# EPISODE 552

[INTRODUCTION]

**[0:00:00.3] JM:** Zcash is a payment and consensus system that allows users to transfer money to each other with strong guarantees of privacy. Zcash implements the same core features of Bitcoin with the added functionality of shielded payments. Shielded payments are private and they are enabled by a novel cryptographic technique zk-SNARKS; zero knowledge, succinct, non-interactive argument of knowledge.

A zk-SNARK allows for the proof that a certain piece of information is valid without revealing any information other than the validity of that information itself. Before you listen to this episode, it might be useful to go back to our previous episode about Zcash with Nathan Wilcox, in which he gives an overview of the technology.

This episode is a deeper dive into how Zcash transactions work and why zk-SNARKS are important. I know that some people are fatigued by the degree of blockchain episodes we've had. There are not very many left, so if you really were appalled at the amount of blockchain episodes as some people who have written in acrimoniously have been, please fill out our listener survey. You can go to go softwareengineeringdaily.com/survey. Or if you're a fan of them, also fill out the survey.

For more feedback, you can also e-mail me jeff@softwaredaily.com. I really would love to hear from you and note any critiques you have of the show, or feedback, or suggestions, whatever you've got. You can always join our Slack at softwareengineeringdaily.com/slack. You can message me there or hang out with the community.

Also, if you're sick of cryptocurrencies, in the meantime you can check out our back catalog of episodes at softwaredaily.com, or by downloading our apps, which have all of our episodes including our greatest hits. This is a curated set of the most popular shows, and the apps will soon have podcast features like offline downloads and bookmarking that you probably expect from your podcast player.

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

[SPONSOR MESSAGE]

**[0:02:08.4] JM:** You’re a successful developer and you couldn’t have gotten to where you are without help in your education and career. Maybe you’re thinking about ways to give back in the community where you live. The TEALS Program is looking for engineers from across the country to volunteer to teach computer science in high schools. Work with a computer science teacher in the classroom to bring development concepts to life through teamwork and determination.

Pay your success forward by volunteering with high school students in your area by encouraging them on the computer science path. You can make a difference. If you’d like to learn more about the Microsoft TEALS Program, or submit your volunteer application, go to tealsk12.org/sedaily. That’s T-E-A-L-S-K-1-2.org/sedaily.

Thank you to the TEALS Program for being a sponsor of Software Engineering Daily.

[INTERVIEW]

**[0:03:13.8] JM:** Sean Bowe is an engineer with Zcash. Sean welcome to Software Engineering Daily.

**[0:03:17.9] SB:** Hello.

**[0:03:18.6] JM:** How did you get involved in the cryptocurrency space?

**[0:03:22.1] SB:** I have been a passive observer of cryptocurrency for quite some time. I got interested in Bitcoin back in 2010 or 2011 and found it fascinating, mostly on a technical level, not really a political level and followed the research and all the new projects in the space for a while before there is like an exponential explosion of new projects and I can’t keep track of everything.

At one point when the zero coin and zero cash protocols, which were a private variety of Bitcoin were published in the academic literature, I followed that pretty closely. That kind of technology especially zero cash relied on some really esoteric cryptography that I found interesting. I followed the project and eventually I accidentally landed a job working at Zcash company.

**[0:04:15.2] JM:** Have you always been a fan of esoteric cryptography?

**[0:04:18.3] SB:** I've always been a fan of cryptography. I originally thought esoteric cryptography was not the kind of crypto that you want to use in production software, but we actually really need this kind of esoteric cryptography to make privacy practical. Definitely all for it

**[0:04:33.9] JM:** I feel like this typical cryptographic concept that we're going to be discussing is important and it is simultaneously hard to explain. It's one of the these situations where it puts the podcaster in unsure territory. It's like, should you try to delve into the very difficult cryptographic concept or should you just skim over it? I think we'll do a little bit of both in this episode, but just to give people a hint at what's to come, can you explain that abstract cryptographic concept that is important to Zcash?

**[0:05:06.4] SB:** Yes. This mathematics and this crypto that Zcash is built on top of are called zk-SNARKS. They are basically a zero knowledge proving system, zero knowledge proofs that you know some information that is constrained in some way, but these proofs don't reveal what that information is to keep it completely private.

Zk-SNARKS are a zero knowledge proof that do this with proofs that are extremely small and extremely cheap to verify, even if the statement is extremely large in size. Zcash uses this to make our transactions private. We in a sense encrypt the transactions on our network and use zk-SNARKS to prove that the transactions are correct without revealing what's inside them, which them extremely good privacy guarantees.

**[0:05:59.2] JM:** This is something that is probably not only useful for money transfer, is that correct? The ability to prove that something is valid without proving anything else other than that things validity, that seems like a widely applicable concept in computer science.

**[0:06:21.7] SB:** Yeah. It's something that's really probably one of the few important and interesting things in crypto, especially going for the next decade or two, you're going to see a lot of things that are built on top of zero knowledge proofs. They might not be built on top of zk-SNARKS, because some better approving systems will likely appear that have better tradeoffs and security assumptions, but overall you're going to see zero knowledge proofs play a huge role in networking protocols and communication and all sorts of things.

**[0:06:49.4] JM:** Just to wet people's appetites, what are some applications that you could build with this zero knowledge proving facility?

