WORLDQUANT UNIVERSITY

Compiled Content Module 5

MScFE 670

Data Feeds and Technology

Revised: 09/07/2019

Table of Contents

Module 5: Cryptocurrencies

Module 5 begins by exploring the different approaches to anonymity in blockchain technologies, first by looking at anonymity in the context of Bitcoin and then by introducing Zcash to demonstrate zero-knowledge proofs and their implications for anonymity. The module then introduces two smart contract platforms, Ethereum and EOS, and concludes by discussing their approaches to decentralized applications.

Unit 1: Is Bitcoin Anonymous?

Introduction

Welcome to Module 5. In this module, we will look at some specific cryptocurrencies that have harnessed blockchain technology. They aim to create thriving ecosystems of value, by allowing people (or even machines) to interact with each other financially with less friction. First, we are going to take a closer look at Bitcoin to understand exactly how anonymous it really is. Following this discussion, we will examine a cryptocurrency called Zcash, which increases anonymity even further. Thereafter, we will discuss two smart contract platforms, Ethereum and EOS. These two platforms are taking different approaches to allow people to create decentralized applications. In Modules 6 and 7, we will rely on this knowledge when we cover smart contracts and as we develop some of our own, unique, decentralized applications.

How anonymous are Bitcoin transactions?

Transactions and the exchange of value are extremely important to the healthy functioning of the economy. Whether people are buying their morning coffee, insurance, fuel, or almost anything else, transactions underpin our way of life. Each transaction signifies our willingness to exchange our money for something we perceive as valuable. This is why trade sanctions can be effective, and why an outage of a service like Visa can cause chaos[1].

Most retail transactions have taken place with cash, credit cards, or perhaps via a traditional EFT. We are starting to see more and more transactions facilitated via another medium:

[1] "Visa meltdown leaves Britain raging as cashless customers fume ...." 2 Jun. 2018, https://www.thesun.co.uk/money/6430640/visa-down-network-crashes-uk-europe-card-payments/. Accessed 7 Feb. 2019.

cryptocurrencies. These cryptocurrencies, like Bitcoin and Ethereum, reach their maximum throughput of transactions, and hundreds of new cryptocurrencies are being created. Recall that, as discussed in Module 4, a blockchain is essentially a clever mechanism for keeping a decentralized record of transactions, and **cryptocurrency** is a term for the "currency" that is being used in these respective transactions.

With many recent examples of personal data being targeted or being leaked, it comes as no surprise that people's concerns extend to the privacy of their cryptocurrency transactions. Furthermore, physical cash is, for the most part, anonymous and not traceable, so it is natural to expect the same from cryptocurrencies they may want to use. With this in mind, which cryptocurrency would you use to send a transaction you did not want anyone to know anything about? This exact question appeared on a CoinBase poll several months ago. Take a moment to consider the question yourself and perhaps formulate an answer. When you are ready, have a look at the results in Figure 1 below[2] and see how your thoughts compare with those of the general public.

Figure 1: Which would you use?

Interestingly, the results show that 36% of participants believe Bitcoin is the way to go for transaction anonymity, making it the most popular choice. In reality, Bitcoin is not the best

[2] These were the results at the time of the screenshot and, of course, may have changed.

cryptocurrency to use if you are looking for transaction anonymity. This is not to say it is worse than an EFT, for example, for privacy, just that some cryptocurrencies are built better to maintain privacy.

Blockchains are inherently public

As we know, Bitcoin is a *public* transaction ledger. This means that every Bitcoin transaction that has ever taken place can be viewed openly and easily by anyone in the world with a couple of clicks. Try this yourself. Visit https://blockchain.info/ and have a look. If we take a closer look at one of the blocks, and we pick a transaction, it will look something like this (see Figure 2):

Figure 2: Public transaction information

The above transaction indicates that 0.01428359 Bitcoin was sent from the public address 13xffWvwUGgCVVAYXMwtY7LesHAEn4DhMK to the public address 1NzE3UqrsemkmK u3NKnds8xjefPMJKYia8.

Pseudo-anonymity

This might seem quite anonymous, since we likely do not know who 13xffWvwUGgCV VAYXMwtY7LesHAEn4DhMK is, and to an extent this is the case. This is what we term **pseudo-anonymity**. Nothing about the public address, "13xffWvwUGgCVVAYXMwtY7LesHAEn4DhMK", itself helps us to identify who that person actually is. This is akin to you signing up to a social media platform and using some completely random username, say 8wrgn1o349. Nothing about your username directly identifies you. However, this is not all there is to the story.

Let's first look at a major implication of the pseudo-anonymity in many cryptocurrencies such as Bitcoin or Ethereum. If anyone is able to find out who exactly owns that public address, then they have the ability to easily go and find every transaction that has ever taken place with that public address. This means that one's entire transaction history can be scrutinized, something that many people would likely feel is a violation of their privacy.

How might someone find out about my public address? Well, for one, you could request payment from someone and thereby be required to provide them with a public address for payment. Just

like that, they know who you are, and could go peruse all your previous transactions that have taken place with your public address.

At this point, you might think: "Couldn't we just generate a new public address and ask for payment to be sent to that new address?" This is a possible workaround. However, a problem would arise when you send Bitcoin from this new address to any of your associated, public addresses. Any time a transaction occurs between two addresses, we can draw a link between them and possibly infer a relationship. This extends to a very powerful concept known as **transaction graph analysis**: we can begin to draw links between all possible Bitcoin transactions and start to infer various patterns and relationships. The FBI, for instance, used these techniques to find the founder of Silk Road, the notorious online illicit marketplace. Below is an example of a transaction graph.

