EPISODE 529

[INTRODUCTION]

[0:00:00.3] JM: Bitcoin is an immutable append-only blockchain ledger that reaches consensus through proof of work. The contents of the ledger are financial transactions; people sending and receiving Bitcoin currency to each other. Since Bitcoin, there have been other cryptocurrencies that have similar properties, like Ethereum and the IPFS file coin system.

Similar to Bitcoin, they use a decentralized proof of work-based system with a currency reward. The ledger being maintained in those systems is not purely financial. A currency is a necessary component to maintaining the blockchain's validity.

Over the next month, we will be exploring a variety of blockchain-based technologies. Some interviews will be high-level conversations that assume only a basic familiarity with cryptocurrencies. Some of them will be deeply technical and assume a strong understanding of Bitcoin and Ethereum.

Some episodes like today's will be aimed at the developer who is in the process of going down the rabbit hole. If you are finding yourself reading about Bitcoin and Ethereum a few hours a day, or maybe more, but you're still struggling to grasp the basics, this episode is for you. It's meant to be a complement to other introductory resources such as Mastering Bitcoin by Andreas Antonopoulos.

Cryptocurrency systems are revolutionary. They will unlock completely new applications in the very near future. If you're trying to understand these decentralized currencies and applications, the best place to start is with Bitcoin. Then you can get in to Ethereum and IPFS and Zcash and everything else. It really helps to get a fundamental understanding of some of these concepts from Bitcoin.

That's why I was happy to have Daniel van Flymen back on the show. Daniel writes about Bitcoin and blockchains and he was previously on for one of our most popular episodes called

Blockchain Building, in which he talked about how useful it can be to build a blockchain-based system as an educational exercise.

Today, Daniel discusses the basics of Bitcoin transactions. What happens when you send money? How are transactions represented on the blockchain? How do full nodes and light clients interact with each other? These are difficult topics to understand and they're difficult to understand purely over audio. As I said, this episode is best listened to as a companion resource for someone who is already studying cryptocurrencies.

If you're looking for an internship, you can apply to the Software Engineering Daily internship program at softwaredaily.com/jobs. If you're looking to recruit engineers, you can also post jobs for your company there as well. It's completely free to post jobs and to apply. Also if you're looking for a job you can apply there.

We're hoping to find interns to contribute to the software daily open source project. If you want to see what we're building, you can go to softwaredaily.com. You can check out our apps in iOS or Android. They have all 650 of our episodes with recommendations and related links and discussions. If you find this episode too dense, there's certainly something in the Software Engineering Daily app for you.

With that, let's get on with this episode.

[SPONSOR MESSAGE]

[0:03:17.0] JM: LiveRamp is one of the fastest-growing companies in data connectivity in the Bay area. They're looking for senior level talent to join their team. LiveRamp helps the world's largest brands activate their data to improve customer interactions on any channel or device.

The infrastructure is at a tremendous scale. A 500 billion node identity graph generated from over a 1,000 data sources, running a 85 petabyte Hadoop cluster and application servers that process over 20 billion HTTP requests per day. The LiveRamp team thrives on mind-bending technical challenges. LiveRamp members value entrepreneurship, humility and constant personal growth.

If this sounds like a fit for you, check out softwareengineeringdaily.com/liveramp. That's softwareengineeringdaily.com/liveramp. Thanks to LiveRamp for being a sponsor of Software Engineering Daily.

[INTERVIEW]

[0:04:23.8] JM: Daniel van Flymen is a blockchain engineer. Daniel, welcome back to Software Engineering Daily.

[0:04:28.7] DVF: Thanks for having me, Jeff.

[0:04:30.5] JM: The last show that we did together was about building a blockchain and that was a very popular show. I think it was popular, because you were quite good at explaining some of the basic aspects of Bitcoin. I would like to do that again in this episode. I'd like to discuss Bitcoin transactions.

As you and I were talking about before the show, the stuff is hard to explain over audio. These episodes, the lower-level episodes are not for everyone. They're mostly to provide an audio resource for somebody who is really going deep into this material and really trying to understand it. I fully admit that this stuff will be hard to explain over audio and I think of it more as a complementary resource.

With that said, let's get into it. Let's talk about Bitcoin transactions. In Bitcoin, the blockchain is a ledger of blocks and those blocks contain transactions. Each of the transaction is a data structure, as well the block is also a data structure. What are the fields in that transaction data structure and then describe the block data structure.

[0:05:39.2] DVF: Right. Each block as you mentioned contains a bunch of transactions. Essentially, what a transaction looks like is a transaction has a bunch of inputs, a bunch of outputs, the counter the inputs, the counter the outputs and a version number, so that any client that's validating a transaction knows which set of rules to validate it against. Does that make sense?

[0:06:02.2] JM: Yeah, certainly. Is that all there is to it?

[0:06:05.0] DVF: Essentially yes, without going too deep into what an input and an output looks like. We can think of a transaction as inputs coming from other transactions and outputs, which essentially spend the funds in the transaction and maybe you sending "value to someone else." Those outputs would be consumed by transactions further down the line.