**[0:06:57.6] SB:** Well obviously, you can build interesting protocols like Zcash, where you have transactions, but you can also build other kinds of protocols such as – I think they're useful for things like Filecoin, which is some kind of proof of storage system, where you prove that you actually have the data, or that you're storing the data. Zero knowledge proofs are useful for those protocols as well.

They're also useful for scalability. Any protocol that is a broadcast network, something like Ethereum for example, where everyone – every fully validating node on the network receives every transaction and validates it, it's a lot easier to imagine these protocols when zero knowledge proofs are in play, because the actual validation time can be pretty much removed and you could just verify a proof instead of verifying all these transactions much more efficient. Scalability is also a huge advantage of zero knowledge proofs.

Also, there is a lot of interesting academic work surrounding zero knowledge proofs, things like proving that you've applied some filters to a picture, but that the original picture and the filters are what you say they are. You didn't' modify the picture in any other way is a interesting type of application for zero knowledge proofs, kind of integrity checks and things like that.

**[0:08:15.3] JM:** Why is that different in something like a checksum?

**[0:08:18.2] SB:** A checksum has the ability to statistically identify errors and although, actually some zero knowledge proofing schemes do use checksums in a sense, but it's different because we can make much more general statements other than this information is correct, or this is what I intend to do descend you which is what a checksum does.

What a zero knowledge proof does is it allows us to prove any general statement as correct that I know the premium edge of a SHA-256 hash, that I know the solution to a Sudoku puzzle, things like these, that really generally applicable to all sorts of problems that other kinds of cryptographic primitives can't tackle.

**[0:08:58.6] JM:** You mentioned that this Filecoin example, so Filecoin this is a system for decentralized peer-to-peer storage essentially, the Filecoin IPFS system. You can imagine one node asking another node, "Hey, do you have this blob of storage?" Node B would want to be able to say, "Yes, I do have that storage. Here is the proof that I have that storage." You're saying that zero knowledge proofs could be useful for that.

There's another example where why wouldn't you just have the – I don't quite understand why you wouldn't have that node just show the hash of that – if you have the hash of that file, what's the difference between having that hash and having a proof that file is in possession of node B?

**[0:09:47.0] SB:** Yes, so I guess the node B can actually just compute the hash and then the delete the data so they don't have to do anything. Another alternative is that they could compute the hash with some kind of [inaudible 0:09:58.1] at the request of node A, and this would only be possible if node B actually had the file contents.

Of course, node A isn't really assumed to actually have the data so they can't verify the integrity of the hash. Zero knowledge proofs are pretty much the only way to solve that problem.

**[0:10:18.5] JM:** Right. Okay. Let's get into the use case of zero knowledge proofs that we're going to be exploring, which is Zcash. Zcash is a privacy coin. I think people know why a privacy coin would be useful. You don't want everybody to be able to trace all of your transactions, and Bitcoin is pseudonymous. It is possible to have some traceability to Bitcoin transactions. Explain why is that? Explain how identities of Bitcoin users and transaction histories can be discovered or identified?

**[0:10:54.6] SB:** In Bitcoin, the model for transactions consist of two components, an input and an output and you can have as many of each in a transaction as you want. The outputs are essentially contracts that allow you to spend some money. They have some money associated with them, and the inputs linked to previous outputs and satisfy those contracts are those scripts or those programs or those predicates or whatever you want to call them. That operation is effectively spending the money.

In this model, there is a directed acyclic graph of value moving through the network. When this graph can show a lot of information about people's financial history, who they're sending money to, where the money comes from, how much money they're transacting. People have devised a lot of different ways to hide this information.

In the Bitcoin network currently people create new contracts, basically new addresses, new payment addresses for receiving money every single time that they want to receive some money, so that the outputs have something different usually. This isn't enough. The graph still shows an enormous amount of information. The values are still transparent, which is a deliberate thing for Bitcoin, because they're concerned that if you hide the values behind some kind of cryptographic technique, then you would never be able to audit whether or not the – if the crypto is broken, you'll never be able to audit to make sure that money wasn't created – counterfeit money wasn't created.

That's something that is near and dear to the Bitcoin land. In other privacy-oriented systems such as Monero and Zcash, we make some tradeoffs and make some cryptographic assumptions in order to hide things like the value, and in order to obscure this graph a little bt more so that almost all the information related to a transaction can be hidden completely.

**[0:12:48.4] JM:** Is part of the reason you don't need that audit ability of Bitcoin because we have agreed that Bitcoin works and we do work like, "All right. This system works. We don't need to audit it all the time for counterfeiting, because we trust that counterfeiting is not taking place in the Bitcoin network." It was something like Zcash we could just say, "Here is a new experiment. We're raising the stakes a little bit."

**[0:13:12.2] SB:** Yeah, that's what it is. It's almost a philosophical thing. I think in any privacy-oriented cryptocurrency you're going to make some tradeoffs that say, "Okay, if there is a crypto bug, I hope that we can fix it." In Bitcoin's case, they don't want to make that or take that risk. In Zcash's case and Monero's case what our users get is increased privacy, significantly increased privacy. That's something that we need as a society.

**[0:13:38.9] JM:** In Zcash, there are normal transparent transactions and these are transactions that are just like they exist in Bitcoin, because Zcash is a fork of Bitcoin. It is a fork, right? It's just a direct fork of the code and augmentations.

