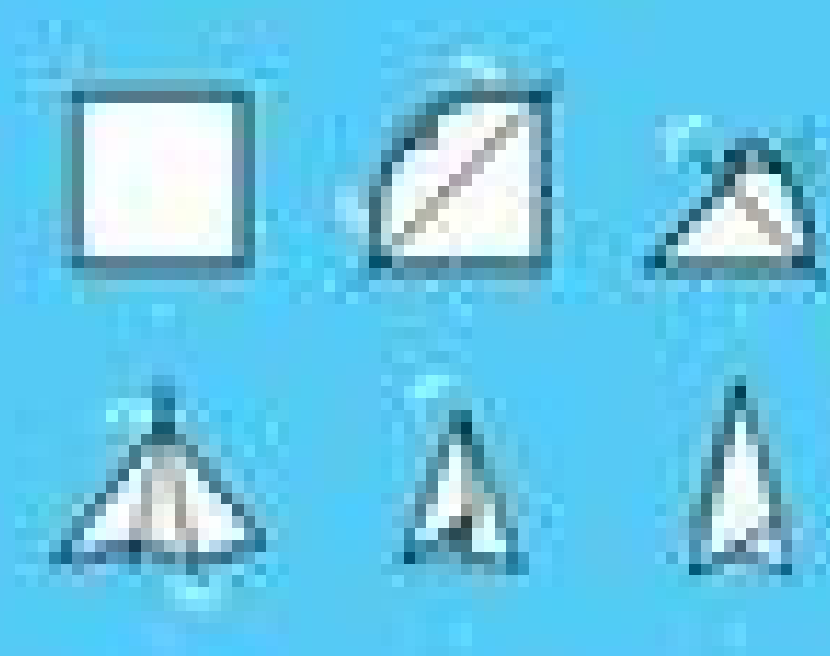


A DEVELOPER'S GUIDE TO ETHEREUM



THIS CONTAINS THE FULL SOURCE CODE

A Developer's Guide to Ethereum

Copyright © 2018 SitePoint Pty. Ltd.

Ebook ISBN: 978-1-925836-01-1

Cover Design: Alex Walker

Project Editor: Bruno Škvorc

Notice of Rights

All rights reserved. No part of this book may be reproduced, stored in a retrieval system or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical articles or reviews.

Notice of Liability

The author and publisher have made every effort to ensure the accuracy of the information herein. However, the information contained in this book is sold without warranty, either express or implied. Neither the authors and SitePoint Pty. Ltd., nor its dealers or distributors will be held liable for any damages to be caused either directly or indirectly by the instructions contained in this book, or by the software or hardware

products described herein.

Trademark Notice

Rather than indicating every occurrence of a trademarked name as such, this book uses the names only in an editorial fashion and to the benefit of the trademark owner with no intention of infringement of the trademark.



Published by SitePoint Pty. Ltd.

48 Cambridge Street Collingwood

VIC Australia 3066

Web: www.sitepoint.com

Email: books@sitepoint.com

About SitePoint

SitePoint specializes in publishing fun, practical, and easy-to-understand content for web professionals. Visit

<http://www.sitepoint.com/> to access our blogs, books,

newsletters, articles, and community forums. You'll find a stack of information on JavaScript, PHP, design, and more.

Preface

Blockchain technology has certainly been hyped over the past few years, but when you strip all of that away, what can actually do with it? This book provides you with an introduction to Ethereum, an open source platform that's based based on blockchain. It enables developers to build and deploy decentralized applications that can be relied on to work without fraud, censorship or interference from third parties.

We start off by explaining what blockchain is and how it works, and also look at some potential practical applications for blockchain technology. We then move on to looking at the Ethereum platform specifically. Far more than just a cryptocurrency or smart contracts platform, Ethereum is becoming an entire ecosystem for building decentralized applications.

Who Should Read This Book?

This book is for anyone interested in using the Ethereum platform for development. No prior knowledge of blockchain is assumed.

Conventions Used

CODE SAMPLES

Code in this book is displayed using a fixed-width font, like so:

```
<h1>A Perfect Summer's Day</h1>
<p>It was a lovely day for a walk in the park.
The birds were singing and the kids were all back
at school.</p>
```

Where existing code is required for context, rather than repeat all of it, `⋮` will be displayed:

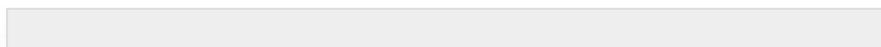
```
function animate() {
  ⋮
  new_variable = "Hello";
}
```

Some lines of code should be entered on one line, but we've had to wrap them because of page constraints. An `↵` indicates a line break that exists for formatting purposes only, and should be ignored:

```
URL.open("http://www.sitepoint.com/responsive-web-
↵design-real-user-testing/?responsive1");
```

You'll notice that we've used certain layout styles throughout this book to signify different types of information. Look out for the following items.

TIPS, NOTES, AND WARNINGS



Hey, You!

Tips provide helpful little pointers.

Ahem, Excuse Me ...

Notes are useful asides that are related—but not critical—to the topic at hand. Think of them as extra tidbits of information.

Make Sure You Always ...

... pay attention to these important points.

Watch Out!

Warnings highlight any gotchas that are likely to trip you up along the way.

Chapter 1: Blockchain: What It Is, How It Works, Why It's So Popular

BY BRUNO ŠKVORC

This introduction to blockchain was originally published at [Bruno's Bitfalls website](#), and is reproduced here with permission.

What is “the blockchain”? How does it work, why is it popular, and why do so many people claim it'll revolutionize the world?

In this article, we'll explain blockchain technology with a banal example that's nonetheless valid for most cryptocurrencies currently in circulation. Not familiar with the concept of cryptocurrency? See [here](#)!

Mario and Luigi

Mario needs to send \$100 to his brother, Luigi, because Luigi being Luigi, he got into some debts at the other end of the world.

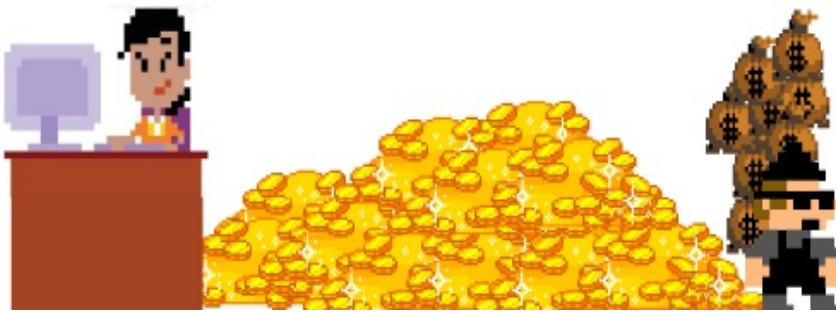


Mario walks into the bank and says “I’d like to send \$100 to Luigi”. The teller says “Account card please”, “ID please”, and “done”, in that order.



In this *centralized* scenario, the bank is the *central* authority over Mario’s and Luigi’s money. Both Mario and Luigi trust the bank to transfer the amount, and believe the numbers shown on their bank account statements. They trust the bank — despite the fact that all the bank has to do is change a number in a database. It’s all digital, after all.

However, when we're dependent on such a central authority, that authority poses a certain threat. It can disappear with our money, it can be evil and not increase Luigi's value while decreasing Mario's value, thereby keeping the difference, or it can just be clumsy and make a mistake. Our finances depend directly on their competence.



One way of preventing such scenarios is skipping the bank entirely and having our own system of tracking value and the travel of value from one location to the next.

Imagine a piece of paper on which we're noting down the status of our bank account. If only Mario and Luigi are using that system, it's hard to keep it fair. If one of them becomes greedy, the system is already compromised. Hence, such a *distributed* (non-centralized) system needs to have enough participants to make it viable — a minimum of three.

Papers

Let's assume we have five participants: Yoshi, Mario, Luigi, Wario, and Bowser, and that each of them has their own piece

of paper.



Mario wants to send \$100 to Luigi. To do this, he lets everyone know (by loudly proclaiming): “I’m sending \$100 to Luigi! Please take note, everyone!”



At that moment, every participant checks Mario’s account to make sure he’s got enough value on it to send to Luigi (yes, every account’s status is public) and if so, writes this

transaction down on their piece of paper. Transactions of this type are written down onto the participants' pieces of paper until they run out of room. In other words, *every* transaction between *any two people* is logged on *every participant's paper*.

Before we file the filled-up paper away into a folder or filing cabinet and grab a new, blank one, we need to seal the filled one with a special code.

Seals and Mining

This “seal” guarantees that the contents of the paper are true.

How do we get this seal? With a special algorithm (mathematical operation) which, when we feed it some input data, always produces the same output *if the input data is the same*.

Let's take the following as an example:

$$x_1 + x_2 + \dots + x_n = z$$

In other words, a simple sum.

Let's assume that our paper's values are *true and valid* (i.e. all the transactions are confirmed) if and only if a given summation operation produces the number 10000.

$$1000 + 6000 + 3000 = 10000$$

In this case, the input data is 1000, 6000, and 3000, while the *seal* is 10000.

Okay, so let's assume that our participants above came to the following agreement: if, when you sum up all the numbers on the paper and *a specific* combination of other numbers, you get 10000, then the transactions on that paper are valid and can be considered confirmed.

For example, if the paper contains the following five transactions:

- Mario -> Luigi = 100
- Bowser -> Yoshi = 200
- Yoshi -> Luigi = 100
- Mario -> Yoshi = 500
- Luigi -> Wario = 100

The sum is 1000, so we're looking for a number that gives 10000 when added to 1000. This remaining 9000 can be achieved with many combinations:

- $5000 + 4000$
- $1000 + 1000 + 1000 + 1000 + 1000 + 1000 + 1000 + 1000 + 1000$
- $2000 + 3000 + 2000 + 2000$
- etc...

A computer can't intuitively tell which numbers will produce the desired number. To get to this result, a computer has to *randomly guess* between different combinations of numbers under 10000 until it gets one combination that produces

10000. Thus, the first among our participants to correctly guess a combination of numbers that produces 10000 when summed with all the transaction values on the paper will be the one to tell everyone else the result.

Suppose that Yoshi found the combination $4000 + 5000$. He tells everyone: “I’ve got a 10000! Try 4000 and 5000!” Given that it’s very easy to verify the correctness of Yoshi’s numbers by simply inputting them into the algorithm, the other participants verify this. All the other participants’ papers that, during this check, now also produce 10000 when summed with 4000 and 5000 effectively validate the list of transactions in front of them. By doing this, a consensus is reached that the papers are all valid.

If someone’s paper *doesn’t* produce 10000 when summed with 4000 and 5000, we have a problem. If, for example, Bowser either purposely or accidentally logged a different transaction — say, that Mario gave Luigi \$200, and not \$100 — then the sum will not match the requirements.

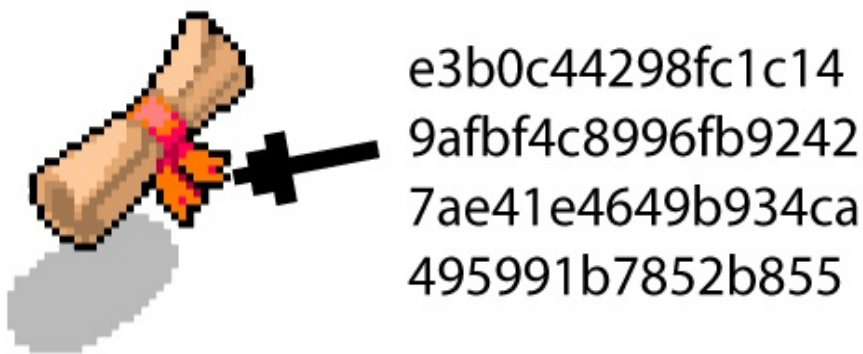


Bowser's paper is thereby considered invalid, and if he wants to continue participating in this system, he'll have to discard his paper, copy someone else's valid paper, and promise to be more careful in the future. On the other hand, Yoshi, who was the one who found the winning combination, gets a reward of, for example, \$5 from the system. The system *produces* the \$5 out of thin air as a reward for the lucky participant.



This *production* of money out of thin air is called **mining** in the cryptocurrency world.

While this was a dramatically simplified example, the only real difference from the real blockchain (beside the fact that in reality it's all digital and automatic) is the fact that the algorithm being used to produce the *seal* is different — a more complex one that can accept both numbers and letters, and outputs code like
90bdaa79bbccacf8558edcbb30df48d7fc920eeb75
a28f883de4100f58a99b49.



The *seals* are called **hashes**, and the algorithms producing them — like SHA256 which produced the above code — are called **hashing functions**. Try opening [this link](#) and inputting any amount of text into the field. Whether you put in a single word or the entire Bible, it'll always produce a hash of exactly 64 characters.

Hence, to get the hash that seals our paper, we enter into the algorithm all the transactions present on the paper. They become a hash. Since there's an infinite number of combinations of characters we can provide as input for the

algorithm, guessing the original input set of data based purely on the output data is a mathematical impossibility.

Specifically, in the Bitcoin blockchain, the consensus of the network and all its participants is that as long as a produced hash (which we get from combining the last paper's hash and all the transactions of the current paper, plus a random bit of numbers and letters) starts with a given number of zeroes, it's valid. For example, if paper 1 has the hash of 000000000000000000000000058edcbb30df48d7fc920eeb75a28f883de4100f58a99b49, paper 2's valid hash will be the one which (when the current paper's transactions and some random data are added to the last paper's hash) contains an equal number or more zeroes at the front.

To get this combination of random characters needed for producing a new valid hash, a computer must make guesses. Modern computers are very fast and easily try out thousands of combinations per second, but this still isn't fast enough given that the number of possible combinations is near infinite. As an aside, the aforementioned hash shows us that the difficulty of guessing a new one is ranked 17, because there are 17 zeroes at the front of it. In time, the difficulty will increase and a new valid hash will need more and more zeroes at the front.

When the paper of each participant is marked as valid with the hash, it's put away into a folder — a *ledger* of sorts — and a new blank one is pulled out.

IN BLOCKCHAIN PARLANCE

In blockchain parlance:

- One paper is a **block**.
- One block contains many **transactions**.
- One block always follows a previous block, forming a chain — a **blockchain**. Validated blocks are put away into a **ledger** (synonymous with **blockchain**).
- Computers guessing the combinations are called **nodes**. A node that guesses the hash combination gets a reward in the form of the blockchain's tokens — in our particular example, a few Bitcoins.
- The guessing for combinations is called **mining**, because we're *digging* for new value in a big pile of random guesses. Instead of muscles for effort and pickaxes for tools, we use electricity, time, and a computer's calculating power. The characters in our story above are all in little go-carts precisely because it's all a race: whoever first confirms a block with a valid hash wins, and gets the very valuable reward. Likewise in the blockchain world, the most powerful computers (or pools of computers) usually win the block rewards.
- The existence of a new valid hash is considered **proof of work**. This is a model that most cryptocurrencies today use in order to make easy guessing or cheating financially non-viable because of too high electricity costs. Some protocols like Ethereum are moving to **proof of stake** in which the pointless wasting of electricity is being minimized and the punishment for invalid participation is made stricter, but that's a topic for another post.

Conclusion

All cryptocurrencies are based on *blockchain technology*.

Blockchain is what makes it possible for them to be transparent, definitive (impossible to falsify or duplicate) and somewhat limited in maximum amount produced. Unlike fiat currency (USD, Euro, etc.), cryptocurrencies cannot simply be printed by their creators, except in the case of some scam

currencies like Ripple (XRP) or OneCoin. (More on those in another post.)

Blockchain technology allows for distributed control over the financial system of a society — local or global — and helps with avoiding middlemen. This is one of the main reasons why cryptocurrencies have exploded in popularity so much. Due to the distributed nature of the blockchain and millions of users all around the world, all of whom serve as “accountants” and validators, many consider cryptocurrencies to be indestructible and unstoppable. Sadly, that’s a different kind of delusion — one that we’ll cover later (see links in the following section).

What Next?

Here’s some further reading:

- What are cryptocurrencies: learn what it’s all about and where the value is coming from.
- Bitcoin is not unstoppable: find out how easy it is to actually stop and destroy a cryptocurrency.
- Bitcoin is not finite: learn why the theoretical limit of 21 million Bitcoin ever produced simply isn’t true, practically speaking.
- Bitcoin is not anonymous: learn why people often (incorrectly) say that Bitcoin is anonymous and useful for the black market and criminal funding.
- What is Ethereum: learn about the most powerful blockchain today.

Chapter 2: What is a Bitcoin Node? Mining versus Validation

BY BRUNO ŠKVORC

This introduction to Bitcoin Nodes was originally published at Bruno's Bitfalls website, and is reproduced here with permission.

You'll often hear the word *node* thrown around in blockchain and cryptocurrency circles. If you've read our intro to blockchain (and we highly recommend you do that), one of the characters in the comic there that's writing down transactions on a piece of paper is actually *a node*. That introduction is quite simplified, however — so let's explain the concept of nodes in a bit more detail now.

Validation Nodes

One node is a computer running specific software. In the case of Bitcoin, one node is a Bitcoin program which connects to other Bitcoin nodes, *i.e.* other Bitcoin programs on the same machine, or on other machines which can be across the street or on the other side of the planet. There are several types and

several versions of Bitcoin software. By picking a specific version of a specific Bitcoin program, a user “votes” for certain changes in the Bitcoin protocol. For example, if a bunch of users suggest the increase of 21 million total BTC to 42 million, the majority of the network is required to vote “yes” by installing the software implementing this change. Code changes are, thus, democratic.

Where this idea falls apart is in the fact that there are very few Bitcoin nodes out there — a mere 10000 currently.

Estimated size of the Bitcoin network (active peering nodes in the network)

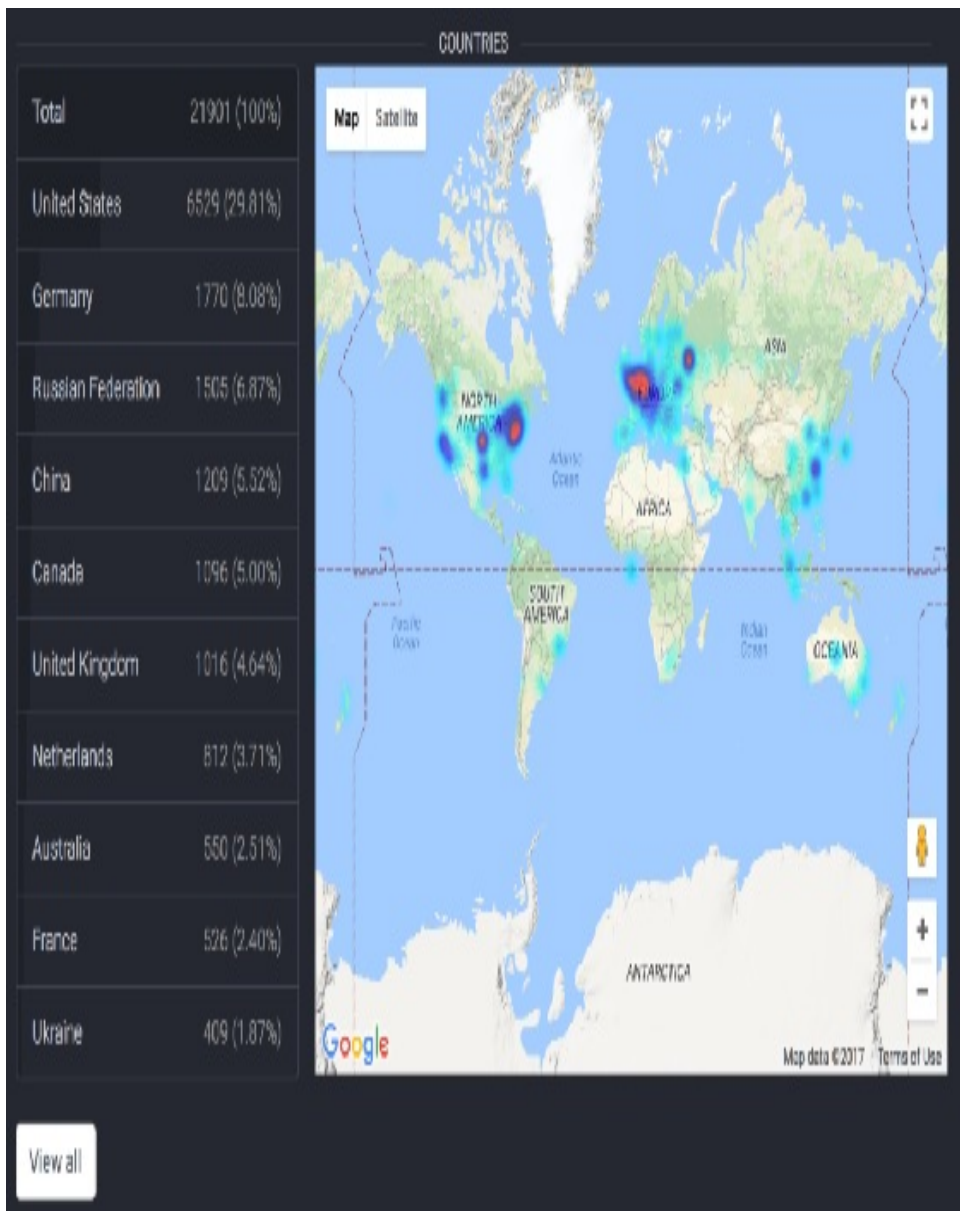
The map shows concentration of 10041 Bitcoin active nodes

Top countries with their respective number of active nodes

1	United States	3174
2	Unknown	1082
3	Germany	828
4	United Kingdom	585
5	Canada	582
6	Russian Federation	542
7	China	345
8	Netherlands	313
9	France	271
10	Australia	249
11	Ukraine	159
12	Sweden	124
13	Austria	109
14	Romania	107
15	Italy	105
...	Other	1468



In contrast, Ethereum — a cryptocurrency five years younger — already has twice as many:



Neither number is very impressive from a global perspective. According to some calculations, running a Bitcoin node on AWS (Amazon’s cloud service) costs around \$10 per month. This means that taking over 10000 brand new nodes takes \$100,000 per month, or only \$1.2m per year — which is

pocket change to any Bitcoin early adopter.

A list of node software you can install, along with their pros, cons, and special features, can be found here.

It's important to note that validation nodes are purely an expense for the users running them. They give their users nothing. Bitcoin Core, for example, needs around 150GB of disk space, 2GB of RAM, and a fast and uncapped internet connection with at least 50KB of constant upload speed available just to run. It's not uncommon to need to upload over 200GB of traffic per month when running a single node. Validation nodes are volunteer nodes and are useful for the system's decentralization, but as they become ever more expensive to run, so too does the number of nodes drop. Add to that the mounting disillusion with Bitcoin's theoretical decentralization due to the fact that bankers seem to have taken over the protocol's development, and the fact that Bitcoin's price is being pumped by crime syndicates, and it's no surprise that the number of nodes dropped by 20% in a single month — from 12000 to 10000. As more nodes disappear, so does centralization. A hostile takeover becomes more and more probable.

Mining Nodes

A mining node is a validation node which also uses the hardware of your own or a rented machine to guess the combinations of numbers and letters needed to validate and verify a block. A mining node can *team up* with other nodes

and send guesses to a common pool (pool mining) to increase chances of guessing, but then counts as only one node.

Because most new miners opt to join a powerful pool to maximize their chances of mining a block and getting rewards, we're seeing a very serious technological centralization happening in which 20 of the most powerful pools are mining almost all the Bitcoin.

Here's a list of the biggest mining pools. Notice that the first one mines 25% of all the Bitcoin in existence.

A mining node is the only bit of software which can “produce” new Bitcoin, and running one in a way that makes it worth your while requires either a very strong computer or free electricity. If you'd like to give mining a go, the list of BTC mining software can be found here.

Conclusion

A mining node is a node which contributes to the network by guessing the combinations needed to “seal” the blocks of transactions and thus confirm them, producing new Bitcoins in the process. A validation node is a node which validates this information, makes sure it's true, and passes the information along to other nodes, thus enabling the transfer of monetary value from location A to location B. Mining nodes are a *subset* of validation nodes, because every mining node is also a validation node.

This difference is only manifested in the PoW consensus

system and becomes technically unnecessary in PoS. With PoS, every node can be a validation node, and mining nodes as such no longer exist: new tokens are created based on another principle. For more about this, please read our [PoW vs PoS article](#).

Chapter 3: How the Lightning Network Helps Blockchains Scale

BY BRUNO ŠKVORC

This introduction to the Lightning Network was originally published at [Bruno's Bitfalls website](#), and is reproduced here with permission.

Bitcoin is currently impractical to use because of slow and expensive transactions plaguing its blockchain. Most people use it as a store of value (the digital gold fallacy) or to trade on exchanges. A concept known as the Lightning Network was introduced as a solution to this *scalability* issue.

The Basics of the Lightning Network

The Lightning Network was first described in a 2015 whitepaper by Joseph Poon and Thaddeus Dryja. The concept, however, was actually introduced by Satoshi Nakamoto in an email to Mike Hearn in 2013.

The Lightning Network works through *payment channels* which are actually multi-sig wallets (multiple signature). A multi-sig wallet is just a Bitcoin address which requires a signature or private keys of several owners before money in that address can be spent. You can view them as bank vaults which require the turning of two different keys at the same time in order to open.



A multi-sig wallet can, for example, be the common Bitcoin address of a married couple, wherein they both have to sign a transaction in order to spend their common Bitcoin.

The purpose of payment channels is regular execution of smaller payments and avoidance of high transaction fees. Examples of relationships ideal for LN channels are employee-employer, consumer-producer, utility provider-

utility consumer, coffee drinker-coffee shop, *etc.* The idea is letting a customer open a payment channel with his coffee shop, regularly paying for coffee without having to wait for confirmation (10 to 60 minutes currently).

How the Lightning Network Works



Let's explain with a step-by-step example. Our imaginary scenario is as follows:

Bob wants to pay Alice to write articles for him. The deal is 10 BTC for a total of 100 posts, or 0.1 BTC per post.

In a traditional Bitcoin system, that would require one hour on

average with a fee of \$5 to \$500 per transaction, depending on how backlogged the network is. Because both Alice and Bob are bitcoin maximalists, they chose to open an LN channel rather than go with a cheaper and easier to use altcoin.

