What Satoshi Did Not Know

Gavin Andresen(✉)

Bitcoin Foundation, Washington, D.C., USA
gavinandresen@gmail.com

1 Introduction

When Bitcoin was invented six years ago (cf. [8]), Barack Obama had just been inaugurated president and Lady Gaga had just released her first big single. If you are 20 years old, that probably seems like forever ago. If you are 48 like me, that seems like not all that long ago. I first heard about Bitcoin in 2010, and was attracted to it because it combined economics, peer-to-peer networking and crypto in a really interesting way.

I'm going to talk about what we have learned over the last six years. Satoshi knew a lot, but he wasn't omniscient – I think there were a lot of things, both big and small, that he didn't know when he was inventing Bitcoin. I will finish by talking about some things that I think we still do not know.

2 What Satoshi Didn't Know

I think one of the things Satoshi did not know is *would it bootstrap?* Would anybody, besides geeks like me and him, be interested in this complicated piece of technology? Is there some way of creating value out of nothing? Because that's the thing that trips up most people: how can you bootstrap a currency from literally zero value? It had no worth for the first year of its life. I don't think Satoshi knew if this was possible or not. If you go back and look at some of his early writings, he was completely wrong about the ways it might bootstrap. He assumed it would be used as a spam filter device for email, something like a practical HashCash [1] system. But he was wrong about that. The way it bootstrapped was a guy buying a couple of pizzas for 10,000 bitcoins and people taking that leap of faith that it could actually be successful and might be worth a dollar or two at some point in the future.

I think something else Satoshi did not know is *was it legal?* When I first got involved in Bitcoin in 2010 we still did not know the answer and that actually worried me a lot. I was thinking: could I get arrested for working on this technology? Would I be thrown in jail? I was pretty sure I wouldn't – it was an open source project, I was not trying to rip anybody off and nobody was paying me any money. It didn't seem like there was a whole lot of room for authorities to

G. Andresen — I would like to thank Malte Möser for his help in preparing this transcript.

R. Böhme and T. Okamoto (Eds.): FC 2015, LNCS 8975, pp. 3–10, 2015.
DOI: 10.1007/978-3-662-47854-7_1

come down hard, arrest me and throw me in jail. But I am sure Satoshi thought about that and it's something he didn't know on the social side.

The situation is much better today. Last year, the SEC published an advisory about "Ponzi schemes using virtual currencies" [11]. And the important word in there is "using". They are not saying that virtual currencies are Ponzi schemes, they are just warning investors that if you see a too good to be true scheme using Bitcoin or Litecoin or Dogecoin or any of the other altcoins, then be careful. You are probably getting ripped off. That's a much better situation than we were in six years ago, where I think, if you asked the SEC, they probably would have classified Bitcoin as a Ponzi scheme.

3 Penny-Flooding

I think something else Satoshi didn't know was *how annoying people are on the Internet*. We all know this, but until you actually create a system and launch it and see all of the different ways people attack it, I don't think you really *know* it. There were some properties of the first Bitcoin releases that were subject to abuse and I think Satoshi didn't completely internalize how willing people would be to abuse this thing, even if there is absolutely no economic incentive for doing so. People will abuse your system just to be annoying if you are popular enough.

Early in Bitcoin's life we had a big problem with what we called "penny-flooding". Penny-flooding works as follows: I set up two machines and send bitcoins from one to the other all day long. If transactions are free, there is nothing stopping me from doing that. I can just flood the network with transactions that accomplish nothing, transferring them back and forth for ever and ever. Near the end of 2010 that was a pretty big problem on the Bitcoin network. Figure 1 shows the number of transactions on the Bitcoin network in 2009 and 2010. You can see this huge spike at the end of 2010, and there are a couple of other spikes which were earlier penny-flooding attacks. The vast majority of transactions were one or more annoying people doing this just because they could.

We actually found a solution to this which is now called proof-of-stake, although we didn't call it that back then. This is probably my most proud email from Satoshi, where I came up with the idea of using the age of a transaction and the size of the transaction in terms of bitcoin value to prioritize transactions and rate limit free transaction based on this scarce resource of old bitcoins. And Satoshi said "You may have finally solved one of the most challenging problems". There are a number of altcoins that have taken the same idea and use proof-of-stake for other things. We think that's a bad idea, and you can talk to Andrew Miller, Greg Maxwell, or Andrew Poelstra about why; they have looked pretty hard at proof-of-stake systems and thought about whether they can possibly work. But certainly, for supporting free transactions on the network, if we go back to that graph, it worked really well. The penny-flooding stopped as soon as we implemented this change and things got back on track.

