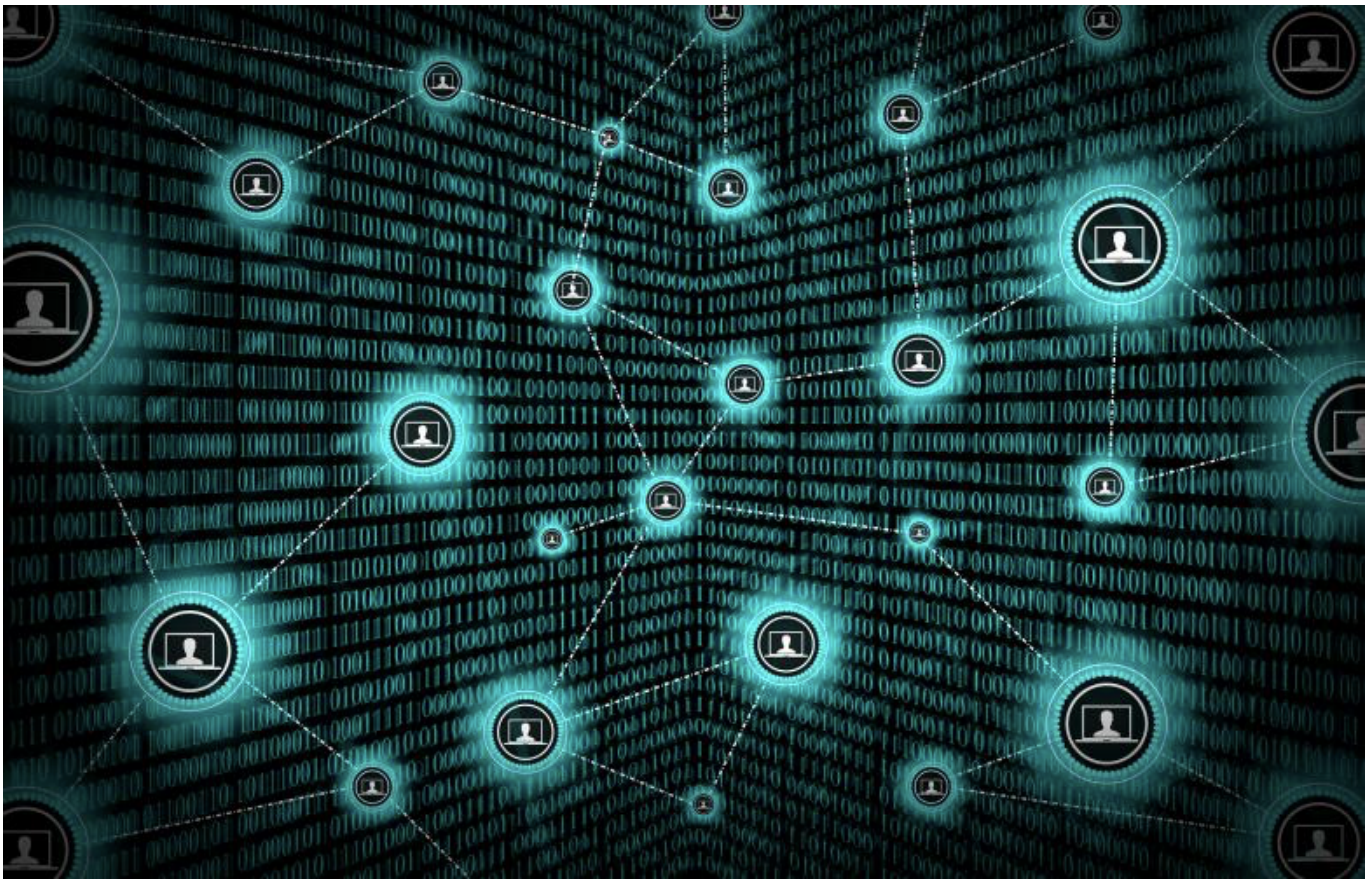
zk-SNARKs — A Realistic Zero-Knowledge Example and Deep Dive

Adam Luciano Follow
Jun 9, 2018 · 11 min read



Welcome back! Let's dig deeper into zk-SNARKs.

Note: unless you have familiarity with zk-SNARKs, I would suggest reading Part 1 of this blog series.