[0:06:29.3] JM: When transactions are spent, you're spending transactions, you can also spend fractions of a transactions. It's not that you're actually spending "Bitcoin." You're actually spending what's called UTXO, the unspent transaction outputs. Rather than my entire balance sitting in a wallet, which is what most people imagine, like you've got a wallet and your wallet contains all your Bitcoin, your balance is actually the sum of the UTXO; the sum of the transactions across the blockchain that you have the keys to.

Why is Bitcoin designed this way, where you have these transactions just strewn throughout the blockchain that are waiting to be spent, rather than having some centralized account balance?

[0:07:16.4] DVF: Okay. Let's talk quickly about UTXO. UTXOs stand for Unspent Transaction Outputs. As we just said, a transaction has a bunch of inputs and these inputs are actually references to earlier outputs. You can think of – coins is actually a bad word to use too. UTXOs actually send Satoshis. 1 Satoshi is the smallest unit in the Bitcoin system. 1 Satoshi is a 100 millionth of a Bitcoin.

Basically, if you're using the electron wallet, or if you're using Coinbase, or if you're using any client software it's the job of your software to keep track of these unspent transaction upwards. When you see something like you have – let's say that you have 5 Bitcoins in your wallet, what it actually means is 5 Bitcoins is the sum of all outputs that have been sent to your public address.

It's the job of your client to traverse the blockchain and see, "Oh, okay. Someone made a transaction that sends 1 Bitcoin to my public address. Another one that sent to, another one that

sent to. I have a sum total of 5 spendable Bitcoins, because I'm referencing transactions whose outputs were sent on my public address.

It's the job of my wallet software to keep track of unspent transaction outputs. What a confirmed transaction actually is is that enough blocks have been added to the ledger, such that I can consider my transaction to be in the authoritative blockchain. It's a matter of time and it's a matter of blocks being added for a transaction to be confirmed.

[0:08:57.0] JM: When I have a wallet client on my phone, or on my computer and that wallet client is calculating the balance that I have across the blockchain for a specific public private key payer, what is that wallet doing? What's the operation that the wallet does to traverse the blockchain?

[0:09:18.5] DVF: The way that different clients would implement this in different ways, but there's a remote procedure call in the Bitcoin protocol. I think it's called Get [inaudible 0:09:26.3] or Get [inaudible 0:09:27.8] Set Info. What they do is your wallet will speak to the peer-to-peer network as a whole and try get the transaction outputs for a certain block for a particular block.

Your wallet's only job basically is to display the sum total of those unspent outputs for you as a balance. They usually show a pessimistic balance, because user tends to only be interested in what they can spend at the moment. It would only show "confirmed transactions."

[0:09:58.1] JM: Did you say it only has to look at one block?

[0:10:00.7] DVF: No. You can either have – I mean, now we're going to get into light nodes or full nodes, but let's say that you are running a full blockchain node. In other words, you have a full copy of the entire blockchain on your local machine. What you would do, what your client would do is go through the whole blockchain and find all the unspent outputs that can be unlocked by your private key. In other words, it's looking for all outputs whose value has been sent to your public address, which is a hash of your public key.

[0:10:33.5] JM: Yes. As you said, within the output section of a transaction there is the script that displays the public key that that transaction is destined for. You can scan through all those transactions and be able to compare the public key of the wallet to the public key that the

transaction is – the UTXO of a specific transaction on a block is assigned to, and then you can just do a sum of all those different transactions, is that right?

[0:11:06.7] DVF: Correct. Yes, that's right.

[0:11:09.3] JM: The light client versus the full node, so light clients I believe those are types of Bitcoin nodes that only contain the block headers, is that right?

[0:11:20.9] DVF: Yes, that's correct.

[0:11:21.9] JM: Okay. What's in a block header?

[0:11:24.2] DVF: A block header is essentially the meta-data of the block. If we're thinking of the entire blockchain as a ledger, the headers of each block are going to describe everything about that particular block that a light client might need just to verify a transaction.

[0:11:41.3] JM: It won't be able to actually do a sum of all of my money across the blockchain in the light client, because I don't have the actual transaction data. What I can do is know if a full node is lying to me with my light client data.

[0:11:59.7] DVF: Okay. There's a bit of a gray area there and it depends on the clients again. Essentially, a light client should be able to tell you what your full balance is. Only difference is a light client might not know if an input is fake or not, because the main difference between light clients and full nodes is a light client doesn't implement a consensus mechanism. In other words, a light client doesn't follow the rules, so to speak, of Bitcoin.

They have no way of knowing if an input is fake. I believe in the Bitcoin docs, they call this economic strength of a full node. I encourage you, if you're listening to this to have a quick look at the Bitcoin docs, because they do a good job of describes what light clients can and can't do.

[0:12:40.8] JM: Do the light clients have access to the transaction data that are in the blocks?

[0:12:47.4] DVF: Yes.

[0:12:48.7] JM: Okay.