STEP 1: OPENING A CHANNEL



Bob creates a regular Bitcoin transaction on the main chain which defines the following:

- who he's opening the channel with
- how much BTC he's sending into the channel (10 BTC)
- after how long a period of time (one week in this case) he has the right to take the 10 BTC back if Alice does not respond.

The latter is actually a sub-transaction in the main transaction

with a “timelock” function, which makes sure that, in spite of both parties having confirmed it, the money is not moveable for a week.

So, Bob sends Alice two transactions — one in which he suggests opening a payment channel with a 10 BTC deposit on a multi-sig which is opened with that transaction, and one in which he says the 10 BTC go back to him if there’s been no activity in the channel for a week.

STEP 2: ACCEPTING THE OPENING OF A CHANNEL



Alice receives two transactions in which she can see that Bob is offering 10 BTC on the multi-sig address with the two of them as the participating parties. She can also see he’s added

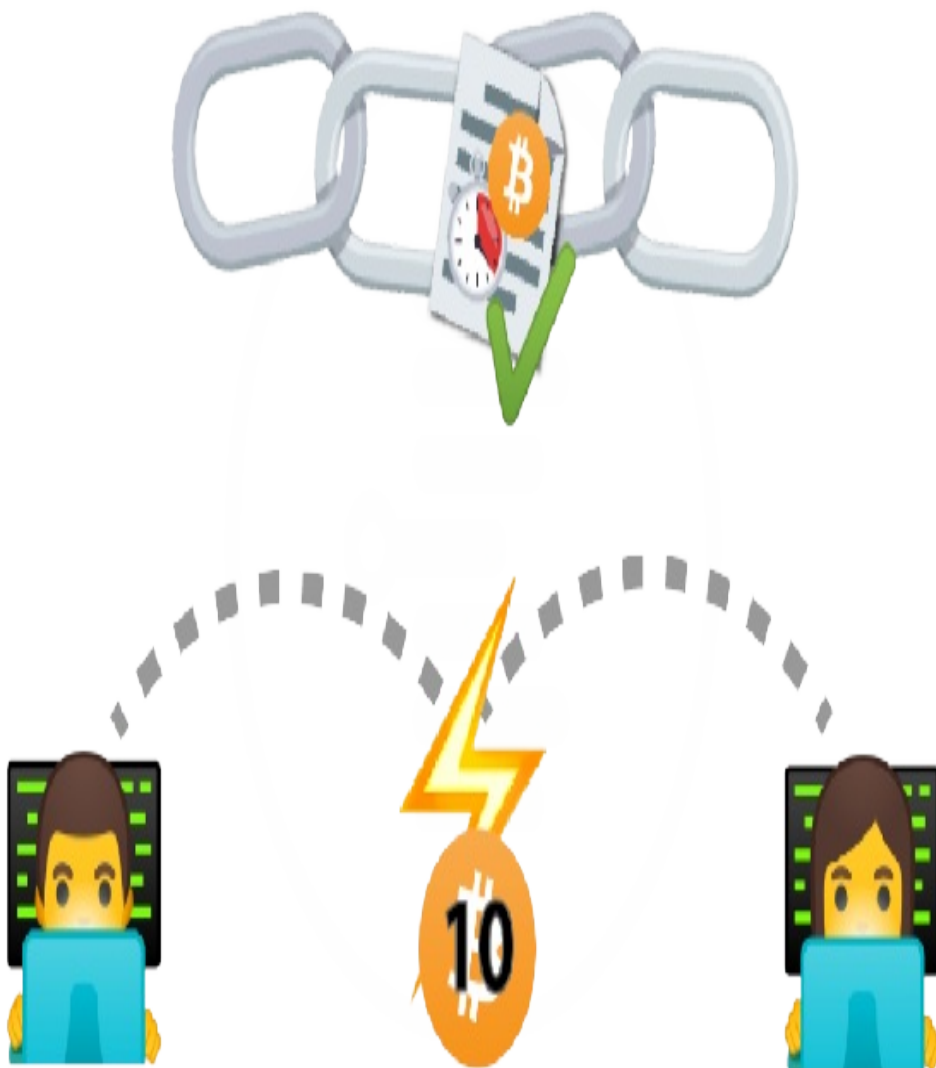
the condition to return the money to him after one week of inactivity. She accepts this and **signs** the transactions, after which she **broadcasts** the transactions and they're sent to the main blockchain, finalizing the channel's creation.



*It's important to define two concepts here: **signing** a transaction and **confirming or broadcasting** a transaction. A signed transaction is merely ready for sending to the blockchain and constitutes an agreement between parties. It's*

not visible on the blockchain. A broadcast or confirmed transaction is sent to the blockchain and closes the payment channel, settling balances.

Signing the first transaction opens the channel and causes Bob's 10 BTC to be deposited into the multi-sig address. Signing the other, despite allowing Bob to take all 10 BTC back, can only become active after a week.



Alice and Bob now have a week to do the first bit of business.

3. SENDING THE FIRST TRANSACTION

Alice wrote an article and Bob likes it. He pays 0.1 BTC by doing the following:

- He generates a new transaction which states “I’m sending 0.1 BTC from the multi-sig address containing 10 BTC, and I’m sending 9.9 to myself”. Alongside it, he generates another transaction: “If the previous transaction is not broadcast within one week of signing it, then I send myself all 10 BTC from the multi-sig address”.
- He sends the transactions over to Alice for signing via Lightning Network nodes without sending them to the main blockchain. Remember: transactions are only finalized in the blockchain once both parties sign and broadcast them.
- Alice receives the transactions and checks conditions: 0.1 BTC for an article, OK, and a week to accept and get 0.1 BTC which means I have a week to send a new article in.



Pending					
Tx1	✓	₿	10	₿	✓
	✓	9.9	→	0.1	



Alice doesn't have to sign these new transactions. By not responding to them, she will trigger the week-long timeout which will return the money to Bob and cancel their arrangement. To keep the deal open, she needs to keep it "on the table" by just signing it and not broadcasting.

This "leaving on the table" is the part the Lightning Network nodes are taking care of. The software accepts and signs the transactions, but only on the LN layer, not the main Bitcoin blockchain. After a signature from Alice, the *new state of the channel is the valid state*.



Pending					
✓	₿	10	₿	✓	
Tx1	✓	9.9	→	0.1	✓



The transaction is, at any given point in time, immutable and the conditions described in it must be satisfied before any change can occur: either a week needs to expire, or the transaction needs to be broadcast by either Alice or Bob to finalize the latest distribution: 0.1 - 9.9 BTC.

4. SENDING THE SECOND TRANSACTION

Alice sends a new article three days later and it's up to Bob to send a new 0.1 BTC. Seeing as it's not possible to alter an existing transaction and Bob is unable to send another 0.1 into the same transaction as before, he generates a new one which says: "From the multi-sig address with 10 BTC I'm sending 0.2 BTC to Alice and 9.8 to me" and another "If Alice doesn't sign and broadcast this state within a week, I get all 10 BTC".



Pending					
	✓	₮	10	₮	✓
Tx1	✓	9.9	→	0.1	✓
Tx2	✓	9.8	→	0.2	



Alice now has the opportunity to either sign and broadcast the new transaction, thereby taking 0.2 BTC and finishing the work relationship, or continue with it, send more articles, and get more of such incremental transactions. She can also stay offline or idle for a week and lose it all.



Pending					
	✓	₿	10	₿	✓
Tx1	✓	9.9	→	0.1	✓
Tx2	✓	9.8	→	0.2	✓



5. DOCKING PAY

In the second article, Alice accused a competing company of plagiarism but didn't do enough fact checking. This harmed the company's reputation and Bob decides to dock 0.05 off her pay.

Bob generates a new transaction which says "From the multi-sig address with 10 BTC I'm sending 0.15 BTC to Alice and 9.85 BTC to myself" alongside a transaction saying "If Alice does not sign and broadcast this within a week, I get all 10 BTC".



Pending					
	✓	₮	10	₮	✓
Tx1	✓	9.9	→	0.1	✓
Tx2	✓	9.8	→	0.2	✓
Tx3	✓	9.85	→	0.15	



Alice now has the following options:

- tolerate the pay cut and sign the transaction, continuing with the work
- broadcast the last signed transaction, the one for 0.2 BTC and get more money than Bob is currently offering, but this would finish their relationship
- not respond and lose it all.

5A. ALICE CHEATS

Let's assume that because of the second article's incident the relationship has been permanently damaged and Alice wants out but she already signed the 0.15 BTC transaction. Can she decide to sign the 0.2 BTC one instead that still on the table?



	Pending					
	✓	₮	10	₮	✓	
Tx1	✓	9.9	→	0.1	✓	
Tx2	✓	9.8	→	0.2	✓	
Tx3	✓	9.85	→	0.15	✓	



The LN is set up in such a way that signed (but not sent) transactions are ranked by age. Trying to send an older signed transaction is a punishable offense in the network which sends all the money from the multi-sig to the non-offending party.

This security system makes sure only the latest signed transaction can be broadcast, but has other implications as well: it requires the users to be constantly online to have their nodes be able to communicate with one another. To prevent users from having to be online at all times, the concept of Watchtowers was introduced, which we'll explain in another post.

5B. BOB CHEATS

Let's assume Bob is the cheater. Alice sent the third article, but Bob decides to punish her further by not paying at all. Alice has no idea whether or not she'll ever be paid again, and wants to end the relationship. She can simply broadcast the last signed transaction and the 0.15 BTC will be sent to her. She'll only lose as much money as she spent on writing a single (third) article.



	Pending					
	✓	₿	10	₿	✓	
Tx1	✓	9.9	→	0.1	✓	
Tx2	✓	9.8	→	0.2	✓	
Tx3	✓	9.85	→	0.15	✓	



But what if this further offends Bob and he decides to send in another transaction in which he says “from the multi-sig address I’m sending 0 to Alice and 10 to me, and if she doesn’t sign and broadcast within a week it’s all mine anyway”.

This is where the aforementioned concept of signed and broadcast transactions comes into play. For an LN transaction to be valid, it needs to be signed by both parties. If it’s not signed, the last signed transaction is valid for broadcasting.



	Pending				
	✓	₮	10	₮	✓
Tx1	✓	9.9	→	0.1	✓
Tx2	✓	9.8	→	0.2	✓
Tx3	✓	9.85	→	0.15	✓
Tx4	✓	10	→	0	



This makes Alice safe from ultimate fraud.

5C. ALL GOOD

The alternative that's best for everyone involved is for Alice to do her job perfectly and Bob to send incremental transactions. Let's say that Alice had another mistake or two and ended up earning 9.9 BTC, so that's what Bob's latest transaction says: 9.9 BTC to Alice, 0.1 BTC to him. Both are happy with this agreement and Alice signs *and broadcasts* this transaction.



	Pending				
	✓	₿	10	₿	✓
Tx1	✓	9.9	→	0.1	✓
Tx2	✓	9.8	→	0.2	✓
Tx3	✓	9.85	→	0.15	✓
	...				
Tx101	✓	0.1	→	9.9	✓



The following occurs:

- Alice signs and broadcasts the transaction and it's written to the blockchain.
- The transaction costs a transaction fee, docked from the total being paid out. Seeing as it's only a single tx now, and not 100 of them, it's only the cost of a single tx.
- Within ten minutes to an hour, the transaction will be finalized on the Bitcoin blockchain and the money will be transferred, the channel closed.



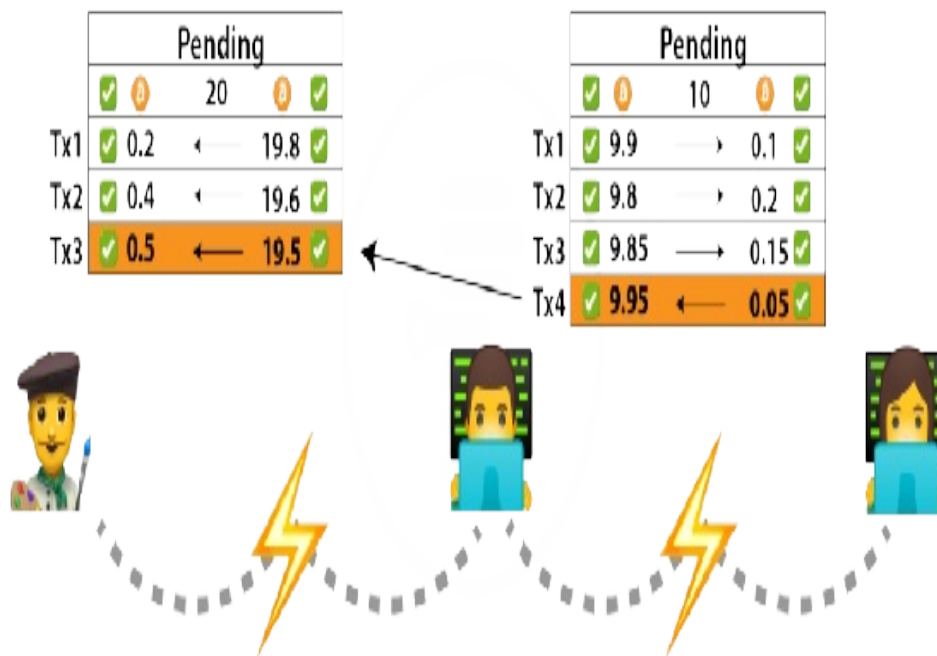
It's important to note that, while this process sounds incredibly complex, they're working on masking this complexity and hiding it from users. The payment channels are expected to be

invisible in end-user software.

Network and Routing

In the example above, Alice and Bob have a clearly defined relationship with an end goal in mind. In theory, payment channels will stay open indefinitely because of *routing*. The idea is to connect several nodes and then route payments from one to the other, allowing payments to be sent to nodes you're not connected with through nodes that are. The software should be able to find a way to get there.

For example, if Alice is working with Roko, a graphic designer, and Bob is working with him too, let's assume Alice will need a 0.1 BTC graphic design work done. She can send this to Roko through Bob as long as Bob has at least 0.1 BTC in the channel open towards Roko, and Alice has 0.1 BTC in her balance towards Bob. Bob is running a mediator Lightning Network node in this case.



The problem is that if someone already paid Roko through Bob and Bob no longer has a 0.1 BTC balance towards Roko, no one else can pay Roko through Bob; they need to find another route. [This video explains it well.](#)

This is an argument for keeping channels open indefinitely and permanently locking funds into the LN, because it's assumed LN's popularity will be so big anyone will be able to pay anyone anywhere. But all this has its own set of problems which we'll cover in another post.

Conclusion

In essence, the LN is a practical solution to some problems that currently *don't exist* in the Bitcoin network. Does this make it an efficient way of scaling Bitcoin? We think not, because some problems seem impossible to solve. We do,

however, feel like the concept is interesting and are looking forward to seeing where the developers can take it.

Chapter 4: The Top Nine Uses for Blockchain

BY MATEJA KENDEL

Blockchain is often being shoehorned into projects it shouldn't be a part of, so it's hard to find out how viable it is long-term, and whether or not it can actually end up being *used* for something.

With that in mind, we've compiled this list of top nine uses for blockchain that are not only viable, but also necessary and highly probable.

1. Financial Freedom

The first and most obvious use is decentralization of money. Bitcoin's original purpose was a cheap and easy but trustless way to transfer monetary value to anyone in the world at any time of day. As that vision slowly faded away, others took up the mantle and now there are many cryptocurrencies that do just that. In underdeveloped countries that skipped landlines and moved straight to smartphones with data connections, e-banking is now possible as long as those people are willing to be their own banks.

But how viable is this? If large corporations like Facebook, Google and Twitter can be commanded to ban cryptocurrency-related advertisements, the unbanked will need jailbroken and rooted phones to install some of the software essential for an entry into cryptocurrency. This might present a hurdle we've yet to conquer. The various app stores are, after all, still highly centralized.

Still, blockchain's ability to not only let people set up their own trust funds (smart contracts with Ethereum) or organize their own savings and retirement funds is something the world won't be able to ignore for long — for better or worse.

2. Decentralized Data

The blockchain running on several thousands of machines all over the world makes it possible to do wonderful things with data. If you suddenly get the ability to run code, store data, or process mathematical operations on every machine in the world, you can easily find scenarios where this might become useful.

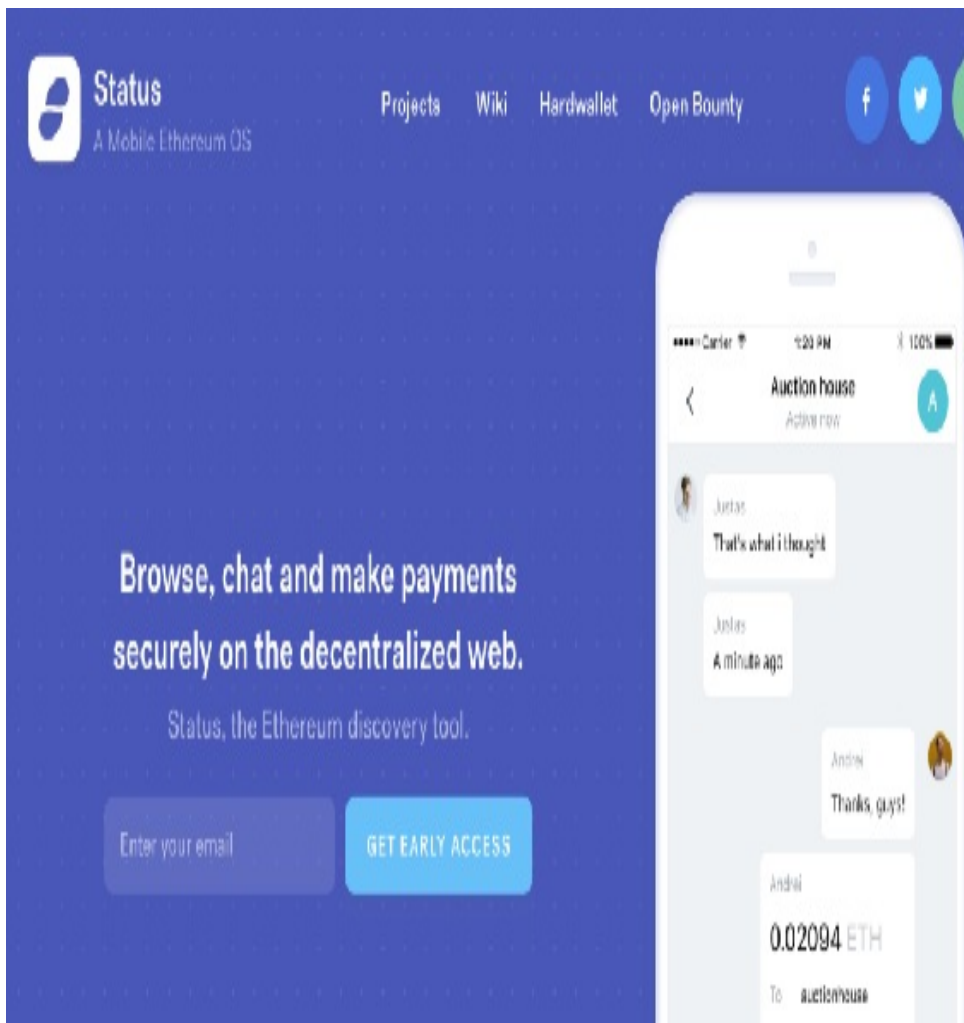
Projects like Golem and iExec make it possible to distribute compute-heavy tasks like rendering graphics or executing CPU instruction respectively, by splitting tasks into thousands of smaller pieces and returning a verified solution to each, forming the master solution at the point of origin. Imagine not having to rent a supercomputer and being able to map the human genome in seconds using distributed computing and paying the executors in pre-purchased tokens!

The potential for data storage is another big deal in blockchain. While we do have Bittorrent and similar solutions, there's no incentive for keeping files *seeded* and thus recoverable for a long time. Projects like IPFS, Siacoin, Filecoin, Storj, Maidsafe, Datacoin and others are developing ways to store files across multiple machines in an encrypted and private but incentivized and safe manner. Siacoin, for example, pays users to rent out their extra hard drive space. The users are paid in Sia tokens and right now, though file access is slower than S3 and similar private solutions, it's cheaper and shows a lot of long-term potential. Even for databases, there's BigchainDB and similar options.

Finally, games. With all the data and files being distributable and with the security of the Ethereum blockchain backing the network, many games with collectibles have cropped up. There are still challenges to be resolved, but with scaling solutions like Loom's dappchains, high-throughput games are finally becoming a possibility.

3. Decentralized Internet and Chat

By utilizing the decentralized data aspect mentioned above, we can store websites on various computers around the world and retrieve them in a decentralized way.



Where up until now we needed a central server (or cluster thereof) to host websites or transmit messages online, Ethereum's Swarm and Whisper protocol is making blockchain-based chatting and distributed file retrieval possible.

Status and Toshi, the first platforms to fully dive into these concepts, have grown into platforms for launching decentralized applications. They are, in effect, their own app stores with blockchain APIs exposed, but which can also communicate with the smartphone they're installed on. This makes decentralized fetching of websites and sending of

messages possible.

A Facebook or YouTube which can no longer censor content based on region, or a chat platform which can no longer be shut down when a revolt happens: all this is now a reality. It's a UX nightmare and we're working on this, but it's here and it works.

For an example of a decentralized web application running fully on a blockchain, see [DelegateCall](#).

4. Immutable Records

Another use case that's rather straightforward is immutable records. This can be done in two ways: verification style and full-store style.

Verification is when a user keeps the original of a file somewhere, preferably signed with a private key of some authority or actor or provable identity, and then hashes it. This hash is stored into the blockchain, and when re-hashed the file can be easily verified for accuracy. However, this presumes the existence of the file somewhere outside the blockchain.

Full-store style is when the file itself is encrypted with a private key and stored wholly on the blockchain somehow — perhaps using one of the decentralized data solutions above. Then, the file remains forever accessible *and* verifiable.

This is useful in a whole bunch of areas. for example, criminal records shouldn't be wipeable, and you should be able to

identify an offender at any time in any jurisdiction, even automatically when crossing the border. The same goes for health records: there's no reason for them not be interchangeable and verifiable by every health institution on earth. Encrypting a health record is trivial, and when required it can be decrypted on the spot with the private key of a patient which can be something as simple as a fingerprint or iris scan.

Ownership of real estate or deeds is also a perfect use case: the ownership can change hands, in which case the new owner is written after the old one with the private key of the old one, signifying a legal exchange, while an auto-auditable paper trail remains.

5. Automation of Civilization

Referencing the above auto-auditing, we can use the blockchain to remove a big part of the middleman industry. Real estate agents, resellers, a lot of lawyers and accountants ... even tax inspectors. If a country transitions to a well developed central-bank-issued cryptocurrency, all the transactions will immediately become auto-auditable with 100% accuracy. This means no tax reports, no inspections, no fines, no lost money trails. It's a cashless society 2.0.

6. Democracy (Voting)

Elections are often plagued with corruption and scandals, not to mention the long queues and the younger generations not voting because it's simply too impractical. Moving elections

onto the blockchain enables them to be not only fair and verifiable, but also opens the door to merit-based voting, which many say is the only thing able to save democracy — whereas right now everyone has one vote ...

In a merit-based democracy, which is easy to automate with blockchain, a farmer could have five times the vote power of a lawyer as far as crop dusting limits and regulations are concerned. What's more, this lets the government ask the people to vote for anything as often as they like, without enormous campaign costs and the parades that follow: a simple notification on an electronic device, and the user could reliably vote from the comfort of their own home.

7. Loyalty Points

A more trivial but no less popular (and widely in use!) purpose for the blockchain is tokenization of loyalty points. With the almost free infrastructure of the blockchain in place and scaling solutions like Loom's dappchains in the works, many companies are transitioning to a loyalty point system where users collect tradable tokens.

If I have three Domino's tokens and my friend has five, why wouldn't I be able to send my three tokens to her so that she can order our free eight-token pizza online by simply sending the tokens to Domino's Ethereum address?

8. Royalty and Proof-of-authorship

authorship

Building on the aspect of immutable records, blockchain can also digitize ownership of creative content. By hashing a digital work like a digital painting, a song, a movie, or anything else and uploading it to the blockchain, you can prove that you're the original author of said work — irrevocably. This technology is fully admissible in court as proof, because the blockchain is immutable and ultimate in its truth about a certain state. And that state says that you proved yourself as owning said work at a moment in time.

One example of a running application that does exactly this is Po.et.

9. Company Management

Last but not least, there are DAOs. A DAO is a decentralized autonomous application, and kind of ties into the Democracy point above. Where a government could use the blockchain to tally votes, a company could replace its entire board by blockchain users. This is what DAOs do today.

A DAO will distribute tokens to its investors, and these tokens will be used as voting rights. Any decision the company needs to weigh will be voted on by token holders. For example, imagine an investment DAO which can invest into other projects using the Ethereum collected during the investment phase. A user suggests that project YetAnotherICO deserves an investment. This user can send this in as an official suggestion (all automated) and others can vote on their call

within a timeframe. Depending on how many tokens there are in an address, that's how much weight that address' vote has. If the majority votes in favor, the suggestion goes through.

An interesting use case for a modern DAO is AmpNet, which lets people sell electricity peer to peer, while a famous failed case (due to improperly written code) is the DAO incident.

Bonus: Supply Chain

You may have heard many pundits mention the supply chain as a good use case for the blockchain. The idea is that a given good is tracked from origin to destination using QR codes and sensors, so a customer in a store can simply scan a product's QR code and see its whole origin trail (hence, the company OriginTrail).

However, this is not a viable option as long as there are humans in the pipeline. Humans are, by default, fallible. They are the weakest point of any automated system, and if a truck driver whose fridge has blockchain-connected temperature sensors on the crates of tomatoes in the back and the driver's salary depends on his success rate, you can be sure that most drivers will rather stop at a gas station, get ice, and pour it directly onto the sensors when the AC fails than let a robot decrease their salary.

Only when humans have been completely removed from both the production and transportation of goods will we be able to use blockchain for the supply chain.

Conclusion

Blockchain is a viable solution to many problems, but an ill-fitting one for many more. When considering whether or not your project needs the blockchain, be sure to consult blockchain experts *before* you proclaim your project blockchain based.

Don't lock yourself into a technology for marketing reasons. Really study whether or not it's viable. Even if it is, you might be ahead of your time in your problem solving (like with supply chain), so playing it safe and waiting is better than rushing.

