A Treatise on Altcoins

Andrew Poelstra

9 Jan 2014

1 Preamble

Why am I writing this document? Because when I first entered the world of cryptography, there were certain common-sense maxims which were passed around such as *to design a cryptosystem*, *you must first think like a cryptanalyst* or *anybody can make a cryptosystem which they themselves cannot break*. For most people, these maxims could be summarized simply as *don't roll your own crypto*. Anyone who flouted this golden rule, without decades of schooling and experience, was rightly dismissed as a crank or a troll.

Of course, there were a few people who didn't subscribe, and they would spend years repeating their half-baked ideas, conspiracy theories, factoring algorithms and NSA-proof cryptography on sci.crypt. These people were often ridiculed but just as often sincerely advised to seek mental help. I hope for their sake that some of them have since done so. At no point were their ideas taken seriously, used, or, god forbid, invested in.

However, shortly after the turn of the 21st century, Adam Back discovered a novel type of cryptography called *proof-of-work* which enabled a *distributed consensus* cryptosystem. This cryptosystem was used in 2009 by Satoshi Nakomoto to develop the first decentralized *cryptocurrency* — Bitcoin — which was also the first experimental cryptosystem to see billions of dollars poured into it by people who had no understanding of its mechanisms.

By that time, the benefits of doing cryptography in the open had long since been made clear, so Bitcoin's reference implementation was fully open-sourced. This allowed anybody to see the code, and anybody to fork it to develop their own cryptosystems. Of course, "developing your own cryptosystem" is the purview of only cranks and researchers, so it was reasonably assumed that none of these "altcoins", as they were called, could ever be plausibly presented for public use.

Boy, were we ever wrong on that one.

The purpose of this document is twofold:

 If you are a member of the public interested in cryptocurrencies, this document discusses what cryptocurrencies, and cryptosystems in general, are. It discusses the miracles and dangers of modern cryptography, and the serious risks associated with cryptosystem-tweaking by unqualified (and even qualified!) people.

Since Bitcoin has introduced direct monetary value to new cryptosystems, it is not only cranks doing stupid things with it, but also liars and thieves. This document also discusses that side

of altcoin development.

2. If you are, or are planning to, develop and release an "altcoin" to the public, this document reminds you that you are playing with fire. This sort of behavior was cute on sci.crypt, a community populated mainly by cryptographic experts where there was no risk that your charlatanism would be mistaken for anything legitimate, and where there was no ability to store value in your scheme anyway.

The Bitcoin community differs in both those respects. Your crankery is not cute. You are not a cryptographer, and yet are releasing a homebrew cryptosystem, misrepresenting your own qualifications, and encouraging others to store value in your creation. These actions are incompetent, dishonest and reprehensibly dangerous.

If somehow you are doing this through honest cluelessness, I dream that you'll read this article and realize the error of your ways.

2 What are cryptosystems?

Modern cryptography, as a field, studies the ability and techniques of controlling information flows independently of containing data flow. For example, using public-key cryptography it is possible to broadcast data such that the information contained is only accessible to a single person.

Until the advent of modern cryptography, philosophical questions, such as "where" the information actually is, were considered just that: philosophical questions. Intuitively, if you write some information down, it's right there on the page in front of you, available to anybody who can read it. In light of this intuition, it is something of a miracle that modern cryptography should be able to exist at all. And given that we evolved this intuition which has served us perfectly well until very recently, it should be expected that modern cryptography is an extremely subtle and perilous practice. Indeed, this is the case.

This cryptographic idea of "separating information flow from data flow" can be put on good mathematical footing, and much progress has been made in this direction ¹, though there are still many fundamental open problems². By reading papers in this field, one gets a sense for the difficulty of making concrete statements about such subtle concepts, and for the precision with which one's assumptions must be made.

A *cryptosystem* is a collection of algorithms which work together to achieve some cryptographic goal. A typical cryptosystem consists of three algorithms: key generation, encryption, and decryption. Cryptosystems are typically published alongside security proofs which reduce the problem of "breaking" the cryptosystem (e.g. learning some bits of the input to the encryption algorithm from its output) to some "hard" mathematical problem such as finding a discrete logarithm of an elliptic curve group element. These proofs are intricate, subtle, and their relation to reality is a subject of

¹See, for example, Shafi Goldwasser and Silvio Micali, Probabilistic Encryption, Special issue of Journal of Computer and Systems Sciences, Vol. 28, No. 2, pages 270-299, April 1984. Available online.

²For example, functions such as SHA256d which are easy to calculate but hard to invert are called *one-way functions*. However, SHA256d is merely assumed to be one-way, but no proof has been found — in fact, no proof has been found that *any* one-way functions exist!

intense controversy³. An important thing to note is that these proofs also consider the cryptosystem as a whole: change one algorithm even slightly, and the security proof of another algorithm could be completely invalidated.