[0:12:49.5] DVF: As far as I understand, the light client will only have access to actually the Merkel tree that's built from all the transaction hashes in a block. The way that it does this is called SPVU, which is simplified Simplified Payment Verification. Its only job is to determine if the hash of a transaction is in the blockchain or not. Remember, it's not going to validate that the transaction and the rules governing the transaction are valid. It's just going to tell you if a transaction is in or not.
[0:13:19.0] JM: Does that mean that the light client itself can, or cannot calculate my account balance?

[0:13:26.4] DVF: It can calculate your account balance, but it can't tell you if that account balance is valid or not. This is a source of confusion and this is why in some way light nodes are dependent on full nodes, because a full node is going to enforce the consensus mechanism which forms the core of the network. If I'm a light client and someone sends me – let's just say that I'm listening for transactions coming in, I have no way of knowing if a transaction is fake or not.

[0:13:56.0] JM: I understand that. If I at any given time, if I have a light client and let's say I am partitioned from the network, let's say I spin up a light client and I ask my light client what is my account balance. Light client does have the necessary transactions on it to sum to my balance. It does have the transaction data.

[0:14:20.9] DVF: Yes.

[0:14:21.6] JM: All right. Got it. Let's say that the only UTXO that I have access to on the network is a single UTXO where someone originally paid me 20 Bitcoins. I want to purchase something from you for 1 Bitcoin. I just want to make a transaction that transfers 1 Bitcoin to you. What is my transaction going to look like?

[0:14:48.3] DVF: The one rule that we have to remember is that UTXOs, if you start an input to a transaction remember, references a previous output. If we're going to use a previous output in

a transaction, we cannot spend a fraction of it. We have to spend the whole thing. If we're referencing a previous output, in other words we're referencing somewhere where someone sent me 20 Bitcoins to my public address, then that 20 Bitcoins must be used in the transaction that we're dealing with now.

What I would do is I would reference your public address. I would send you 1 Bitcoin in the transaction and then I would create another public address that's owned by me to spend other 19. This is a very simplistic sense, because there is also the idea of a transaction fee, which is actually the change that is not spent in the transaction. Essentially, what I would do is send 1 Bitcoin to your public address. Let's say 18 to my public address and then there would be one left over, which the miner would then collect.

[0:15:55.3] JM: You mentioned creating and additional address. What's the purpose of that?

[0:16:00.2] DVF: Basically, an output is an address where coin should go. They're not wallets. They're just hashes of public keys that are generated by wallet software. They don't even have to be on the network. They could be generated locally if you want. Essentially, a new address should be created for each transaction. I'm sure if you've seen, if you've used Coinbase, or if you've used any public wallets, every time that you receive coins, you generate a new receive address.

[0:16:27.4] JM: Yes.

[0:16:28.1] DVF: You can create as many as you want essentially.

[0:16:30.8] JM: Yes. This is because most wallets are these hierarchical wallets these days, where you have – the wallets are designed such that you can create many, many different addresses within that wallet.

[0:16:47.5] DVF: Well the rule of thumb is that a new address should be created for each new transaction. This is both for privacy and security. Remember, your public address, in other words when you make a transaction and output as a public address, that public address is generally some sort of hash of your public key.

[0:17:04.3] JM: Yes. Those are different – you're going to have different public keys for each of these different transactions, is that what you're saying?

[0:17:12.0] DVF: No. I would keep the same public key.

[0:17:14.6] JM: Okay. Same public key, different addresses.

[0:17:17.2] DVF: Correct.

[0:17:18.1] JM: Okay.

[0:17:19.0] DVF: These are designed in a special way. They don't have ambiguous characters in them, like zeros and Os. They have checksums encoded in them to prevent someone from making a typo. In fact, I read somewhere that the chance of actually having a valid address that contains a typo is something like one in 4 billion.

[0:17:40.1] JM: Well, that's pretty rare.

[0:17:41.6] DVF: The other reason – I'm just saying the other reasons for using this public address.

[SPONSOR MESSAGE]

[0:17:55.3] JM: This episode of Software Engineering Daily is sponsored by Datadog. We know that monitoring could be a challenge. With so many services, apps and containers to track, it's harder than ever to understand application performance and to troubleshoot issues. Built by engineers for engineers, Datadog is a platform that's specifically designed to provide full stack observability for modern applications.

Datadog helps dev and ops teams easily across all their servers, containers, apps and services to monitor performance and make data-driven decisions. To get a free t-shirt and start your free trial of Datadog today, visit softwareengineeringdaily.com/datadog to get started.

[INTERVIEW CONTINUED]

[0:19:21.2] JM: Can you clarify a little bit more, what's the difference between an address and a public key?

[0:19:26.9] DVF: An address is created from your public key. An address forms the output of a transaction. A transaction can have many inputs and many outputs. If I want to send coins to different public addresses, I need your – I would ask you. I would say, "Hey, Jeff. Can you please give me your public address in order for me to send these Bitcoins to you?" You would say, "Okay, Dan. Let me generate one with my wallet software." You would generate an address and you would give me that.