**[0:13:53.5] SB:** Yeah. It is a code fork of Bitcoin.

**[0:13:56.0] JM:** Code fork. Right. I didn't specify this.

**[0:13:58.7] SB:** A lot of different kinds of forks and in fact, we've stopped using the word fork at the Zcash company, or at least hard fork and soft fork and these words, because they're very confusing for everyone that we'd speak to outside of our company. We stopped using the terms.

**[0:14:14.7] JM:** Right. In any case, there are these transparent transactions that are just like in Bitcoin and then there's also the private shielded transactions. The purpose of a shielded transaction is to allow payments with a shielded payment address. Describe how a unit of currency gets spent in Zcash.

**[0:14:34.8] SB:** The transactions that allow you to make transparent payments and payments to and from transparent transactions are the same transactions technically that allow you to make payments to and from shielded Z addresses. They're integrated into the same unit of a transaction. In Zcash, we have this component at the end of a transaction after all the transparent inputs and outputs. The Bitcoin style components of the transaction, after all that we have a vector of these things called join splits, which are units of shielded transfer where two inputs, two shielded inputs are spent and two new shielded outputs are created in a sense.

In the system, the shielded outputs aren't referenced by the inputs directly in the inputs of the join split. The join split proves that some shielded output was created in a previous transaction, but it doesn't identify which one it is. It could've been any of the previously created shielded outputs that are being spent by any given join split. Yeah, that's pretty much it.

**[0:15:40.6] JM:** Right. Well, let's take a step back. When I send you Sean some Zcash in a shielded transaction, I am sending a note to your shielded payment address. What are the components of a shielded payment address?

**[0:15:57.2] SB:** The shielded address technically contains a couple keys. One key is allowing you to send some information to the recipient that allows them to spend the money. The other key is a spend authorization key. It's the thing that allows the recipient to spend it exclusively. In a newer version of Zcash we're actually merging these two keys, but it's irrelevant.
The address is used to send a few components called R and row and the value and a memo to the recipient. These are a bunch of variables and components of the transaction that the recipient needs to k now.

One of the interesting ones is the memo. You can send your recipient an encrypted memo on the blockchain. The other components such as this R and this row are just randomness that are created to keep the note completely private from outside observers, and also allow you to in row's case it's used to ensure that you're not sending money to the recipient twice that can't be spent – can only be spent once, which is a weird hedge case.

**[0:17:03.7] JM:** Okay. After I send a note to your shielded payment address – by the way, I guess we should define that term note. What is meant by the term note?

**[0:17:14.3] SB:** A note is just basically a public key and a value. There's some other components also, but they're just there for cryptographic purposes to make it function. Essentially, you can think of a node as a key that's allowed to spend the money along with how much money is in the node.

[SPONSOR MESSAGE]

**[0:17:37.8] JM:** QCon.ai is a software conference for full-stack developers looking to uncover the real-world patterns, practices and use cases for applying artificial intelligence and machine learning in engineering.

Come to QCon.ai in San Francisco from April 9th to 11th, 2018 and see talks from companies like Instacart, Uber, Coinbase and Stripe. These companies have built and deployed state of the art machine learning models, and they've come to QCon to share their developments.

The keynote of QCon.ai is Matt Ranney, a Senior Staff Engineer at Uber ATG, which is the autonomous driving unit at Uber. He's an amazing speaker. He was on SE Daily in the past. If you want to preview for what he is like, then you can check out that episode that I did in conversation with him.

I've been to QCon three times myself and it's a fantastic conference. What I love about QCon is the high bar for quality, quality in terms of speakers, content and peer sharing, as well as the food and the general atmosphere. QCon is one of my favorite conferences. If you haven't been to a QCon before, make QCon.ai your first.

Register at qcon.ai and use promo code SE DAILY for $100 off your ticket. That's qcon.ai and you can use promo code SE DAILY for a $100 off. Thanks to QCon for being a sponsor of SE Daily. Check out Qcon.ai to see a fantastic cutting-edge conference.

[INTERVIEW CONTINUED]

**[0:19:24.1] JM:** Is that equivalent to a UTXO in Bitcoin?

**[0:19:28.1] SB:** Yeah, that's pretty much equivalent. Though UTXOs and Bitcoin allow you to predicate to spend on much more sophisticated conditions. In our case, it's just knowledge of a secret key.

**[0:19:39.9] JM:** Okay. Got it. No sophisticated scripting in Zcash quite yet.

**[0:19:43.6] SB:** Yeah.

**[0:19:44.2] JM:** After I send a note to your shielded payment address, you can spend the Zcash that is in that note, because you have the private key that is associated with that address. Describe how I would make a shielded payment from top-level, fully fledged explanation. How would I make a shielded payment in more detail?

**[0:20:08.8] SB:** Top level or bottom level. Do you want me to explore the cryptographic approach to constructing?

**[0:20:15.0] JM:** Yeah, it's more of the cryptographic approach.

**[0:20:18.0] SB:** Okay. The original note that was placed – that was created that was sent to you was placed inside of a Merkel Tree. There is this big Merkel Tree that every node in the network maintains and adds new notes to every time a new shielded note is created. They actually add a commitment to the note is it's kept private. It's a cryptographic commitment that binds to the contents of the note.

This cryptographic commitment contains some randomness, so the sender had to have sent you some random information in an encrypted memo that allows you to use that to open the commitment. You're not going to open the commitment publicly, you're going to do it with a zero knowledge proof. You're not going to identify where in the Merkel Tree your note is, you're going to do with a zero knowledge proof as well.