Chapter 5: Introduction to Ethereum: A Cryptocurrency with a Difference

BY BRUNO ŠKVORC

This introduction to Ethereum was originally published at Bruno's Bitfalls website, and is reproduced here with permission.

Ethereum, launched in 2015 by Vitalik Buterin, is a special blockchain with a special token called Ether (the ETH symbol in exchanges).

Ether is used as fuel (literally, *gas*) to power the Ethereum network and execute *smart contracts* (see below). Ether is paid to miners (the people running the network) in *gas cost* to make the features below possible, and thus isn't a **cryptocurrency** as much as **cryptooil**.

Here's how it's different from bitcoin, and what makes it stand out in the world of cryptocurrency.

Finality

Bitcoin has a theoretical limit of 21 million bitcoins ever produced, which makes it a deflationary currency: its total amount in circulation shrinks over time as people lose access to their wallet, stockpile it, *etc.*

Ether has no limit, but does have a fixed ratio of production. Currently, a little over 15.5 million Ether is mined every year, which comes down to 5 ETH every second — far more than bitcoin's current rate of 25 BTC every 10 minutes or so. After Ethereum switches its mining type — more accurately known as **consensus type** — to PoS (see below), the production rate will decrease dramatically, approaching zero.



All this will make Ethereum's value stop growing disproportionately, thus reaching market price stability which lends itself perfectly to the practical uses described below.

Smart Contracts

Bitcoin supports simple scripting. For example, you can write a small program with bitcoin which will support transactions that have multiple addresses as input, or that need multi-sig (a signature from several people before funds are released). The language used for this is not *Turing complete*, however, because it doesn't have *loops*. (If you don't know what this means, it doesn't matter.)

Ethereum programs, on the other hand, are programmed in Solidity — a language that *is* Turing complete, thus allowing for any kind of program to be written in it, given enough resources (within reason). When a program is written in Solidity, it needs to get sent to the blockchain, which costs *gas*, paid in Ether. The bigger and more complex the program, the more expensive it is to *deploy* it to the blockchain. Thus, inefficiency costs money; it's in the interest of everyone involved to keep those programs as small as possible.

Vitalik himself best describes the smart contracts with a vending machine analogy:

A vending machine [...] basically implements the conditions of some kind of an agreement. And the conditions of the agreement here are simple. You put \$2 in, water comes out.

You do not put \$2 in, water does not come out. If you do not put \$2 in but water does come out then that's bad. And a vending machine is basically an encoding of this set of rules, that also comes with a mechanism that keeps it at least kind of secure. Secure enough for \$2 water bottles.

When we pay an amount of Ether into a smart contract, that smart contract can then decide what to do with this Ether: send it to address A based on one condition, address B based on another, lock it in place for a period of time, refund it, move it around based on external input, trigger external output based on this Ether, *etc.*

A practical example would be replicating Kickstarter. Kickstarter is a site which lets creators gather funds for their projects before they're launched. The basic rule is, if a given amount of money is exceeded in a given number of days, the project was a success and the money can be released to the makers. Otherwise, the money is refunded. This simple condition is very easy to replicate with smart contracts, eliminating human error, greed, and the middleman from the equation, offering up a truly decentralized way of fundraising.

Applications built with smart contracts are called decentralized apps or **dapps**.

What are ERC20 Tokens?

A very important feature of Ethereum is the ability to create new tokens on the Ethereum blockchain. Tokens are a

“cryptocurrency” of sorts built with specific smart contracts, and used like any other — sending to and from addresses. These tokens are sent to Ethereum addresses, not addresses of a new cryptocurrency’s blockchain. It’s because of this that tokens aren’t really a cryptocurrency per se, but the result of logic executed via a smart contract. Saying a token is a cryptocurrency is actually just as inaccurate as claiming that a program is a programming language.

A token can be a concert ticket, loyalty points in a shop, in-game currency, *etc.*

As more and more tokens started to appear, their format was standardized into ERC20 — a set of rules on how to develop them to make them easily consumable by various exchanges and systems. This means that all ERC20 tokens have some common features like a ticker symbol for exchanges, an icon, *etc.* making them easy to list on various websites by reusing the same code used for a previous ERC20 token.

Creating ERC20 tokens in combination with smart contracts is Ethereum’s revolutionary feature which is poised to completely change the way we do business. Tokens make autonomous companies possible, they allow for partial purchases of digital goods or land, they even allow for the creation of autonomous cars which drive themselves, pick up customers, drop them off, collect payment, and effectively *pay themselves off*. The number of potential use cases is so large we haven’t even scratched the surface.

POS

The third biggest difference from bitcoin is that Ethereum will be moving to an alternative way of *mining* called PoS (Proof of Stake) rather than PoW (Proof of Work).

With bitcoin, a unit of proof of work is the hash gained by doing massive calculations. We have a separate post about PoW and PoS, but suffice it to say here that a PoS system isn't wasteful as far as electricity goes, which is an important feature considering China's mining dominance.

PoS has its own problems, of course. In this system, a user of Ethereum trying to be a **node** (this new type of miner) **stakes** their Ether to guarantee the correctness of their calculations. If they try to game the system and fake some calculations that aren't valid, the other nodes will call the node out, and the stake gets lost. Otherwise, the staker gets their stake back after a few months (yes, months!) plus any fee earnings they scored from transactions or from executing smart contracts. The size of the stake deters malicious actors; the initial stake is said to be 1000 Ether.

Conclusion

To sum up, Ethereum differs from bitcoin in the following ways:

- it'll soon transition to Proof of Stake instead of Proof of Work
- it'll soon be quantum-proof
- it supports the creation of sub-tokens on the network

- it supports custom logic in the blockchain for full financial automation (smart contracts) — example [here](#).

There's another simile we like to mention when talking about BTC vs ETH:

- Bitcoin is gold. It's complicated to obtain, expensive and slow to transfer, but highly deflationary and finite ([for now](#)). This makes it a good long-term investment, depending on who you're asking.
- Ethereum isn't silver to bitcoin's gold as many would assume. Ethereum is oil. Other products are made with the help of Ethereum, and ERC20 tokens correspond to plastics, make-up, paint, rubber ... Ethereum is the base of an entire new industry, and while there are several alternatives (Tezos, EOS, Rootstock, NEO), none of them have a developer or user community as wide and strong as Ethereum does.

Chapter 6: A Deep Dive into Cryptography

BY BRUNO ŠKVORC

This Deep Dive into Cryptography was originally published at Bruno's Bitfalls website, and is reproduced here with permission.

The media is jam-packed with content about cryptocurrency and everyone is raving about the importance of public and private keys. You've heard of encryption, but do you know what it actually is and how it works?

This post will take you back to the basics and explain encryption, describe the different types, and demonstrate algorithm examples, all in a newbie-friendly way. If you've ever wanted to understand this but it seemed too complicated, you'll love this post.

Cryptography

Cryptography is, in a nutshell, turning a message into such a format that it makes sense only to the recipient, and not anyone who might get their hands on it in between.

What is the problem we're trying to solve?

Suppose there are two people who want to exchange encrypted messages, regardless of the communication channel they're using — letters, SMS, email ... They first have to come to an agreement about a set of rules to apply when encrypting or decrypting the message. These rules are a set of one or more *functions* which must have a *counterfunction*. That is, if a given function is used to encrypt a message, then there must exist a counterfunction to decrypt it. In mathematical parlance, such a function is **bijective**.

The sender of the message applies a function to a set of information and gets an encrypted version of this information which can then be sent to the recipient. The recipient applies the counterfunction to extract the true information out of the encrypted version of the message. If someone in the middle intercepts this communication but they don't have the counterfunction with which to decrypt the message, they'll be unable to read it.

ALICE AND BOB

Let's make this clearer with the help of a trivial example. Suppose Bob wants to send an SMS containing "I LOVE YOU" to Alice, but can't risk it being seen by someone else. (Someone picking Alice's phone up would see the message.) To successfully exchange encrypted messages, Alice and Bob need to come to an agreement about the way of encrypting/decrypting messages. Let's assume this is the agreement:

Every letter in the message will be replaced with the double-digit index number of that letter in the English alphabet. “A” will be “01”, “B” will be “02”, *etc.* “00” will indicate a space character. This is their encrypt/decrypt function, and it’s bijective.

code: symbol	code: symbol	code: symbol
00: space	09: I	18: R
01: A	10: J	19: S
02: B	11: K	20: T
03: C	12: L	21: U
04: D	13: M	22: V
05: E	14: N	23: W
06: F	15: O	24: X
07: G	16: P	25: Y
08: H	17: Q	26: Z

Bob will therefore be sending the message:

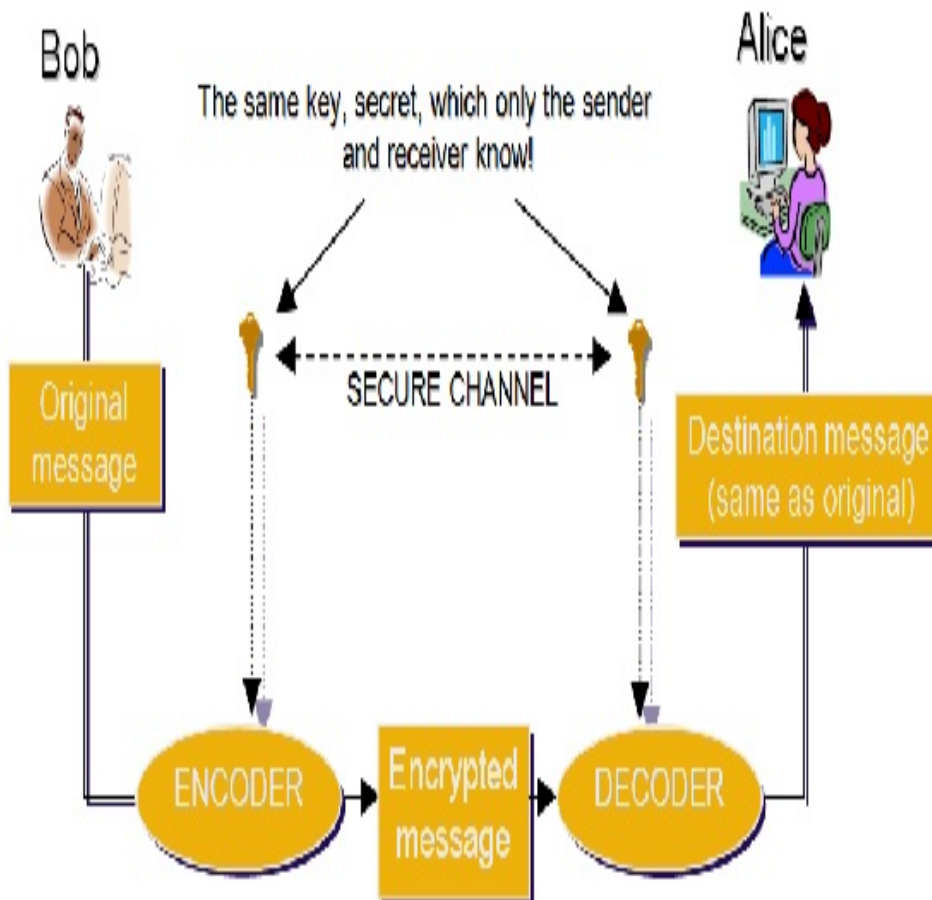
```
09001215220500251521
```

Should someone intercept this message, it won’t make much sense to them. Their love will remain a secret. Alice, on the other hand, will be able to easily decrypt it and blush.

Symmetric Encryption (with a Private Key)

Private Key

The above outlined approach is called **symmetric private key encryption**. “Symmetric” indicates that the message can be encrypted and decrypted using the same key (secret). The key (secret) is actually the letter-to-number-to-letter replacement function we described. It looks a little like this:



A system like this might seem perfect at first glance. Of course, a more complex bijective function is required to make this communication truly secure. The system as such is perfectly safe for as long as Alice and Bob can meet and arrange a method of encryption/decryption beforehand. But

what if, like in most cases today, the people communicating aren't actually in close physical proximity, or maybe don't even know each other? How can they safely exchange a secret key without risk of it falling into the wrong hands?

This is the biggest downside of such a shared key encryption: there has to be a pre-established secure channel via which to exchange the key.

A POSSIBLE SOLUTION

It's clear that, unless a secure channel already exists, it's nearly impossible to safely exchange the private key. If the channel did exist already, then there's no need for another one.

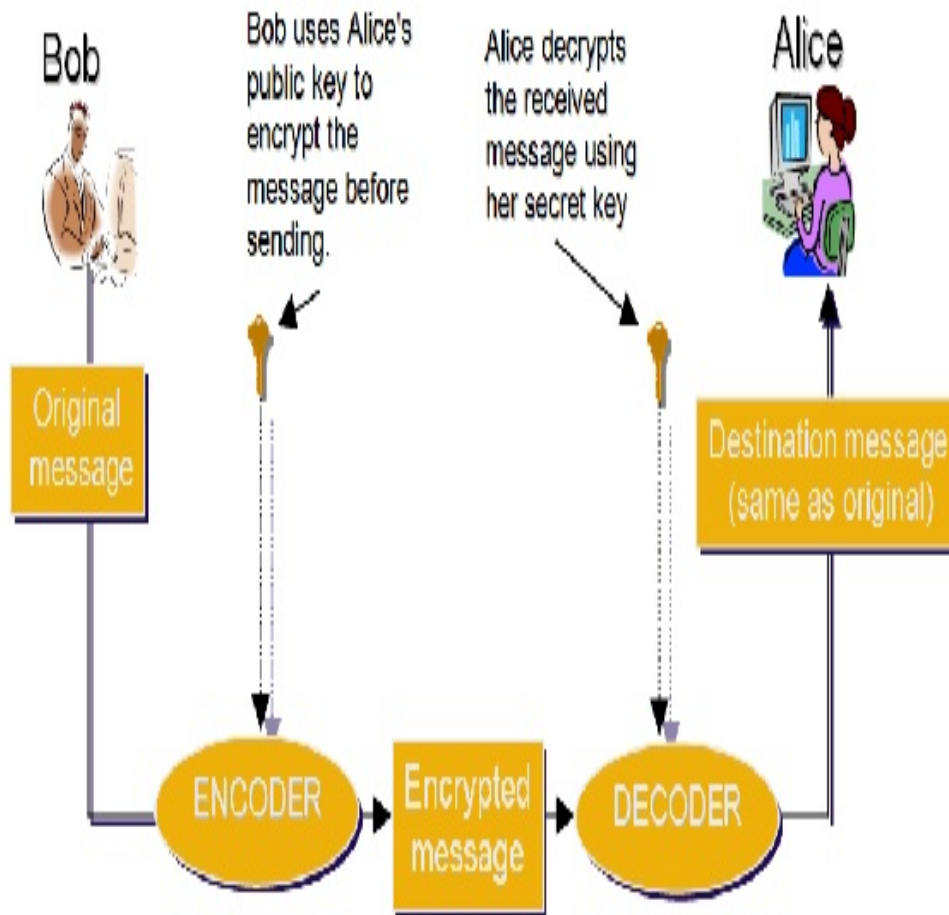
The solution is not to find a safe way of exchanging the key, but to eliminate the need for such an exchange altogether. This can be accomplished by adding another key into the mix. One of them would be used only for encrypting, the other for decrypting.

The encrypting key could be available to everyone. In fact, it *must* be available to everyone, because without it it's impossible to encrypt messages and send them to the recipient. This key is called the *public key*.

The other key is used only for decrypting and shouldn't be sent to anyone. Only the recipient of encrypted information has it, and we call this one the *private key*.

Asymmetric Encryption (with a

Asymmetric Encryption (with a Public Key)



Looking at the image above, we can see that there's no need for a secure channel to exchange keys through. The asymmetry is in the fact that it's impossible to encrypt and decrypt the message with the same key. A separate one is needed for each action.

If Bob wants to send Alice an encrypted message, he has to have her public key. This public key could be given to him directly by Alice, or she could just publish it on her website

where anyone who wants to send her an encrypted message can find it. When Alice receives the message encrypted with her public key, she uses the private key to decrypt it, which makes the message readable again.

If Alice wants to send a reply, she now needs Bob's public key. The procedure is identical: she encrypts the message, sends it, and only Bob can decrypt it with his private key.

It's important to note that to make this kind of communication feasible, the keys need to be generated with procedures complex enough to offset any computer's ability to guess them for an unreasonable amount of time.

HOW DOES IT WORK?

If you don't fully understand the terms in the section below, don't worry. Read through them superficially; they'll be explained on an example immediately afterwards.

To make it impossible to decrypt a message with only the public key, or to derive the private key from the public key, one-directional mathematical functions are used. A one-directional function is such that $f(x)$ can be calculated for any x but not the other way around. For example, if we know the *sum* is 950, we can't guess which numbers we've summed to get it because the number of possible combinations is infinite.

There are many algorithms to calculate such functions. We'll demonstrate one such algorithm below: it's called the RSA

algorithm, based on the initials of the names of its creators in 1977 (Ron Rivest, Adi Shamir, Leonard Adleman).

1. Select 2 primary numbers — numbers divisible only by 1 or themselves. For example:

1. $p = 61$

2. $q = 53$

2. Multiply them $n = p \times q$:

$$n = 61 \times 53 = 3233$$

3. Calculate the lowest common multiple $\lambda(n) = (p-1) \cdot (q-1)$:

$$\text{lcm}(p-1, q-1) = \text{lcm}(60, 52) = 780. \text{ This can be done with the algorithm presented [here](#).}$$

4. Pick any number between 1 and the multiple calculated previously, so that this number is relatively prime or coprime to the initial two numbers.

Numbers are coprimes if the only positive number that divides both of them is 1. In this case, that's $e = 17$.

5. Calculate the modular multiplicative inverse of $e \pmod{n}$. We look for d such that:

1. $(d \times e) \pmod{n} = 1$

2. $(d \times 17) \pmod{780} = 1$

3. $d = 413$

6. These calculations produce all the necessary components of a public and private key set. The public key is the pair (n, e) and the private key is (n, d) . Thus, the public key is $(n = 3233, e = 17)$. The public key serves to encrypt the messages via the following formula:

$$c(m) = m^{17} \pmod{3233}$$

The private key is $(n=3233, d=413)$. It's used to decrypt a message:

$$m(c) = c^{413} \pmod{3233}$$

Now that we have the encryption and decryption functions, we can come back to our Alice and Bob scenario. Since these functions can only process numbers, we need to turn letters into numbers first. The ASCII table — a standard used by computers when working with letters on computer systems — can come in handy. Let's only focus on uppercase letters here.

code: symbol	code: symbol	code: symbol
32: space	73: I	82: R
65: A	74: J	83: S
66: B	75: K	84: T
67: C	76: L	85: U
68: D	77: M	86: V
69: E	78: N	87: W
70: F	79: O	88: X
71: G	80: P	89: Y
72: H	81: Q	90: Z

For every ASCII code from Bob's message, we need to calculate the $c(m)$. As the message is "I LOVE YOU", the ASCII of the first letter is 73. $c(73)$ is:

$$c(m) = m^{17} \bmod 3233$$

$$c(73) = 73^{17} \bmod 3233$$

$$c(73) = 47477585226700098686074966922953 \bmod 3233$$

$$c(73) = 1486$$

Let's also calculate the rest.

Space: $c(32) = 32^{17} \bmod 3233 = 1992$

L: $c(76) = 76^{17} \bmod 3233 = 2726$

O: $c(79) = 79^{17} \bmod 3233 = 1307$

V: $c(86) = 86^{17} \bmod 3233 = 1906$

E: $c(69) = 69^{17} \bmod 3233 = 28$

Y: $c(89) = 89^{17} \bmod 3233 = 99$

U: $c(85) = 85^{17} \bmod 3233 = 2310$

The encrypting function calculates the mod 3233 (division remainder when dividing with 3233) so the result of a letter's encryption can't be more than 3232, which in turn means four is the maximum number of digits in each letter's encrypted version. Therefore, we *pad* each number with zeroes on the left side: 1486, 1992, 2726, 1307, 1906, 0028, 1992, 0099, 1307, 2310.

The full sendable message is:

```
1486199227261307190600281992009913072310
```

Alice has her private key, and can use it to decrypt this. She'll use the previously defined inverted function $m(c) = c^{413} \bmod 3233$ where c is the encrypted message. The inverted function shows that mod 3233 is being calculated, and that every letter can be at most 3232 and have four digits. Thus, Alice knows the message needs to be divided into sets of four, the leading zeros of each being meaningless:

```
1486 1992 2726 1307 1906 (00)28 1992 (00)99 1307  
2310
```


Let's decrypt 1486:

$$m(c) = c^{413} \bmod 3233$$

$$m(1486) = 1486^{413} \bmod 3233$$

$$m(1486) = 1,1060335282256977039647849058382e+1310 \bmod 3233$$

$$m(1486) = 73$$

We got the number 73, which according to the ASCII table matches the letter "I". By following the same process, we can decrypt the rest of the message:

$$m(1992) = 1992^{413} \bmod 3233 = 32, \text{ ASCII } 32 = \text{razmak}$$

$$m(2726) = 2726^{413} \bmod 3233 = 76, \text{ ASCII } 76 = \text{L}$$

$$m(1307) = 1307^{413} \bmod 3233 = 79, \text{ ASCII } 79 = \text{O}$$

$$m(1906) = 1906^{413} \bmod 3233 = 86, \text{ ASCII } 86 = \text{V}$$

$$m(28) = 28^{413} \bmod 3233 = 69, \text{ ASCII } 69 = \text{E}$$

$$m(1992) = 1992^{413} \bmod 3233 = 32, \text{ ASCII } 32 = \text{razmak}$$

$$m(99) = 99^{413} \bmod 3233 = 89, \text{ ASCII } 89 = \text{Y}$$

$$m(1307) = 1307^{413} \bmod 3233 = 79, \text{ ASCII } 79 = \text{O}$$

$$m(2310) = 2310^{413} \bmod 3233 = 85, \text{ ASCII } 85 = \text{U}$$

Let's re-iterate why asymmetric encryption is better than symmetric encryption. With symmetric encryption, there's only one key which is used to both decrypt and encrypt messages. If participants want to communicate securely, they first need to exchange that key safely. If they're physically remote, this becomes a big problem. With asymmetric encryption, there's a public key which we can freely give out, and which people can use to encrypt messages intended for us. Our private key remains secret, and is used only to decrypt the

messages we receive. There's no need for a secure channel via which to exchange a secret key, making asymmetric encryption much safer than symmetric encryption.

But would it be possible to calculate the private key from the public key?

GUESSING THE PRIVATE RSA KEY

Let's look at the functions one more time:

- Encrypt: $F(x) = x^e \bmod (p \times q)$
- Decrypt: $F^{-1}(c) = c^d \bmod (p \times q)$

The encryption function — that is, the $(p \times q, e)$ pair — represents a public key. If we know this pair and we want to calculate the private key, we need to find the numbers p and q . That means we need to factorize the product of $p \times q$. If we assume the numbers p and q are 1024 bit numbers, then their product is a 2048 bit number — a number with 617 digits if represented as a decimal number. To factorize such a big number, even today's most powerful supercomputer would need an absurd amount of time, making the process a mathematical impossibility.

It's not *technically* impossible to factorize the number. There are special algorithms developed for that exact purpose, and the most efficient one right now is GNFS (General Number Field Sieve). It's especially handy for factorization of numbers with more than 110 digits. The table below lists the time required to factorize a big number expressed in MIPS (*Million*

Instructions per Second). Accordingly, one MIPS-year is the number of computer operations executable in one year by a computer with the power of 1 MIPS. That number is 3.1536×10^{13} .

Key length	MIPS-years needed to factorize it
512-bitni	30.000
768-bitni	200.000.000
1024-bitni	300.000.000.000
2048-bitni	300.000.000.000.000.000.000

If we assume that the applied algorithm multiplies 1024 bit numbers, producing a 2048 bit product and that today's most powerful personal computer CPUs have around 300,000 MIPS, the number gets pretty high. Even today's most powerful quantum computers with 100 million MIPS of power (which are a rarity, if they even exist), would need 3,000,000,000,000 years to factorize a 2048 bit number.

MERSENNE PRIMES

Given that big prime numbers are the main idea behind the RSA algorithm, it's increasingly more important to find big primes. There's a subclass of primes called Mersenne primes, and they look like this: $2^n - 1$. They were named after a French friar who was the first to (wrongly) identify 11 of them. There's a big movement around finding the biggest possible Mersenne primes these days. (You can find the details at Mersenne.org.) The last (49th) and biggest Mersenne number

so far was found on January 7th, 2016. It's $2^{74.207.281}-1$ and has 22,338,618 digits. The one before it was found on January 1st 2013: it was $2^{57.885.161}-1$ and has almost 5 million fewer digits than the 49th number.

One more fun fact about prime numbers: the EFF, Electronic Frontier Foundation, has bounties on big primes. Their [website](#) lists these bounties, and shows that the \$50,000 reward for finding a 1-million-digit prime and the \$100,000 reward for finding a 10-million-digit prime were given away, with the \$150,000 and \$250,000 rewards for 100 million and 1 billion digits respectively are still pending. Got a good algorithm idea? Maybe you can be the one to find the number!

Asymmetric Encryption and Bitcoin

So what does all this have to do with cryptocurrency, other than sharing the “crypto” name?

While the below applies to almost all cryptocurrencies, let's take the most famous one — Bitcoin — as an example. Bitcoin also uses a public/private key pair. The public key — or at least a form of it — is the address to which you can send some Bitcoins. Like with encrypted messages where a public key is needed so that someone can send us a message, with Bitcoin a public key is needed so that someone can send us money. On the other hand, a private key lets us confirm, approve, and perform a transaction with which we send some of our bitcoins from our address (account) to someone else's

public key (address).

The idea is similar, but Bitcoin has a different approach at calculating the private and public key. The private key of Bitcoin is a 256-bit number. We're generally dealing with a big number randomly picked from a set of 2^{256} numbers. That's a little over 10^{77} possible choices. This might not seem like much, but considering the estimation that there are 10^{80} atoms in the whole universe, the size of this number can still make us pause and think. Just counting all those numbers at a speed of one billion per second would take more time than the age of the universe.

So we have a private key which only we know and which can't be guessed by anyone or any computer in a reasonable amount of time. In the Bitcoin protocol, the private key is used to calculate the public key using the ECC algorithm, *Elliptic-curve cryptography*. This algorithm is based on a curve the function of which can be mathematically expressed as $y^2 = x^3 + ax + b$. The result is the public key. This URL and this URL explain the algorithm in detail if you're interested.