The idea there is really pretty deep. It really is a core idea behind Bitcoin that scarcity plus utility equals value. If something is scarce and it is useful then it has

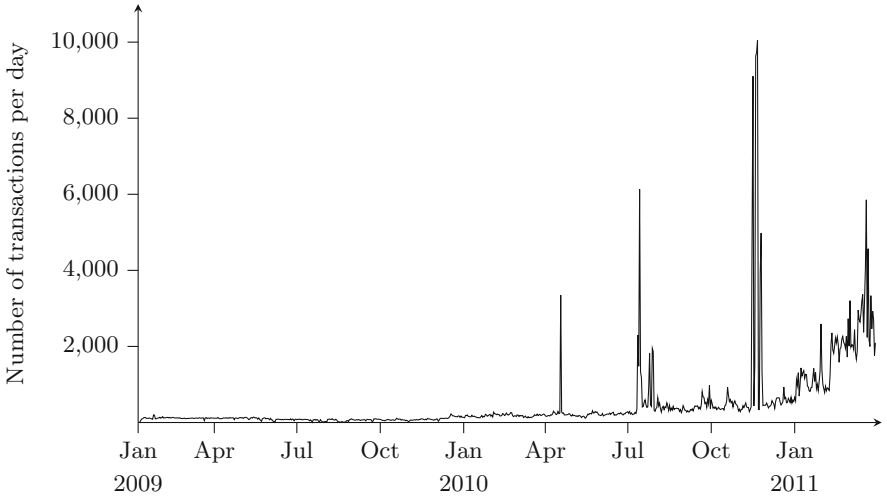

Fig. 1. Transaction volume in the early days of Bitcoin

value. In the Bitcoin system, bitcoins are the primary scarce resource: there are only a certain number that are ever created, there are only a certain number in circulation and the cryptography makes sure that those rules are followed. Transaction priority turns out to be another scarce resource that we can use in the Bitcoin system, and that's what we used to prevent penny-flooding. It is not obvious that this is a scarce resources, but if you want one bitcoin that is a year old and hasn't been spent, then that's a scarce resource, even if you don't spend it.

I think one of the more unexplored parts of the Bitcoin system is other uses of old unspent transaction outputs. There has been a lot of thinking that has started to happen, there hasn't been lots of code written, there hasn't been a lot of systems implemented, but some of the ideas might be leveraging them for further denial-of-service attack prevention. For example, you could imagine requiring some bitcoin deposit to merely join the network. Of course we don't want to do that because we want the network to grow. But you could imagine that if we had a really big problem with Sybil attacks [4], then requiring that somebody somehow gets some bitcoins that they control, as a kind of gate to join the network, might make sense. Another idea we have talked about for years is bitcoin deposits as pseudonymous identity anchors: if I hold a hundred bitcoins that are five years old and I can prove that I hold them, that might be a useful type of anchor for a persistent identity. And as long as I hold them I will maintain that identity. There are all sorts of issues there, but we've talked about this and I don't think anybody has actually tried to do it yet.

It is interesting to think about whether you could spend bitcoins as denial-of-service prevention for other systems that aren't Bitcoin-related. I would be very interested in seeing that. So far most people think about using bitcoins as

a token that you spend to get access to some other system, but you don't have to spend them, just having them might be good enough in a lot of cases.

I will note that spent transaction outputs are not a scarce resource, those are easy to create, it is easy to spend bitcoins over and over again. So if you are thinking about this, do think about spent versus unspent, and the edge case of when bitcoins go from unspent to spent which is where all the interesting things happen.

4 Was Satoshi a Cryptographer?

When I see popular press articles about Satoshi, he is often called a cryptographer. But is he really a cryptographer? I don't think so. There is actually very little cryptography in Bitcoin. We've got ECC keys, ECDSA signatures and hashing. And thats about it. All of Bitcoin is built from those primitives. If you actually look at the code, the way Satoshi used cryptography is fairly naive. He used OpenSSL, but he didn't use it very deeply. We are using OpenSSL deeply now, and there are all sorts of nasty little edge cases down there. I think if he were a cryptographer he would have been much more careful there. There is no mentioning of Schnorr signatures [10], Lamport signatures [7], none of the stuff I would expect a real cryptographer to mention in the source code. I don't see any comments like "I didn't use Lamport signatures here because they are too big". And there is a quote from Satoshi, that I ran across when I was preparing this talk, where he said "Crypto may offer a way to do 'key blinding'. I did some research and it was obscure, but there may be something there. 'group signatures' may be related" [9]. I think that shows you that with what Satoshi knew about cryptography, he probably wasn't a cryptographer, I don't think he was up on the very latest crypto research.