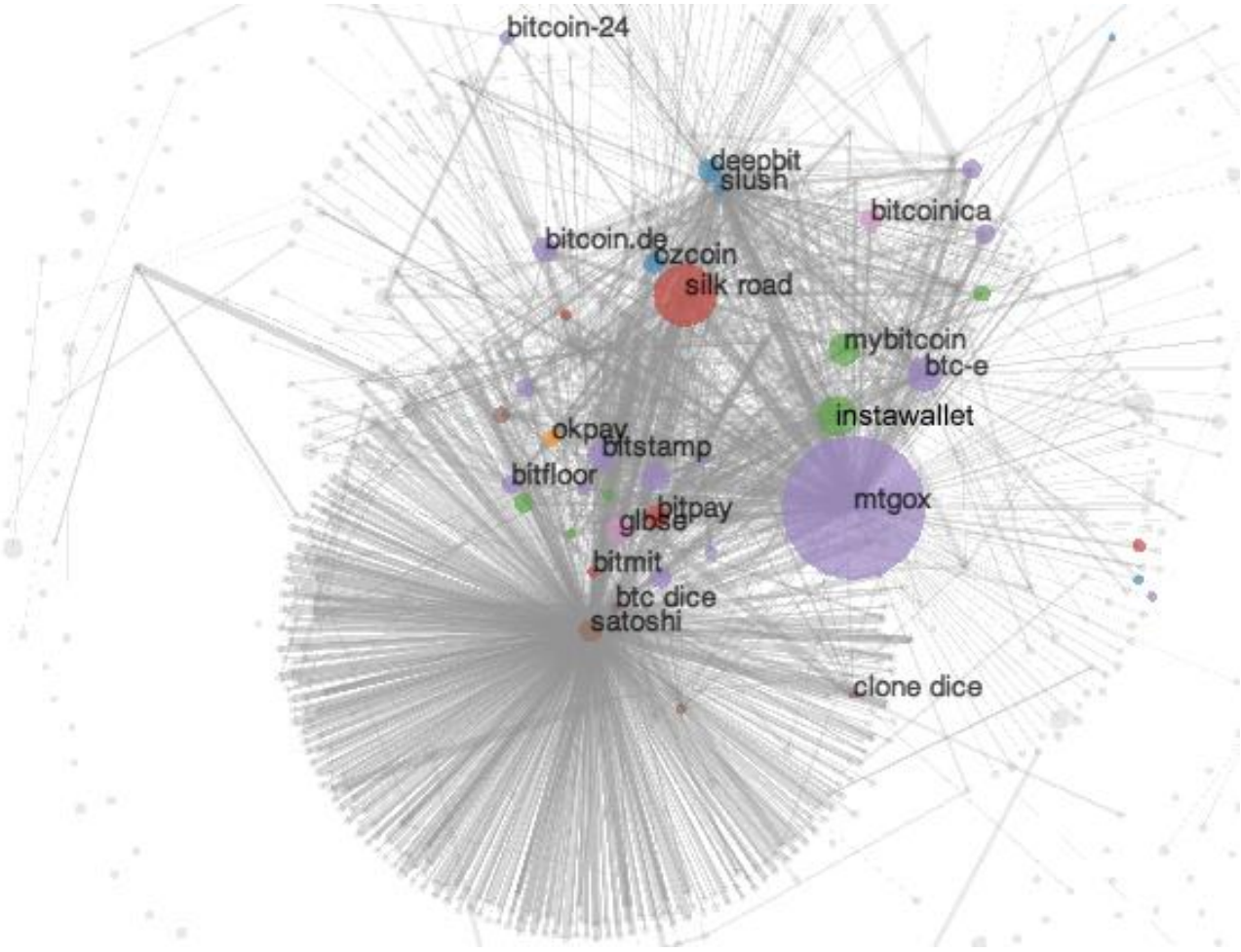


Figure 3: Example transaction graph

The lines represent the links between certain public addresses due to transactions, the nodes are Bitcoin addresses, and the lines are transactions, so you can see

how some addresses participated in many thousands of transactions. The really big circle we see is something called a **Satoshi dice**. A Satoshi dice can be understood as a gambling game where a user can send some Bitcoin to a public address with the hopes of winning some Bitcoin in return.

Given the public nature of public keys, we can conclude that Bitcoin does provide pseudo-anonymity to users. Yet, like any other pseudo-anonymous system (e.g. Facebook, Snapchat, Reddit, etc.), your interactions on the system have the potential to link your real-world identity to your pseudo-anonymous identity. Other cryptocurrencies have been created that aim to provide full anonymity. The two most popular are Zcash and Monero. Both are cryptocurrencies that aim to provide users with a completely anonymous transaction system. However, they use different techniques to achieve this. A key difference between them is that in Monero all transactions are anonymous, but in Zcash you have the option of making anonymous or public transactions. We will talk about Zcash in the next section.

Unit 2: Zcash

Introduction

In the previous section of notes, we described how some cryptocurrencies, like Bitcoin, may not be as anonymous as people believe. Now we will turn our attention towards Zcash, a cryptocurrency aiming to provide users with complete anonymity while still ensuring the safety and integrity of the blockchain.