With Bitcoin, the public key isn't the address to send money to, but it can be easily calculated using the following formula:

```
address = RIPEMD160 (SHA256 (public key))
```

This URL contains details about the process if you're interested.

Conclusion

CONCLUSION

It's important to remember the difference between encoding and encrypting. Encoding is the process of making the sent message as identical to the original as possible to minimize errors. Encrypting is making the message unreadable to everyone but the intended recipient.

There are several different kinds of encryption, the main ones being asymmetric and symmetric. We covered the former in this article, and explained that in contrast with symmetric encryption, asymmetric encryption introduces a public and private key rather than just one private key shared between the parties, making it possible to securely communicate across insecure channels.

The advantage of asymmetric encryption becomes obvious in cryptocurrency, where the public key is used to receive funds and check balance and transactions, whereas the private key is the only way to actually sign messages and send the tokens.

Chapter 7: 3 Bitcoin Alternatives Compared: Ethereum, Cardano and NEO

BY DAVID ATTARD

If you've been digging into blockchain platforms, cryptocurrencies and related technologies for a while, you'll find that there are endless references to Bitcoin. While this is currently the most popular cryptocurrency using blockchain technology, it's not the only one.

In this article, we're going to have a look at some of the more popular blockchain platforms apart from Bitcoin. Rather than looking at the platform as a cryptocurrency, we're going to look at them from a technology perspective.

Given that there are many platforms out there, with many others coming out on a daily basis, we're taking a look at some of the ones which already have significant traction:

- Ethereum
- Cardano
- NEO

What is Ethereum?

Ethereum is a blockchain-based distributed computing platform which powers the “ether” cryptocurrency.

Ethereum was founded by Vitalik Buterin, crowd funded in mid-2014, and released a year later, making it one of the earlier platforms and cryptocurrencies.

Here is a video by the founder of Ethereum explaining the platform.

While this was initially an alternative to Bitcoin which was looking to advance the technology per se, an exploited vulnerability in the DAO project (a decentralized autonomous organization meant to provide a business model for the platform) caused the Ethereum project to be forked into two separate blockchains, each with its own cryptocurrency, resulting in Ethereum Classic and Ethereum.

Ethereum is today still one of the main cryptocurrencies apart from Bitcoin, but its real technology advancement was that it was the first technology to implement smart contracts.

The real value of Ethereum, therefore, comes not from the currency but from the actual platform.

While Bitcoin uses the blockchain to verify the ownership of the digital currency, the Ethereum platform allows developers to implement truly distributed applications, removing the “centralized” client-server architecture of the internet. It’s the first platform to implement the smart contract concept.

The Ether currency, while also being a tradable cryptocurrency, is actually used to fuel the network. Participants in the network use ether to pay for services performed by the Ethereum network.

In reality, the cryptocurrency is a side-effect (albeit an essential one) of the Ethereum platform.

Given the smart contract concept, Ethereum can be used to build truly decentralized applications.

Ethereum uses the concept of a virtual machine to be able to do work — the Ethereum Virtual Machine (EVM) — which can run the “tasks” required to build a decentralized application.

Ethereum Strengths:

- A very popular platform and cryptocurrency which keeps growing and was one of the earlier technologies. Big early mover advantage.
- Has the possibility to perform work and implement distributed applications rather than just being a cryptocurrency platform.
- Implements useful technology advantages such as smart contracts.

Ethereum Weaknesses:

- Proof of work requires mining, which is energy intensive. This is being solved with a transition to Proof of Stake.
- The DAO hard-fork of Ethereum was, and still is, a contentious issue which split the Ethereum community. Other very expensive bugs have happened, but all of them were user errors.

What is Cardano?

What is Cardano ?

Cardano is another decentralized, fully open-source public blockchain and cryptocurrency project, which powers the ADA cryptocurrency.

The idea behind the project is to improve previous blockchain platforms by developing a smart contract platform which delivers advanced features through a research-first approach.

Essentially, rather than developers working mostly on their own, the concepts are peer-reviewed, making sure the ideas implemented have been reviewed and proven to work *before* being implemented.

Here is a lengthy but interesting overview of Cardano by Charles Hoskinson, CEO of Input Output Hong Kong.

Cardano uses a different approach to “mining” than other blockchain platforms. Rather than requiring energy-intensive resources as proof of work (one of the major drawbacks of Bitcoin and other mining-based cryptocurrencies), the new algorithm eliminates this intensive mechanism, yet it still has been proven to be mathematically secure — to the level of Bitcoin (the most secure platform to date).

This is what’s called proof of stake. A number of blockchain platforms have tried to implement such a solution, and the pioneer of the method — still in operation today — is Peercoin. Cardano uses what’s called Ouroboros to find a “leader”, which is determined not through mining but through a multi-party implementation of a coin-flipping protocol.

This means that the Cardano platform brings all of the benefits of the blockchain platform, but eliminates one of the major disadvantages we've witnessed with blockchain, the insanely wasteful use of energy required for mining purposes.

Cardano was developed by blockchain development firm IOHK (Input Output Hong Kong) and is led by Charles Hoskinson, former co-founder of Ethereum.

The aim of Cardano is to be used as a platform to build financial applications powered by the Ada cryptocurrency.

Cardano provides a blockchain implementation which makes use of all of the advantages of blockchain and decentralized, distributed computing without suffering from the major drawbacks which have been dogging other platforms and major cryptocurrencies.

Cardano is still in the process of being developed.

Cardano Strengths:

- Not intent on speculation of cryptocurrency but more for the advancement of the blockchain technology per se.
- Resolves a number of issues with popular blockchain platforms, such as Ethereum and Bitcoin, by implementing a type of mining which doesn't require much energy.
- Supported by both academic and technology research, such that there's a strong, reliable community around it.

Cardano Weaknesses:

- At the time of writing this article, the technology is still in relatively early

days and is not intended for production. Whether or not it will be able to fulfill its promises remains to be seen.

What is NEO?

NEO is another blockchain designed to build a scalable network of decentralized applications.

The difference and specific strength of this platform is that it supports commonly used programming languages such as .NET, C++, Go, Java and Python. This is done through the creation of a customized version of Docker, called the neoVM, which can run the code on different secure executable environments.

The other differentiating feature of NEO is that it's backed by the Chinese government. While this is something which can go both ways, the backing of such a powerful institution almost guarantees the success of this platform.

NEO uses dBFT (delegated Byzantine Fault Tolerance) as a proof of work mechanism, which is a methodology that's less energy-intensive than Ethereum mining, very similar to Proof of Stake. NEO is also not susceptible to the hard-fork problem which split Ethereum, because the mechanism requires a two-thirds majority rule to operate while the minority gets discarded.

NEO has another real-world advantage. It uses the concept of X.509 Digital Identities to be able to associate real-world identities to the blockchain, which is important to implement

applications which require proof of identity, such as KYC/AML (Know Your Client / Anti-Money Laundering) requirements where regulations are involved.

NEO Strengths:

- Backed by the Chinese government, giving it a strong backing and legitimacy of use.
- Implements secure identities, allowing the association of real identities to the blockchain.
- Can be used by popular programming languages in use today.
- Less energy intensive than traditional blockchain mining.

NEO Weaknesses:

- The backing by the Chinese government might mean that technological decisions are taken with political influence rather than independently.

Conclusion

As can be seen from the above comparisons, blockchain technology is slowly but surely evolving from a speculative currency into a real digital platform which can be used to build scalable, distributed applications.

The above three platforms are some of the more popular ones which can be used to develop applications today, but all of them are evolving at a rapid pace.

In the near future, we'll start seeing the true maturity of the technology and the release of truly distributed blockchain-based applications.

Chapter 8: Compiling and Smart Contracts: ABI Explained

BY MISLAV JAVOR

Most smart contracts are developed in a high-level programming language. The most popular currently is **Solidity**, with **Vyper** hoping to take the throne in the near future.

However, the mechanism driving Ethereum can't understand the high-level languages, but instead talks in a much lower-level language.

The Ethereum Virtual Machine (EVM)

Ethereum smart contracts are sets of programming instructions being run on all the nodes running a full Ethereum client. The part of Ethereum that runs the smart contract instructions is called the EVM. It's a virtual machine not unlike Java's JVM. The EVM reads a low-level representation of smart contracts called the **Ethereum bytecode**.

The Ethereum bytecode is an assembly language made up of multiple **opcodes**. Each opcode performs a certain action on the Ethereum blockchain.

The question is, how do we go from this:

```
pragma solidity 0.4.24;

contract Greeter {

    function greet() public constant returns
(string) {
    return "Hello";
    }

}
```

to this:

```
PUSH1 0x80 PUSH1 0x40 MSTORE PUSH1 0x4
CALLDATASIZE LT PUSH2 0x41 JUMPI PUSH1 0x0
CALLDATALOAD PUSH29
0x10000000000000000000000000000000000000000000000000000000000000000
000000000 SWAP1 DIV PUSH4 0xFFFFFFFF AND DUP1
PUSH4 0xCF AE3217 EQ PUSH2 0x46 JUMPI JUMPDEST
PUSH1 0x0 DUP1 REVERT JUMPDEST CALLVALUE DUP1
ISZERO PUSH2 0x52 JUMPI PUSH1 0x0 DUP1 REVERT
JUMPDEST POP PUSH2 0x5B PUSH2 0xD6 JUMP JUMPDEST
PUSH1 0x40 MLOAD DUP1 DUP1 PUSH1 0x20 ADD DUP3
DUP2 SUB DUP3 MSTORE DUP4 DUP2 DUP2 MLOAD DUP2
MSTORE PUSH1 0x20 ADD SWAP2 POP DUP1 MLOAD SWAP1
PUSH1 0x20 ADD SWAP1 DUP1 DUP4 DUP4 PUSH1 0x0
JUMPDEST DUP4 DUP2 LT ISZERO PUSH2 0x9B JUMPI DUP1
DUP3 ADD MLOAD DUP2 DUP5 ADD MSTORE PUSH1 0x20
DUP2 ADD SWAP1 POP PUSH2 0x80 JUMP JUMPDEST POP
POP POP POP SWAP1 POP SWAP1 DUP2 ADD SWAP1 PUSH1
0x1F AND DUP1 ISZERO PUSH2 0xC8 JUMPI DUP1 DUP3
SUB DUP1 MLOAD PUSH1 0x1 DUP4 PUSH1 0x20 SUB PUSH2
0x100 EXP SUB NOT AND DUP2 MSTORE PUSH1 0x20 ADD
SWAP2 POP JUMPDEST POP SWAP3 POP POP POP PUSH1
0x40 MLOAD DUP1 SWAP2 SUB SWAP1 RETURN JUMPDEST
PUSH1 0x60 PUSH1 0x40 DUP1 MLOAD SWAP1 DUP2 ADD
PUSH1 0x40 MSTORE DUP1 PUSH1 0x5 DUP2 MSTORE PUSH1
```

```
0x20 ADD PUSH32
0x48656C6C6F000000000000000000000000000000000000000000000000000000
000000000000000000 DUP2 MSTORE POP SWAP1 POP SWAP1
JUMP STOP LOG1 PUSH6 0x627A7A723058 KECCAK256 SLT
0xec 0xe 0xf5 0xf8 SLT 0xc7 0x2d STATICCALL
ADDRESS SHR 0xdb COINBASE 0xb1 BALANCE 0xe8 0xf8
DUP14 0xda 0xad DUP13 LOG1 0x4c 0xb4 0x26 0xc2
DELEGATECALL PUSH7 0x8994D3E002900
```

Solidity Compiler

For now, we'll be focusing on the Solidity compiler, but the same principles apply for Vyper or any other high-level language for the EVM.

First things first: install [Node.js](#).

After you've done this, go to your terminal and run this:

```
npm install -g solc
```

This will install `solc` — the Solidity compiler. Now make an empty directory. In that directory create a file called `SimpleToken.sol` and put the following code:

```
pragma solidity ^0.4.24;

contract SimpleToken {

    mapping(address => uint) private balances;

    constructor() public {
        balances[msg.sender] = 10000000;
    }

    function getBalance(address account) public
    constant returns (uint) {
```



```

        return balances[account];
    }

    function transfer(address to, uint amount)
    public {
        require(balances[msg.sender] >= amount);

        balances[msg.sender] -= amount;
        balances[to] += amount;
    }
}

```

This is the simplest token smart contract, but it has several important features that will be useful for this tutorial. They are:

- public functions
- private functions
- properties

After you've done this, run the newly installed `solc` on your file. You do this by running the following:

```
solcjs SimpleToken.sol
```

You should get an output similar to this:

```
Invalid option selected, must specify either --bin
or --abi
```

And your compilation should fail.

What just happened? What is `bin` and what is `abi`?

`bin` is simply a compact binary representation of the

compiled bytecode. The opcodes aren't referenced by `PUSH`, `PULL` or `DELEGATECALL`, but their binary representations, which look like random numbers when read by a text editor.

ABI — Application Binary Interface

Once our `bin` output is deployed to the blockchain, the contract will get its address and the bytecode will be pushed into Ethereum storage. But a large problem remains: how do we interpret the code?

There's no way of knowing, from bytecode only, that the contract has functions `transfer(:)` and `getBalance(:)`. It's even less clear whether these functions are `public`, `private` or `constant`. The contract is deployed *without context*.

Calling such a contract would be next to impossible. We don't know where each function is in the bytecode, which parameters it takes, or whether we'll be allowed to call it at all. This is where the ABI comes into play.

The ABI is a `.json` file that describes the deployed contract and its functions. It allows us to contextualize the contract and call its functions.

Let's try running our `solcjs` once again. Run the following commands:

```
solcjs SimpleToken.sol --abi
solcjs SimpleToken.sol --bin
```

Your directory should now have a structure like this:

```
.
├── SimpleToken.sol
├── SimpleToken_sol_SimpleToken.abi
└── SimpleToken_sol_SimpleToken.bin
```

The `SimpleToken_sol_SimpleToken.abi` file should look like this:

```
[{
  "constant": false,
  "inputs": [{
    "name": "to",
    "type": "address"
  }, {
    "name": "amount",
    "type": "uint256"
  }],
  "name": "transfer",
  "outputs": [],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function"
}, {
  "constant": true,
  "inputs": [{
    "name": "account",
    "type": "address"
  }],
  "name": "getBalance",
  "outputs": [{
    "name": "",
    "type": "uint256"
  }],
  "payable": false,
  "stateMutability": "view",
  "type": "function"
}, {
  "inputs": [],
```

```
"payable": false,  
"stateMutability": "nonpayable",  
"type": "constructor"  
}]
```

We can see that the file describes the functions of the contract. It defines:

- their name: the name of the functions
- their payability: whether you can send ether to them
- the outputs: the return value(s) of the function
- their state mutability: whether the function is read-only or has write access.

This is all reasonably easy to understand from reading it. But earlier I mentioned that the ABI also defines how the user can call the functions — that is, the *location* of the function in relation to the smart contract address.

Knowing the name of the function is not enough; we also need to know *how* (where) to call it.

This is done by running a deterministic algorithm on the function properties we mentioned earlier (the name, the payability, the outputs etc.). The details of this function can be found [here](#).

Example

The ABI is the description of the contract interface. It contains no code and cannot be run by itself. The bytecode is the executable EVM code, but by itself it is without context.

In order to call functions in smart contracts, we need to use both the ABI and the bytecode. Luckily for us, this is all abstracted away when we're interacting with smart contracts by using one of the provided frameworks.

An example using the `web3.js` framework would look like this:

```
var simpletokenContract =
web3.eth.contract([{"constant":false,"inputs":
[{"name":"to","type":"address"},
{"name":"amount","type":"uint256"}],"name":"transfer","outputs":
[],"payable":false,"stateMutability":"nonpayable",
"type":"function"}, {"constant":true,"inputs":
[{"name":"account","type":"address"}],"name":"getBalance","outputs":
[{"name":"","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},
{"inputs":
[],"payable":false,"stateMutability":"nonpayable",
"type":"constructor"}]);
var simpletoken = simpletokenContract.new(
{
  from: web3.eth.accounts[0],
  data:
"0x608060405234801561001057600080fd5b50620f4240600
0803373fffffffffffffffffffffffffffffffffffffffff167
3fffffffffffffffffffffffffffffffffffffffff168152602
00190815260200160002081905550610252806100666000396
000f30060806040526004361061004c576000357c010000000
0000000000000000000000000000000000000000000000009
00463ffffffff168063a9059cbb14610051578063f8b2cb4f1
461009e575b600080fd5b34801561005d57600080fd5b50610
09c600480360381019080803573ffffffffffffffffffffff
ffffffffffffffff169060200190929190803590602001909
291905050506100f5565b005b3480156100aa57600080fd5b5
06100df600480360381019080803573ffffffffffffffffff
ffffffffffffffff1690602001909291905050506101d
e565b6040518082815260200191505060405180910390f35b8
06000803373ffffffffffffffffffffffffffffffffffffff
f1673fffffffffffffffffffffffffffffffffffffffff16815
2602001908152602001600020541015151561014257600080f
d5b806000803373ffffffffffffffffffffffffffffffffffff
```

```

ffffffff1673ffffffffffffffffffffffffffffffffffffffff1
68152602001908152602001600020600082825403925050819
05550806000808473fffffffffffffffffffffffffffffffff
ffffffff1673fffffffffffffffffffffffffffffffffffffffff
f1681526020019081526020016000206000828254019250508
19055505050565b60008060008373fffffffffffffffffffff
ffffffffffffffff1673fffffffffffffffffffffffffffffffff
ffffffffffffffff1681526020019081526020016000205490509
190505600a165627a7a72305820c9da07d4976adbf00a4b5fe
4e23330dbaf3cdcbfd4745eed78c702bf27d944060029",
    gas: '4700000'
  }, function (e, contract){
    console.log(e, contract);
    if (typeof contract.address !== 'undefined') {
      console.log('Contract mined! address: ' +
contract.address + ' transactionHash: ' +
contract.transactionHash);
    }
  })
})

```

First we defined the `simpleTokenContract`, which is a *description* of how the contract looks from the outside. We did this by passing it the ABI of the `SimpleToken.sol`.

Then we created an `instance` of the contract `simpletoken` by calling the `simpleTokenContract.new(...)` and passing into it the *data* of the contract (the executable code).

`web3.js` combined the two in the background and now has all the required information to call functions on our smart contract.

Conclusion

In this short overview of smart contract compilation, we explained ABI and how smart contracts deployed on the

Ethereum blockchain can get invoked. While you'll never actually have to use this directly, it's worth being aware of it, as too much abstraction can lead to bugs.

Chapter 9: Ethereum Wallets: Send and Receive Ether with MyEtherWallet

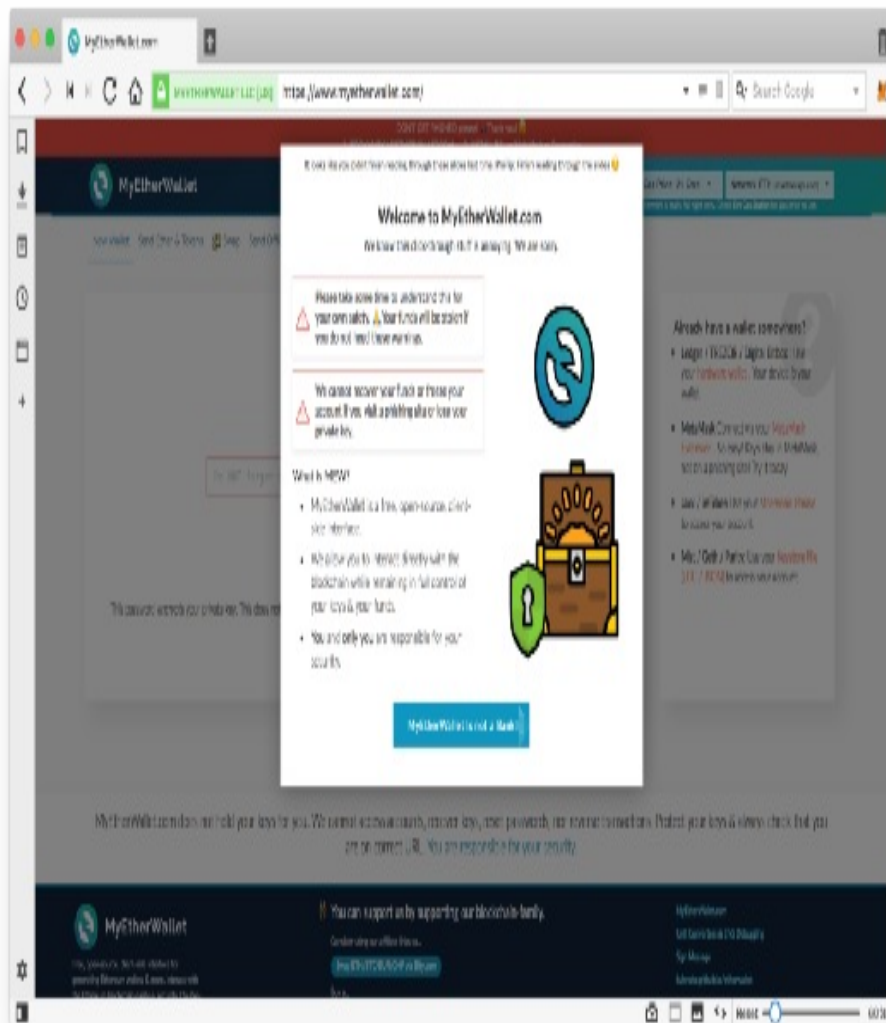
BY BRUNO ŠKVORC

In this chapter, we'll go through the process of generating your first Ethereum wallet and sending and receiving ether through the MyEtherWallet (MEW) interface.

Bookmark This Site!

After opening the site, please bookmark it so you never have to type it into your browser again. There are many bad people who want your ether, and they've bought domains similar to the domain of this site hoping you'll fall into that trap by mistyping!

Immediately after loading, the page will display some warnings about them not being a bank and basically not being able to help you if you lose your funds because they don't have access to them. The gist of it is that you're responsible for your own funds.



Creating and Reading an Address

The first screen of MEW is the “input password” screen. Input a password for a new wallet. (Make it something complex but easy to remember or use a password generator like LastPass.)

Create New Wallet

Enter a password

Create New Wallet

The address is immediately generated in the background, but MEW won't let you see it yet. For as long as you haven't saved the Keystore file somewhere safe, you cannot proceed. Click *Download Keystore File (UTC / JSON)* and store it on a USB drive, then hide it from curious family members. The file will be called something like UTC - -2018-01-26T10-39-56.592Z - -d877263725d9247352661e44c444b21dc5a28581. The first part is the date of creation, the second is the address itself. After you've stored it safely, the next screen will display the private key.

Save Your Private Key.

20106c154daa9d9741f1ccb171b5045628bf5b76af0e27a28a9380d98ee6610d

Print Paper Wallet

****Do not lose it!**** It cannot be recovered if you lose it.

****Do not share it!**** Your funds will be stolen if you use this file on a malicious/phishing site.

****Make a backup!**** Secure it like the millions of dollars it may one day be worth.

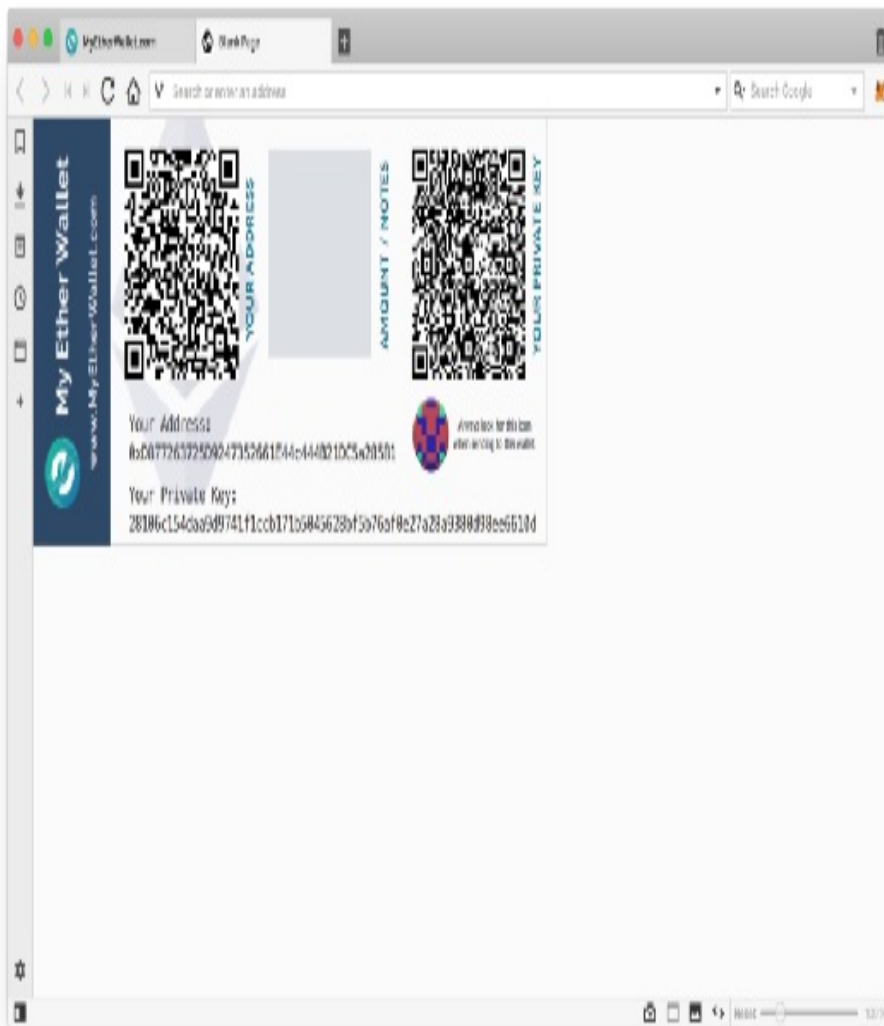
Save Your Address. →

Use One Method or the Other

The private key is equivalent in access level to the file you downloaded previously. Both can be imported into wallet software and used to unlock an address to spend money on it. But there is an **important difference**: the private key **does not require a password!** If you enter a private key into another tool like **MetaMask**, the address will immediately become unlocked without a password and can be used for sending Ether and tokens. If you import the previously downloaded file into any wallet software, it will require a password before letting you access it fully. Therefore, it's safer to only keep one method around: either store the file and remember the password, or store only the private key in a very secure location. Destroy the other method. The fewer ways to access your wallet there are, the safer the money is.