That address is generated from your public key. When you would make another transaction further down the line, you would then prove to the network that you own that address by using a signature.

[0:20:07.1] JM: I see. If there are unique addresses for all of these different transactions that I'm interacting with across the blockchain, why do people say that Bitcoin is pseudonymous? Why do people say that if you're using – since it uses the same public key all the time, people can derive who you are. It seems like if you're using different addresses all the time, people would not be able to derive that.

[0:20:33.2] DVF: Well, yeah to an extent. Transactions are not encrypted. You can always see which address Bitcoins are being sent. If you happen to know those addresses are generated

by Dan, which would know if I in the future if I wanted to consume those unspent outputs in a new transaction, I would have to prove that I own all of them.

By clever aggregation, we can then see, "Okay, well in block 2,000 Dan referenced all of these transactions, referenced all of these outputs. Thus, we can assume that he owns all those Bitcoins."

[0:21:10.8] JM: When you reference all those transactions, you have to use your public key in order to do that?

[0:21:16.9] DVF: I have to sign the transaction proving that those outputs are now the inputs for the transaction that I'm creating. We can get into that if you want. I mean, that's what the script is all about.

[0:21:29.0] JM: Yeah, why don't you delve into that a little further.

[0:21:31.5] DVF: Right. Let's say that I'm making a transaction right. Each transaction in Bitcoin contains a script. Why do transactions have scripts? Well, a script is basically just a list of rules for unlocking, or being able to spend outputs in a transactions. It's what's needed for me to spend transferred coin – if someone sent me coins in the outputs of their transactions, I need to validate a list of rules included in a script to unlock those coins.

These rules can be fairly arbitrary, but typically the vendor must always provide a public key, which is hashed, which is actually the destination address in the script. They must also provide a signature to prove ownership of the private key corresponding to the public key. In other words, if someone has sent me coins and I want to unlock them, in other words, so let's imagine we have transaction A and transaction B. Transaction A has outputs that are being sent to me; let's just say to my address.

I need to prove that I can spend those coins. In the input to the transaction that I'm currently making, transaction B, I need to show that firstly I have ownership of those addresses. I'll provide a signature. Then if there are any other conditions in the script, I would have to prove those too. These are also sometimes called encumbrances.

[0:22:53.3] JM: Okay. Let's make this real, like in an example. If Daniel and Alice and Bob have all paid me 1 Bitcoin in the past and we've got some transactions on the blockchain where they have issued me 1 Bitcoin for different transactions, then I go to Charlie and I want to purchase a Lamborghini from Charlie for those 3 Bitcoins, what do I do? How do I aggregate those three UTXOs that are destined for addresses that I control and transfer those to Charlie?

[0:23:35.6] DVF: Right. Okay. Then what you would do, let's talk even simpler than that. I just want to do 1 Bitcoin to another person just that we can see exactly where the script fits in. We have Alice and Alice needs to pay Bob. What she does is she asks Bob for an address. Bob uses his client software and generates an address.
Let's say that Bob wants to receive payment to a script address, and we'll get into that for a second, but for whatever reason Bob is supplying a set of rules that would need to be satisfied in order to collect them out. Bob wants to receive payment to let's just say a pay to script hash address. What that means is Bob creates something called a redeem script and then uses that to create a redeem script hash, which is then encoded into an address. Are you following these so far?

[0:24:26.9] JM: I do follow so far.

[0:24:28.9] DVF: This is why it's probably a lot easier to look at a diagram. It's difficult to explain over voice.

[0:24:33.6] JM: Totally. Full disclosure, this is an experimental podcast. People find these too deep in the weeds, then we won't do this in the future. What I found is that people actually really do like these deep in the weeds type of episodes. Yeah, let's soldier on.

[0:24:50.0] DVF: Okay. All right, cool. Alice pays Bob, Alice wants to pay Bob, Bob supplies an address and the kind of address Bob supplies is very special. It's actually called a P2SH address. What Bob does is he creates something called a redeem script. He creates the script using a scripting language, which Bitcoin has a bunch for awkward, which we'll get into in a little while.

What he does is he creates a redeem script and that redeem script might be something super, super simple. Let's say that redeem script might be what number when added to four gives me five? My answer is one. What he does is he creates a hash out of the script and that hash is basically the address that he's supplying to Alice.

He gives Alice this address and Alice submits a transaction to the blockchain for 1 Bitcoin to this address. Let's say at some time later, Bob wants to send money to Jeff because the price of Bitcoin is a million dollars and Jeff wants to buy a Lambo, right?

[0:25:47.5] JM: Sounds good.

[0:25:48.4] DVF: Bob then creates a transaction with an input that references the redeem script he created earlier. What then happens is when Jeff wants to buy his Lambo, Jeff when he's sending money to the Lambo dealer, needs to prove that the answer to that puzzle that we supplied earlier is one. Does that make sense?

[0:26:08.8] JM: Got it. Yes, absolutely.

[0:26:11.1] DVF: These scripts, you could think of them as puzzles. They typically used for multi-sig wallets which I don't really want to get into because that's going to complicate things even more. You could think of these scripts as puzzles. They're a list of criteria that need to be satisfied in order for someone to prove that they can spend money that's being sent to them.