What is Zcash doing differently to afford users anonymity? They are utilizing something known as zero-knowledge proofs. **Zero-knowledge proofs** allow network validators to prove that a transaction from one public address to another is a valid transaction – i.e. that the paying public address has enough funds – without revealing anything about the two public addresses or the amount in question. Therefore, even though we have a public transaction ledger, we can infer nothing at all from these transactions, as they all convey *zero-knowledge*.

How do zero-knowledge proofs work?

Recall that a zero-knowledge proof proves the validity of a transaction while conveying no knowledge about said transaction in the process. First, let us consider the situation as illustrated in Figure 4 below.

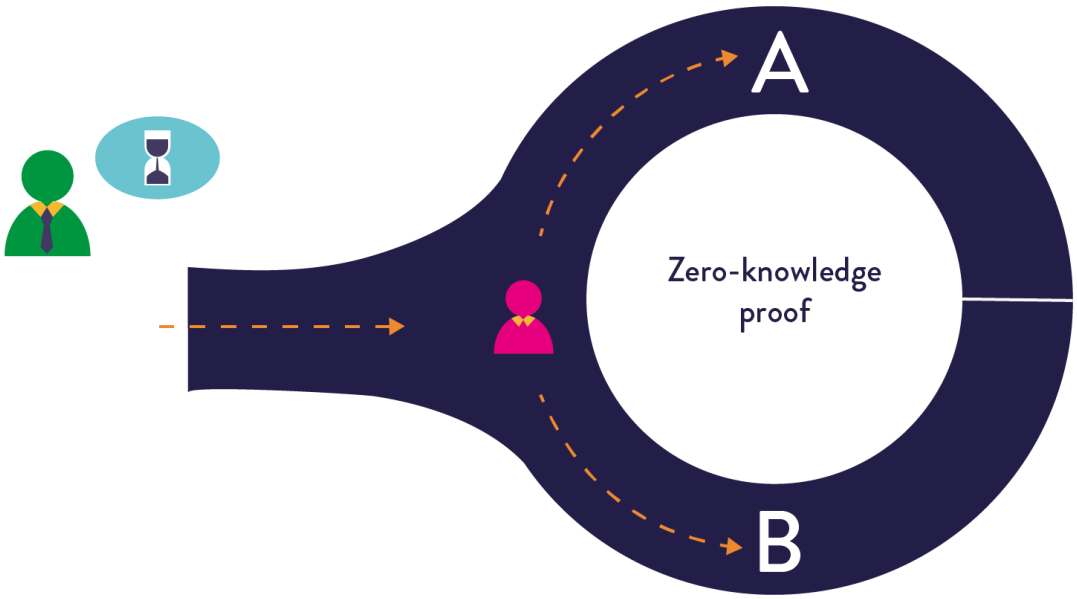


Figure 4: Zero-knowledge proof visualization

Imagine a cave shaped like a doughnut with a wall on the far side from the entrance stopping you from being able to pass through and walk around the cave in a circle unless a magic password is said. Sarah (in pink) wants to convince John (in green) that she knows the password. One way she could do this could be to walk with John to the wall, shout out the magic password, and if she passes through, then John could be convinced that Sarah knows the password. The problem with this conventional approach is that John could learn the magic password, which is something Sarah could perhaps not like him to know. This approach is akin to the way Bitcoin works, where everyone sees every transaction so that we can be certain of the Bitcoin value each public address holds. Yet this comes at the cost of everyone being able to know this information.

Because Sarah does not want John to know the magic password, Sarah thus needs a way to prove to John that she knows the password without actually telling him

what the password is. What she needs is a **zero-knowledge proof**, proving something (here the fact that Sarah knows the password) while conveying zero-knowledge (here without John learning the password).

Sarah could achieve the zero-knowledge proof in the following manner: she could enter the cave and arbitrarily choose to go to either side A or B. John would then come to the cave entrance and ask Sarah to either exit from side A or from side B. Now if Sarah does indeed know the password she will be able to pass through the wall and emerge from the requested side. However if she does

not know the password, she has a 0.5 probability (50% chance) of being able to come from the side requested by John. John would now simply repeat the experiment n times, meaning Sarah could only pass his tests without knowing the password at probability 2^{-n}. Repeating this experiment just 15 times would mean Sarah would only have a 2^{-15} or 0.0031% chance of passing John's test without actually knowing the password. We describe this as an **argument of knowledge**, as Sarah cannot actually prove she knows the password. However, by repeating the experiment multiple times and letting n tend toward infinity, there is an arbitrarily small, and thus ignorable, chance that Sarah could convince John she knows the password without actually knowing it.

This principle has been applied to Zcash. Technically, Zcash uses **zero-knowledge succinct non-interactive arguments of knowledge** (zk-snarks) – a very clever variant of zero-knowledge proofs that doesn't rely on this kind of challenge response.

Zk-snarks

In order to get a high-level understanding of how zk-snarks could work for anonymous transactions, we first need to explore a possible function used to transfer funds between addresses. Below we have the transfer function for the standard ERC20 token implemented in solidity[3]. For now, read it as if it is R code, even though it is solidity code. It is similar, and simple enough that you should understand it.