Quick recap: There are two primary parties in a zk-SNARK, a prover and a verifier. Using a zk-SNARK a prover wants to assure a verifier about a statement of knowledge, without

revealing what that exact knowledge is to the verifier.

Discover and review best Blockchain softwares

What is a zk-SNARK?

Prior to walking through a more technical example, let's walk through what properties a zk-SNARK must satisfy first:

1. **Completeness**: if the statement is true, and the verifier and prover are honest, the proof is accepted.
2. **Soundness**: if the statement is false, a cheating prover cannot convince an honest verifier that it is true, except with some tiny probability.

In addition to the above, a zk-SNARK (zero-knowledge Succinct Non-Interactive ARgument of Knowledge) needs to be:

1. **Zero-knowledge:** if the statement is true, a verifier does not learn anything beyond the fact that the statement is true.
2. **Succinct**: The size of the proof needs to be small enough to be verified in a few milliseconds.
3. **Non-Interactive**: Only one set of information is sent to the verifier for verification, therefore there is no back and forth communication between the prover and verifier.
4. **ARgument**: A computationally sound proof: soundness holds against a prover that leverages polynomial-time, i.e. bounded computation.
5. **of Knowledge**: The proof cannot be constructed without access to the witness (the private input needed to prove the statement).

Given the above conditions, let's walk through an example of a zk-SNARK and also incorporate how it would be used in combination with a blockchain. Using the first example between Bob and Alice in post #1: Alice wants to transfer money from one of her bank accounts to another and needs Bob to initiate the transfer — let's describe how that scenario can be implemented using zk-SNARKs and a blockchain.

Note: The below example is not scalable, but provides a good overview of how a zk-SNARK can be approached.

Four actors in our scenario:

Alice — Prover

2.

Acme Bank — Verifier, Trusted third party for bank account attestation transaction, & a trusted third party for zk-SNARK setup

3.

Bob — Acme Bank call center representative

4.

Self-Sovereign Identity Solution — Data provider of digital identity information to Acme Bank App, and a trusted third party for zk-SNARK setup

Below is a diagram example of how Alice can transfer money from one of her bank accounts to another, without providing Bob at Acme Bank information to authenticate her identity:

1. Create a smart-contract enabled blockchain, for this example let's use an Ethereum instance.

2. A program is created that returns a '1' if the program evaluates to true (program example at end of blog post).

2a. The inputs include public inputs that are used by multiple parties (Alice, (Self-Sovereign Identity Solution), and Acme Bank), and private inputs (the secrets, aka witness) that only the prover (Alice) provides.

2a_i. Public: Name, Phone Number, Blockchain Digital Identity Address, Bank attestation of Alice being a current bank account owner — completed recently; timeout of 1 hour (to be described further below)

2a_ii. Private (Alice or Self-Sovereign Identity Solution): Social Security Number (SSN), Date of Birth (DOB)

2a_iii. Private (Alice only): Secret Pin

b. Setup the zk-SNARK parameters and generate prover and verifier keys.

2b. Use multi-party computation (MPC) between Self-Sovereign Identity Solutionand Acme Bank to generate the public parameters to setup the zk-SNARK. MPC allows for multiple parties to participate in the trusted setup phase to generate the parameters for the zk-SNARK — this minimizes risk vs. one party creating the keys. If one of the users who participates in the trusted setup phase is not compromised, the parameters generated are secure. If all of the users who participated in the setup are compromised, the parameters can be used to generate false proofs, and thus eliminates the value of the zk-SNARK process.

2c. Verifier keys and the verifier program are embedded in a smart contract that is instantiated in the Ethereum blockchain instance.

3. Alice open up Acme bank's app, uses her fingerprint to authenticate her identity to the app, and then enters the app.

3a. She goes to the 'Contact Us' area in the app and taps on the 'Call Acme with Prior Authentication' button.

3a_i. After Alice taps on the 'Call Acme with Prior Authentication' button, behind the scenes: 1) A request is sent from Alice's Acme bank app to Acme Bank asking if Alice is still a bank customer. 2) Acme Bank then sends a transaction to the blockchain stating that Alice is still a current customer.