[0:26:30.3] JM: Okay. What about under the condition where there are multiple Bitcoins that I want to aggregate into a single transaction to somebody?

[0:26:40.1] DVF: You would simply reference those outputs; the inputs to your transaction. Remember, an input is a reference to a previous output and so what you would do, let's say that like a bunch of people is sending you money, each one of those outputs, remember these, you must reference them as – you can't spend money that you don't have. You must reference those outputs to the inputs of your current transaction.

Once you've referenced enough, let's say that you want to send someone 5 Bitcoins, you would have to reference at least – you'd have to reference outputs that provide you with 5 Bitcoins. Once you've referenced those, then you may submit a transaction to the network that contains those outputs.

[0:27:22.6] JM: Do I have to think about any of these when I am composing the transaction myself, or does my wallet client take care of assembling all of these information?

[0:27:33.3] DVF: No, your wallet client would keep track of exactly which inputs would be required in order for you to send let's say, Dan 5 Bitcoin. You can view them on the public blockchain in sites like blockchain.info. You can actually go click on a bock and observe all the transactions within a block and you would actually see for each transaction which inputs they are referencing – sorry, which outputs they're referencing.

[0:27:58.6] JM: In order for money to be originated on the network, there has to be an output that originates at some point. The original output that was created on the Bitcoin blockchain was the genesis block. The genesis block got mined and the resulting transfer for Bitcoin to the ownership of that original miner, that's called a Coinbase transaction. Explain what happens in the creation of a Coinbase transaction.

[0:28:31.4] DVF: A Coinbase transaction is actually called a generation transaction. It's always the first transaction in a block. When a miner mines a new block, the first transaction is a transaction that references no previous outputs. In other words, let's say that I'm mining Bitcoin and the current Bitcoin reward for mining a block is 12 and a half Bitcoins. Those 12 and a half Bitcoins must come to one of my public addresses; must come to a public address that's owned by the miner. This is what's actually called a Coinbase transaction.

The Coinbase is actually the free text area within this transaction. It can contain arbitrary data. I think famously, the Satoshi's first transaction – I think it said something like the chancellor's on a brink of a second bailout for banks back in like 2008. Yeah, in short the Coinbase transaction, first transaction in a block sends money to the miner of the block.

[0:29:28.5] JM: Right. Let's say if we're back in 2008 and Satoshi just mined that first block and he wants to send some of the UTXO from the transaction that was freshly mined to somebody else, what does he do? How does he send it to somebody else?

[0:29:47.1] DVF: Well, he wouldn't do it in that same block. He would then advertise a new transaction, submit it to the network and that transaction – the inputs from that transaction would reference the output of that first block that he mined.

[0:30:00.0] JM: Got it.

[0:30:00.7] DVF: Does that make sense?

[0:30:01.4] JM: Yes. Yes. Basically the same. It's the same as the previous example that we had.

[0:30:06.6] DVF: Yeah. I mean, this is oddly just curiously, this is how Satoshi could prove that he is Satoshi is by submitting that transaction to prove that he owned the private key that generated the output.

[0:30:17.1] JM: Say that again?

[0:30:18.6] DVF: Let's say that someone wanted to prove that they were Satoshi, the way that they would do that is possibly by submitting a transaction that referenced the output of that first block, because he would have to supply the private key for it when he signs the transaction.

[0:30:33.1] JM: Yes. Yes, yes, yes. Got it. If you transferred me some UTXO, is there a way to look back at the entire history of that UTXO and see all the way back to the generation transaction, or the Coinbase transaction that originally mined it?

[0:30:53.2] DVF: Sure. I mean, this is transaction validation so what a full node does. You can traverse the inputs from each transaction back until where they were created. This is basically what solves the double spending problem.

[0:31:04.5] JM: Okay. Can you now delve a little bit into the difference between full nodes and light clients?

[0:31:11.6] DVF: Sure. A Full node is a node participating in a peer-to-peer network called Bitcoin, that contains the entire blockchain. This full node usually has ports open on the network, can receive transactions, can listen to transactions. Miners necessarily are full nodes. Someone mining Bitcoin would have to have a copy of the full blockchain on a node, listening for transactions to include in the next block they're mining and communicate with other nodes. That's why the ports have to be opened in order for them to participate in this network.

A full node is considered a reference implementation for Bitcoin. A light node, you might want to run a light node if you maybe don't have enough disk space, I think that the total blockchain size is about a 160 gigs at the moment. Maybe you don't want to participate in all the network traffic. If you use a blockchain explorer, you'll see that they're about – I think they're about 8 to 9,000 reachable full nodes on the network. That might not suggest that they're more. I mean, the ports might just be blocked, or they might just be listening. Essentially, a full node just has the whole blockchain. That's it.