You can generate a Paper Wallet on the same screen. This will

open a new tab with a neatly generated printable layout of two QR codes. One is the public key — the address to which you can send funds and tokens — and the other is the private key mentioned above.

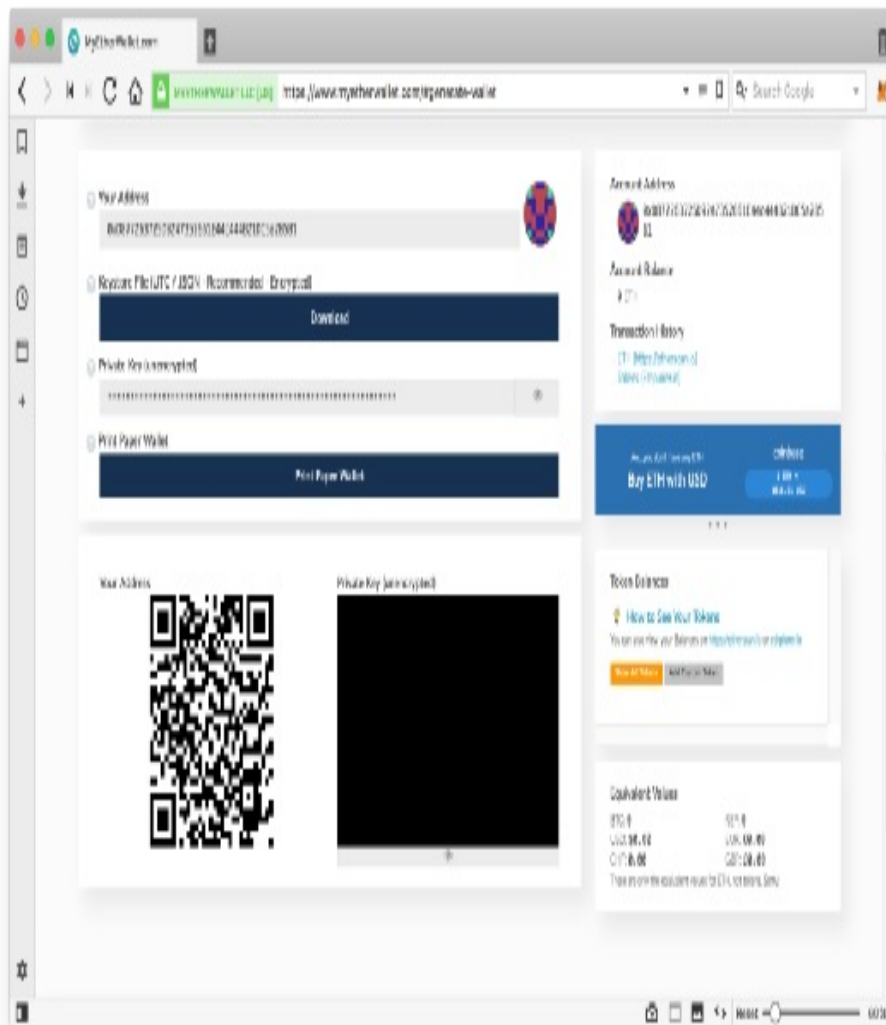


If you're planning to use this address for long-term holding of ether, it's smart to print it and store it in a safe as if you had a bond or a big wad of cash.

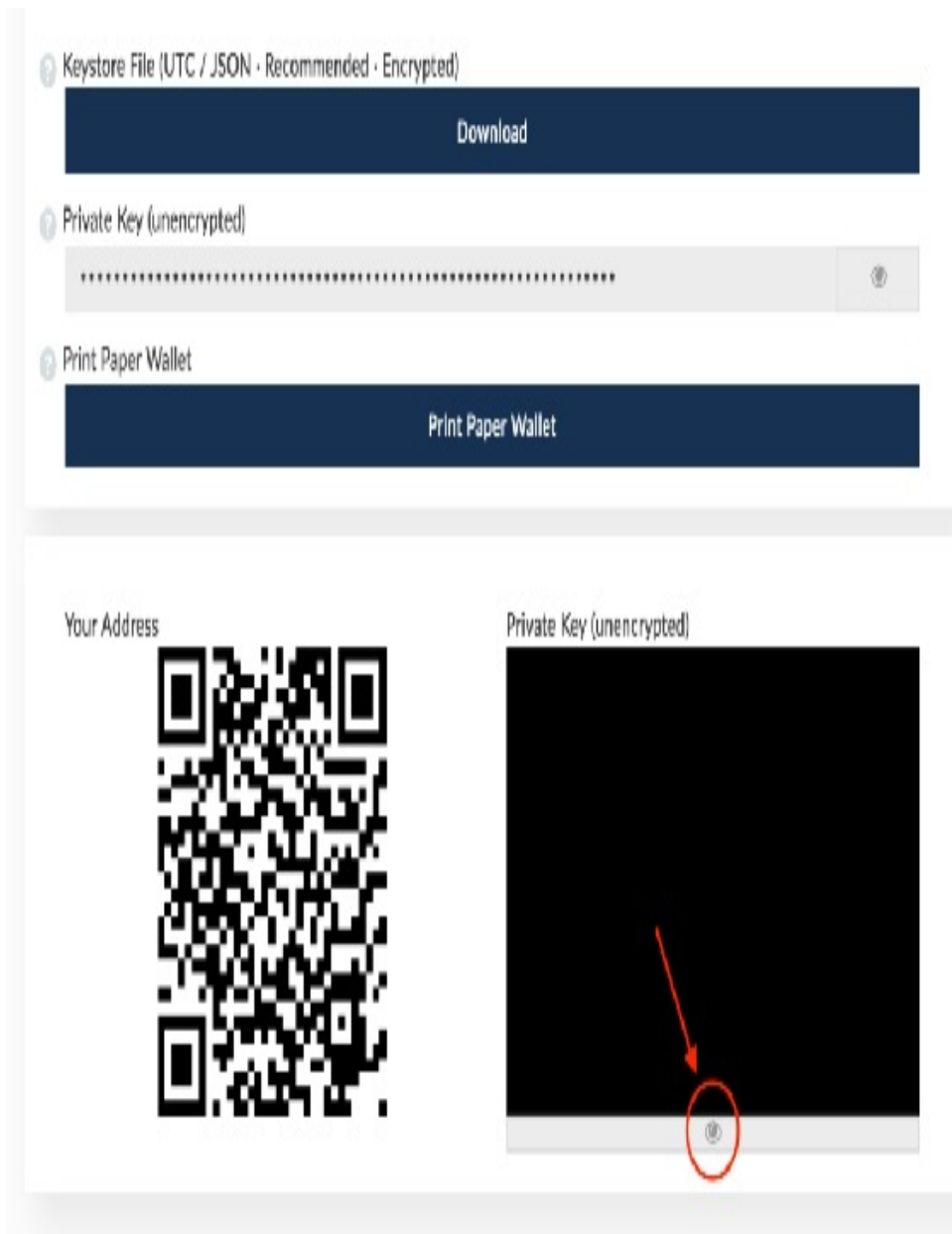
After these steps have been completed, MEW lets us unlock the wallet and view its info. This can be done in two ways:

- Picking a Keystore/JSON file will provide you with a form through which you upload the file you downloaded previously. You'll then be asked for a password.
- Picking a private key lets you paste the private key directly into the field, unlocking the address for usage.

Unlock your wallet and you should see something like the image below:



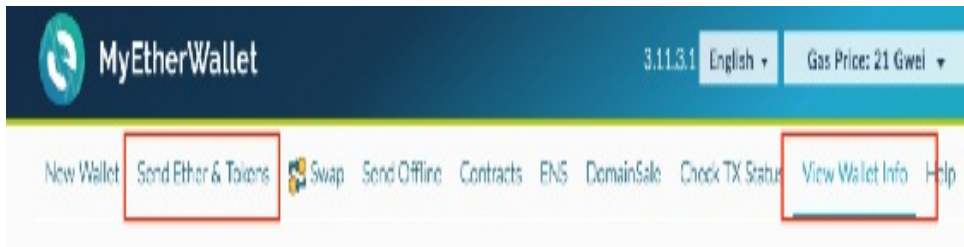
This interface lets you print the wallet again if you lost or destroyed the previous one, or to re-download the JSON file if you lost it and accessed the wallet with the private key. The QR code for the private key can be read by clicking the eye icon under the black rectangle in the bottom of the screen.



On the right, you can see your account's balance: 0 Eth. A little lower you can see the *Token Balances* frame which shows you how many of which tokens you have. Because of a large number of available tokens out there, you first need to click *Show all tokens* to load them.

This screen is visible because you're on the *View Wallet Info* option to which MEW automatically redirects you after

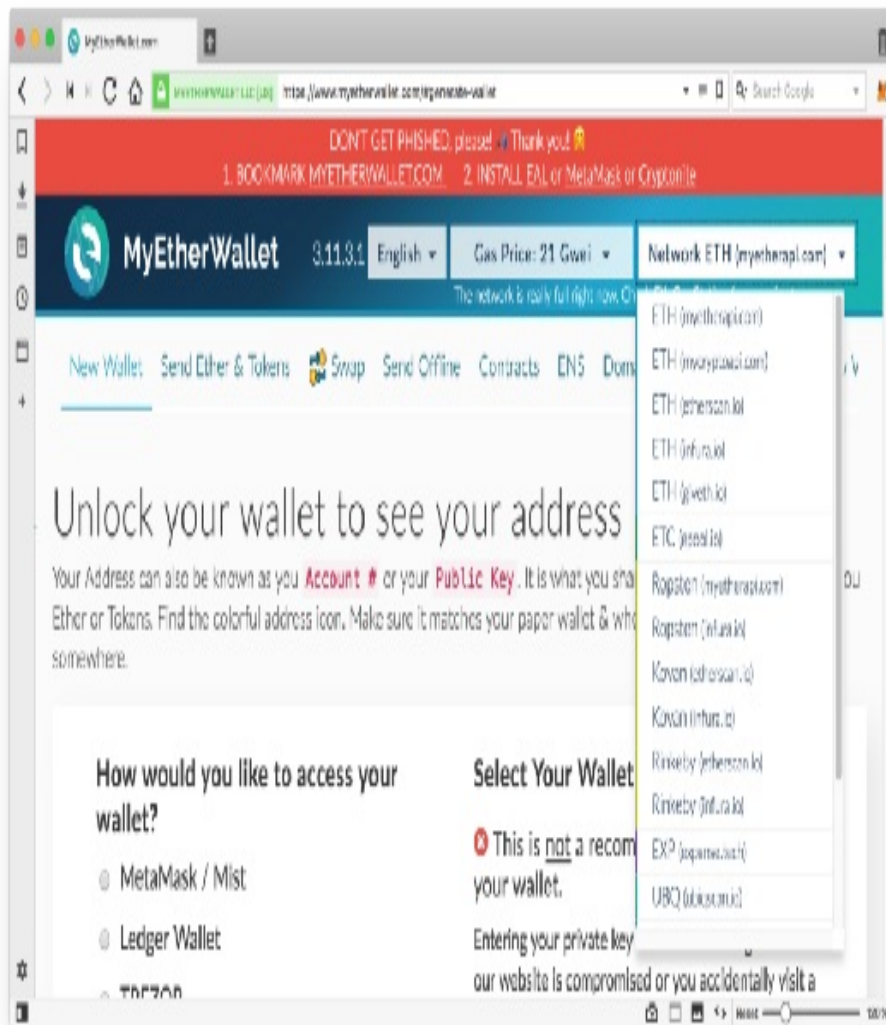
creation. When sending tokens or ether, the option to use is *Send Ether and Tokens*.



Receiving Ether

On the *View Wallet Info* screen, the QR code in the lower left is the address to which tokens and ether can be sent. The value of the code is equivalent to the value in *Your Address* at the top, and the *Account Address* value in the right sidebar.

To receive some Ether, let's switch to the Ropsten test network. In the upper right corner of the screen you can pick the network to connect to:

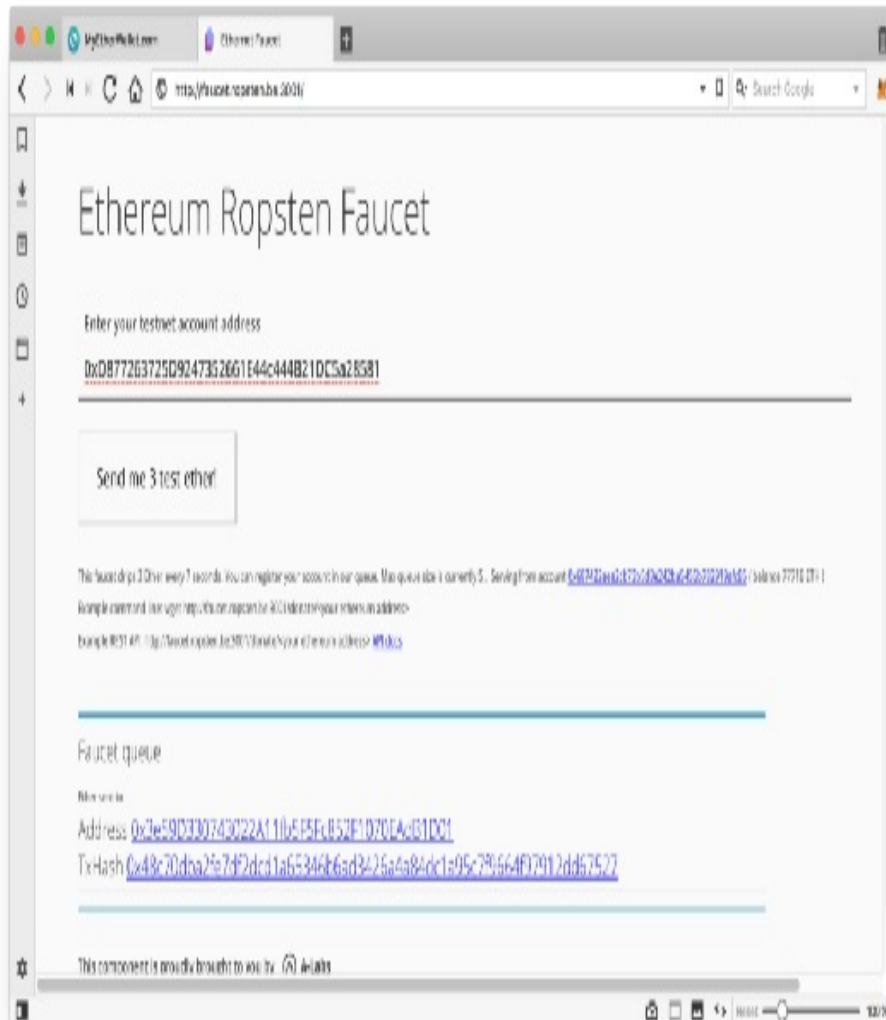


Pick any Ropsten. The screen will refresh and you'll have to re-enter the wallet. After re-entering via the *Send Ether & Tokens* tab, the interface will be a little different. No QR codes and the balance will be in ROPSTEN ETH, not ETH.

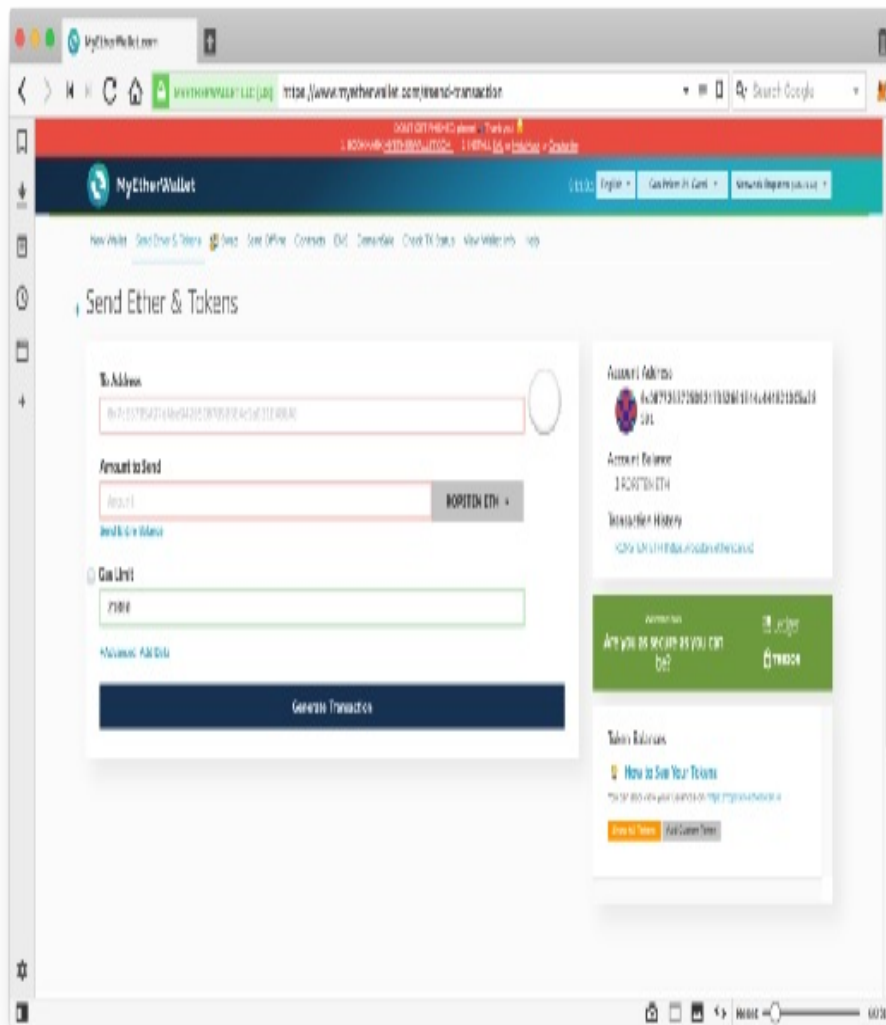
Testnets are environments for playing around and testing the Ethereum blockchain without financial consequences. You can find out more about them [here](#). Open the Ropsten faucet next.

It's a page which lets us request free Ropsten Ether:
faucet.ropsten.be:3001

In the recipient field, enter your wallet’s address. Copy it from the *Account Address* part of MEW.



The Ether should arrive within a minute or two, depending on how busy the testnet is.



Sending Ether

Now let's send some Ether. In a new tab, open MEW again and generate another wallet, following instructions from before. Copy this newly generated address' public key (*Account Address*) to the *To Address* field of the original unlocked wallet with test Ether.

+ View Wallet Info

1 Your Address

0x01a1a031450d0f7053030c450f75505af2feC10f



2 Private Key (unencrypted)

.....



3 Print Paper Wallet

Print Paper Wallet

Account Address



0x01a1a031450d0f7053030c450f75505af2feC10f

Account Balance

0 ROPSTEN ETH

Transaction History

ROPSTEN ETH 0 (https://ropstenall.org/tx/0)

Learn more about
protecting your funds.



Your Address



Private Key (unencrypted)



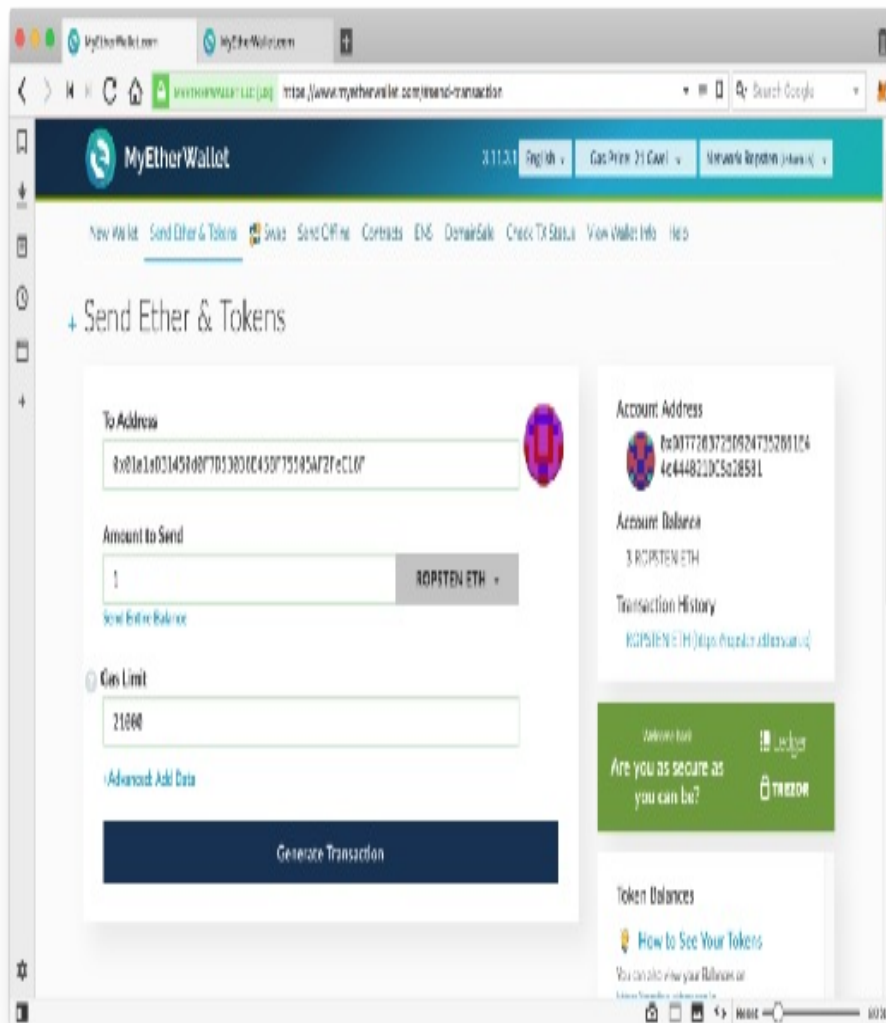
Token Balances

[How to See Your Tokens](#)

You can also view your Balances on
<https://ropstenethercan.io>

[View All Tokens](#)

[Add Custom Tokens](#)



Set the amount to less than maximum and click *Generate Transaction*. After it's generated, confirm it with *Send Transaction*.

+Advanced: Add Data

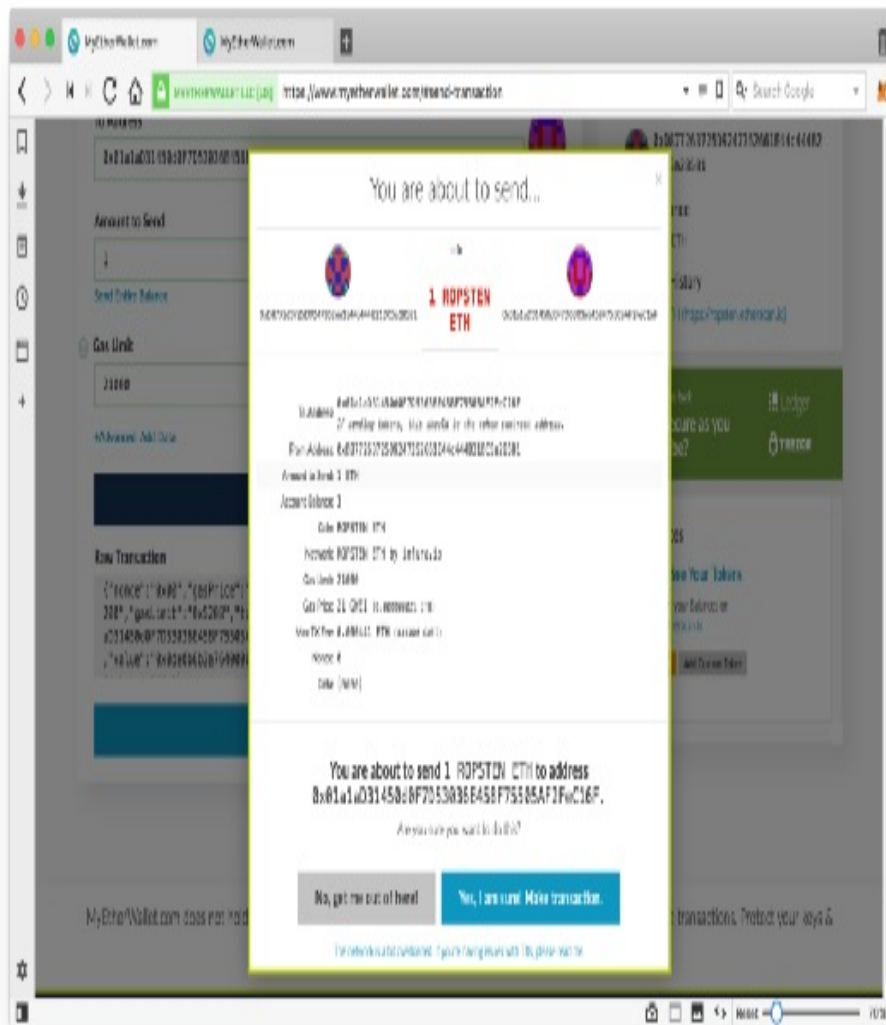
Generate Transaction

Raw Transaction	Signed Transaction
<pre>{"nonce":"0x00","gasPrice":"0x04e3b29200","gasLimit":"0x5208","to":"0x01a1aD31450d0F7D53036E458F75505AF2FeC16F","value":</pre>	<pre>9b49c686d6524b63aa8ed0dd8ae038fdd6cba0674b1150d214cdd21d17b55bc7ae5ec3faf409e6d09e9d78ed0d425f466f3810</pre>

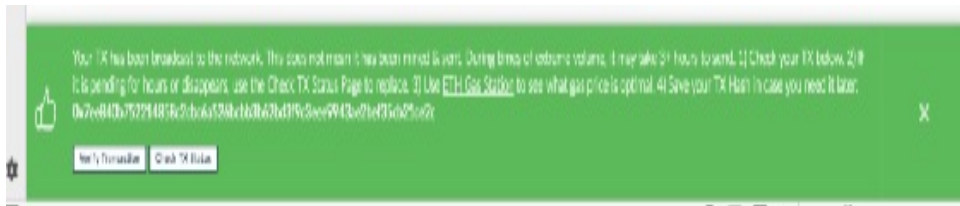
Send Transaction

Another confirmation window will appear, listing all the thus far provided information:

- sender and recipient
- amount
- balance of sender's account
- coin being sent
- network being used
- gas limit and price — that is, the cost of the transaction
- nonce and data (irrelevant for now).