3a_ii. Alice is asked the following questions in the mobile app (some of these questions (social security number and date of birth) can be answered via a digital identity application providing the data to the Bank app, like Self-Sovereign Identity Solution):

3a_ii_1. What is your social security number? Alice enters: 123–45–6789

3a_ii_2. What is your date of birth? Alice enters: 1/1/2000

3a_ii_3. What is your secret pin? Alice enters: 7327; Why a secret pin? This is in case someone gains access to the app somehow and utilizes the digital identity application to populate the answers for the social security number and the date of birth, and then tries to interact with the Bank as Alice while not actually being Alice. The secret pin acts as a last hurdle that cannot be known by an outsider to validate one's identity.

4. Alice's answers (private inputs aka witness) are merged with the public inputs of Alice's name, phone number, public digital identity address, and bank's attestation that Alice is a current customer to generate a zero-knowledge proof (zk-SNARK).

4a. The proof is sent to the verification smart contract in the blockchain.

5. The smart contract returns 'true', which Acme Bank is monitoring and logs the response of 'true' in their own internal system.

6. A button in Acme's mobile app is now available for Alice to call Acme Bank. Alice taps on the button, and then is transferred to an Acme Bank call representative, Bob to initiate the bank transfer.

The above example provides a breakdown of how a zk-SNARK could work when Alice wants to call Acme Bank to initiate a transfer from one of her accounts to another. The

above example eliminates Alice sharing sensitive identity information with Bob, and additionally reduces average handle time for Acme bank, thus reducing costs — a win-win for both Alice and Acme Bank.

Third Party Trust and Proof of Current Customer

One key element in the above example of a zk-SNARK working in the real world is the bank attestation that Alice has an account with Acme Bank. This is a key element that is often overlooked when providing examples for zk-SNARKs — when there is a third party that needs to be involved with a zk-SNARK process, how is that incorporated? There can be different approaches, however we can incorporate a form of proof of existence, that we will call Proof of Current Customer that allows the above example to become a more realistic implementation.

A Proof of Current Customer could work in the following way for Acme bank:

1. When Alice first becomes a customer or is already a customer, the bank would make an attestation that Alice is a customer of the bank by sending a transaction to Alice's public digital identity address in the relevant blockchain. (The transaction would contain an encrypted data payload that Alice's Acme Bank mobile app would read to indicate that Alice is a customer of Acme Bank.)

2. The issuer identity (Acme Bank) would be validated by checking Acme's public address. In the blockchain, there would be a public-address registry in a smart contract (the registry would store the public addresses and their associated identities), thus Acme Bank could be identified as the issuer of the original attestation. The registry could only be updated by certain selected owners, or the creator of the contract registry.

3. Next, a new transaction is sent from Acme Bank to Alice's public digital identity address with an encrypted data payload that Alice is a customer of Acme Bank, and she can call Acme Bank without prior authentication for the next 60 minutes.

Note: There are multiple ways to approach the above. For example, instead of using simple transactions, we could also utilize a document stored in a decentralized storage system that holds the relevant data, which can then be used to verify the underlying customer data and digital signature of Acme Bank.

In the above example, Alice was able to communicate with Bob at Acme Bank without sharing information about herself and authorize the transfer of money from one of her bank accounts to another. zk-SNARKs and blockchains can work really well together: bridging privacy, security, and transparency to exchange and verify information.

In addition to the above scenario, zk-SNARKs can be used in a wide array of areas, for example:

1. Verification of computation (centralized, decentralized)

2. Anonymous cryptocurrencies or cryptocurrencies that can enable use of zk-SNARKs in smart contracts, example: Zcash, Ethereum (protects user privacy)

3. Proof of provenance between public/private blockchains (a. Instead of recording all data of a transaction that occurs in a private blockchain onto a public blockchain, a proof can be stored on a public blockchain. This enables companies to keep their sensitive data secure, while proving provenance of a specific transaction.)

4. Authentication without passwords (as shown above in Alice and Bob example)

5. Sharing information about one's identity conditionally, for example: Alice is > 21 is true or false? (Age is not revealed), a zk-SNARK can be used to prove that Alice is over 21 while minimizing the trust needed between parties involved (ex: a. Sharing of PII, Health Data, Loan data)

Wider utilization of zk-SNARKs may change how data is stored. Perhaps companies may not need to keep as much data about their customers/users as they currently do. In the future, organizations could interact with blockchains and use zero-knowledge proofs to communicate with current processes, thus alleviating data leaks, further increasing privacy of user data, and reducing risk for organizations. Sharing of confidential data between people can also be further reduced. The applications of zero-knowledge proofs and self-sovereign identity will allow users to more fully protect their data, and further minimize data leakage as organization that own important aspects of one's identity data will start to decline.