You'll start by creating a zk-SNARK proof that proves that you know a path in that Merkel Tree to a commitment, that you know the randomness and you know how to open that commitment, and that you know the spending key associated with that note, all in zero knowledge. With the zk-SNARK proof, you place it inside your transaction along with whatever else you need to do to send your payment to the next person and submit that to the network.

**[0:21:36.8] JM:** You mentioned this abstraction of a note commitment. What is the purpose of the note commitment?

**[0:21:43.5] SB:** If the note was published, it would be the same almost as a UTXO. Everything would be public. A note commitment is basically and just think of it as a hash of the note along with some randomness, which prevents someone from finding out what value, or what key is inside without knowing that randomness as well.

**[0:22:01.4] JM:** Sorry, then is the note and the note commitment, are these abstractions that are attached to the same transaction?

**[0:22:09.4] SB:** The note commitment is the only thing that's public in a transaction. The note itself is an abstraction that we can think of internally when we're trying to discuss how values transferred around and how things are spent and stored in the system, but as far as outside of the transaction, or the transaction itself we're only really interested in the note commitment.

**[0:22:32.0] JM:** The note commitment says that this note has been committed to Sean and that's a field that commits the note to you being the owner. How do you interact with that note commitment in order to spend the note that is associated with that note commitment?

**[0:22:49.6] SB:** With your zk-SNARK proof, you reconstruct the commitment. In zk-SNARKS, we can prove that we perform the computation without revealing the inputs. In this case, you have a note commitment, which is just a hash digest. You prove that you know how to open it by constructing the hash and producing that hash, so you perform a computation over your key and your value and the randomness and if you get the same commitment, then your zk-SNARK proof will be valid. If it isn't the same commitment, then your zk-SNARK proof will not be valid. It will be rejected by the network.

**[0:23:26.2] JM:** You mentioned that you're doing calculation over the – your private key, and that's equivalent to you basically signing. You have ownership over that note commitment/note and you can spend it, so that – so you're signing it and you said there was some randomness associated with that, spending process as well. What is the randomness derived from?

**[0:23:47.3] SB:** The randomness is just used to hide the value of the note and the key that's associated with it, because the commitment is published publicly. If you didn't use some randomness in a commitment scheme then people would be able to figure out what the contents of the note are.

**[0:24:04.4] JM:** Okay. Each note also has a nullifier. What does the nullifier do?

**[0:24:11.2] SB:** The nullifier prevents you from spending the note twice. It is a deterministic output of some computation over the note that is unique to the note. If you attempt to spend twice, you're going to produce the same nullifier. Two notes can't have the same contents, so which we ensure in a different mechanism.

You can think of the nullifier as a hash of the note along with some key, the secret key that you know. Obviously if you just hash the note and produce the nullifier then people could see when you spent it. If you hashed some secret information inside the note and produce that as the nullifier then the person that sent you the money can see when you spent it. If you hashed one of the contents of the note, that was called row, which is unique to every note. If you hash that along with the secret key, then nobody but you can see when the note was spent.

Yet, the nullifier is unique and deterministic, so all the nodes in the network when they see a nullifier they can write it down in a big list and then check every time they get a transaction to see if they've seen the nullifier before. If they have, then they know the note is being spent again, which is not allowed.

**[0:25:26.4] JM:** Okay. Just to rehash that, what is the relationship between the note commitment and the nullifier?

**[0:25:32.8] SB:** The nullifier uniquely identifies the operation of spending the note commitment, or spending the note that's associated with a note commitment. Notes essentially are like adding a record to a ledger and a nullifier essentially takes the record away forever.

**[0:25:52.0] JM:** If the nullifier and the note commitment are there together, can an outside observer derive which notes have been spent from looking at the note commitments and the nullifiers that are around the blockchain?

**[0:26:07.4] SB:** The nullifiers are only revealed when you're spending, and the note commitments are only revealed when you're sending money to someone. A nullifier and its associated note commitment never appear next to each other. Otherwise, this would identify the note commitment directly.

When we're spending money, we're proving that we know a note commitment that has such and such nullifier that's in the tree. We have this note commitment in the tree that all the nodes maintain and we don't reveal which note commitment it is. Whereas, in the output case, we're just creating a new note and we're using the zero knowledge proof to prove that it contains the correct value that we haven't just created like a billion dollar note and are trying to counterfeit money and that it has the correct structure that the row is unique, which is guaranteed with a mechanism that's complicated and so on.

**[0:27:00.9] JM:** Just like in Bitcoin, a transaction in Zcash has inputs and outputs and I believe it does have scripts, right? I guess, the scripting language is just more limited. Unlike Bitcoin, a Zcash transaction can also have one or more of these join split descriptions. You mentioned that a little bit earlier. What is a join split description?

**[0:27:26.9] SB:** A join split is the act of performing a shielded transaction. It's going to have two nullifiers associated with two notes being spent and two note commitments associated with new notes being created. The join split does some other things as well. It allows money to flow in from the outside and flow out from inside of this shielded universe. We call it a value pool.

There is also some other information that's placed alongside of a join split. For example, an encrypted memo to the recipient of each note and some ephemeral Diffie Hellman keys for sending that information to the recipients. Also as a signature that allows you to authorize the transaction completely.

**[0:28:13.6] JM:** Can you give an example of what the join split description is doing in a transaction in a little more detail?