[3] Note in Modules 6 and 7 we will dive deeper into the solidity code: the smart contract language used in Ethereum.

```
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    require(b <= a);
    uint256 c = a - b;

    return c;
}

function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a);

    return c;
}

mapping (address => uint256) private _balances;

function _transfer(address from, address to, uint256 value) internal {
    require(to != address(0));

    _balances[from] = _balances[from].sub(value);
    _balances[to] = _balances[to].add(value);
    emit Transfer(from, to, value);
}

function transfer(address to, uint256 value) public returns (bool) {
   _transfer(msg.sender, to, value);
   return true;
}
```

Figure 5: Snippets taken from the from the ERC20.sol contract implemented by Zeppelin Solidity

The above is made to look more complicated than it is because it is split up. It can be rewritten as follows:

```
1  mapping (address => uint256) private _balances;
2
3  function transfer(address to, uint256 value) public returns (bool) {
4      require(to != address(0));
5
6      // Do the subtraction:
7      require(value <= _balances[msg.sender]);
8      _balances[msg.sender] = _balances[msg.sender] - value;
9
10     // Do the addition:
11     _balances[to] = _balances[to] + value;
12     require(_balances[to] >= value);
13
14     // Emit a transfer event to the blockchain
15     emit Transfer(msg.sender, to, value);
16
17     return true;
18 }
```

Figure 6: Snippets taken from the from the ERC20.sol contract implemented by Zeppelin Solidity – rewritten

Take a moment and try to understand how the above two pieces of code are the same. Challenge yourself – you may be surprised at how far you get.

The function `transfer` takes in three parameters, namely:

1. `to`, which is the address to which the transfer of funds is going.
2. `_value`, which is the value of the transaction.
3. A third parameter is implicit: `msg.sender`. The address from which the transfer of funds is coming is the sender of this transaction (i.e. the message sender `/msg.sender`). It then returns a Boolean: true if the transaction was successful, otherwise false.

Let's examine lines 4, 7, and 12 a bit more closely. These are important `require` statements.

Line 4: In the first require statement, we check to make sure the user is not sending funds to the 0 address by mistake, since this is the default value if no `to` is specified. This is helpful because there is no bank to phone and ask for transactions to be reversed if an error is made. However, this is not strictly required for an ERC20 token, and many Ethereum tokens do not check this.

Lines 7 and 12: Here you need to prevent an integer overflow. Basically, for an unsigned integer (`uint`), which is always positive, so in `uint3` which consists of the numbers (0, 1, 2, 3, 4, 5, 6, 7) 2-

4 is 6 and 5+5 is 2 since there is no number less than 0 and no number greater than 7. You have to make these kinds of checks when dealing with money; there are just no two ways about it.

Lastly, the Transfer event gets logged on the blockchain as something that software can listen to. For example, a DApp you are using to make this transfer could listen to that event and display a message saying "transfer complete" when the event triggers. It doesn't affect any logic within the transfer.

Now that we understand these require statements, we understand the difficulty of achieving anonymous transactions. We need to explicitly see the address that the transaction is coming from, as well as the value in question, so that we can check that the address has enough funds for the transaction in question. Our problem now becomes how we could implement the above transaction scheme, ensuring these require statements are met, but without anyone finding out the details.

Zcash has implemented their system in such a manner that there are four possible cases that can occur:

1. Transactions are public (as seen in Bitcoin).
2. Transactions are completely anonymous. (The from address, to address, and value are all shielded.)
3. Input is shielded. (The from address is shielded, while the to address and value are transparent.)
4. Outputs are shielded. (The to address and value are shielded, while the from address is transparent.)

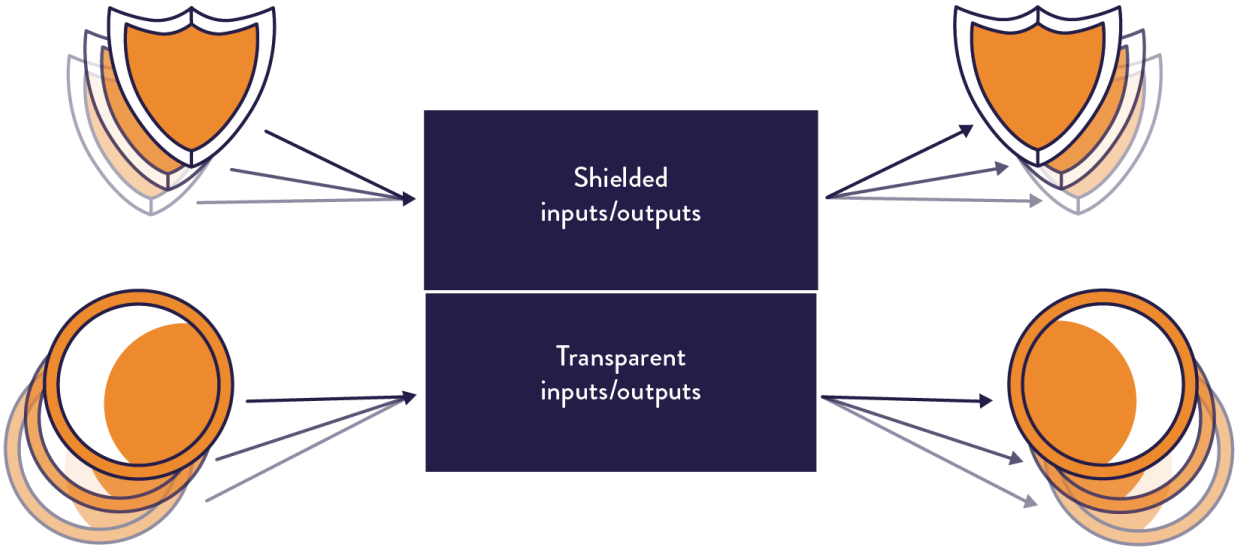


Figure 7: The four cases

This scheme of shielded inputs/outputs and transparent inputs/outputs is nice for a variety of use-cases. In our code example of explaining zk-snarks below, we choose to showcase the third variant above: the from address is shielded and the to address and value are transparent. This scenario would make sense if, say, someone wanted to make an anonymous donation to a charity but would like the donation amount to be publicly visible. Here is a basic zk-snark, which is a mathematical definition consisting of three equations:

