A New MimbleWimble Blockchain:

Our Experience Creating EPIC Cash

Table of Contents

1 Introduction

EPIC (Epic Private Internet Cash) is a community effort to create a MimbleWimble-based blockchain. We think that MimbleWimble is important to the future of cryptocurrencies, and we wanted an implementation that can serve as a basis for spreading its use. Epic Cash's goals include a conservative monetary policy, financial support via the Epic Foundation for public education, blockchain research and community development and a commitment to using multiple cryptographic algorithms in our blockchain for proof-of-work.

This paper describes our goals and strategy, the technical details and our experiences with the MimbleWimble implementation, and future directions for development.

2 Goals and Strategy

2.1 The importance of multi-algorithm blockchains

Epic is a multi-algorithm blockchain. Each individual block shows proof-of-work in a single algorithm, but the algorithm changes from block to block.

Using multiple algorithms for a proof-of-work chain is an idea which has sometimes been treated skeptically by the blockchain community. We believe that conventional wisdom regarding this can be shortsighted and that many of the supposed shortcomings of multi-algo chains can actually be a source of strength.

In this section, we will examine three such objections to multiple algorithms and show that, in each case, with some thought and careful design, the conventional wisdom can be turned completely on its head.

2.1.1 An overview of Epic's multi-algorithm implementation

In this section we will cover how Epic features a flexible policy framework for configuring the set of ciphers to be used for proof-of-work.

Our initial implementation uses our Feijoada protocol: the blockchain policy allocates to each cipher some whole number of blocks, and these portions sum up to 100. Analogy: at the start of a set of 100 blocks, bottles are set up for each cipher and beans are put into the bottles according to the policy. The blockchain progresses through the ciphers according to a proportional round-robin and, with each block, takes a bean out of that cipher's bottle. At the end of the set, the bottles are empty, and the process repeats; if a new policy has been put in place, then that is the time when it will take effect.

The set size of 100 allows fairly fine-grained policies while still allowing the policy to be changed fairly frequently. The policy is driven by a configuration file which is kept uniform among blockchain users and miners by means external to the blockchain.

We also have a randomized version of Feijoada, where the total allocation is the same but the order is varied according to the previous block hash.

2.1.2 51% attacks

A typical criticism of multi-cipher blockchains:

I'm concerned about the security of dual PoW against 51% attack, relative to single algorithm PoW. (There are also various attacks requiring less than 51%, and the discussion below mostly applies to them as well. But for simplicity's sake, assume

51% or greater hash rate is the minimum threshold for a successful 51% attack.)

https://github.com/zcash/zcash/issues/3672

We do not believe that this concern holds up to scrutiny. In fact, we argue that multi-algo blockchains show far more resistance to 51% attacks.

To see why, let's consider the normal course of a 51% attack:

1. An attacker with +51% of the hash power in Algorithm X spends his coin on the public network.
2. Subsequently, he forks the chain from an antecedent block, omitting his spending transaction, and instead transfers those same coins to a different account, applying his superior compute power in Algorithm X to demonstrate more proof-of-work on his chain than on the public chain.
3. The public chain continues mining Algorithm X, but with less computing power.
4. After some time, the attacker releases this chain to the public, and since it shows more proof-of-work, the public accepts this fork, undoing his spending from step #1.

The power in this scenario comes from the attacker's ability to **exclude** transactions from the blockchain history. To see how multi-algorithm mining helps, let's consider the analogous scenario in a multi-algorithm chain:

1. An attacker with +51% of the hash power in Algorithm X spends his coin on the public network.
2. Subsequently, he forks the chain from an antecedent block, omitting his spending transaction, and instead transfers those same coins to a different account, applying his superior computing power in Algorithm X to demonstrate more proof-of-work on his chain than on the public chain.
3. The public chain continues mining Algorithm X, but with less computing power.
4. The public chain then, per its algorithm policy, switches to Algorithm Y, and Y miners contribute a block which includes the first spend in its history.
5. The public chain then, per its algorithm policy, switches to Algorithm Z.
6. After some time, the attacker releases this chain to the public, but since he does not have 51% of the chain's hash power in Algorithms Y and Z, he does not show more proof-of-work. The public does not accept this fork, leaving his spending from step #1.

If he only has 51% of the hash power in a single algorithm, then the attacker's dominance ends the moment that the blockchain switches to another algorithm, and so does his ability to dominate the hashrate.