**[0:28:20.5] SB:** It just contains the proof, which uses a zk-SNARK proof in order to demonstrate that the transaction is valid, that he nullifiers are valid, that the commitments are valid and so on. It contains some other information and all of the nodes on the network use this to verify the proof. They also place these nullifiers in their list to make sure that they're unique and they place these commitments in the tree.

There's nothing else very special about the join split description. One thing that's important about it is that it's indistinguishable from any other join split description for the most part. Or reattempt to do that as best as we can.

The nullifiers and the commitments and the proofs don't reveal anything about what the transaction is doing at all. You can try to compare it with another join split description. It will look basically exactly the same. This is a property called ledger and distinguishability. It's at least what the zero cash authors call it. That's an important privacy feature of the system.

**[0:29:24.0] JM:** It maybe sound like a really stupid question, but why is it called a join split transaction?

**[0:29:31.9] SB:** Because we couldn't really come up with a better name, I guess. Join splits join two inputs together and then split it out into outputs. We had no idea what to call it. The zero cash paper actually calls it a pore, and we weren't really sure that that was the best word to use. We've changed a lot of the terminology if you look at the zero cash paper and compare it to what Zcash deployed.

We've changed things like coin to note, and nullifier was originally called serial number. Think of it as a serial number on a banknote. We've changed some of the terminology, but yeah it's an arbitrary choice of terminology.

In the new system that we're building to replace the existing crypto that we use in Zcash for building something that's a lot faster, in this system the inputs and outputs are in Bitcoin there in separate vectors. There isn't such a thing as a join split in this new system, which is called sapling. They're just independent.

**[0:30:30.8] JM:** Okay. When a note is spent and we'll talk about sapling a little bit later in the podcast, but just to stick to the basics for a little while longer, when a note is spent the spender is only proving that there is some note commitment in existence that maps to that note, but the spender is not revealing which note commitment maps to that note. That means that the note cannot be linked to the transaction that created that note, is that right?

**[0:31:04.6] SB:** Right.

**[0:31:05.2] JM:** Why is that important?

**[0:31:07.2] SB:** In Bitcoin, your inputs directly identify the previous outputs and this means that someone who sent you some money can see when you spend it, which is itself all it takes really to completely destroy your privacy and some certain adversarial scenarios. In other systems such as Monero, you identify multiple previous outputs and say, “I’m spending from one of these.”

Unfortunately, this isn’t good enough if the number of previous outputs that you’re identifying is small, because you can make multiple payments and then statistically if the two payments will be close together in this transaction graph, so you can identify that they were from the same person likely.

In Zcash, our system ensures that you can’t tell which note is being spent, the so-called anonymity set of your transaction, which is the set of all of the previous participants that you’re blinding yourself amongst is every previous participant, which is important, extremely important for security and privacy.

**[0:32:18.8] JM:** Right. There is also this term, the note traceability set, which is the set of previous notes that could link a note input to a transaction. If I am a controlling government and I want to follow your trail of payments in a cryptocurrency, how would I want to use the note traceability set?

**[0:32:39.6] SB:** The note traceability set is basically the same as the anonymity set. It’s just two different words for the same thing.

**[0:32:45.1] JM:** Okay.

**[0:32:47.7] SB:** It depends on whether or not you like the word anonymous in this case.

**[0:32:51.3] JM:** Okay.

**[0:32:52.5] SB:** If your note traceability set is small, then when you're in some interactive scenario with some kind of adversary who's sending money and receiving money from you, the payments that is adversary is sending will appear close together in this graph. You can form a graph based on connecting the inputs and outputs of transactions together.

In systems like Monero, that graph becomes a little blurry, but it's still not good enough, because statistics, because you can just link to transactions that are likely from the same person because they wouldn't – with high probability they wouldn't be interacting with outputs and inputs that are associated with other transactions that you're interacting with with that person, so the two transactions are close together in this graph, we would say, that would immediately distinguish someone.

I don't actually feel comfortable talking about the kinds of attacks you can deploy with this, because of ethical reasons, but there is some serious ways that you can – that people don't really understand that you can actually abuse this blurry graph to the anonymized people that will hopefully be revealed and mitigated in practice in software. We try to avoid this entirely in Zcash by making sure that the graph is completely okay.

[SPONSOR MESSAGE]

**[0:34:18.9] 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 done 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.

[INTERVIEW CONTINUED]

**[0:36:01.2] JM:** If people want to lose their confidence in weak anonymization systems, all you have to do is look that Netflix price fiasco, where Netflix revealed a bunch of data about supposedly anonymized data about movies that people had watched, and it was – there was some vector to deanonymizing it. I remember that example as one of like, okay actually if you're not thorough about anonymizing a data set, it is probably not anonymous.

**[0:36:36.1] SB:** That's right. Absolutely.

**[0:36:37.9] JM:** Okay. I want to move into zk-SNARKS and I'm just going to give people a heads up, like if they already don't have this sense – there are some episodes of this podcast that it's going to be really hard to really get a lot out of the podcast if you're just listening in the gym, or washing dishes and you have no frame of reference on the content.

I think that this content is particular useful to somebody that's already studying this stuff and they're just looking for some supportive complimentary material. That is the target audience for this kind of episode. I honestly don't know if maybe it is useful to people who are washing dishes and if in any case, you can send me an e-mail, I would love to know.