Generator (**C** 'circuit', **λ** 'toxic waste'){
(**pk**, **vk**) = G(**λ**, **C**)}

Prover (**_from** *'secret input'*, **_to** *'public input'*, **_value** *'public input'*) {
𝕀𝕀= P(**pk**, **_from**, **_to**, **_value**)}

Verifier {
V(**vk**, **_to**, **_value**, 𝕀𝕀) == (∃ **_from** s.t C(**_from**, **_to**,**_value**)}

Take some time to examine the above. Understanding how exactly the functions work is out of the scope of this course, but it is important to understand the inputs and outputs of each function and what they mean.

Walking through the functions

The first function is the generator function. It is a once-off function used to initialize the whole setup. What, then, is C? C, the **circuit**, is the program that we would like to run. In this case, C is equal to the `transfer` function that we introduced earlier. Why do we use the term "circuit"? Well, as Vitalik Buterin, the creator of Ethereum, points out, zk-snarks cannot be applied to any computational problem directly; rather, you have to convert the problem into the right "form" for the problem to operate on. That means our current `transfer` function will not do in its current form. Instead, we need to follow a non-trivial process of converting it into something called a **quadratic arithmetic program** (QAP).

The pipeline of this process can be seen in the diagram below, and here you can see where we get the term "circuit". If you want to get more details on how this transformation actually works, head over to Vitalik's blog post.

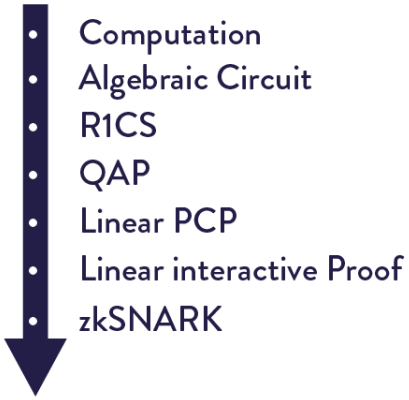


Figure 8: The pipeline

Now that we better understand the first input, let's move onto λ, the "toxic waste." First off, it is termed "toxic waste" because it is something that needs to be disposed of very, very carefully, presenting an extreme danger if it is leaked. λ is the initial parameter, along with the circuit C, used to generate pk and vk, the public proving key and the public verifying key respectively. These initial parameters, λ, need to be disposed of, because if someone knew these parameters they could construct arbitrary proofs, making it possible for them to send money to themselves from whomever they choose – not ideal to say the least. This is why this step (of generating the pk and vk) is often called the **once-off trusted set-up**. We need to trust that whoever performed this step properly disposed of λ.

How did Zcash do this? They did a multi-party computation ceremony. You can read the technical paper on how this was done here: (https://d3fpmqdd8wxk96. cloudfront.net/downloads /mpc whitepaper.pdf). Otherwise, the excerpt below will suffice in explaining their aim:

> "In order to ensure the toxic waste does not come into existence, our team (Zcash) designed a multi-party computation (MPC) protocol which allows multiple independent parties to collaboratively construct the parameters. This protocol has the property that, in order to compromise the final parameters, all of the participants would have to be compromised or dishonest."

The outcome of step 1 is that we now have pk and vk (the public proving key and the public verifying key respectively). These will be needed in the proving and verifying functions that we will employ next.

Let's now turn our attention toward the prover function, presented again here for ease of reference.

> Prover (**_from** *'secret input'*, **_to** *'public input'*, **_value** *'public input'*) {
> $\mathbb{\Pi}$= P(**pk**, **_from**, **_to**, **_value**)}

As we described earlier, in this case the sender does not want anyone to know what their address is, and so this must be a secret input. (Remember how in the Ethereum example it was impossible to hide the `_from` since it is always available as `msg.sender`.) The value and `to` address are both public inputs. The user would use these three parameters along with the public proving key (pk) generated in the previous step in order to produce a proof, $\mathbb{\Pi}$.

For the sake of space, we are going to treat the actual mathematics behind the prover function as a black box. We are also going to do the same for the verifier function below. If you want to unpack these black boxes, see the resources provided in the references section.

Provided here for ease of reference, the verifier function is:

> Verifier {
> V(**vk**, **_to**, **_value**, $\mathbb{\Pi}$) == ($\exists$ **_from** s.t C(**_from**, **_to**,**_value**)}

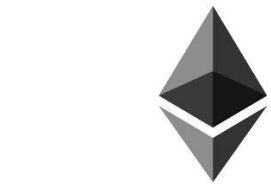
This function uses the public verification key along with the public inputs, the `to` address and value, and finally the proof generated in the previous step. Verification is extremely fast, which is critical: network validators can quickly determine the validity of transactions before including them in the next block.