Confirm the sending. After sending, a green confirmation message should appear at the bottom of the screen with a *Verify* link which lets you look at the transaction on Etherscan, Ethereum's most popular block explorer. Etherscan has access to the Ropsten network and can inspect it.



This is how the TX looks for our case above:

Transaction [0x7ee840b752214858c2c6c6a5266cbb3b62bd3f9c3eee9943ae2be735cb21ce2c](#) [Home](#) / [Transactions](#) / [Transaction Information](#)

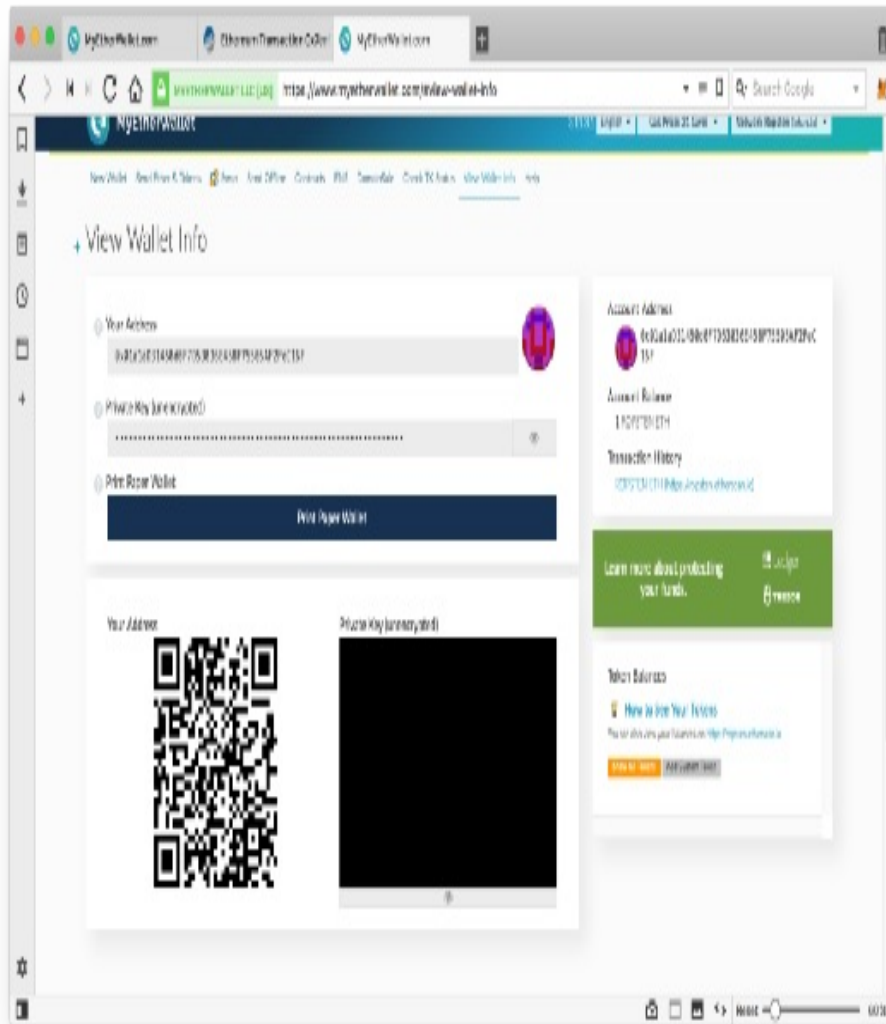
Overview

Transaction Information (Pending Confirmation)

Tx Hash:	0x7ee840b752214858c2c6c6a5266cbb3b62bd3f9c3eee9943ae2be735cb21ce2c
Block Height:	(Pending)
Time Stamp:	(7:00 hr 01 min 28 secs ago (Jun-28-2018 11:16:33 AM))
From:	0xd871265775d85247282861e64c444a21dc5a08501
To:	0x01a1ad31400a077053080e40876505ad2f0c10f
Value:	1 Ether (\$0.000000)
Gas Limit:	21000
Gas Used By Tx:	Pending
Gas Price:	0.000000021 Ether (21 Gwei)
Actual Tx Cost/Fee:	Pending
Cumulative Gas Used:	Pending
Nonce:	0
Input Data:	<div>0x</div>

Pending means the transaction is still being processed. The higher the gas price, the faster it turns into Success. Once it's successful, the ROPSTEN ETH state of the second, new

address, will change (sometimes a re-unlock is necessary so MEW registers the changes).



Conclusion

MyEtherWallet is a simple and safe tool for interacting with the Ethereum blockchain. You can use it to interact with contracts, register domains (like bitfalls.eth), track accounts,

activate smart contracts, send Ether, even run decentralized apps (more on that in a followup post).

But while MEW is a pretty safe option, nothing can match the safety of hardware wallet like Ledger Nano S. Seeing as MEW's interface can seem a little complex, there's a simpler alternative as well — MetaMask.

Chapter 10: Ethereum: How Transaction Costs are Calculated

BY BRUNO ŠKVORC

This article on Ethereum Transaction Costs was originally published at [Bruno's Bitfalls website](#), and is reproduced here with permission.

When sending a Bitcoin transaction, its fee is proportionate to its size. The more inputs and outputs, the more expensive it is. Add to that the factor of pending transactions, and transaction fees can skyrocket based on those two factors alone.

With Ethereum, given that we're talking about a programming language within the protocol, it's possible to be very computationally demanding with very little text or code (something which would be very cheap in the BTC-verse). Let's look at this loop, for example:

```
while (i++ < 1000) {  
    j = j + i;  
}
```

This loop means “for as long as *i* is smaller than 1000,

increase it by 1 and then sum up i and j and write the result into j , then do it all again.” This loop will execute 1000 times if i is 0, or more if it’s a negative number.

To pay for this computational cost in a fair way — since it has to be executed on all miners’ machines at once and they spend their resources and time on it — the concept of **gas** was introduced. Gas is used to pay for the execution of these so-called smart contracts (Ethereum programs) inside the EVM. For example, $i + j$ above is a summation operation which costs 3 gas every time it’s executed, so 3000 gas if executed 1000 times.

To explain *gas* properly, let’s first cover the *EVM*.

EVM

EVM stands for *Ethereum Virtual Machine*. But what is a virtual machine anyway?

VIRTUAL MACHINE

A virtual machine is software running on a specific computer which contains another operating system completely encapsulated inside the main one. For example, a virtual machine allows you to run Windows inside Linux, Linux inside Windows, Windows on macOS like in the image below, or any other combination.



We use virtual machines to separate the environment in which we do our everyday computer use from the environment we work or program in. This lets us keep viruses at bay (they have no way of breaching the virtual machine and getting to the main operating system), helps prevent infinite loops from crashing our main operating system, and holds hard-drive corruptions like the infamous WannaCry ransomware at bay. Additionally, VMs let us use Windows games on Linux, for example, or allow us to program in different versions of the same programming language's environment easily, without mixing them up.

EVM

The Ethereum Virtual Machine is built into the software running on the Ethereum protocol. It executes smart contracts — Ethereum programs written in the Solidity language. The

EVM is contained in the full nodes of the Ethereum network, inside of which it executes these Ethereum-user-written programs.

Any miner of Ethereum simultaneously executes smart contract code. What this means is that Ethereum programs (dapps — decentralized apps) are executed on everyone's computer at the same time (decentralized).

Execution of these programs isn't free, however. Miners spend their own electricity, time, and hardware to do this. To pay them for their effort of executing computer instructions (like “store the value 5 into the variable X”), the concept of *gas* was introduced.

Gas, Ether, and GWei

Gas is a unit of cost for a particular operation a computer needs to execute, and it executes this instruction when we broadcast a transaction which contains an Ethereum program in order to run a *dapp*. For example, summing two numbers costs 3 gas. Multiplying them costs 5 gas. Storing a 256bit word into the blockchain costs 20000 gas, which means storing 1kb of data costs 640000 gas.

Just like the USD has cents, so too does ether have its own basic unit: wei. If we take wei as the basic unit of ether, we get the following table of definitions:

unit	wei
...	...

wei	1
kwei <i>ada</i> femtoether	1.000
mwei <i>babbage</i> picoether	1.000.000
gwei <i>shannon</i> nanoether / nano	1.000.000.000
szabo / microether / micro	1.000.000.000.000
finney / milliether / milli	1.000.000.000.000.000
ether	1.000.000.000.000.000.000

Tip: use [this converter](#) to convert between Ether units.

According to [this informative site](#), the current average price of gas is 10 GWei (10 gigawei). Seeing as 1 GWei is one billionth of an ether, the aforementioned cost of storing a 1kb word is $640000 * 10$, which is 6.4 million GWei. That amounts to 0.0064 eth which, at a price of \$450 per ether, amounts to around \$2.88.

The text from the beginning of the above table all the way to >>this specific marker<< is around 1kb. So storing this little bit of text into the blockchain actually costs \$2.88. However, that's just the storage cost! Our smart contract could also have some logic, like summing or multiplying and then storing numbers, or triggers that activate on a specific mined block *etc.* Obviously, storing data into the blockchain itself is incredibly expensive. Storage is better off in BigchainDB or [IPFS](#), while blockchain is a better solution for global processing and verification of that data.

WHY GAS?

Why pay costs in gas and not ether directly?

All the gas prices of all the possible operations the EVM can perform are hard-coded in the Ethereum protocol and in the clients (programs) we connect to it, like Geth, Eth, Parity, *etc.* If the code listed them in ether, then we'd have to update the code every time ether's value fluctuated to keep the price of computing efforts in a normal range and keep the system usable, which is obviously unsustainable.

By adding this *gas* layer on top of the costs, and paying for gas with GWei, we're given the option to alter the amount of gas to use in a transaction and the amount of money to pay for it. It's fully under our control, without throwing the system off balance.

This leads us to our last section.

LIMIT/COST VS PRICE

Gas limit is the *maximum amount* of gas we're willing to spend on a transaction. Most software we use to broadcast Ethereum transactions has the ability to auto-estimate the amount of gas that'll be necessary to execute a function. It'll usually suggest a figure right off the bat. For example, simple monetary A->B transactions usually need only 21000 gas. More complex ones which call specific smart contract functions might run into hundreds of thousands or even millions of gas. The spent amount of gas is called **gas cost**.

We, as a user, can modify the amount of gas we want to spend on a transaction and reduce it, but if the transaction runs out of gas during execution, we lose the gas we sent in. It's been spent and the transaction is rejected. On the other hand, if we provide more gas than is needed, the rest is refunded to us. Hence, it's always better to send more gas than you might need to execute a transaction.

Gas cost is the GWei price per unit of gas.

Thus, the total cost of an Ethereum transaction is actually the amount of necessary gas multiplied by the price in GWei per gas unit. This is the *maximum* transaction fee we'll pay; any extra gas is refunded, so fees are often vastly overestimated.

Let's look at an example.

MetaMask Notification

CONFIRM TRANSACTION

Main Network

Bitfalls main

4dA2e8...f22F

0.870743 ETH

401.53 USD

b1690C...7d8C

Amount

0.387617 ETH

178.74 USD

Gas Limit

135963

UNITS

Gas Price

25

GWEI

Max Transaction Fee

0.003399 ETH

1.57 USD

Max Total

0.391016 ETH

180.31 USD

Data included: 36 bytes

RESET

SUBMIT

REJECT

In the above image, we're executing a transaction which, due to its complexity, estimates it'll need 135963 gas. With a 25 GWei price, the maximum transaction fee we'll pay is \$1.57, which is trivial when looking at the total amount we're sending (0.38 eth = \$178). If we raise the price of gas to 250 GWei, the transaction becomes proportionately more expensive:

MetaMask Notification

CONFIRM TRANSACTION


Main Network

Bitfalls main


4dA2e8...f22F

0.870743 ETH

401.53 USD



>



b1690C...7d8C

Amount

0.387617 ETH

178.74 USD

Gas Limit

UNITS

Gas Price

GWEI

Max Transaction Fee

0.033990 ETH

15.67 USD

Max Total

0.421608 ETH

194.42 USD

Data included: 36 bytes

RESET

SUBMIT

REJECT

A higher tx fee effectively encourages miners to process our transaction before others in the queue, thereby speeding up execution. If we're not in a rush, we can leave the price of gas at its starting value and the transaction's confirmation shouldn't take more than 10 minutes. But if we're in a rush and want it done in seconds (if, for example, we're dealing with an auction), it's easy to increase the price of gas and sacrifice some money for a guarantee of success.

Conclusion

Gas is the unit of work expended per computational operation in the Ethereum Virtual Machine. It's paid for in ether, the token of the Ethereum protocol, and each computational operation has a different gas cost. The gas price (in GWei or 1 billionth of an ether) varies according to the network congestion and the user's preference for a speedy confirmation.

For a smart contract which we want to deploy on the Ethereum network, two terms are important: gas limit, and gas price. Gas limit is the maximum amount of gas we're willing to spend on executing the transaction. The amount of gas actually required (known only once the transaction has been executed) is called *gas cost*. Gas price is the price per unit of gas, expressed in GWei (or billionths of ether). The total cost of a transaction will be the product of gas cost and gas price, while the maximum transaction fee will be the product of gas limit and gas price. The difference is refunded to the transaction's

sender to keep the system fair and usable.

Chapter 11: Proof of Stake vs Proof of Work

BY BRUNO ŠKVORC

This article on Proof of Stake vs Proof of Work was originally published at [Bruno's Bitfalls website](#), and is reproduced here with permission.

You've probably heard of PoW (Proof of Work) and PoS (Proof of Stake). You may have heard of DPoS (Delegated Proof of Stake) and PoA (Proof of Authority) as well. We'll discuss each method in this article.

Proof of Work

With Proof of Work, your miner (the computer or group of machines under your control) does the following things:

- It takes from the internet the order to process some transactions. In other words, it takes from the miners geographically closest to it a set of transactions it's supposed to verify, in which it's written who's sending how much of a [cryptocurrency](#) to whom.
- The miner then builds a [block](#) — a list of transactions that need to be validated. How many transactions per block depends on the size of those transactions. Those which send from many addresses to many addresses are much larger than those sending money from one to maybe one or two addresses, as [previously explained](#).

- The miner combines all the data from this block (it literally glues them together), adds some more data into the mix, and then tries to guess the final bit of data which will result in a valid hash when hashed. For example, in Bitcoin, the hash has to be prepended by a certain number of zeroes. The computer, thus, does the following: “Try summing up all of this and the number 1. Incorrect? Okay, try summing up all of this and number 2. Incorrect? Okay, try...”
- The computer’s processing power will dictate how many of such guesses per second it can do.
- After a successful guess, the computer gets a block reward, which is currently 12.5 BTC in Bitcoin, or 6.18 XMR in a system like Monero, *etc.*

The profits of mining this way will vary by hardware, software, and currency. We’ve gone into some detail about that here.

The advantages of PoW are:

- Outside factor effect. With the PoW mechanism, the production and circulation of money requires external factors like power and hardware. It’s not possible to get the expense of power or production of hardware back. Why this is important will be explained in the PoS section below.
- It’s simple to pool mine. It’s easy to just grab another computer’s calculated hashes, combine them into one big pool of hashes, and have many computers hashing together, splitting the profits.
- It’s useful for areas with surplus electricity, like China with its hydroelectric dams.

The disadvantages of PoW are:

- PoW isn’t possible on smaller and weaker devices like smartphones. Not only do these devices lack the space to store hundreds of gigabytes of blockchain data, but they’re also not computationally powerful enough to mine effectively. The battery would be emptied very quickly, not really accomplishing anything.

- PoW mining is slow. With Bitcoin, it's one block every ten minutes, and the transactions that fit inside that block will be processed. Anything else has to wait for the next block. This causes long waiting periods or expensive transactions (those that attach a higher transaction fee are processed faster).
- PoW is already spending enormous amounts of electricity. Simply mining a single block costs more electricity than some countries need in a whole year. This will only get worse. The dependence of a cryptocurrency on electricity is unsustainable in all but the most stable environments. This dependency also means that a more expensive electricity bill or a government-imposed limit to the types of spending electricity can be used for can stop an entire cryptocurrency.
- PoW allows for the centralization of mining. China already has 80% of the world's Bitcoin hashing power, and if their cartels join forces, we've got an 80% attack, not a 51% attack.
- Because the block reward keeps decreasing, miners keep getting fewer and fewer tokens of a mined blockchain. At the same time, as more people are mining, the mining difficulty increases, so it gets harder and harder to mine. This makes mining more and more expensive compared to profits, and fewer people bother with it, exiting the system. The currency self-sabotages. Less hashpower among the miners also makes the 51% attack more likely.

As an example, Bitcoin might still rise to some \$25000 or \$50000 through the next five years or so, but as transactions get moved *off chain* onto solutions like the Lightning Network (designed to transact small amounts on the side, without waiting for validation from the main chain, thus moving fees off the main chain too), mining becomes even less profitable. With block rewards approaching 0 and transaction fees all but gone, this will further escalate the miners' departure from the network, opening it up to a 51% attack or total stagnation.

Proof of Stake (PoS)

With Proof of Stake, no complex equations need to be guessed, so powerful computers are no longer necessary, and with them, there's less need for electricity.

Examples of PoS coins: Ethereum (soon), BlackCoin, CoinMagi, Diamond, Mintcoin, OKCash, HyperStake, Quotient, *etc.*

Let's take Ethereum as an example. Proof of Stake works by randomly selecting a **validator** — an account with enough Ether to be considered a stakeholder, someone who's invested into the ecosystem. Initially, that would be 1000 Ether. The more Ether a validator has, and the longer it has been on the candidate's account, the more chance that they'll be picked. This validator then stakes the Ether (locks it up for a period of months) and guarantees to uphold the laws of the ecosystem — to truthfully validate transactions. Once a new transaction arrives, it's added to the block, validated, and the block is sent to the other validators for confirmation. For this work, the validator gets the transaction fees of the transactions they processed.

Because there's no more need to guess combinations, and processing transactions is easy and cheap, it's easy to generate fake ones. But because validators have to reconfirm the information (like students in a school correcting each other's exam), it's almost impossible to think a bunch of validators will all confirm a malicious one's report. For this to happen, the malicious group must not only be randomly selected at the same time (impossible), but also have 51% more Ether than the rest of the randomly selected validators, who could have anywhere from the minimum to an astronomical amount of Ether.

Furthermore, if a validator is found to be malicious, they lose

their stake and are kicked off the network. Cheating becomes an incredibly expensive sport. Someone with enough money invested into the Ethereum ecosystem to become a validator has no reason to sabotage this ecosystem, because their holdings lose value (due to the scandal they would cause if successful, or through losing their stake if the cheating attempt failed).

The advantages of PoS are:

- speedy processing of transactions
- contrary to PoW, not harmful to the environment
- not vulnerable to a state attack: no need for enormous amounts of electricity
- can be performed on smaller and weaker devices because there's no need to download the whole blockchain, and since there's no need for much computational power either, can be easily adopted by the mainstream.

The downsides of PoS are:

- No external factors. Given that the stake is a part of the system itself, the whole game is internal. This means that someone with enough money to invest exclusively into the destruction of this system can do so by investing only money, as opposed to Bitcoin, where they need to invest money, time, expertise, hardware, electricity, and more — all external factors.
- The rich get richer. Those who have had their Ether the longest (the age of the Ether in an account is as much of a factor as the amount is) also have the best chances of becoming validators. This means their chances to earn more Ether on top of their existing pile also increases. This is different from Bitcoin's "rich get richer" system because there, the rich have to keep investing in hardware and knowledge to remain competitive. It also hurts more to sabotage the network.

Delegated Proof of Stake

(dPoS)

In this PoS type, 101 delegates are picked by the community by voting with the cryptocurrency in question — for example, 1 Lisk, 1 vote. Some blockchains have a different number than 101, but that's the default.

Delegates cannot modify transactions, only delay their inclusion into a block, but this has safety nets built into the protocol, so prolonged exclusion of a transaction becomes expensive. This is also the only “technical” power these delegate validators have, and should they abuse the power, the community can instantly see it and vote them out.

Delegates get rewards for validating transactions, just like in PoS, so cheating again makes no sense because they lose both their stake and their role in the system. The rewards they do get they can spend on lobbying, spreading the word about the currency, cash it out into earnings, *etc.*

Some DPoS systems work in such a way that they can define a burn rate: a percentage of tokens to be destroyed upon getting a reward. For example, a burn rate of 40% will **destroy** 40% of the tokens received by a delegate. Destroying tokens leads to deflation, which leads to increased value of the remaining tokens — both those tokens in the hands of the delegates, but also the tokens in people's wallets around the world. It's like everyone who uses the currency gets dividends, because their money becomes more valuable automatically. The remaining 60% of tokens can, of course, still be used for anything the delegate wants to use them for.

Advantages of dPoS:

- A more even distribution of block rewards. People will elect only those delegates who give them the most rewards, so the causal users and not just big holders will be rewarded.
- Real-time voting security. Any malicious action can be immediately detected by the voters and the malicious delegate can be voted out of the system.
- As with PoS, it's very environmentally friendly and easy to execute on smaller and weaker devices. It's harder to stop because it doesn't depend on external factors controlled by the state, like electricity.

Disadvantages of dPoS:

- It's possible that the delegates get organized into cartels. This has already happened with the Lisk blockchain which, even though it hasn't been released as a full product yet, has a cartelized dPoS in which 51 of the 101 delegates have joined and formed a "Lisk Elite Group" in which they vote for each other and don't distribute their gains to any casual user who hasn't voted for all of them. This is identical to the modern political system.
- Because fewer people are in charge of keeping the network alive, it's easier to stop it or organize a 51% attack (see the point above about cartels).

DPoS is used by currencies like Bitshares, Crypti, Lisk, RISE, ARK, *etc.*

Proof of Authority

Proof of Authority is when certain nodes are selected as block validators and they stake neither their money nor their electricity, but their reputation.

The way it works is as follows: when setting up the network/blockchain, certain nodes are selected as allowed to

seal blocks. These nodes will usually provide some kind of proof of identity to prove that they're trustworthy. Plenty of such unrelated nodes need to be selected for the system to become viable. If a node is found to be in violation of protocol rules, that node is kicked off the network and cannot rejoin because it is banned based on identity. The stake is that it cannot play with others any more.

PoA is only useful in private or closed or specific-use-case blockchains. For example, testnets use it (like Ethereum's Kovan) and cartels might use it. Airlines, hotels, and banks are ideal candidates: they don't have to like each other or trust each other, but it's in their best interest to work together and be truthful with each other in order to force the prices to stay up so there's no direct competition. A PoA blockchain is the perfect solution: the identity violating the rules — for example, United Airlines — is kicked off the consensus and can no longer participate, and this destruction of trust is much more harmful to them as a business than the effect their cheating in the network could have had. The involvement of identity in the staking mechanism directly encourages the validators to remain truthful.

Conclusion

There are other methods, too — like Proof of Importance in use by NEM (XEM). It's similar to PoS in that it values wallets with over 10000 XEM and calculates their importance score based on their holdings and their participation in the network: the more transactions are bound to an address, the

more important it is as an active participant. But this too has obvious downsides.

None of the methods is perfect, and each has its own set of problems, keeping extremely smart people busy. When and how much progress we'll see remains to be seen, but one thing is certain: the progress is coming, and it'll be the turning point at which cryptocurrency goes mainstream.

Chapter 12: Ethereum's Casper: Ghostbusting Proof of Stake Problems

BY TONINO JANKOV

Every complex society has many cases where a big number of players need to reach consensus. The main advantage of such a society is that people can agree and act together on different issues, and can solve more and bigger problems than individuals can do alone.

Society is based on consensus. Trade, diplomacy, finances — they all have this prerequisite of consensus. This has created a need for different mechanisms of governance to reach consensus. And with it, often, comes a lot of abuse, manipulations and rogue players.

To prevent different manipulations, we've had centralized institutions that gave us efficiency in reaching consensus and making decisions. The necessary tradeoff between democracy and efficiency is well known among the students of different modes of government.

While different levels and models of centralized decision-making have lead us to sophisticated, functioning society, it

has also lead to accumulation of power and resources in the hands of the few. In modern days, things like the Internet and cryptography have given us power to explore new models of reaching consensus.

Blockchain is one of the modern inventions that's supposed to enable us to reach consensus without centralization and trusted parties. It is, in its many incarnations, still in its baby stages, but we can safely say that if it delivers on just a part of its promises, it will help us have a more efficient society without the tradeoffs.

Byzantine Fault Tolerance

When we start designing (or imagining) decentralized institutions, decentralized organizations, we can reduce (in programmers' terms) all the related logic and data structures down to *voting* and a *ledger* of decisions/votes.

The ideal of decentralization involves applications that are robust and have no single point of failure. No central authority, and *yet*, we should be able to guarantee the integrity of the ledger. For example, we need to know that our bank accounts, or vote results, or some online text, or the content of some online conversation, or some digital identity *have not been tampered with*.

Without central, trusted players, *this can be hard*.

In a decentralized, untrusted environment, we need to provide for cases of rogue players, who will try to manipulate the

consensus. To simplify: everyone needs to have a vote, but no one should be able to cheat the system using their voting right.

When a decentralized system — which *consists of untrusted parties* — is able to function when there are treacherous parties in its midst, we say it has Byzantine fault tolerance.

In cases of creating a decentralized financial system, or critical decision-making systems, incentives for *cheating the system* are huge, so these systems need to be bulletproof.

Cryptocurrency solutions of the recent years have addressed the “Byzantine generals problem” by various consensus mechanisms, combining cryptography and various game theory strategies.

Consensus Mechanisms

In blockchain applications - and more narrowly, cryptocurrencies — different consensus mechanisms are, really, ways to solve the “rogue players” problem of trustless systems we outlined above — the *Byzantine generals problem*. These algorithms are different ways to deter the players from cheating the system. These players are usually known as **nodes** — computer systems with instances of the blockchain, or which validate new transactions and *mine* new blocks.

Consensus mechanisms are devised to make it non-lucrative, or not worth the effort, to cheat the system.

Read more about these mechanisms in this post.