[0:32:21.1] JM: Let's revisit the mining process. If you're a full node, you have this mem pool. You don't just have access to the entire blockchain that has been accepted by all the nodes or the majority of the nodes. In the past, you have the set of pending transactions, which is sitting in memory. It's called the mem pool. At any given time, the full nodes are pulling transactions out of the mem pool. They're assembling those transactions into blocks and they're trying to solve the cryptographic puzzle associated with that set of transactions.

When they are trying to solve that puzzle, do they validate those transactions before they start trying to solve a puzzle, or do they wait until after they've solved the puzzle or try to validate them?

[0:33:13.8] DVF: Well, think of it from a miner's point of view. If you are mining, if you are trying to mine a block you would be spending a lot of energy doing so. Obviously, the worst thing that could happen is for that block to be invalidated by the rest of the network. You have now lost the

energy that you spent creating that block and you've lost the reward. It's in their best interest to validate all transactions before the block is mined.

[SPONSOR MESSAGE]

[0:33:43.1] JM: Software Engineering Daily is brought to you by ConsenSys. Do you think blockchain technology is only used for cryptocurrency? Think again. ConsenSys develops tools and infrastructure to enable a decentralized future built on Ethereum, the most advanced blockchain development platform.

ConsenSys has hundreds of web3 developers that are building decentralized applications, focusing on world-changing ideas like creating a system for self-sovereign identity, managing supply chains, developing a more efficient electricity provider and much more.

Listeners, why continue to build the internet of today when you can build the internet of the future on the blockchain? ConsenSys is actively hiring talented software developers to help build the decentralized web.
Learn more about consensus projects and open source jobs at consensys.net/sedaily. That's C-O-N-S-E-N-S-Y-S.net/sedaily. Consensys.net/sedaily. Thanks again, ConsenSys.

[INTERVIEW CONTINUED]

[0:34:59.8] JM: Indeed. What is the process by which if I'm a full node, I pull a transaction out of the mem pool, I'm trying to assemble a block. How do I validate one of these new transactions from the mem pool against the past blockchain?

[0:35:13.8] DVF: Okay. You would basically traverse the blockchain and make sure that the originating outputs to the input of that transaction are valid. There are various ways you can do that. You could determine if – Remember the UTXOs don't necessarily need to be tracked if they've completely spent. An output doesn't need to be tracked if it's spent. All we need to know is that let's say that I have a new transaction going into my block, all I need to know is that that transaction's inputs were valid. Does that make sense?

[0:35:45.9] JM: Well, maybe you could clarify that a little bit more.

[0:35:48.4] DVF: Right. Maybe it's not immediately clear, but the total mine Bitcoin ever is sits in UTXOs. Maybe it's not that clear, but –

[0:35:56.5] JM: Yeah. No, that makes sense. At any given time, the sum of all of the UTXOs on the blockchain that are waiting to be spent, that is equal to the number of Bitcoins that have been mined.

[0:36:08.2] DVF: Correct. Yes, that's true. Once an output has been spent, we don't really need to keep track of it anymore. It's been spent, it's done and so we would just need to for a new transaction make sure that the input to it was valid.

[0:36:23.1] JM: Although there will now be some other UTXO that you need to track, or some other set of UTXOs that you'll need to track who sum to the same value of that original UTXO that you were previously tracking.

[0:36:34.9] DVF: Correct. Given a new transaction, I don't need to validate the outputs. The output might be to an address that doesn't – that no one's claimed on the network. Come only validating the inputs. Let's say that you generate an address on an offline wallet, I submit a transaction to your address and I submitted in the current time period. Years could go by before you try to claim that UTXO. It's only when you try claim it that the miner or the network as a whole has to reach consensus about whether or not that transaction is valid.

[0:37:10.0] JM: Again, I've got – I'm a full node. I've got this mem pool of transactions. I pull one out and I want to validate it against the previous blockchain and make sure this is a valid transaction. What am I looking at and how am I traversing the blockchain? How expensive is it to traverse that blockchain?

[0:37:31.2] DVF: Okay. Let's just clarify something. Confirm transaction is a transaction that is in the blockchain. In other words, enough blocks have been added to the ledger for me to make a 99% guess and say, "Your transaction is valid. It's currently in block 5. We are currently mining

block 20," say that for example. An unconfirmed transaction is a transaction that is sitting in the mem pool.

What that means is that a miner has not yet selected it, or maybe is currently selecting it in the current block that they are mining. Given a new transaction, there are a number of reasons why I might choose to include it. Number one is the advertised fee of that transaction is there to incentivize me to include it in the current block that I'm mining.

Let's say that you've added all these inputs to a transaction and there is enough change in the transaction that's high enough to, let's say well give me 1 Bitcoin if I choose to mine it. Now I included it. Now I want to make sure that that transaction is actually a valid transaction, that it's not junk.

What I'm going to do is I'm going to look at the inputs for that transaction and I'm going to make sure that the signature of the transaction proves that I actually own the UTXOs that those inputs referenced. Does that make sense?

[0:38:48.0] JM: Yes.

[0:38:49.1] DVF: The input to my transaction is being validated by the miner and by the network and by everybody that's available to me. I'm looking at those inputs which referenced previous outputs and I'm proving to the network that I actually own that address. I'm doing that by means of a signature.