With that heads up, we can dive into zero knowledge proofs. Zero knowledge proofs are core to the functionality of Zcash and they can prove that a statement is true without conveying any information, other than the information is indeed true. They're called zero knowledge, because you don't – at least, I'm guessing they're called zero knowledge because you don't convey any other knowledge other than the proved statement. It is zero knowledge proven other than the proving statement. Describe a scenario in Zcash when I would need to prove that something is true, without conveying any other information.

**[0:38:03.1] SB:** One example of this is a digital signature has nothing to do with zk-SNARKS. A digital signature is a zero knowledge proof. You're proving that you own some key, or that you know some secret that's associated with some public key without revealing the secret key. It's a limited zero knowledge proof. The kind of zero knowledge that are really interesting are obviously these generic ones.

Then the generic ones, I like to think of it as us proving that we perform the computation without revealing all of the inputs to the computation. We can reveal some of them and we can choose to hide most or all of them if we would like to. This is essentially the same as saying I'm proving that a statement is true without revealing why it is true. It's basically just another way of looking at it.

In Zcash, we use zk-SNARKS to prove that transactions are correct. One of the components of this, or a very common component is proving that we know the pre-image to a SHA-256 hash for example. What we do with our zero knowledge proofs with our zk-SNARKS is that we perform the SHA-256 operation over the pre-image and produce a result and that proof, the proof that a company is this demonstrates to the verifier that yes, we do know pre-image, the SHA-256 hash. It's really cool.

**[0:39:27.7] JM:** By the way, that term pre-image, does that just mean whatever it is you're hashing before it gets hashed?

**[0:39:33.2] SB:** Right, right. The input into the hash function.

**[0:39:36.7] JM:** Okay. I want to try to play analogy out of my head and you could tell me if this is a really misguided way of thinking about things. Private public key cryptography, most people listening are going to know what that is. They may or may not remember how it works, so I took a class in college where we did the math to doing the private and public key cryptography.

I saw mathematically, okay this actually does work, and then I forgot everything about how it works, other than that it does work and you can sign things with your private key that can be publicly verified as to the owner of that key – of the private key having signed it. Over time, you get used to this abstraction. It becomes easier in your head to think about it. Like okay, I just need to sign for example to authenticate my computer to push to Github.

People are used to doing that. They don't really question it anymore. They're comfortable with how it works. They may not know how the math, or in fact they almost certainly don't know how the math works, but they're comfortable with the process. If I understand zk-SNARKS to some degree, so there is this parameter generation process, where you make a string at the beginning of the network creation and this is the parameter generation process. This is in some ways analogous to calling whatever that RSA thing is in your laptop to make your private and public keys.

**[0:41:08.8] SB:** Yeah, it's a key generation.

**[0:41:10.5] JM:** It's a key generation process, except instead of producing a private public key payer for a single user, it's producing a signing scheme for the network basically. The network is going to be able to authenticate things in the future. It's like instead of individuals signing things, you have the network signing things. Is that a reasonable way to think about it?

**[0:41:39.5] SB:** It's a reasonable way to think about zero knowledge proofs as a signature of knowledge. In some situations though that has separate cryptographic connotations than what zk-SNARKS may actually have. The way that I like to think of zk-SNARKS and this parameter generation is we take the statement that we're trying to prove and we construct a verifying key and a proving key using that, using that statement.

This verifying key allows you to verify proofs and the proving key allows you to construct new proofs that can be verified by the verifying key. These verifying keys and proving keys are specific to the statement. You can't just create a statement, or create some parameters and then use them for some different statement. You have to use them for the specific statement that you had originally intended.

This doesn't necessarily come inside with the network. In Zcash, it did because we launched the network, we constructed these parameters for it. For example, there are some forks of Zcash and clones of Zcash, they just use our parameters. They didn't need to perform their own ceremony, but perform their own parameter generations setup. They could just use out parameters. They could use their own and make their own as well if they wanted to.

**[0:42:56.0] JM:** To enable zero knowledge proofs between different parties, there is this setup phase. This is the parameter generation phase, where you need to create a common reference string that is shared between a proofer and a verifier. I guess, what I shouldn't have said is this idea of the network proving things, because all you really need is two people who – you just need a prover and an identifier and they need to agree upfront on this common reference string. Then once they have that common knowledge, that's all that they need to be able to have zero knowledge proofs between each other. Is that right?

**[0:43:33.8] SB:** That's mostly right. I guess, I would say that the setup phase allows you to construct this common reference string that you're referring to that can be used to create as many proofs as you would like. The interesting thing about zk-SNARKS is that they're non-interactive.

The prover can construct a proof and give it to the verifier, and the verifier can use this common reference string, or this verification key to verify the proof. They don't need to interact. The verifier doesn't need to do some challenge response protocol with the prover in order to see if the prover is honest.

The reason why this is important for Zcash if that the prover and the verifier can't have an interactive conversation. The prover is broadcasting a transaction and everyone on the network is passing that transaction around and verifying it. It's being placed in a blockchain forever, so that in the future you can join the network and verify all the proofs. If you had to interact with the prover every time, that would just never work. It has to be a non-interactive proof. That's what the common reference string accomplishes.

In zk-SNARKS, zk-SNARKS are interesting and unique in the sense that they require this common reference string to be constructed in a very delicate way. Otherwise, it would be possible to create false proofs. If you – essentially the common reference string, I like to think about it as a mathematical environment. When you're constructing a proof, you're proving that you know how to traverse this mathematical environment in a sense.