Ethereum Consensus Algorithms

Perhaps the most mainstream consensus algorithm used today is PoW (Proof of Work). This algorithm stakes the integrity of the network on the difficulty required to solve hashing puzzles. The most basic explanation of BTC Proof of work is that a mining node takes transactions and tries to hash them together with a number — a **nonce** — so that the resulting hash begins with a predetermined number of zeros. The higher the number of beginning zeros, the higher the difficulty. Mining nodes try by brute force to find the smallest possible nonce that, hashed together with transactions in the block, satisfies a given condition.

Ethereum developers have made decentralization and democratization its goals — and therefore, in contrast to Bitcoin, ASIC resistance. This has resulted in using ETHASH, an algorithm which emphasizes the speed of manipulating memory, instead of raw computing power, and this gives some advantage to using graphic cards and reduces incentive of developing ASICS (application specific integrated circuits) — dedicated, specialized hardware developed specifically for the purpose of solving PoW problems in particular blockchains.

The other feature of Ethereum's PoW is the *Ghost Protocol*, which is implemented to combat centralization by distributing rewards for orphaned blocks — blocks mined by miners who lost the race to create the next block first, but still mined a valid block.

“Ghost” is short for Greedy Heaviest Observed Subtree and is a rather complex way of addressing the issues we mentioned.

Proof of Work has many flaws. For the system to function, ideally, every node has to be able to verify the entirety of the blockchain transactions — meaning, it has to store and validate the entire blockchain history. As Ethereum’s Wiki says, Ethereum, like Bitcoin,

suffers from the flaw that every transaction needs to be processed by every node in the network.

Proof of Work is, further, extremely energy-inefficient, and with the enormous energy demands required just for the network to function, it becomes a big liability, and the network becomes fragile. Proof of Work also (even with ETHASH) still struggles with centralization trends, with miners pooling in their resources, and with network relying on these mining pools, and other infrastructure problems.

Casper, the “Friendly Ghost”

Ethereum’s upcoming Serenity release plans to transition the Ethereum blockchain to the PoS (Proof of Stake) consensus protocol. This protocol is called Casper, because it implements or adapts many principles of Ghost PoW we mentioned. Currently, both Vitalik Buterin and Vlad Zamfir are working on this release, and many final details still haven’t been finally decided.

The essence of Casper is staking and betting. Validating nodes

will stake a certain amount of cryptocurrency — let's say 1000 ETH, or more — which will enable them to partake in validation of transactions and production of blocks. By eliminating the search for a solution to cryptographic puzzles, we can expect much bigger scalability of the network.

Validating nodes will produce blocks and *bet* on validity of transactions. These transactions/blocks will then be verified by other nodes. If validating nodes place their bet on promoting some invalid transactions as valid — in other words, if the validating nodes *try to cheat* — they will forfeit their stake, along with their place in the validation process.

This approach of *betting*, and possibly *losing the stake*, is Ethereum's way of addressing the *Nothing at Stake Problem*, which occurs when the best strategy for a validator is to support every possible version of blockchain history (of transactions) without risking anything. In other words, to demotivate miners to back fraudulent transactions.

Without this solution, if the miners only have prospects of getting rewards for backing *any* version of blockchain history — as the Ethereum's PoS FAQ puts it —

if all actors are narrowly economically rational, then even if there are no attackers, a blockchain may never reach consensus. If there is an attacker, then the attacker need only overpower altruistic nodes (who would exclusively stake on the original chain), and not rational nodes (who would stake on both the original chain and the attacker's chain), in contrast to proof of work, where the attacker must overpower

both altruists and rational nodes

So this is a way to ensure the integrity of the blockchain ledger and its transactions.

When nodes have staked an amount like 1000+ ETH, their incentive is primarily to keep their stake. Having in mind that validating nodes are chosen randomly, so rogue players cannot be sure they will be able to overtake the validation process, this makes it very costly to even try to cheat the system.

In the future, the required stake may decrease, to allow for greater decentralization of the network. The Ethereum founders mentioned the possibility of it coming down to as low as 10 Ether with time.

Advantages of Proof Of Stake

- It's energy efficient.
- Smaller incentives are needed to keep the network going, due to smaller costs of mining new blocks (both energy-wise and hardware-wise).
- Reduced need for incentives solves, or reduces, a whole range of problems that need to be solved in PoW system, like difficulty adjustments, and generally reduces complexity of mechanisms needed to regulate the network.
- Compared to 51% attacks in PoW system, where the attackers don't lose their main weapons of attack, their hardware, attackers who try to overtake the Ethereum network stand to lose the main source of their power — their stakes.

Problems with PoS

One of the main problems that people point out regarding PoS is, again, a certain level of centralization, or “the rich get richer” problem. Rewards from the network will go only to those who already have a high amount of capital.

During a recent conference in Curacao, VMware’s prominent distributed system’s expert Dahlia Malkhi pointed out these problems, emphasizing that it’s “giving power to people who have lots of money”.

But those with their eyes on the crypto industry for the last couple of years have seen their fair share of “whale games” and market manipulations even in those cryptocurrencies which boast by their different decentralization mechanisms. It’s hard to claim any mechanism will inherently secure fairness *and* efficiency *and* security and fault tolerance.

So perhaps being perfectionist here is unrealistic, and speculations about consequences of different design decisions aren’t always reliable. Besides, the rich get richer in PoW, too. In fact, it’s only possible to become an effective miner if you’re rich. In Ethereum, everyone will be able to be a “miner” in time.

Conclusion

We still don’t know the final shape this Ethereum upgrade will take, and even its core developers don’t seem to be always on the same page about it. There are two versions that are being juggled around, but Serenity is coming, and with it both

Casper and other improvements of the protocol that improve its scalability and efficiency. If it is rolled out with the “Constantinople” update — as is speculated — we could see Casper published by the end of 2018.

Chapter 13: Decentralized Storage and Publication with IPFS and Swarm

BY TONINO JANKOV

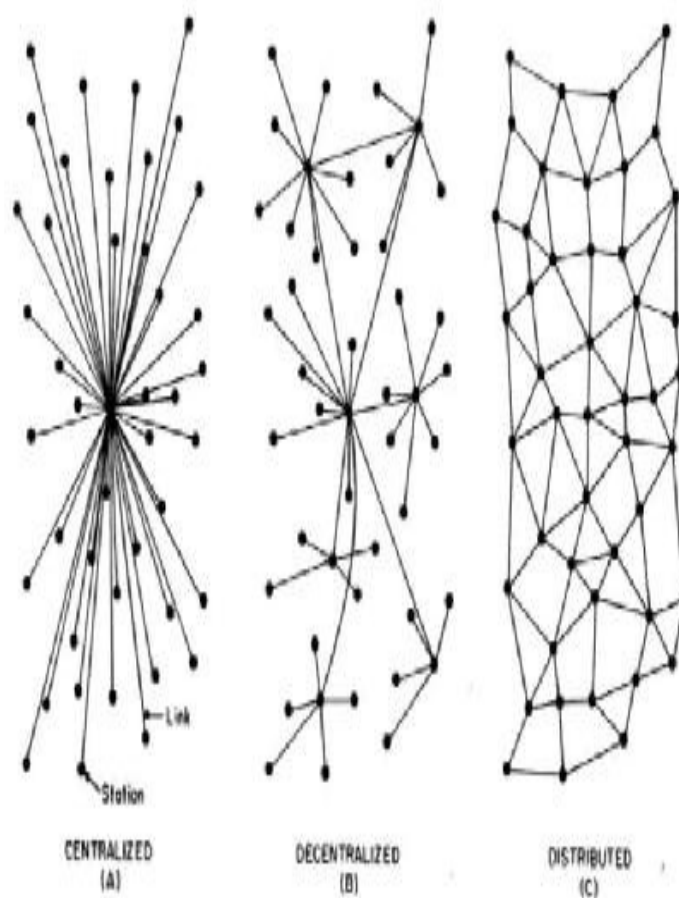
In this article, we outline two of the most prominent solutions for decentralized content publication and storage. These two solutions are IPFS (InterPlanetary File System) and Ethereum's Swarm.

With the advent of blockchain applications in recent years, the Internet has seen a boom of decentralization. The developer world has suddenly gotten the sense of the green pastures that lie beyond the existing paradigm, based on the server–client model, susceptible to censoring at the whims of different jurisdictions, cloud provider monopolies, *etc.*

Turkey's ban of Wikipedia and The “Great Firewall of China” are just some examples. Dependence on internet backbones, hosting companies, cloud providers like Amazon, search providers like Google — things like these have betrayed the initial internet promise of democratization of knowledge and access to information.

As this article on TechCrunch said two years ago, the original

idea of the internet was “*to build a common neutral network which everyone can participate in equally for the betterment of humanity*”. This idea is now reemerging as Web 3.0, a term that now means the decentralized web — an architecture that is censorship proof, and without a single point of failure.



As Gavin Wood, one of Ethereum’s founders, in his 2014 seminal work on Web 3.0 put it, there is “*increasing need for a zero-trust interaction system*”. He named the “*post-Snowden web*”, and described four components to it: “*static content publication, dynamic messages, trustless transactions and an*

integrated user-interface”.

Decentralized Storage and Publication

Before the advent of cryptocurrency — and the Ethereum platform in particular — we had other projects that aimed to develop distributed applications.

- **Freenet**: a peer to peer (p2p) platform created to be censorship resistant — with its distributed data store — was first published in 2000.
- **Gnutella network**: enabled peer-to-peer file sharing with its many client incarnations.
- **BitTorrent**: was developed and published as early as 2001, and Wikipedia reports that, in 2004, it was “*responsible for 25% of all Internet traffic*”. The project is still here, and is technically impressive, with new projects copying its aspects — hash-based content addressing, DHT distributed databases, Kademlia lookups ...
- **Tribler**: as a BitTorrent client, it added some other features for users, such as onion routed p2p communication.

Both of our aforementioned projects build on the shoulders of these giants.

IPFS

The InterPlanetary File System was developed by Juan Benet, and was first published in 2014. It aims to be a protocol, and a distributed file system, replacing HTTP. It’s a mixture of technologies, and it’s pretty low level — meaning that it leaves a lot to projects or layers built on top of it.

An introduction to the project by Juan Benet from 2015 can be found in [this YouTube video](#).

IPFS aims to offer the infrastructure for reinventing the Internet, which is a huge goal. It uses *content addressing* — naming and lookup of content by its cryptographic hash, like Git, and like BitTorrent, which we mentioned. This technique enables us to ensure authenticity of content regardless of *where it sits*, and the implications of this are huge. We can, for example, have the same website hosted in ten, or hundreds of computers around the world — and load it knowing for sure that it's the original, authentic content just by its hash-based address.

This means that important websites — or websites that may get censored by governments or other actors — don't depend on any single point, like servers, databases, or even domain registrars. This, further, means that they can't be easily extinguished.

The Web becomes resistant.

One more consequence of this is that we don't, as end users, have to depend on internet backbones and perfect connectivity to a remote data center on another continent hosting our website. Countries can get completely cut off, but we can still load the same, authentic content from some machine nearby, still certain of its authenticity. It can be content cached on a PC in our very neighborhood.

With IPFS, it would become difficult, if not impossible, for

Turkey to censor Wikipedia, because Wikipedia wouldn't be relying on certain IP addresses. Authentic Wikipedia could be hosted on hundreds or thousands of local websites within Turkey itself, and this network of websites could be completely dynamic.

IPFS has no single point of failure, and nodes don't need to trust each other.

Addressing the content is algorithmic — and it becomes uncensorable. It also improves the efficiency. We don't need to request a website, or video, or music file from a remote server if it's cached somewhere close to us.

This can eliminate request latency. And anyone who's ever optimized website speed knows that network latency is a factor.

By using the aforementioned Kademlia algorithm, the network becomes robust, and we don't rely on domain registrars/nameservers to find content. Lookup is built into the network itself. It can't be taken down. Some of the major attacks by hackers in recent years were attacks on nameservers. An example is [this particular attack in 2016](#), which took down Spotify, Reddit, NYT and Wired, and many others.

IPFS is being developed by Protocol Labs as an open-source project. On top of it, the company is building an incentivization layer — Filecoin — which has had an initial coin offering in Summer 2017, and has collected around \$260

million (if we count pre-ICO VC investment) — perhaps the biggest amount collected by an ICO so far. Filecoin itself is not at production-stage yet, but IPFS is being used by production apps like OpenBazaar. There's also IPFS integration in the Brave browser, and more is coming ...

The production video-sharing platform d.tube is using IPFS for storage, while Steemit is using it for monetization, voting, *etc.*

It's a web app that's waiting for wider adoption, but it's currently in production stage, and works without ads.

Although IPFS is considered an alpha-stage project, just like Swarm, IPFS is serving real-world projects.

Other notable projects using IPFS are Bloom and Decentraland — an AR game being built on top of the Ethereum blockchain and IPFS. Peerpad is an open-source app built to be used as an example for developers developing on IPFS.

Swarm

According to Viktor Tron, of the Ethereum Foundation, “basically, Swarm is BitTorrent on steroids”.

Swarm, by Ethersphere, aims to solve the same problems as IPFS. According to its GitHub page —

Swarm is a distributed storage platform and content

distribution service, a native base layer service of the Ethereum Web 3 stack. The primary objective of Swarm is to provide a sufficiently decentralized and redundant store of Ethereum's public record, in particular to store and distribute Dapp code and data as well as block chain data.

Viktor Tron is currently behind Swarm as its lead developer. He was one of the first employees of the Ethereum Foundation. Ethereum Foundation is funding the project development, along the lines of Gavin Wood's vision of Web 3.0 that we quoted. So, Swarm is more integrated with the Ethereum ecosystem, and along with Whisper and Ethereum Virtual Machine, it's aiming to build a next-generation platform for distributed apps, or Dapps.

Swarm is in an earlier stage of development than IPFS. To quote Viktor Tron —

IPFS is much further along in code maturity, scaling, adoption, community engagement and interaction with a dedicated developer community.

Once Swarm becomes production-ready, it will provide an incentivization layer and integration with Ethereum's smart contracts, which should give plenty of room for creativity and innovative applications.

Neither the incentivization layer of Swarm nor of IPFS (Filecoin) are currently ready for use.

Reasons for Optimism

At the time of writing (May 2018), Swarm's lead developer has announced the release of POC3, which keeps its roadmap on the clock, and gives reasons for optimism regarding Swarm becoming production-ready in 2018.

While IPFS aims to build a *protocol*, and is a lower-level, more generic project, Swarm ties into the Ethereum's Web 3 vision, with more focus on censorship resistance: it *“implements plausible deniability with implausible accountability through a combination of obfuscation and double masking”*.

This reminds us of the Freenet project, where those hosting certain content don't necessarily have access to it, or know what it is.

Swarm, with its incentivization mechanisms, is aiming to provide higher level solutions. It —

exploits the full capabilities of smart contracts to handle registered nodes with deposit to stake. This allows for punitive measures as deterrents. Swarm provides a scheme to track responsibilities making storers individually accountable for particular content.

Compared to IPFS, Swarm has a lot of focus on these mechanisms. On the one hand, this includes incentives for long-term storage of not-so-popular content, and on the other, incentives for highly popular, high-bandwidth content. These two require two different approaches to penalties/rewards.

In Swarm's case, this requires working on cryptographic constructs known as Proof-of-Custody, which make it possible

“to have a compact proof, proving to any third party that you store a blob of data without transferring the whole data and without revealing the actual contents”. So proving a storage of some content doesn't require the full download of that content every time.

Swarm even has an Accounting Protocol, SWAP, currently in development, as one level of incentivization.

Currently, before incentivization mechanisms are published, which is expected to happen in 2018, Swarm functions like a cache: less popular content can get deleted, and there's no insurance against that.

Swarm will be usable as cloud hosting, while IPFS relegates this to projects that will be built on its infrastructure. IPFS leaves it to the implementors/developers to find the actual storage devices.

IPFS itself, as lower layer, has no guarantees of storage. While Swarm includes this in its roadmap, the IPFS team, in comparison, plans this on the Filecoin level, but it's just in idea stage at the moment.

There's a two-part YouTube interview with Tron where he explains the Swarm project in less technical terms.

There are two projects that build further on IPFS and Swarm that are worth mentioning in the context of Dapps: distributed applications. Since both projects allow for only a limited level of dynamic content, database-oriented projects built on top of

these distributed systems add significant value.

OrbitDB is a “serverless, distributed, peer-to-peer database” that uses IPFS for its data storage.

It’s a database that works both for Node.js and in browsers. Its development is active, and is being sponsored by Protocol Labs. After its \$260 million fundraising in 2017, the future of OrbitDB — just like that of IPFS — looks promising.

OrbitDB is part of the Node.js/npm ecosystem.

Wolk is a project/token that’s building a database — SWARMDB — using Swarm’s codebase. Behind it is a Californian startup, Wolk Inc., that managed to raise around 7,100 ETH in its ICO in 2017. WOLK promises a censorship-resistant distributed database powered with WLK token as its incentivizing layer. It provides a Go, Node.js and HTTP interface.

They claim Swarm and Bancor as their partners.

While it’s hard to predict success and adoption of these projects, or ascertain their quality, as IPFS and Swarm progress and become more production-ready and reach wider adoption, it’s pretty certain we’ll see more projects like these.

Swarm’s Orange Paper is an interesting, albeit a very technical read.

A longer comparison of the two projects can be found here.

Commonalities

Things that both IPFS and Swarm share are **hash-based content addressing**, which we described before. And while this provides git-level version control of the content, hosted on both systems, and censorship-resistance, deleting the content is something that remains to be solved.

Immutability provides guarantees of authentic content, but changes to the content produce new addresses, so to provide editing capability, additional layers are necessary.

From the perspective of different web apps, *both projects support only static content*. So, there's no back-end apps with interpreted languages, like PHP, Python, Ruby, or Node.js. For Swarm, this is where EVM comes into play, but EVM also has its own inherent limitations.

Conclusion

Both IPFS and Swarm are promising projects, although one can't help but wonder if the developers have set overly ambitious goals. If they succeed with their development roadmaps, and achieve wider adoption, there's no doubt this will bring big changes to the Internet as we know it.

Chapter 14: Ethereum Messaging: Explaining Whisper and Status.im

BY TONINO JANKOV

This article will show how Ethereum, rather than being just a cryptocurrency or smart contracts platform, is actually developing into a whole ecosystem.

There are four components to the post-Snowden Web: static content publication, dynamic messages, trustless transactions and an integrated user-interface. — Gavin Wood

In the 1990s, the Internet sprang into existence and, year by year, it has revolutionized the way we communicate, the way we do business, the way we consume news and TV. In many ways, it has democratized access to information, and drastically lowered global communications costs, but it has also raised the average citizen's expectations in regards to access to communications, news, and privacy.

Websites like Wikileaks, Facebook, Twitter, in the second wave — dubbed Web 2.0 — have, along with other websites, like Youtube, Linkedin, and many personal publishing platforms like WordPress, changed the publishing of

information, and made it hard to hide. Whatever news is out there is bound to be revealed, sooner or later.

Governments and officials have gone down because of this. For good and for bad, what began as leaked cables published by Wikileaks in 2010, fomented public revolt in countries like Tunisia, Libya, and sparks later turned to fires that changed the face of the Middle East. At that time, sparked by leaks of governments cables, western countries saw a series of protests as well.

In his book *When Google Met Wikileaks*, Julian Assange outlined often unholy links between politics, tech giants and the intelligence community. Commenting on Eric Schmidt's and Jared Cohen's book *The New Digital Age*, he rightly notes:

while Mr. Schmidt and Mr. Cohen tell us that the death of privacy will aid governments in "repressive autocracies" in "targeting their citizens," they also say governments in "open" democracies will see it as "a gift" enabling them to "better respond to citizen and customer concerns".

The crackdown of financial institutions on Wikileaks then showed us how vulnerable to censorship we are — even in the age of the Internet.

Then came the Snowden revelations, and the public's illusion about the actual extent to which our privacy is breached on an everyday basis was flushed down the toilet.

It was in this environment that Ethereum was created. Some months after the Snowden revelations, Gavin Wood, co-creator of Ethereum, wrote an article outlining his vision of Web 3.0 — a web which utilizes the internet infrastructure we already have, and cryptography that's available, along with the blockchain, to build a better internet. This is to include content publishing, messaging, and value transactions — in a decentralized, censorship-proof way, with privacy guaranteed.

Whisper

In the article, Wood outlines an *identity-based pseudonymous low-level messaging system*, a system that will give its users — both people and DApps — hash-based identities, privacy assurances, encrypted messages, cryptographic guarantees about senders, and messages with a defined time-to-live. This system has, for lack of a better word, modular privacy and anonymity, and guarantees of “darkness” — allowing users to opt-in or out of different privacy features. It uses the infrastructure of the Ethereum network.

Whisper is being built as a *protocol*, meaning that it lays the foundation for higher-level implementations, DApps, built on it, with different variations, using different features of the protocol, and different settings. It's currently at POC 2 stage, being usable in current versions of geth and Parity. The usage on the mainnet is restricted by the number of running, production Ethereum nodes that have the Whisper protocol enabled. The protocol is, we can say, in alpha stage. Many specs will change.

Both Ethereum and Whisper client nodes use the DEVp2p Wire Protocol for their P2P communication. In particular, the RPLx protocol is used, which is described as —

a cryptographic peer-to-peer network and protocol suite which provides a general-purpose transport and interface for applications to communicate via a P2P network.

The Node-discovery algorithm of a decentralized, censorship resistant network is its major part. Ethereum uses adapted Kademlia UDP for this, similar to Bittorrent network's peer discovery.

Because of the evolving specs, the best place for the current definition of the protocol is Ethereum's wiki — currently Whisper POC 2 Spec page.

Whisper combines aspects of both DHTs and datagram messaging systems (e.g. UDP).

When designing a system that aims for *complete darkness* — meaning guaranteed privacy and anonymity — there are performance tradeoffs. This is, we presume, the reason for the choice of UDP, which is lower level, but at the same time faster than TCP, and gives greater control.

This line from the spec that may explain many of the design decisions:

It is designed to be a building block in next generation DApps which require large-scale many-to-many data-discovery, signal negotiation and modest transmissions with an absolute

minimum of fuss and the expectation that one has a very reasonable assurance of complete privacy.

As the spec says, there's an important distinction between encryption of messages and pitch-black darkness, which is what the designers of Ethereum are trying to achieve. Today we know that well-funded actors are able to break privacy guarantees even of networks like Tor. For many purposes, merely knowing the destination of someone's communication can end the needed privacy guarantees, without ever breaking the encryption of the content. (A political party insider communicating to Wikileaks would be one example.) Metadata about our communication, analyzed in sufficient, bulk amounts, can give a lot of data, and sometimes annul the effect of encrypted content.

Recent GDPR legislation in Europe somewhat reflects this.

This is why there's a need to reach deep — to code a new system starting at a very low level.

Whisper's POC2 promises a "100% dark operation" — which is a bold claim.

They continue:

This applies not only for metadata collection from inter-peer conduits (i.e. backbone dragnet devices), but even against a much more arduous "100% - 2" attack; *i.e.* where every node in the network were compromised (though functional) save a pair running DApps for people that wanted to communicate

without anybody else knowing.

Protocol Elements

The main elements are *Envelopes*, *Messages* and *Topics*.

Envelopes are packets that *contain* time-to-live (in seconds), *expiry* (in Unix time), *topics* (“these might, for example, correspond to “twitter” hash tags or an intended recipient’s public key hashed with some session nonce or application-identity”) and *nonce* (which provides for proof-of-work requirements for message senders in future implementations). And then there’s the *message data field*.

The **Message data field** within an envelope contains the actual message — the *payload* — and *flags* and *signature*. Payloads are encrypted by the sender and decrypted by the recipient, both in one of two ways.

The protocol provides for ranking of peers by the nodes, and ranking of the messages themselves by the *work spent* in obtaining the *nonce* that we mentioned before. Proof of bigger work should afford a message bigger priority on the network.

Nodes can advertise their topics of interest to each other. Senders and recipients can opt in or out of different privacy features versus performance features, because this is sometimes a tradeoff.

no TTL and not designed for asynchronous data publication.

- Tox: Higher level (IM & AV chat) replacement.

Basic Design

Uses the "shh" protocol string of DEVP2P.

Rest coming soon, once I've finished prototyping. Gav.

Considerations for Defeating Traffic Analysis

(from Ickiverloren) All existing protocols for location obscured instant messagi

Since the spec is currently fluid, and the implementation is being worked on, it's hard to precisely distinguish what's already implemented, what's on the way, and what's on a level of proposal.

Status.im

Status calls itself "*A mobile OS, built for Ethereum*". It's an Ethereum client that's meant to bring the richness of Ethereum capabilities to smartphones. It was introduced in 2016 at Devcon2 in Shanghai.

Under the hood, the mobile app runs a full implementation of geth. DApps are being run on user's mobile phones. They can be added to chats.