And that is it. These are the basics behind zk-snarks and how it is used in order to keep your Zcash transactions anonymous.

The purpose of this section of notes is to challenge you and expose you to interesting innovations in areas such as mathematics that allow us to solve real-world problems, such as blockchain anonymity. It should be noted that the above functions and explanations are high-level, educational examples. They have been provided here in order to give you a very basic understanding of how zk-snarks work and are not by any means used in practice. In practice, the implementation would be much more complicated and take many more factors into account. Refer to the full Zcash whitepaper in the references section for a more technical explanation of Zcash.

Unit 3: Ethereum

Introduction



Recall our definition of a blockchain as a complete, tamper-resistant, and public transaction ledger that is replicated and held by a distributed set of nodes. So far, we have been content with the fact that all transactions in the ledger are transactions detailing transfers of Bitcoin amounts or some other cryptocurrency. As people became more and more aware of the power of blockchain technology and how it provides complete transparency and accountability to a payments-type system like Bitcoin, people naturally wondered how this could be extended to other use-cases. In short, people wanted to know how we could further harness the power of blockchain technology.

Ethereum

Ethereum was born out of this vision. Ethereum was designed to be an open software platform, based on blockchain technology, that would enable developers to create decentralized applications. While the purpose of Bitcoin was to simply track ownership of Bitcoins (a currency), the Ethereum blockchain focuses on creating a decentralized, general-purpose platform for running code enabling a diverse range of decentralized applications.

How is this done?

The Ethereum blockchain has something called the **Ethereum Virtual Machine** (EVM), which is Turing complete, and allows the execution of smart contracts. A virtual machine is a computer that runs on top of the operating system, rather than directly on the computer hardware This makes it easier to make sure the code executes exactly the same way on all different types of computer, which is of course desirable for Ethereum, where consensus is required.

Smart contracts are simply computer code encoded and stored in the blockchain, that are then executed by the Ethereum Virtual Machine (EVM). These pieces of code can be published on the Ethereum blockchain and thereby exist on a distributed set of nodes, where they will be executed by a network of computers, resulting in possible ledger updates. Since the EVM is deterministic, they run exactly as programmed – without any possibility of censorship, downtime, fraud, or third-party interference.

To show the desirability of this, we can consider a brief, high-level example. Assume a chess tournament is encoded as a set of smart contracts in Ethereum. Assuming that enough of the game logic lives in the blockchain, the blockchain can hold the tournament's prize money and ensure that the true winner of the tournament can receive this prize money. This could be desirable for some kinds of international tournaments where the players are in different countries and don't want to trust a central entity with the distribution of the prize money.

Since **decentralized applications** (or **DApps**, as they are often termed) run on a blockchain and utilize smart contracts for their logic, they cannot be manipulated or controlled by any centralized entity. This is especially desirable in some arenas – e.g. voting. The actual process of voting is facilitated by centralized parties that have been entrusted to neither tamper with the votes nor do anything else to harm the integrity of the voting process. Voting DApps, however, would take away the need for trust in this process while still achieving everything desired.

What makes Ethereum powerful is that standards have emerged to allow different contracts to communicate with each other. These standards are created and agreed upon by the community. One such standard is the ERC20 standard. If you use this standard you can be sure that your code will be interoperable with others.

ERC20 token standard

The ERC20 token standard is a simple smart contract that allows users to create a token on the Ethereum platform. Do you remember the fragment of Ethereum code we looked at earlier in the Zcash section? The `mapping (address => uint256) private _balances;` line of code is the core of what a token contract is: it simply records the balances that each address owns, and has a `transfer` function. The contract details the name, supply, and ownership of the token. It also details how the tokens may be transferred between individuals, among other things. This enables users to create tokens for their specific DApp or project using the ERC20 standard on the Ethereum blockchain, as opposed to the difficult process of creating their own blockchains. This is why many projects are ERC20 tokens.

Other token standards exist as well, depending on the needs of the project. ERC721, for instance, is a non-fungible token standard, meaning each token is unique and tailored toward DApps that may need distinguished digital assets. Cryptokitties is a DApp that allows users to collect and trade digital cats. In this case, each cat would be a token, and we obviously want all cats – i.e. tokens – to be unique. Thus, the ERC721 token standard would be applicable in this case.

In Modules 6 and 7 you will be learning more about developing smart contracts in Solidity for the Ethereum blockchain. However, before we get there we are going to dive deeper into the working of Ethereum and another prominent decentralized application platform: EOS. First, let's introduce arguably the biggest challenge facing blockchains today: scalability.

Scalability

Scalability alludes to how the system could expand to be able to support global, mass adoption, as opposed to the limited, early adopters currently using the network. One of the most common criteria that helps to measure scalability is the notion of **transactions per second** (tps). While Bitcoin is currently able to process 7 tps, Ethereum can manage about 15 tps. In comparison, payment giant VISA can handle 24 000 tps. With such a large difference in tps, decentralized technologies are still a fair way from rivalling conventional payment technologies. As you can imagine, the entire world would not be able to use Bitcoin, as a payment system speed of 7 tps would never match the global demand for transactions.

The problem of scalability is closely linked with the consensus protocol used by the blockchain. Recall that a consensus protocol is a component of blockchain that is necessary to ensure all decentralized parties consistently reach an agreed-upon state of the shared ledger. Since a centralized database need only worry about updates, transaction processing speed is effectively only limited by the rate at which this single node can update this centralized ledger.