Nonetheless, we interact daily with many cryptosystems, most of which we do not even think about. We assume the security of these systems partially because there is legal recourse if they are broken, partially because this is the way that we've always done things, and it seems like security breaches are reasonably uncommon, and partially because we assume that very smart people designed these systems to be hard to break. After all, there's a lot of money riding on them.

Of course, even if these things are true about the cryptosystems we use to check our email, do our banking, buy groceries and store private data, there is no reason to believe in them for novel cryptosystems designed by anonymous people to do unprecedented things. This is largely responsible for the "crank" label assigned to so many pointless projects on sci.crypt.

3 What are cryptocurrencies?

A *cryptocurrency* is a cryptosystem designed to facilitate the transfer of value, by transferring scarce goods defined within the system itself. The prototypical example is Bitcoin, which transfers signing authority and maintains a global ledger of value associated to this authority. The primary innovation of Bitcoin was the creation of this ledger, which is updated and verified in a completely decentralized fashion, with all parties agreeing on the atomicity of transactions and their ordering in time.

Out of necessity, cryptocurrencies are enormous cryptosystems and contain many smaller cryptosystems as components. This makes them fearsomely complex, and their security correspondingly difficult to verify, but the fact that Bitcoin has held up for over five years gives evidence that this complexity can be managed.

Adding to the complexity of the cryptosystem itself is the fact that exchanging value necessarily involves economic considerations. Therefore cryptocurrencies must be analyzed not only for computational soundness and security, but also for *economic* soundness and security. That is, is the cryptocurrency designed so that the incentives are aligned with the goal of security the system, and *not* with the goal of undermining it?

To illustrate the complexity of Bitcoin, and to give an overview of its workings (which we will cover in more detail in Sections 5 and 6), we have broken the cryptosystem down into its constituent algorithms. The cryptosystem in its entirety is run by every validating "full" node on the network. We assume the existence of a communications network by which nodes are able to exchange data. (In practice, Bitcoin nodes use a peer-to-peer network, and communicate by the "Bitcoin protocol".)

Such a breaking-down is necessarily subjective and gives an incomplete view of the system⁴, but is didactically necessary. It is important to emphasize that this is *one cryptosystem* and the security (economic and computational) of every component is tied to that of every other. Therefore, anyone hoping to change a single component must understand the entire system and have the technical background to analyse and implement the change.

³See Alexander Dent, Fundamental Problems in Provable Security and Cryptography, retrieved from the IACR preprint archive, 2006. http://eprint.iacr.org/2006/278.pdf

⁴Robert Pirsig, Zen and the Art of Motorcycle Maintenance, 1974

We now give an overview of each component of Bitcoin, leaving detailed cryptographic discussion to future sections.

Setup. When a Bitcoin node is first started it creates two data structures, a weighted hash tree called the *blockchain* and a database called the *utxo set*, both of which are initially empty. Elements of the blockchain are called *blocks*; elements of the utxo set are called *utxos* or *unsigned transaction outputs*. The motivation for these terms will become clear.

It then contacts another node to request the highest-weighted path in its blockchain. For each block in this path (which must start with the so-called *genesis block* whose hash is hardcoded into the node), the node runs its Block Verification algorithm, which updates its chainstate.

Relay. Each time a node sees a transaction on the network, it runs its Transaction Verification algorithm. If this passes, the node passes the transaction to each of its peers (after a small delay, to prevent flooding attacks).

Similarly, each time a node sees a block on the network, it runs its Block Verification algorithm. If this passes, and if the new block is part of the highest-weight blockchain path, the node passes the block to each of its peers.

Script Evaluation. Internal to Bitcoin is a stack-based scripting language. It has the capacity to push and pop data, branch on simple conditions, and also execute some cryptographic primitives.

(TODO: describe script in some detail. At least cover CHECKSIG.)

Transaction Verification. At its heart, a Bitcoin transaction is composed of two main parts: the inputs and the outputs. As the inputs refer to the outputs of other transactions, we cover them first.

Both inputs and outputs are constructed from *scripts*, which are instructions in a Bitcoin-specific stack-based programming language. This language is very small and does not support looping, so that it can be consistently implemented and easily audited.

Outputs are fairly simple: they consist of (a) a value, which is the number of Bitcoins the output represents, and (b) a script which reads values from the stack then either passes or fails. A typical script might expect the stack to contain a digital signature, for example. All that is needed to validate outputs is that their scripts use the defined script opcodes.

Inputs are more intricate: they consist of (a) a reference to an output of an existing transaction and (b) a script which places values on the stack. To validate an input, it is first checked that the referenced transaction output has not been spent, *i.e.* it appears in the utxo set. Then that output's script is concatenated to the input's script, and the concatenated script is run using the Script Evaluation algorithm. If the algorithm accepts, the transaction is valid.

Further, the total value of outputs must be greater than or equal to the total value of the inputs (input values are defined as the values of their referenced outputs). If the output total is strictly greater than the input total, the difference is called a *transaction fee* and is recaptured by the network.