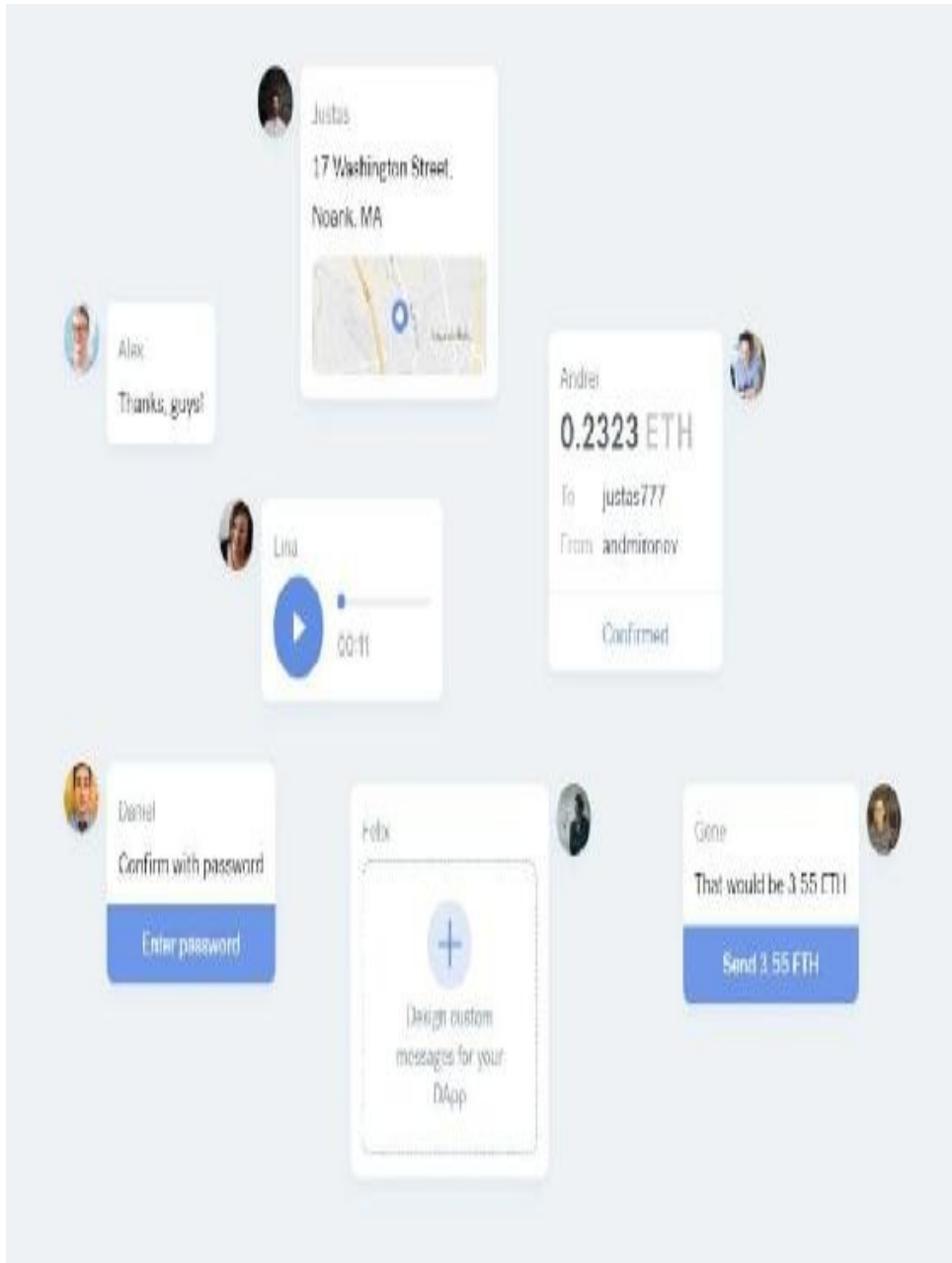
The chat feature is implemented on top of Whisper.

At the time of writing this article (May 2018), Status 0.9.18 is the current release.

Many DApps, such as Gnosis, Aragon, Etherisc, Uport, Ethlance, CryptoKitties, Bancor, Peepeth and others are available within the app.

Status combines a messenger and browser running on top of the Ethereum network and is aiming to be a ÐApp browser, enabling developers to reach the users. As such — along with similar projects that may spring up — it's an important, high-level element in Ethereum and blockchain applications coming to the mainstream. They say their mission is *"to lower the barriers to entry on Ethereum"*.

Besides the Ether wallet, and encrypted chat, it also aims to provide a kind of a social network built on top of ÐApps, and the Whisper protocol, with optional levels of anonymity.

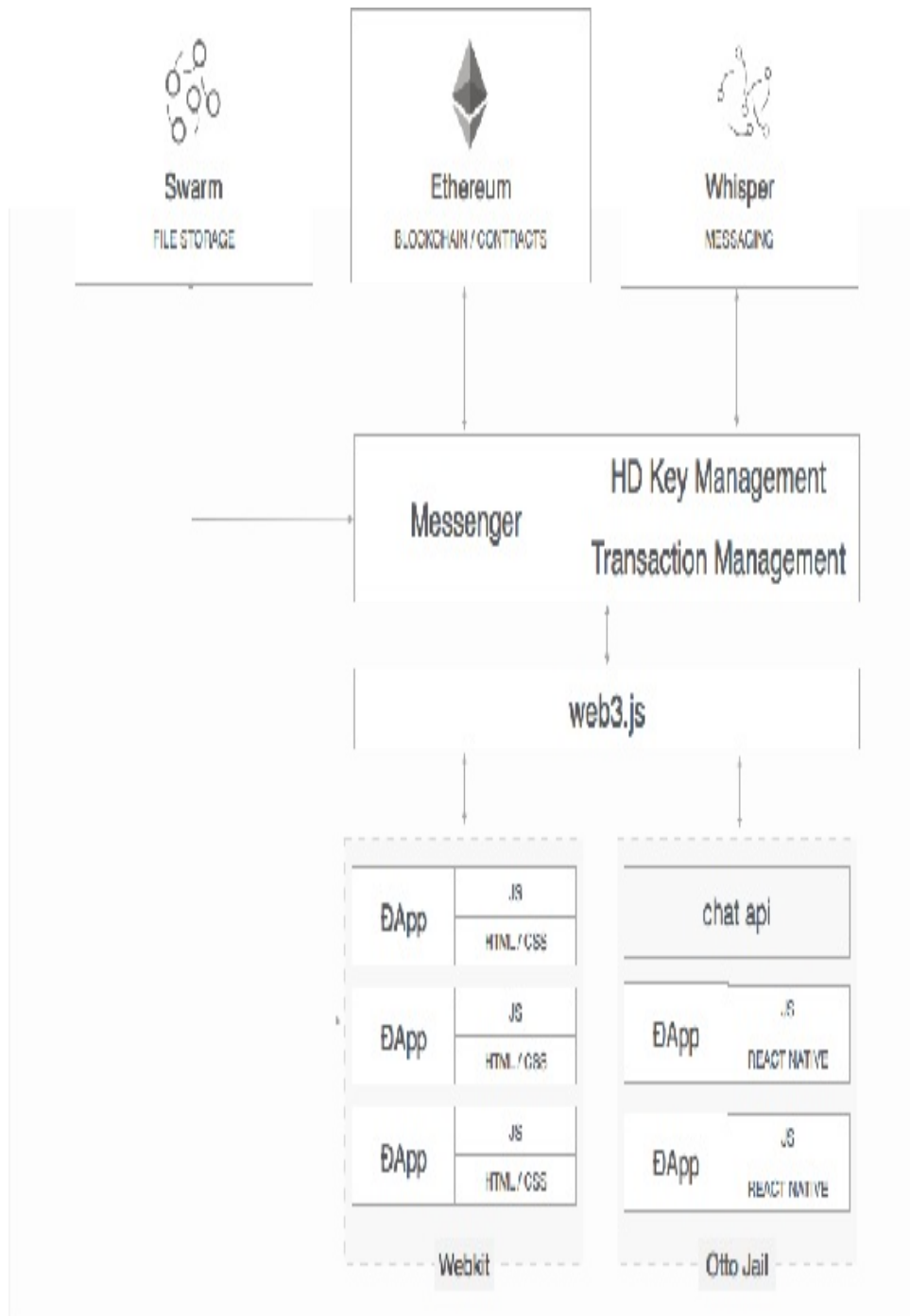


The release article says the following:

In this latest release, group chat has been temporarily disabled so that we can focus efforts on 1–1 and public chat. It will be re-introduced in a future release.

A **P2P exchange** is meant to facilitate fiat-to-ETH real-world

decentralized exchange. Their business model is to charge a small fee for this.



STATUS NETWORK TOKEN

In May 2017, folks from Status announced *the Status Network*, and the token, *SNT* — a utility token which is meant to facilitate the functioning of the network and application. It will serve as the mechanism of governance of the client, decentralized push notification market, username registrations using *Ethereum Name Service*, curation of content, the aforementioned P2P exchange and other features.

The white paper is available [here](#). Judging from *CoinMarketCap* data, trading information about the token, which can often be used to ascertain how active the project is, Status Network, with market cap of ~\$300 000 000 is doing well, and the market seems to have accepted it.

STATUS WIKI

Status.im wiki has a user guide, and informations for developers, either those who want to build a serverless chatbot, or their own ÐApp, and add it to Status.

Their *Open Bounty* offers rewards to developers who contribute to development.

The Ecosystem

By building a space for ÐApps to reach the end users, and bringing the Ethereum technology to the masses on a higher level, Status is building an ecosystem. This is further emphasized in their ***incubator program*** — “*An Incubator Dedicated to Decentralization*”.

Status' **Embark** is a framework for developing DApps. It can be installed via npm:

```
npm install -g embark
```

It's a framework that attempts to unify and ease the usage of all three parts of Web 3 — Swarm, or IPFS, for storage, Whisper or Orbit for communication, Ethereum blockchain with DApps itself, and, at last, front-end technologies like React, Foundation, Angular, and different front-end utility tools.

There's a dashboard, configuration system, a plugins ecosystem. More can be learned [here](#).

The company is incorporated in Zug, Switzerland, with offices in Singapore. The mobile app is being developed, besides the standard Go — the language of geth — in Clojure/Clojurescript, a functional language that lends itself nicely to advanced logic.

Conclusion

Building the Web 3.0 is a big endeavor. Many companies are trying to fix the internet. The success is varying. But the Ethereum project had the vision of changing the Internet from the start. So, to talk about Ethereum just as a cryptocurrency platform, or smart contracts platform, would be an understatement. There is a whole ecosystem being built beside the Ethereum blockchain.

Judging by the adoption of the Ethereum itself, incentives system built in, and the activity around the other two branches — Swarm and Whisper, and projects like Status.im — we have basis to expect some very interesting changes to the Internet as we know it in the coming years.

Chapter 15: Ethereum: Internal Transactions & Token Transfers Explained

BY BRUNO ŠKVORC

When using various Ethereum blockchain explorers like Etherscan to inspect addresses, you may come across certain addresses which have *Transactions*, *Internal Transactions* and *Token Transfers*. To understand the difference between them, we first have to understand the concept of external and internal addresses on Ethereum.

External and Internal Addresses

There are two types of addresses (accounts) in Ethereum: External and Internal.

When a user creates an address, that's called an *external* address because it's used for accessing the blockchain *from the outside* — from the “user world”.

When you deploy a smart contract to the Ethereum blockchain, an *internal* address is generated which is used as a

pointer to a running blockchain program (a deployed smart contract). You can target it from the outside for calling functions, or you can target it from the inside so another deployed contract can call functions on an already deployed contract.

It's important to note that *all* transactions on the Ethereum blockchain are set in motion from external accounts. Even if one smart contract is supposed to call another and that one in turn calls another, the very first transaction *must* be done by an external account. There's currently no way to automatically call a transaction from the outside, though solutions are being worked on.

The key difference between external and internal accounts is the following:

External addresses have private keys and can be accessed by users. Internal addresses cannot be accessed directly as a wallet, and can only be used by calling their functions.

Transaction Types

This brings us back to transaction types. Let's inspect this address.

Overview

10.809650361265263 Ether

\$7,100.20 (\$414,067)

72 txns

Mac

Address Watch: Add To Watch List

Token Balances: View \$5.62 +

More Options

Transactions

Internal Transactions

Token Transfers

Comments

17 Latest 25 txns from a total of 72 transactions

View All

TxHash	Block	Age	From	To	Value	(To/Fee)
0x542ac5b400e411...	6652282	7 days 2 hrs ago	0x141380c15a82375...	In	5 Ether	1,000%
0x53039a400e402...	6633360	12 days 4 hrs ago	BinanceWallet	In	0.2 Ether	1,011%
0x18e893a19c93c18...	6488034	20 days 8 hrs ago	0xb2222222222222...	Out	0.6 Ether	1,000%
0x386b55a3a0054b...	6488111	20 days 9 hrs ago	0xb2222222222222...	Out	0.13333333 Ether	1,000%
0x386b55a3a0054b...	6488108	20 days 9 hrs ago	0xb2222222222222...	Out	0.13333333 Ether	1,000%
0x144c02a5283a18...	6488009	20 days 9 hrs ago	0xb2222222222222...	Out	0.13333333 Ether	1,000%

The address has several entries in the *Transactions* tab — some outgoing, some incoming. These transactions are *external* transactions — to and from external accounts. So as per the screenshot, we can see this address received 5 ether 7 days ago from this address and 0.2 ether 12 days ago from a Binance wallet. But if we look at the most recent sender, we'll see some more interesting entries:

Transactions

Token Transfers

Comments


12

Latest 25 items from a total of 83 transactions

View All

To/Hash	Block	Age	From		To	Value	(To/From)
0x6842a6f0d0a41f...	658282	7 days 14 hrs ago	0x14138ccf3a82...	OUT	0xb2b28870398b...	6 Ether	3,00521
0x48cd4032c5b2...	658209	7 days 14 hrs ago	0x7071121e038e98...	IN	0x14138ccf3a82...	1.1675558 Ether	3,00482
0xc44d3ba3e38e6b...	3561879	7 days 15 hrs ago	0xc5e2c3175e2e01...	IN	0x14138ccf3a82...	3.00042 Ether	3,00128
0xad1a78204ad7b2...	3561811	7 days 16 hrs ago	BinanceWallet	IN	0x14138ccf3a82...	0.7554553 Ether	3,00178
0x4e0d3440d4825...	4830288	132 days 16 hrs ago	0x14138ccf3a82...	OUT	EOSTokenContract	0 Ether	3,002485
0x78d7d4fced7a7e...	4830277	132 days 16 hrs ago	0x14138ccf3a82...	OUT	EOSCrowdsale	0 Ether	3,002683
0x6dd7bd38f1a2f6...	4830251	132 days 16 hrs ago	0x14138ccf3a82...	OUT	EOSCrowdsale	3.10 Ether	3,003708
0x80ec5cb0c07813...	4830243	132 days 16 hrs ago	BinanceWallet	IN	0x14138ccf3a82...	3.18541382 Ether	3,01138
0x5a00413f058b328...	4830781	138 days 16 hrs ago	0x14138ccf3a82...	OUT	EOSTokenContract	0 Ether	3,002485
0xd11632cdd7776b...	4807880	138 days 16 hrs ago	0x14138ccf3a82...	OUT	EOSCrowdsale	0 Ether	3,004086

This address has been sending out Ether to individuals, but it did something else, too: it contributed to the EOS crowdsale and withdrew EOS tokens. Most of these transactions send 0 ether; they just call functions. For example, [this transaction](#) shows that almost half a year ago our protagonist called the `claim` function on the EOS Crowdsale contract, and this resulted in the crowdsale sending that person 312 tokens in return.

To:	Contract 0xd3a808e544bc68db5db3a091b171a77407ff7cef (EOSCrowdsale) 
Token Transfer:	312,803,736,865,917,934,133 (\$4,567.01)  EOS Token from 0xd3a808e544bc68db5db3a091b171a77407ff7cef to 0x14138cc3a82375...
Value:	0 Ether (\$0.00)
Gas Limit:	80000
Gas Used By Txn:	81259
Gas Price:	0.00000005 Ether (50 Gwei)
Actual Tx Cost/Fees:	0.00406265 Ether (\$2.84)
Nonce & (Position):	53 (166)
Input Data:	Function: claim Quint256 day)

The details of this transaction don't matter much. We're just looking at it to define the *Transactions* tab properly:

The *Transactions* tab lists all transactions *initiated* by external accounts, regardless of who initiated them — the receiver or the sender.

Now let's look at the second tab of our first address: *Internal Transactions*.

The Transactions Tab

Not all addresses will have this tab. It's only present when an internal transaction actually happened on an account.

Transactions	Internal Transactions	Token Transfers	Comments		
Internal Transactions as a result of Contract Execution					
[7] Labeled 3 Internal Transactions					
ParentTxHash	Block	Age	From	To	Value
0xdd832030e41f81...	400418	168 days 6 hrs ago	0x73b2ed78a87b71...	→ 0xb2b28f870336b4eaa0acc73ce02757fc428dc9	0.40 Ether
0xb844409a0e838e...	400417	168 days 6 hrs ago	0xe213cd93d0c0d4...	→ 0xb2b28f870336b4eaa0acc73ce02757fc428dc9	0.40 Ether
0x1072a232e1a0fc...	4001018	172 days 23 hrs ago	0xaccab068e5c29a...	→ 0xb2b28f870336b4eaa0acc73ce02757fc428dc9	0.04 Ether

Let's look at one of these — for example, this one.

To:

Contract [0xb2b28f870336b4eaa0acc73ce02757fc428dc9](#) (ENS-Registrar)

TRANSFER: 0.04 Ether from [0xaccab068e5c29a04986a05e76296a208a83e507e500c07a566d](#) to [0xb2b28f870336b4eaa0acc73ce02757fc428dc9](#)

ENS Auction/Realized for [0x8666a1a0001c88c0a8a523804a986a05e76296a208a83e507e500c07a566d](#) with 0.07 Ether locked up

Value: 0 Ether (\$0.00)

Gas Limit: 200000

Gas Used By Txn: 98791

Gas Price: 0.00000021 Ether (21 Gwei)

Actual Tx Cost/Fee: 0.002095811 Ether (\$1.48)

Nonce & (Position): 2 | (17)

Input Data:

Function: `finalizeAuction(bytes32 _hash)`

This was a bid on the ENS (Ethereum Name Service) domain service, which allows entities to register an `eth` domain like `bitfalls.eth` so people can send ether straight to it rather than to a long and cryptic address like `0xbE2B28F870336B4eAA0aCc73cE02757fcC428dC9`. The transaction describes itself as being sent to the ENS -






Registrar contract, which then transferred 0.94 ether to the address which initially formed the auction, and then the contract called the finalization function.

But how is this an *internal* transaction if it still needed to be initiated by a transaction from an external account? The original initiating TX was external, yes, but this particular transaction is just one in a chain of transactions that happened inside the blockchain, from contract to contract. As the contract automatically sent ether back due to being triggered by another contract (the ENS auction process), it was logged as an internal transaction because the transfer of ether was the result of logic that was built into the smart contract and was not sent by someone from the outside. Therefore:

The *Internal Transactions* tab lists all transactions *initiated* by internal accounts as a result of one or more preceding transactions.

Token Transfers

Finally, there's the *Token Transfers* tab.

Transactions	Internal Transactions	Token Transfers	Comments		
Latest 3 ERC20 Token Transfer Events					
TxHash	Age	From	To	Value	Token
0xad10034a00e758...	8 days 8 hrs ago	0x00000000000000...	 0xad2281873339b...	1,500	ERC20 (BOBx)
0xa4cc4278d456803...	13 days 2 hrs ago	0xad38412a00e758...	 0xad2281873339b...	32	 Token
0xb1b0d4338d8d8d0...	160 days 25 hrs ago	0xb1b0d4338d8d8d0...	 0xad2281873339b...	5	 Loopring

Somewhat self-explanatory, token transfers are transactions that do only and exclusively that: transfer tokens. The token in question must be an ERC20 token (i.e. have standardized functions) and then it'll be listed here if the only purpose of the transaction is to transfer tokens. Here we can see an example of a spam-token: 1500 BOBx tokens were sent to the address 8 days ago. Token spam (airdrop) is a common method of promoting an ICO project. Airdrops make recipients wonder where the tokens came from. As they Google for the project, its Google SERP rating grows and backlinks are created. Considering it's almost free and very trivial to build your own token, the marketing costs for such a stunt are near zero. \$10 to be more precise, which becomes obvious if we look at the transaction more closely ...

Tx Hash: [0xed10034aa9e1539c24351da0cc37538bcb0fcd09724fa12f94c2f60340793](#)

Tx Receipt Status: **Success**

Block Height: [5557975](#) (48043 block confirmations)

Time Stamp: 8 days 8 hrs ago (May-05-2018 01:58:28 AM +UTC)

From: [0xd51272cc64bb489efc70f11a056ac718aca16b53e7](#)

To: Contract [0x7b9f6063114ba228c2a3fa77cda9becd939ac8](#) 

Token Transfer:

- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0x74a04532e17d6...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0xae283b6ad179a...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0x3fd74915ee83c57...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0xae617c244b7cf95...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0x8a210797a43cd6...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0x05081983515ed0...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0xd8e37e7a434e5d...
- 1,500 EPC20 (Free BOB Tokens - BobaRepair.com Token) from 0x000000000000... to → 0xa51dea98820ad8...

[illegible]

We can see that it was indeed an airdrop with a lot of addresses targeted as recipients.

Conclusion

You now know how to interpret the transaction types in Ethereum on Etherscan, and can with a reasonable degree of certainty say what was transferred to whom, when, and where from, as well as identify what triggered it. This will come in handy in your future explorations of the Ethereum blockchain.

Chapter 16: BigchainDB: Blockchain and Data Storage

BY CHRIS WARD

Since I wrote this post, a lot has changed for BigchainDB and in the blockchain space broadly, so it seemed high time for a revisit and refresh of what effect blockchain can have on one of the more fundamental parts of the traditional computing space: data storage.

Originally built as a technology to replace the Bitcoin blockchain in the Ascribe digital artwork tracking project, BigchainDB expanded into a component of the abandoned IPDB and now as a storage layer for the bold Ocean protocol.

This change in use caused changes to the underpinnings and implementation of BigchainDB, as did the closure of RethinkDB, forcing the team to switch the storage engine to the stalwart MongoDB. The blockchain layer on top of the database that provides the transactional support that helps guarantee a database change has taken place and adds additional control and security remains, but has gained maturity, with BigchainDB set to hit 2.0 in 2018.

All these changes now mean that BigchainDB encourages you to use their public network instead of deploying your instances. This approach is somewhat counter to traditional distributed database practice, but is more in-fitting with the evolution of Blockchain-based projects over the past few years, helps BigchainDB monetize their platform (with an ICO or SaaS model) and is an interesting change. Time will tell if customers are comfortable with storing data in a public network, but with access tokens ensuring security and privacy, it's conceptually not too dissimilar from using a cloud-hosted database.

If you want, you can still install your own BigchainDB instances, but I feel the company will increasingly encourage you not to.

Whichever option you take, you can then use the official Python, JavaScript or community drivers. For example, with JavaScript, install the package with `npm install bigchaindb-driver`, create a connection and write and read assets to the database using appropriate keys for the writer and reader.

You can read full documentation for the driver here and the database here, but the example below creates an article asset for `author` and then assigns it to `assignee`:

```
const driver = require('bigchaindb-driver')
const author = new driver.Ed25519Keypair()
const assignee = new driver.Ed25519Keypair()

console.log('Author: ', author.publicKey)
console.log('Assignee: ', assignee.publicKey)
```

```

const assetdata = {
  'article': {
    'title': 'Blockchain DBs',
    'body': 'Article body',
  }
}

const txCreateAuthorSimple =
driver.Transaction.makeCreateTransaction(
  assetdata,
  [driver.Transaction.makeOutput(

driver.Transaction.makeEd25519Condition(author.publicKey))
  ],
  author.publicKey
)

const txCreateAuthorSimpleSigned =
driver.Transaction.signTransaction(txCreateAuthorSimple, author.privateKey)

let conn = new
driver.Connection('https://test.bigchaindb.com/api/v1/', {
  app_id: '<APP_ID>',
  app_key: '<APP_KEY>'
})

conn.postTransactionCommit(txCreateAuthorSimpleSigned)
  .then(retrievedTx =>
console.log('Transaction', retrievedTx.id,
'successfully posted.'))

  .then(() => {
    const txTransferAssignee =
driver.Transaction.makeTransferTransaction(
      [{tx: txCreateAuthorSimpleSigned,
output_index: 0}],

[driver.Transaction.makeOutput(driver.Transaction.
makeEd25519Condition(assignee.publicKey))],
      {price: '100 dollars'}
    )

    let txTransferAssigneeSigned =
driver.Transaction.signTransaction(txTransferAssignee, author.privateKey)

```

```

        console.log('Posting signed transaction:
', txTransferAssigneeSigned)

        return
conn.postTransactionCommit(txTransferAssigneeSigned)
    })
    .then(res => {
        console.log('Response from BDB server:',
res)
        return res.id
    })
    .then(tx => {
        console.log('Is Assignee the owner?',
tx['outputs'][0]['public_keys'][0] ==
assignee.publicKey)
        console.log('Was Author the previous
owner?', tx['inputs'][0]['owners_before'][0] ==
author.publicKey)
    })
    // Search for asset based on the serial number
of the bicycle
    .then(() => conn.searchAssets('Blockchain
DBs'))
    .then(assets => console.log('Found assets with
title:', assets))

```

Other Alternatives

BigchainDB is no longer the only blockchain-based database in town, depending on your definition, and what you want to accomplish.

FlureeDB wraps a graph-style database with a blockchain layer, which — when you consider the nature of Blockchain — makes some sense. Thanks to the Graph underpinnings, it integrates well with GraphQL and React. It's still in active development, and follows familiar funding models for databases, with a limited community version, and extra capacity, security and support for premium users. FlureeDB's

involvement with Blockchain technology seems to be using tokens as a replacement for money and some form of consensus mechanism. The project is not open source, so it's hard to tell what's under the hood.

From memory, [OrbitDB](#) has existed for about as long as [BigchainDB](#), but designed for simpler application needs. While it uses [IPFS](#) for storage (which some might claim as a database of sorts), it doesn't claim to be a "blockchain database", but rather a choice for decentralized apps.

[TiesDB](#) makes a lot of bold claims on its site, but with little detail about how it accomplishes them, and with a [sparse repository](#) and sparser documentation on how to run the database, it's hard to confirm if it delivers. There's a handful of whitepapers in [the repository readme](#) that you can sift through, but they still mostly cover theory rather than practice. Intriguingly, it also allows you to delete data, which, while a fundamental part of traditional databases, somewhat contradicts blockchain ideals. There's no wrong or right in this decision; some developers have to consider compromises to push blockchain technologies into the mainstream.

[Swarm](#) is an Ethereum component that's the default storage mechanism for distributed apps ([Dapps](#)). It doesn't offer such a seamless way to get started, but if you're already investigating Ethereum for its other components, then [read the documentation for more details](#).

[Filecoin](#) does something different. It offers a mechanism for tracking transactions between blocks of spare storage around

data centers and the Internet. It allows you to use traditional storage, but via a blockchain layer that lets users bid for space you offer and tracks their usage of it.

Both of these technologies are described in more detail in [this post](#).

Part of a Decentralized Future

Ignoring its Blockchain heritage for a moment, BigchainDB supplies functionality missing from current NoSQL and distributed databases. That fact alone may be a reason to try it and may provide a valid business/use case.

For the blockchain aficionados among you, BigchainDB and other alternatives also complete the puzzle for a complete decentralized application stack — with Ethereum for applications, IPFS as a file system and BigchainDB for data storage. The pieces are in place for a different way of developing, deploying and maintaining applications, leading to a fascinating future that I would love to hear your opinions on.