[0:39:05.0] JM: Okay. Perfect. Let's say this full node pulls in a bunch of transactions, it mines a block and it sends that block to the other full nodes and the block propagates and eventually gets confirmed and everything is hunky dory, let's compare that to the process by which a wallet client has to confirm a transaction as being real or unacceptable. When would a wallet client – I know we discussed this a little bit earlier, but when would a wallet client need to validate a transaction, or need to validate a block?

[0:39:46.2] DVF: Okay. Let's just go back one second. Something I should've mentioned is at any given point in time is UTXO set. In other words, the set of all unspent transaction outputs in

the network. You ask me if it's expensive to figure out if the inputs to a transaction are valid. The answer is it's not expensive, because all we would be doing is looking – is linearly searching a set to see if the input to my transaction is in that output; is in the output set.

Now given to your next question, if I have a wallet client, how does the wallet client validate new transactions on the network? Remember that when a miner block, one thing happens, Bitcoin proof of work trying to solve a puzzle that's very difficult to solve, but very fast to verify that the solution is correct. The way that I do that is I would just hash the block. If I hashed the block and a hash comes out correct, in other words it satisfies the difficulty requirements for the current block, then I can be sure that the block is correct.

Now if I wanted to go one step further and start validating everything in it, I could simply valid that all the transactions are correct by simply calculating the Merkel roots and showing that the Merkel roots satisfies the Merkel roots in the header of a block.

In terms of a wallet client, I mean most wallet clients are on full nodes, I would simply be looking – I'd be trusting at the peer-to-peer network is giving me valid block headers, because I'm participating in this network and then I'm just going to make sure that my transaction is in a certain block by verifying the Merkel roots of the block.

[0:41:22.1] JM: Yes. Merkel tree that is definitely something that is something best explained with a diagram, probably by a good teacher. It is absolutely something worth understanding for the sake of understanding Bitcoin. What can we say about a Merkel root that will make sense over audio or perhaps that will be helpful to somebody who has just looked at a Merkel tree. They're like trying to wrap their heads around it. Maybe you could give a few pieces of advice to understanding what a Merkel tree is and why that's important to Bitcoin.

[0:41:57.4] DVF: Okay. Let's say that I'm looking at a block and I have – for the sake of this example, let's say that I have four transactions and I'm choosing four to be an even number, because things are a bit easier if they're even. Basically, these four transactions would be, I would take transaction one, transaction two and I would calculate their hash. I would take transaction three and transaction four and I would calculate their hash.

Now I have two hashes from four transactions. I would then take these two hashes, call them hash A and hash B, and I would find a new hash. I would concatenate them and find the hash of that. That would form the root of the tree. Are you following so far?

[0:42:36.5] JM: I am.

[0:42:37.2] DVF: I have four transactions. Transaction one and transaction two get hashed to produced hash A. Transaction three and transaction four get hashed to produce hash B. I then find the hash of A and B. Now I'm finding a hash of a hash, does that make sense?

[0:42:52.0] JM: Yeah, yeah.

[0:42:53.1] DVF: Now, basically that is the Merkel root, the top – the roots of the tree, the top hash, whatever you want to call it. Because I have this root of the tree, if any single bit in any transaction further down the line, if a bit in transaction three were to change, then the hash of the entire tree would change. Does that make sense?

[0:43:13.0] JM: Absolutely. How does this pertain to light clients versus full nodes?

[0:43:18.7] DVF: I don't need to download the set of all transactions. All I need is the hashes of them. If I have the hashes of them, which are much, much smaller than containing all the transactions, then I can prove that the transactions that foresee that my transaction is within a certain block, because if it wasn't the hash would be different. I can prove that all those transactions haven't been a valid, that haven't been valids about what that – haven't been changed.

[0:43:46.3] JM: A light client that has the Merkel tree, they need the full Merkel tree in order to validate whether any full node is telling them the truth, right? Or any transaction that comes in is valid or not.

[0:44:02.1] DVF: Correct. You could think of it as like a Merkel tree is just a very quick way of determining if a transaction was included in a block. I mean, if you want to read more I strongly

encourage anyone listening to this to read about how the simplified payment verification works in Bitcoin. The stuff is as you said way better explained visually and verbally.

[0:44:25.8] JM: Definitely. Let's talk a little bit more about scripting. We explored so far essentially transactions where there are only two people involved. I am transferring somebody else money, or they're transferring money to me. There are other types of transactions. There is the multi-sig script and in a multi-sig script it's required that multiple signatures are provided to unlock funds.

For example, if I had three children and I want to issue a will that makes sure if I die, my three children can unlock my funds. If I don't trust any of my children individually, then I want to require two out of three of them to claim my money.

I can put my money on the blockchain. I can put it under a multi-sig transaction where it's a two of three multi-sig transaction, or two out of three of them have to sign for it. Is there anything notably different from processing a multi-sig transaction versus a normal single pay to public key hash script?