Unfortunately, if you know how the mathematical environment is constructed exactly, or in some particular ways, then you can subvert it and you can create false proofs, just with the knowledge of this hidden randomness that is used to construct the common reference string. It's something unique to zk-SNARKS. There are other proofing systems that don't have a common reference string at all, or that use different methods for finding one that is transparent and is created in a way that's secure.

**[0:45:44.7] JM:** Again, if I send you a shielded transaction, I have to construct a proof. What am I proving when I send a shielded transaction to you? What is it in my proof?

**[0:45:58.0] SB:** You're sending it to the network, but in your proof you're basically just proving this thing called the join split that we were discussing earlier. You're proving that you have some value in our network and you are authorized to spend it and that you haven't spent it before, and that you're creating some new note commitments and the value is all balanced up. That's basically what you're proving with your zk-SNARK.

**[0:46:24.2] JM:** In my transaction, I've got a join split description and that join split description also has a zk-SNARK proof. The zk-SNARK proof proves several aspects of the transaction. It proves the input and output values are balanced. It proves that the nullifiers and the note commitments are computed correctly. There's some other verifications that it proves. Again, if I am the creator of a transaction, I am the one who is creating the proof. Is that right?

**[0:46:57.6] SB:** Yeah.

**[0:46:58.8] JM:** Okay. Cool. Well, I want to step back a little bit from the technical weeds. Let's talk about the different actors in Zcash, specifically the miners. What are the miners doing in Zcash? Are they doing anything differently than in Bitcoin?

**[0:47:12.8] SB:** Not really. Zcash does use a different technique to prove that miners are actually mining. In Bitcoin, they just try to partially collide SHA-256 hash. In Zcash they're trying to find a – well, they're trying to do that as well as finding solutions to a memory-intensive problem. I'm not too much familiar with, it's based on the generalized birthday problem, but it's something called Equihash, and basically there is a steep memory time tradeoff in constructing, in solving one of these problems.

The goal with this difference between Bitcoin and Zcash is to make it so that miners have to have a lot of memory in order to construct, in order to mine blocks, mine valid blocks and this would presumably deter specialized hardware, the kind that you see in Bitcoin, these ASIC chips that allow you to mine really fast, would maybe deter that. Though I'm beginning to think it's not deterring that. I guess, we'll find out in the future.

**[0:48:22.9] JM:** Well, Ethereum did that right?

**[0:48:24.6] SB:** I think that they did some memory hard proof of work, which is basically what Equihash is. I'm not really familiar with it. Or some GPU-friendly proof of work at first. Other cryptocurrencies like Monero are always trying to ensure that they're so-called ASIC-resistant, so that the specialized chips don't appear on the market. That anyone can mine with their GPUs and nobody can buy a new GPU, because they're so expensive and nobody can play videogames anymore.

**[0:48:56.9] JM:** When you compare Zcash to something like Monero, you gave the comparison earlier where Monero is blurry, but it is only partial anonymization, what's the tradeoff there? Is there some advantage to Monero that it is able to get by trading off in that anonymization?

**[0:49:18.2] SB:** Yeah, by targeting a slightly weaker – well, it's not really slightly I would say. It's significantly weaker privacy by making that tradeoff there, allowing themselves to use a little bit simpler crypto. They don't have to use zk-SNARKS, and they don't have to make strong cryptographic assumptions, the kinds that we do in Zcash in order to protect the integrity of the system.

They just have to make some pretty reasonable old assumptions that are used in a lot of – that are relied on and things like Bitcoin for example. It's basically just avoid the complexity and risk of new crypto. Also, because the zk-SNARKS require this setup and there is a lot of philosophical disagreement on whether this setup is a good thing, or it could be managed, or achieved securely, or whether we should be building systems on top of these kinds of setups at all. There is a huge fight amongst cryptocurrency people over whether it's a good idea to do that.

The people that don't trust these setups, they'll be in the Monero camp and the people that don't mind the cryptographic assumptions, but want the stronger privacy, they'll be in the Zcash camp.

**[0:50:29.6] JM:** When you're saying these systems on top, are you talking about side chains that would enable anonymity?

**[0:50:36.8] SB:** With systems on top you're referring to?

**[0:50:38.8] JM:** Well, you said a few things related to systems on top, like people who didn't want systems built on top. Okay, are you saying that in Monero if people wanted absolutely anonymity they could use some system built on top of Monero?

**[0:50:52.4] SB:** No, I may have misspoke. I guess, you could do that. I think that they're considering something like that maybe, but I don't think it's good enough. I may have misspoke. I'm just referring to how people use Monero or use Zcash. There's just going to be people that disagree on cryptographic assumptions and things like that. That value, the integrity of the system more than the assumptions or vice versa or whatever.

**[0:51:17.6] JM:** Okay. As you mentioned earlier, the next generation protocol implementation is being worked on at Zcash. This is sapling, and you've worked on this. You've put a lot of work into this. Can you talk a little bit about what does that mean? What does that mean that you're building a new protocol for Zcash?