If the algorithm changes more quickly than the double-spend window, even a miner

with absolute dominance of a given algorithm will very quickly see his dominance of the chain melt away, as the other algorithms take their turn, cement the public history, and process any transactions which the attacker has tried to exclude.

There are details to be managed, including the particular policy, how proof-of-work difficulty is evaluated across algorithms, and how long to wait post-transaction before currency is spendable. However, fundamentally, multi-algorithm chains significantly increase the difficulty of 51% attacks, because they make changing the history and excluding transactions, the two key operations of a 51% attack, much harder.

2.1.3 Mining

One can view a blockchain as a marketplace for delivering proof-of-work services to currency holders. Miners pay for the costs of proof-of-work computation, both capital expenses like computers and air conditioners and operating expenses like electricity, and they receive block rewards in return. The currency holders receive the ability to perform transactions, and they pay with block rewards to the miners.

By having a flexible cipher policy, we can take advantage of opportunities in the mining marketplace to acquire the greatest possible hash power. As certain kinds of hardware stop being economical on other chains, we can make room for them on Epic at a lower price, increasing their return on capital while delivering hash power to the network at a lower marginal cost. As other kinds of ciphers stop being effective, we can diminish or eliminate their allotment on Epic, making room for newer providers.

The Epic Foundation will provide this policy for the community, and this policy will be based upon continued research into market conditions and cryptographic security.

2.1.4 Security and long-term resilience

When novel attacks are found against ciphers, they transition from useful to not useful very quickly. MD5, SHA1, and Nimbus are three historical examples of this happening. When a blockchain depends on a single cipher, it is highly vulnerable to such failures.

Even worse, they usually rely on a single implementation of that cipher, and the history of software bugs in cipher implementations is even less confidence-inspiring than the history of their cryptanalysis. (Aside from Heartbleed, OpenSSL has had 10 high-severity bugs in the last five years, which is not very surprising for a codebase with almost half a million lines of C code.)

With our modular architecture, Epic can continually add support for new ciphers. As a bonus, the strong abstractions also allow us to add alternative implementations for existing ciphers into the codebase.

In this way, support for alternative ciphers can be added well in advance of any of our ciphers being broken, and such breaks can be dealt with easily and with minimal disruption of blockchain operations.

2.2 The Epic Foundation: supporting community and ongoing development

An important goal for Epic was to avoid the need for private investors to support ongoing education and development. Instead, a portion of block rewards go to the Epic Foundation to support the community which maintains the software. This arrangement will yield a different mix of costs and benefits than the more traditional profit-centric arrangement.

To achieve this goal, we had to modify the blockchain to split the block rewards between the miners and the Epic Foundation. This task was somewhat complicated by the structure of MimbleWimble, which generally requires that a wallet be online in order to receive payments or rewards. We did not like the security profile of the Foundation wallet being online, so we changed the mechanism to eliminate that requirement. That work, along with the interesting new directions it opens for other use cases, are described below.

3 New Features

3.1 Truly multi-algorithm blockchain

We surveyed existing MimbleWimble implementations for a starting point and selected Grin, an existing blockchain implemented in Rust. Rust has good security properties and a strong type system. Our team consists of programmers accustomed to working in languages with Hindley-Milner-style type inference, so we found Rust's type system to be intuitive and efficient.

Our major technical goal was to make the blockchain truly multi-algorithm for proof-of-work. Grin supports the Cuckoo category of algorithms, consisting of Cuckaroo and Cuckatoo, but these algorithms were custom-designed for Grin and share a very similar design. The abstractions over them in Grin are highly Cuckoo-specific; for example, the abstracted code still relied on the height of the graph for various purposes, which is not a useful metric for non-graph-based algorithms.

For this reason, the additional ciphers (ProgPow and RandomX) use the difficulty adjustment procedure in the Homestead release of Ethereum, which is both simple and well-tested.

We believe that this area holds promise when combined with algorithm selection, which we explain next.

3.2 Algorithm selection (Feijoada)

When proving work with multiple ciphers, there is the central question of which cipher to use for each block. For our initial implementation, we chose a deterministic process. We will describe that process and then describe possible future directions, as we think this will continue to be an interesting area for innovation.