There is one exception to this last rule for so-called *coinbase transactions*. These are special transactions which occur once in each block and may be created with no inputs at all. They are the

mechanism by which new Bitcoins are brought into circulation. The total output size must be less than or equal to the *block reward* plus the total network fees for all other transactions in the block.

Transaction Generation. To create a transaction, a node selects outputs from its utxo set which it has the capacity to spend (for example, outputs whose scripts expect a digital signature, and the node is in possession of the requisite key.) It chooses enough outputs so that their total value is greater than or equal to the amount desired to spend.

It then creates new outputs which the transaction's recipient has the capacity to spend (typically this requires contacting the recipient through another channel, *e.g.* to obtain the hash of a public key for which the recipient has the corresponding private key), and sets their values so that the total is equal to the amount desired to spend.

Any discrepancy between the total input value and total output value is considered as a network fee. To reduce this, the node may add an additional "change" output, which it has the capacity to spend itself.

Block Verification. To verify a block, the Bitcoin node first checks that it is formed correctly and that the correct hash of its contents is in its header. It also checks that the hash of the block is within a small range — the exact range is calculated by observing the timestamps of the block's ancestors and attempting to adjust so that future blocks will be created roughly every ten minutes. See the Block Generation algorithm for more details about this.

It then runs the Transaction Verification algorithm on every transaction in the block, and if any of them fail, the block is invalid.

It is crucial to the Bitcoin cryptosystem that all nodes agree on the result of the Block Verification algorithm — and by extension, that all nodes agree on Transaction Verification, Script Evaluation, and Difficulty Calculation. That is, these algorithms are *consensus algorithms*. More about this will be discussed in Section 6.

The block is weighted in the blockchain according to its difficulty.

Block Generation. Unlike the previous algorithms, Block Generation is not done by most Bitcoin nodes, since it is designed to be very computationally expensive. Today it requires special-purpose hardware to be feasible.

To create a block, a node assembles a list of transactions, which are obtained through the Bitcoin network and all pass the Transaction Verification algorithm. The node also creates a coinbase transaction, which has no inputs and whose outputs can be spent by the node itself.

These transactions are hashed up, and the resulting hash is put alongside a timestamp and nonce in the *block header*. In order that the new block pass the Block Verification algorithm, its hash must fall into a small range defined by the Difficulty Calculation algorithm. To accomplish this, the block is hashed *ad nauseum* for different values of the nonce until a hash is found which falls into the required range. This computation is called a *proof of work*, and Bitcoin's security depends on it satisfying several subtle mathematical properties, which will be discussed in Section 6. Difficulty Calculation. (TODO: discuss difficulty and this range calculation properly)

Basically, if blocks' timestamps are too close together then the difficulty increases, meaning that the range of valid hashes shrinks. Conversely if blocks are too far apart the range grows. The result is a negative feedback loop designed to cause blocks to appear every ten minutes on average.

4 Cryptography is hard.

[tell stories about serious cryptosystems broken in weird ways, eg TLS 1.0 side-channel attack]

5 Cryptography of Bitcoin 1: transactions and signatures.

[explain transactions and signatures, CHECKSIG etc] [explore dangerous/bad ideas]

- Turing completeness, script expressiveness
- Extrospection

6 Cryptography of Bitcoin 2: distributed consensus.

[explain distributed consensus works, risk of forks, incentive issues, etc] [explore dangerous/bad ideas]

- Dependence on architecture (type width, floating point)
- · Speed and clamping of difficulty changes
- Verification versus search speed for PoW
- PoW with different hardware requirements
- Block frequency and size
- Proof of stake
- Update management (dogecoin's fork, lack of broadcast keys, etc)

7 Where do I go from here?

I hope this document has provided some perspective on the intellectual magnitude of tackling cryptographic projects. Even experts shy away from developing new cryptosystems, preferring to use tried and true cryptographic primitives which have withstood the test of time and been analysed in depth by thousands of people. However, there are many open problems and exciting research directions in cryptography and the field is remarkably accessible to those willing to invest a few years into learning its history. As a start, several active and famous cryptographers maintains blogs devoted to presenting cryptographic ideas in accessible ways. Of particular interest are those of Matthew Green and Bruce Schneier. Also current academic research is typically posted to the preprint archive at eprint.iacr.org. It is helpful to skim the abstracts periodically, both to find interesting papers and to get an idea of current trends in cryptographic research.

For an historical account from ancient times through World War II, read David Kahn's tome *The Codebreakers*. This is a very long text, but an enjoyable and engaging read.

Regarding modern cryptography, many classic papers are available online. An incomplete and unordered list of essential reading is:

- Probabilistic Encryption, Goldwasser and Micali, 1984.
- A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms, ElGamal, 1985.
- The Decision Diffie-Hellman Problem, Boneh, 1998.
- *How to Prove Yourself: Practical Solutions to Identification and Signature Problems*, Fiat and Shamir, 1986.

It is also worthwhile to read the Wikipedia article on zero-knowledge proofs, which has plenty of citations, but is more clearly written than any of them \odot .

8 Conclusion.