[0:45:36.3] DVF: Well, yeah. I mean, a pay to public key script, pay to public key hash could support multi-sig, sure. You've got to think of these scripts as they're just rules that if anyone can prove that they satisfy these criteria, they unlock that amount. Usually, P2SH is used for multi-sig wallets, using the example that I gave earlier. You could think of a script as a smart contract in its simplest form.

[0:46:02.4] JM: Yeah. These scripts can be pretty complex. If they get very complex, they start to take up a lot of space and if they have to exist on every single full node, that can be expensive to the network as a whole because the full nodes have to maintain the entire blockchain. One solution to – if you wanted to create a very complex transaction, if you wanted to create something like a smart contract, you can use pay to script hash. Explain what pay to script hash is.

[0:46:35.1] DVF: Okay. Let's just recap here. The transaction has outputs and outputs are paid to a pub-key script. Anyone who can satisfy the conditions can satisfy the conditions so the

script can unlock their amount. Pay to script hash is easily identified, because the public address begins with a three.

They usually for multi-sig wallets and you could think of them like puzzles. Going back to our example earlier, let's say that Alice wants to pay Bob, she asks Bob for his address, he'll supply an address that will start for the three, so that your client will know that it's a P2SH. Bob wants to receive payment, so Bob supplies P2SH address and Bob creates what's called a redeem script and then uses that to create a redeem script hash.

The reason why he creates this hash is so that he doesn't have to store the entire script. He doesn't have to submit the entire script. Anyone satisfying those conditions would just have to produce the same hash. Are you following so far?

[0:47:31.4] JM: Yes, yes. I follow so far.

[0:47:32.5] DVF: Then this hash is basically encoded into the address itself. When Bob gives the address to Alice and Alice pays it, sometime later if Bob wants to spend those Bitcoins to Jeff, Bob would create a transaction with an input and a field in an input is a script sig, as we said earlier contains the redeem script that Bob created earlier.

[0:47:52.8] JM: Okay.

[0:47:53.4] DVF: In other words, Bob doesn't need to supply the whole script. He just needs to supply the script hash.

[0:47:58.7] JM: Yup. Got it. Makes sense. We've gone very deep into several different topics and hopefully it's been quite useful to people. I'd like to begin to wind down our conversation. Let's zoom out a little bit, what are you working on these days? What has changed since we last spoke?

[0:48:16.4] DVF: I'm just participating actively in the space. I'm super excited about – as I said, about healthcare. I think we spoke last time over a company called Blink Health, which is trying

to make the price of prescription medications affordable for people who can't afford them. At the same time, I'm interested in applications of the blockchain to healthcare.

One idea that resonated strongly with me was using blockchains to store EMR data, which a number of – there are a number of people doing it now. I've seen at least two or three ICOs for it on Medcoin, Healthcoin. It's strange to me that you as an individual don't own your healthcare history.

I've been working on different consensus mechanism just as a side project, just in the moment in time. If you remember the first time we spoke, I wrote an article on building, understanding blockchains by building one. This is culminated in a play toy network called Electron Network, which is going to be part two of that original article, where we actually build a full functioning blockchain in a very simple, easy to understand way.

[0:49:24.0] JM: Awesome. That's fantastic. That's a great development. Congratulations.

[0:49:27.0] DVF: Yeah, thank you. It's just to help people understand the stuff better. I strongly encourage you to check out some of Andreas's videos on YouTube of this transaction stuff. It's complicated, because he's got a knack for making it simple.

[0:49:41.9] JM: Yes, he sure does. Okay, Dan. Well, thanks again for coming back on the show. Hopefully people found this useful, and I'm sure the next show that we do will be higher level. Hopefully I will be better equipped – Part of this is training for me personally to get more equipped to have these kinds of conversations with people. I'm sure we'll do it again in the future.

[0:50:03.9] DVF: Thank you so much, Jeff. It was always great talking to you.

[END OF INTERVIEW]

[0:50:10.0] JM: Your enterprise produces lots of data, but you aren't capturing as much as you would like. You aren't storing it in the right place and you don't have the proper tools to run complex queries against your data.

MapR is a converged data platform that runs across any cloud. MapR provides storage, analytics and machine learning engines. Use the MapR operational database and event streams to capture your data. Use the MapR analytics and machine learning engines to analyze your data in batch, or interactively across any cloud, on premise, or at the edge.

MapR's technology is trusted by major industries like Audi, which uses MapR to accelerate deep learning in autonomous driving applications. MapR also powers Aadhaar, the world's largest biometric database, which adds 20 million biometrics per day.

To learn more about how MapR can solve problems for your enterprise, go to softwareengineeringdaily.com/mapr to find white papers, videos and e-books. MapR can leverage the high volumes of data produced within your company. Whether you're an oil company like Anadarko, or a major FinTech provider like Kabbage, who uses MapR to automate loan risk, and has 3 billion dollars of automated loans to date.

Go to softwareengineeringdaily.com/mapr to find out how MapR can help your business take full advantage of its data. Thanks to MapR for being a sponsor of Software Engineering Daily.

[END]