5 The Difference Between Transaction Validity and Meaning

I want to talk a little bit about something else I don't think Satoshi knew. I don't think Satoshi internalized the difference between validating a transaction and understanding what that transaction was about. And I think if he did, he would have designed Bitcoin differently. Transaction *validity* asks the question: should this transaction be allowed into the block chain? And the rules for this are pretty simple. First, a transaction has to be syntactically correct, you have to be able to parse it off the wire. Second, the redemption conditions have to be correct. Today, that typically means an ECDSA signature has to be valid for the previously generated public key. Third, the sum of the input amounts has to be greater than or equal to the sum of the output amounts. You are not allowed to create new bitcoins by creating transactions (there was an integer overflow bug early on that broke that rule – that was bad). Finally, it does not double-spend any inputs, so you do not spend any bitcoins twice. Those are basically the rules

for normal transactions. There are special conditions for coinbase transactions that create bitcoins, but I'm not going to talk about those here.

Transaction *meaning* is slightly different, and I think a lot of people don't understand this. In fact, I don't think I understood this and I don't think Satoshi understood this. Transaction meaning is who is being paid, how much are they being paid and who authorized the payment. Ideally, the meaning of the transaction would only be shared between the parties involved in the transaction and nobody else would have to know anything about it.

Satoshi's original wallet understood two transaction types. The first one is very simple: the person who holds the bitcoins creates a transaction that says "you need to check the signature of the public key" and the recipients, when they want to spend the bitcoins, provide a signature that satisfies those conditions. These pay-to-public-key transactions mix validation and meaning more than necessary; everybody observing the transaction knows the public key needed to redeem them.

Bitcoin transactions are expressed in a simple, Forth-like scripting language. The scriptPubKey in the funding transaction specifies how the funds are locked, and the scriptSig in the redeeming transaction satisfies the locking condition. In the simplest case, the scriptPubKey is just a public key and the OP_CHECKSIG operator, and the scriptSig is a digital signature for that private key.

Satoshi also implemented another transaction type which just moves the public key over to the redeeming side and replaces it with a hash of the public key. He did that to keep the resulting "bitcoin addresses" short enough to easily copy and paste. However, I think if Satoshi actually had understood OpenSSL a little bit better and knew that he could fit public keys into 33 bytes instead of 65 bytes, he wouldn't have bothered. He probably would have just had longer bitcoin addresses and we all would be using the first type of transactions. Almost all bitcoin transactions on the network these days are of the second type ("pay to public key hash").

So, we have this little language that the scriptSig and the scriptPubKey allow and there is an interesting question here: why did Satoshi allow opcodes in the scriptSig? The scriptSig is the part that you provide to prove that you know enough to satisfy the conditions of the transaction. So you could have a scriptPubKey that is `11 equals` and you (or anybody else smart enough to work out that eleven equals eleven) satisfy it with a scriptSig that is just `11`. You could also express that as `6 5 +`, but there is really no reason to do that because the person who is creating the scriptSig can always just run the script, get the results and put the data on the stack instead of putting the operations on the stack. I think Satoshi probably did it this way for legacy reasons, but I think if he had been thinking about the fact that you don't need to know the meaning of a transaction I think he would have designed it differently. I don't think he would have allowed operators in the signature.

If he had been thinking ahead, I think he would have realized that you do not need to provide the redemption conditions in advance. We have actually evolved Bitcoin with another transaction type called pay-to-script-hash, where

the entire redemption script is provided when the redemption happens and you just give a hash of the redemption script when you are funding the transaction. This is a step towards breaking the link between what does a transaction mean and is it valid. You can validate the funding transaction, but you don't know in advance how the locked funds will be redeemed.

The tricky bit, if you start thinking about breaking the meaning with validation, is what do you do with transaction fees. The system really wants to know what the fee is on a transaction so that it can prioritize it and reward the miner. Ideally, you'd have a system that removes the last two of the validation rules that I was talking about earlier. It would be nice if you didn't have to validate that the sum of the inputs is greater than equal to the sum of the outputs, if that was blinded somehow and you could just be assured that no bitcoins are created. It might also be nice to get rid of the condition that there are no double-spends. You could imagine a system where you just throw transactions into the block chain and as long as they are correct it's okay. If there are double spends then end-user's wallet software would throw red flags and say "somebody is trying to rip you off, they are double spending". But if you think that all the way through, that gets complicated. The system does need to know what part of a transaction is fees because those get assigned to the miner. And that is blurring meaning and validation because I am not paying a particular miner, I am paying any miner in the world who happens to mine my transaction. It exposes some knowledge about the transaction. And as we think about systems like Zerocash [2] or other systems that are trying to be completely anonymous, that's an interesting design point that I think people are going to be wrestling with for a long time.