**[0:51:37.3] SB:** We're just trying to make a new shielded transaction format. One that's more efficient to construct transactions with. It has mainly the same privacy guarantees as the previous system. It's just more efficient. In order to do this, we had to change what elliptic curve we use and we had to change the zk-SNARK proofing system we're using, and we had to change the construction a little bit. We're not doing this join split thing anymore. We're splitting it out into inputs and outputs and reusing some more novel cryptographic primitives in order to make things more efficient.

What will happen in sapling is there will be a new address, a new Z address that unlike the old shielded addresses, the old Z addresses, the new ones will be much smaller, and they'll also be more efficient to send money to and from.

**[0:52:26.8] JM:** Maybe you could talk about the actual implementation of that. I talk to people all the time about building enterprise software companies for example, but I don't talk to people as much about the implementation of cryptocurrency. If you could just talk about the engineering, the language choices, I don't know, do you have continuous delivery and what do you do for testing and so on? What is the process of writing code for a cryptocurrency company?

**[0:52:57.0] SB:** For sapling, we've split out the development of sapling's crypto into few different pieces that layers on top of each other, that are trying to reach the Zcash protocol and implementation, which is implemented in C++. All these cryptographic primitives are being in these building blocks are being built in rust, so that we're much more confident in its security and safety and so forth. Yeah, we still get some good performance benefits out of it.

The lowest level primitive is the elliptic curve. Zk-SNARKS use a special elliptic curve called the pairing-friendly elliptic curve. I can go into detail what that does in a sense, but it's a special kind of curve. The curve that we're actually using is designed specifically to make zk-SNARKS more efficient and also more secure than the curve that we're previously using for zk-SNARKS. Also, the implementation is a little more efficient.

This building block is something – is in a library and a rust library called pairing. It has had some extensive testing and cryptographic audits. On top of this, we have a library called Bellman, which I've been actually working on for many years. This library is sort of using pairing, using this elliptic curve to create zero knowledge proofs, to create zk-SNARKS.

It will create them for you and allow you to encode your statements and your circuits and all these other things using Bellman's API. That's receiving some cryptographic review now. On top of that, we build the sapling protocol. The statements that you're trying to prove when you're spending notes, the statements that you're proving when you're creating new notes and some of the other things that surround that, so just the way the keys are created and how they're structured and how value is balanced between inputs and outputs, some of that crypto peers there.

As we get closer to the C++ of Zcash, it's going to start looking a little uglier and more C APIs, so that the C++ code can leverage it, but we're hoping to – I'm hoping to rip out as much of the C++ code and turn it into rust code in the future.

**[0:55:13.5] JM:** Now, I know we're up against time. Taking a step back even further and getting a framing of where we're at. I just have done a bunch of shows on Bitcoin and Ethereum and different people I talk to have different perspective for where we are in terms of adoption and I guess, the maturity of this technology. Are we waiting for some particular technical breakthrough, or is it more about adoption of users, or do we require some technology that allows people to use this currency more easily without being afraid they're going to lose their private key?

What do you think are the barriers to widespread adoption of cryptocurrencies, where you're paying for a cup – or are actually paying for a cup of coffee with Bitcoin or Zcash on a regular basis? Or maybe not. Maybe that's a bad example. Maybe an e-commerce purchase. Yeah, what do you think of the barriers to adoption?

**[0:56:12.6] SB:** In the cryptocurrency space, I think two of the major barriers are – or actually three. There's three. There is privacy. Systems that don't have strong privacy guarantees are not going to see as much adoption, because that's not good. Also, there is usability. Being able to use the system and understand how to use it and not misuse it and lose your money. That's probably one of the hardest aspects of cryptocurrency.

The last one is scalability. These systems work pretty well until they get really popular and then we have to hack our way back to something that actually functions correctly, or that can be used as a payment system, as what was envisioned originally for Bitcoin, some peer-to-peer cash system.

For example, innovations like the lightning network and all these other payment channel ideas that people are coming out with. Those are really exciting. I think they're the most exciting. Unfortunately, they come with some usability problems that I hope will be resolved. I think that's the area with the most promise.

Some other people think that it may be possible to do other things with cryptographic protocols, zero knowledge proofs especially to make these distributed consensus systems more private and also more scalable, because with zero knowledge proofs you can just prove that consensus is being reached correctly without having to sign a bunch of data to everyone, which is really, really awesome potential for that technology and that crypto.

**[0:57:47.0] JM:** Well, Sean thanks for coming o Software Engineering Daily. It's been really great talking to you.

**[0:57:49.8] SB:** It's been great talking to you.

[END OF INTERVIEW]

**[0:57:55.3] JM:** GoCD is a continuous delivery tool created by ThoughtWorks. It's open source and free to use and GoCD has all the features you need for continuous delivery. Model your deployment pipelines without installing any plugins. Use the value stream map to visualize your end-to-end workflow. If you use Kubernetes, GoCD is a natural fit to add continuous deliver to your project.

With GoCD running on Kubernetes, you define your build workflow and let GoCD provision and scale your infrastructure on the fly. GoCD agents use Kubernetes to scale as needed. Check out gocd.org/sedaily and learn about how you can get started. GoCD was built with the learnings of the ThoughtWorks engineering team, who have talked about building the product in previous episodes of Software Engineering Daily, and it's great to see the continued progress on GoCD with the new Kubernetes integrations.

You can check it out for yourself at gocd.org/sedaily. Thank you so much to ThoughtWorks for being a long-time sponsor of Software Engineering Daily. We're proud to have ThoughtWorks and GoCD as sponsors of the show.

[END]