In the decentralized context, we need to protect against double spend attacks (and many other attacks), all while trying to process as many transactions as possible and keep the ledger in a state of consensus among a distributed set of nodes. Instead of only a single centralized database having to process a transaction, in decentralized systems every transaction needs to be processed by every single node in the network. This puts a fundamental limitation on the network's transaction throughput equal to the minimum requirement for any single node in the network.

Currently, Ethereum is using a proof-of-work consensus protocol, yet they have many scaling plans in the pipeline. Those behind Ethereum are aware that it must be scaled before Ethereum can reach its potential of being a truly mainstream platform.

Idea behind proof-of-stake

As previously mentioned, Bitcoin, Ethereum, and many other currencies currently utilize the proof-of-work protocol. This energy-intensive protocol requires computer hardware to solve some cryptographic puzzle in order to produce a valid block that can be added to the chain. Alarmingly, Bitcoin mining uses around as much electricity per year as consumed by the country of Ireland. This is just one argument against the protocol. In addition, the protocol does not lend itself toward scalability, since all computation has to happen sequentially, despite there being thousands of computers in the network, and the internet latency to share all the data between the nodes to reach consensus also prevents higher throughput of transactions.

Instead, Ethereum is moving toward a proof-of-stake system. The idea behind proof-of-stake is that instead of network validators competing to solve a computationally intensive puzzle, validators will deposit some stake (eth) in the system in order to participate in the consensus process. The chance of being chosen to produce a block is proportional to the stake offered by the participant. If the participant produces a block that is deemed invalid by the network, they will lose their stake. The proof-of-stake concept hinges on the fact that participants with a greater stake (or "skin in the game") will be incentivized to:

1. produce valid blocks so as to not lose their bigger stakes, and
2. maintain the integrity of the system, and hence the value of the ecosystem, of which they own part.

Summary

In this notes we have introduced Ethereum and it key features. We have also seen how R code could be implemented to make calls in Ethereum. In the next set of notes, we will talk more about Ethereum's plans to scale before moving on to EOS, another decentralized application development platform that takes a different approach towards scalability (among other things).

Further reading

https://cryptozombies.io/ is a fun tutorial to introduce you to smart contract programming.

Unit 4: Scalability and an Introduction to EOS

Introduction

In this section we are going to explore how Ethereum is approaching the scalability issue, before moving onto EOS, which takes an alternative approach.

Layer 1 scaling solutions

The Ethereum community is attacking the scalability problem from all angles. The first idea is called Layer 1 scaling solutions. Layer 1 scaling solutions involve implementing changes to Ethereum's base layer protocol as depicted in Figure 9.

Figure 9: Layer 1 scaling solutions

As you will remember, in decentralized systems every transaction needs to be processed by every single node in the network. One possible layer 1 solution is known as **sharding**. The idea of **sharding** comes from databases where the state of the database is split into **shards** to allow for greater scalability. In Ethereum, sharding involves segmenting the blockchain state into different **shards** such that individual nodes need only process transactions associated with their assigned shard. We can illustrate the logic of sharding by considering an oversimplified scenario.

Imagine only having two nodes. If each node can process five transactions per second, and all nodes have to process all transactions, it would take two seconds for our system to process ten transactions. If we implemented a sharding-, divide-and-conquer-type strategy, we could have one node process five of the transactions, while the other node processed the remainder. This would allow our system to only take one second to process all ten transactions instead of the previously

needed two seconds. This effectively doubles the throughput of the system. Such an illustrative oversimplification serves to demonstrate the logic behind sharding at a high level. For a more in-depth explanation of sharding, refer to this article.

Layer 2 scaling solutions

The next idea, layer 2 scaling solutions, involves building solutions "on-top" of Ethereum's base layer protocol. This could be in the form of smart contracts that interact with software running off-chain or similar solutions that require no changes to Ethereum's base level protocol. Figure 10 visually represents layer 2 scaling solutions.

Figure 10: Layer 2 scaling solutions

One notable layer 2 solution is the concept of state channels. The idea of a **state channel** is to move some of the state management off of the blockchain, which is normally simply termed **off-chain**. The state is continually signed by the parties involved. At any point in time this state can be settled on chain, where the signatures of the participants are verified.

A subset of state channels are payment channels, the simplest being a unidirectional payment between 2 parties. A bar tab is a simple example. To open this bar tab on Ethereum, Alice creates a payment channel contract that contains the maximum amount of money Alice will be able to spend at the bar. This way the bartender knows that Alice cannot get away with not paying since the money is locked up in escrow.

Then, for every beer Alice purchases, both Alice and the bartender sign the state of how much Alice owes the bar. At any point in time this pair of signatures can be submitted to the blockchain for verification. When this happens, a challenge period is started to prevent Alice from trying to cheat the system by submitting an old signature and paying less. The amount that Alice owes the

bar is always increasing, so the bartender always has incentive to submit the latest signed state to the smart contract during this challenge period. So, provided the Bartender never gives Alice a beer without first getting her signature, it is impossible for Alice to get free beer. When the challenge period ends, the bartender gets what they are owed, and Alice gets what remains of her deposit.

Imagine Alice bought thousands of beers, then with several transactions the net result of thousands of individual transactions is achieved. There are techniques to do this with any arbitrary stat that changes over time.