A key area that can be further improved to minimize trust for users of the zk-SNARK is removal of the trusted setup phase. There is ongoing research in zero-knowledge

cryptography, with technologies such as zk-STARKS being developed that remove the trusted setup phase, further improving the process of using zero-knowledge cryptography to provide proof of knowledge. In the next post, I will dig into zk-STARKs.

I hope everyone enjoyed reading more about zk-SNARKs. If you have any questions about this post or the previous post in this series, please feel free to reach out to me directly: aluciano@cynapseblockchain.com.

In addition, if organizations have questions around ZK-tech or tokenomics, please feel free to review more how I help organizations at https://cynapseblockchain.com/

This post can also be found at Cynapse Blockchain Consulting's website: https://cynapseblockchain.com/2018/09/07/zk-snarks%e2%80%8a-%e2%80%8aa-realistic-zero-knowledge-example-and-deep-dive/

Deeper Dive of Alice and Bob Call with Prior Authentication Example — zk-SNARK program inputs and outputs only (example used ZoKrates):

For those of you who would be interested in what a basic zk-SNARK program looks like, I wanted to provide the following example based on the Alice and Bob example above.

1. zk-SNARKs work well with numbers, so we would need to convert the string inputs to numbers (used following website to convert the text strings: *https://cryptii.com/decimal-text*).

2. Public Name: Alice; Run through a Unicode decimal converter to attain: 65 108 105 99 101, remove all spaces and input: **6510810599101**

3. Public Phone Number: (555–666–7777); Run through a Unicode decimal converter to attain: 53 53 53 45 54 54 54 45 55 55 55 55, remove all spaces and input: **535353455454544555555555**

4. Public Blockchain Address: 0x4ef377462b03b750d52140c482394a6703d0d338; Run through a Unicode decimal converter to attain: 0 48 120 52 101 102 51 55 55 52 54 50 98 48 51 98 55 53 48 100 53 50 49 52 48 99 52 56 50 51 57 52 97 54 55 48 51 100 48 100 51 51 56, remove all spaces and input: **048120521011025155555254509848519855348100535049524899525650515752975455485110048100515156**

5. Public Bank Attestation Setup and Call: **a.** Initial Setup: Acme Bank sends a transaction to Alice's public digital identity address with a data payload indicating that Alice is a customer. This transaction would occur when Alice opens the account, or when Alice authenticates her account on Acme Bank's mobile app for the first time. **b.**When Alice kicks-off the 'Call with prior Authentication' process, the bank would check Alice's public digital identity account transactions for a transaction from Acme Bank indicating that Alice was a customer of Acme Bank. If found, Acme bank would then send a transaction to Alice's public digital identity address with an encrypted data payload that indicates that Alice is a current customer of Acme bank. Alice's phone would watch the blockchain for the new transaction from Acme Bank, and once found, would pull the data payload from the recent transaction, decrypt the data, and hash the data. The hashed data would then be turned into Unicode decimal string for input:
49665667686866675454525052495348527066485368517055505368654852505556654849686554556654525655526953566669697056526565555753665670

6. Private Social Security Number: 123–45–6789; entered as for input: **123456789**

7. Private Date of Birth:1/1/2000; entered as for input: **112000**

8. Private Secret Pin input (this is a secret pin that the user must remember): **7327**

9. The public and private inputs will be put into the below program, which will compare the public inputs with what is written in the zk-SNARK program. If they are correct, the program would then multiply the sum of the social security number and date of birth by the secret pin to generate the target number. If that number is the same as what is in the zk-SNARK program (905388517003), the program would return a 1. The code below written in ZoKrates provides an example:

ZoKrates code for zk-SNARK example

10. After the program executes, a proof is generated that looks like the image below:

zk-SNARK (zero-knowledge proof) output from ZoKrates

11. The above proof is then sent to the verification smart contract along with the public inputs (Alice's name, phone number, blockchain address, bank attestation hash, and expected output '1'). If all of the information is current, the contract would return 'true', with the output stored in the smart contract, so a watcher of the contract can see the results (Alice's mobile app and Acme Bank can see the state change within the contract). The verification contract has the verification key hardcoded inside it, so it can determine from the public inputs and the provided proof if the proof evaluates to true.

Zksnark Zero Knowledge Proofs Zero Knowledge Blockchain Ethereum

About Help Legal