6 Scaling

I don't think Satoshi knew how to *scale the system*. He has this quote "each transaction must be sent over the network twice". And that's just not true. There is this really neat current area of research of set reconciliation (cf. [5]). Basically, set reconciliation is a way for two machines to express differences in information they have using bandwidth proportional to the size of those differences. So, in Bitcoin's case, if I find a new block, I don't have to re-send all of the transactions that are part of that block because you probably already have 99.9 % of them from when they were originally broadcast across the network. I'm really excited about this, invertible bloom lookup tables [6] are now my current favorite data structure, and I am working with a research group at UMass on set reconciliation applied to the Bitcoin system.

7 Recent Cryptography Results

There is a number of cryptography results that have happened since 2009 that obviously Satoshi did not know about because they hadn't been invented yet. We didn't know about fully homomorphic encryption, and we didn't know any practical algorithms for non-interactive zero knowledge proofs or SNARKs (cf. [3]).

Six years is not very long, but if you look at the literature quite a lot has happened since then and we do know a lot more know than we knew back then. And I think there are lots of interesting things that will happen over the next six years, too.

8 What Gavin Doesn't Know

There are a bunch of things that I don't know right now. I don't know if the *identity* problem is going to be solved. I think a lot of what people talk about doing in Bitcoin relies on having a good, pseudonymous identity/reputation system. It seems to me that in a pseudonymous world there are completely unsolved problems. I haven't seen a good system that isn't subject to long-range attacks. And we have certainly seen those happening in the Bitcoin world. We have seen people with stellar reputations on Silk Road, apparently honest, stand-up people pushing drugs that are reliable for a while but eventually decide to steal everybody's money. I don't know if there is a solution to that problem, but it's an interesting problem.

Will there be a practical solution to the *privacy* problem? All of this neat crypto we have is fantastic, but I don't know if it will scale and I don't know if it will be low cost enough for people to actually to decide to use it. I think that's an open problem that will be very interesting to watch.

I think there is all sorts of work that can be done with probabilistic and approximating algorithms. There have been some ideas about using flaky but fast hardware to do Bitcoin mining. Because, as long as you check the work that comes out of Bitcoin mining and you just throw out the bad results, maybe its better and more power-efficient to have a really fast but at the same time really flaky piece of hardware that is running some approximations of the double-SHA algorithm. I think that's interesting and I wonder if there are other probabilistic algorithms that might be incredibly useful. An obvious one is certainly full nodes probabilistically checking transactions. If you can't afford to check every single transaction across the network then just check half of them, and as long as you do it in a random way everybody will be checking all of them several times anyway and it all works out. I have a feeling that there might be other problems where probabilistic algorithms could be really interesting.

Can streaming algorithms be applied to Bitcoin problems? I know that there is a fairly active area of computer science research on streaming algorithms that don't keep all 18 petabytes of data in memory but deals with data as it comes along. I think that might be an interesting area of research.

9 Conclusion

I have learned a lot in the last six years; experience is a great teacher. One challenge going forward will be to continue to learn by doing as the consequences of mistakes rise higher and higher. That challenge will be met by clever people like you, who are constantly thinking and researching and experimenting.

References

1. Adam Back. Hashcash A Denial of Service Counter-Measure (2002). http://www.hashcash.org/papers/hashcash.pdf. Accessed on 10 February 2015
2. Ben-Sasson, E., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: decentralized anonymous payments from bitcoin. In: Proceedings of the 2014 IEEE Symposium on Security and Privacy (SP), pp. 459–474. IEEE (2014)
3. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013)
4. Douceur, J.R.: The sybil attack. In: Druschel, P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp. 251–260. Springer, Heidelberg (2002)
5. Eppstein, D., Goodrich, M.T., Uyeda, F., Varghese, G.: What's the difference? efficient set reconciliation without prior context. In: Proceedings of the ACM SIGCOMM Conference, pp. 218–229. ACM, New York (2011)
6. Goodrich, M.T., Mitzenmacher, M.: Invertible bloom lookup tables. In: 49th Annual Allerton Conference on Communication, Control, and Computing, pp. 792–799. IEEE (2011)
7. Leslie, L.: Constructing digital signatures from a one-way function. Technical report SRI International Computer Science Laboratory, October 1979
8. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System (2008). https://bitcoin.org/bitcoin.pdf. Accessed on 10 February 2015
9. Nakamoto, S.: Re: Not a suggestion. https://bitcointalk.org/index.php?topic=770.msg9074#msg9074.2010. Accessed on 17 March 2015
10. Schnorr, C.-P.: Efficient signature generation by smart cards. J. Cryptol. **4**(3), 161–174 (1991)
11. U.S. Securities and Exchange Commission. Ponzi Schemes Using Virtual Currencies (2014). http://www.sec.gov/investor/alerts/ia_virtualcurrencies.pdf. Accessed on 28 February 2015