EOS

"EOSIO" is another blockchain platform with smart contracts that sacrifices decentralization for some gains in scalability, with **EOS** being the native token

used within the system. Consequently, the platform is often referred to as simply "EOS". The platform was developed by an organization called block.one, who raised 4 billion USD over a year-long initial coin offering (ICO) to build the ecosystem. The EOS main net has been live since June 2018 and continues to grow, improve, and offer a world-class infrastructure for decentralized application development. In the following, we highlight some of the main points that differentiate EOS from Ethereum.

EOS' scalability approach

Since every transaction in a decentralized system needs to be processed by every single node in the network, there is a fundamental limitation on the network's transaction throughput equal to the transaction processing throughput we require for any individual node. The original fear was that by increasing the hardware requirements for any individual node, thereby raising the fundamental limitation on the network's transaction throughput, less powerful nodes (i.e. computers with less RAM, or less powerful CPUs or GPs) will be forced to drop out of the network, leaving only powerful node operators. This leads to a more centralized solution, which goes against the aim of decentralized solutions.

The founders of EOS, however, decided that the network should rather be run by 21 master nodes that all have state-of-the-art networking hardware, effectively ensuring the lower bound on transaction throughput would not be significantly lowered due to simple consumer grade nodes participating in the process. Moreover, instead of the 21 nodes inefficiently competing to solve cryptographic puzzles in order to add blocks, the nodes work together in a round robin fashion, sequentially producing valid blocks.

Some argue that the scalability of the EOS platform comes at the cost of decentralization. Only 21 nodes run the EOS platform, as opposed to the more than 25 000 nodes on the Ethereum network. Some claim that the distribution of network power in the Ethereum network is, in reality, highly concentrated due to there being a small number of mining pools. However, this is a distorted view since mining pools are made up of many individuals. Please see the further reading section for more information.

Delegated proof-of-stake

Delegated proof-of-stake (DPoS) is the consensus protocol used in EOS. As noted earlier, in proof-of stake the chance of being chosen to produce the next block is proportional to the amount resources staked in the system. **Delegated proof-of-stake** (DPoS) functions in a similar fashion. The difference is that the resources (or EOS) staked in the system are used to vote for the 21 master nodes (called **block producers**). Every EOS token counts as exactly one vote and can be used to vote for up to 30 different block producers.

Users with more EOS therefore naturally have a greater say as to who the block producers should be. Since those with more EOS have a large stake in the ecosystem (often being termed "whales" due to their large holdings), the logic is that they would be incentivized to vote for honest block producers that they believe will contribute and grow the EOS ecosystem and hence the value of their stake in the system.

Anyone can register to become a block producer, but industry-grade hardware and a compelling narrative are needed to convince the community to vote you in. Actors are incentivized to become block producers due to the 1% of EOS currency inflation that is allocated to rewarding block producers. For a more in-depth explanation of the system, take a look at the resources provided in the further reading section.

Resource allocation

Another innovation brought forward by EOS is that transactions are made free. In Ethereum, "gas" is required for transactions. How gas works will be addressed in a later module. In EOS, the native EOS token is used to allocate scarce system resources such as CPU and NET (together known as **bandwidth**). Instead of users paying a small amount to use the network, users stake EOS. Staking EOS allocates them a proportional amount of system resources to use the network and thereby facilitate transactions. What's more, users can simply unstake their EOS at a later stage, to sell or use them as they wish.

A notable exception to this, however, is RAM – the network resource used for storage. RAM is scarcer than CPU and NET, and thus it is currently allocated through a marketplace dynamic facilitated by a Bancor algorithm to ensure constant liquidity. The Bancor algorithm means that resources are pooled against each other to prevent big fluctuations in price of those resources.

Summary

Scalability is one of the important research topics in blockchain technology. Blockchain has many envisioned use-cases, but it will not be possible without scalability. Ethereum is not the only smart contract blockchain around, and other blockchains exist that try to improve on scalability in various different ways, by making different tradeoffs.

Further reading

"Are Miners Centralized? A Look into Mining Pools – ConsenSys Media." 9 May. 2018, https://media.consensys.net/are-miners-centralized-a-look-into-mining-pools-b594425411dc. Accessed 10 Feb. 2019.

https://medium.com/blockchannel/counterfactual-for-dummies-part-1-8ff164f78540 (Part 1 of an excellent introduction to state channels.)

https://docs.learnchannels.org

Top block producers: https://www.alohaeos.com/vote

Bibliography

ConsenSys Media. 2019. *Are Miners Centralized? A Look into Mining Pools – ConsenSys Media.* [ONLINE] Available at: https://media.consensys.net/are-miners-centralized-a-look-into-mining-pools-b594425411dc. [Accessed 25 February 2019].

LearnChannels. 2019. *LearnChannels*. [ONLINE] Available at: https://docs.learnchannels.org. [Accessed 25 February 2019].

Medium. 2019. *State Channels For Dummies: Part 1 – BlockChannel – Medium*. [ONLINE] Available at: https://medium.com/blockchannel/counterfactual-for-dummies-part-1-8ff164f78540. [Accessed 25 February 2019].

WIRED. 2019. *Prosecutors Trace $13.4M in Bitcoins From the Silk Road to Ulbricht's Laptop | WIRED*. [ONLINE] Available at: https://www.wired.com/2015/01/prosecutors-trace-13-4-million-Bitcoins-silk-road-ulbrichts-laptop/. [Accessed 25 February 2019].