At all times, the blockchain has a cipher policy. Nodes which do not agree on the policy will not be able to achieve consensus. Each policy consists of a set of one or more ciphers, each accompanied by a portion. The portion is an integer, and all of the portions must sum to precisely 100.

The policy is applied to the blockchain 100 blocks at a time, and the result is basically a round-robin application. On the first block (1, 101, 201, etc), the current policy is used to populate the "bottles" structure in the block. Each cipher is given a bottle, and 100 "beans" are placed in the bottles in accordance with the policy. With each block, the cipher is chosen in a way that keeps the ratio of the beans closest to the ratio of the policy. (In case of a tie, the alphabetically-first cipher has precedence.) The cipher used has a bean removed from its bottle. With the hundredth block, the final bean is removed, and the process begins again with 100

beans. We call this process "feijoada", after the traditional bean dish which all Brazillians, ourselves included, eat practically daily.

The Feijoada procedure results in a deterministic sequence of ciphers, allowing consensus to be maintained easily. It also allows the policy to be reset every 100 blocks, which should be around 100 minutes given our target emission rate of one block per minute.

The policy is kept as a configuration file; this conveniently leaves the matter of maintaining consensus across users a non-technical question. The Epic Foundation publishes a model policy file, and the software by default will regularly check for new versions of the file, confirming its cryptographic signature before using it.

In the future we are interested in allowing multiple algorithms to compete for each block, using a control loop to maintain the portions from the policy. But much care will have to be exercised to ensure that such a procedure remains deterministic and that consensus can be maintained.

3.3 New algorithms

Epic Cash uses ProgPow version 0.15.0 and RandomX version 1.0.3.

We use rust-bindgen as a library to automatically generate rust bindings for C in our build script. That same build script them compiles the libraries for Rust as well. We were able to find an approach that lets us auto-generate the C++ bindings for Rust and allows them to be generated and built in-line with the rest of our build process. Many other projects have taken more difficult approaches, so we are satisfied to have made this approach work.

We eliminated Cuckaroo from our codebase while retaining Cuckatoo.

3.4 Foundation levy

The MimbleWimble protocol requires an exchange of messages between sender and receiver in order to put a transaction into the blockchain; the sender passes inputs to the receiver, the receiver sends the cryptographic residue which establishes his sole ability to spend the output of the transaction, and then the sender must combine them into the transaction.

However, in the case of a block reward, you do not need to match both sides of the transaction, because it is a one-sided transaction. The receiver of the block reward only needs to know the value of the reward, and those values are already known before the blockchain even begins operation.

Thus, the Foundation does not actually need to have its wallet online. Further, if the

Foundation wallet were to be online, processing rewards in real-time, then it is unclear how the blockchain would maintain consensus about whether the Foundation is receiving the block rewards it is supposed to receive.

We therefore added code to pre-generate these Foundation reward messages, which we call "deposit slips", and they are pre-distributed to participants in the blockchain. For each block, they know what transaction to expect for the Foundation reward, and they will reject the block if the reward is not there.

This left one minor problem to be solved, which was the size of these deposit slips, which came in around 10GB for the lifetime of the Foundation reward. That size was manageable, but not ideal. So, we optimized it: instead of one Foundation reward every block, we instead do one Foundation reward every 1,440 blocks, and we make the reward 1,440 times as large. The economics are identical, but now the size of the deposit slips is a much more manageable 7MB for the lifetime of the Foundation reward.

3.5 Packaging

Our primary development platform is Linux, and we packaged the Epic software using dpkg for use on distributions like Debian and Ubuntu. The packaging uses the standard tools, so users can build their own dpkg from the source code in our repository in the standard ways.

Other community members have contributed Windows builds, and we are discussing extending support to those as well as RPM on Linux.

3.6 Cucumber testing

One aspect of the project which we found highly important was the addition of a more powerful testing framework. Grin comes with a good test suite, but it was incomplete and coded directly in Rust.

We wanted a more behavior-driven approach to testing, for several reasons:

- Expressing behaviors in something more like natural language facilitates understanding of system behavior by non-technical individuals;
- Expressing behaviors in something more like natural language also facilitates contributions from non-technical individuals to the test coverage;
- There is significant opportunity for code-reuse between tests;
- Behaviors are generally more expressive than code, lowering the cost and increasing the yield of tests.

We embraced the cucumberrust package to build cucumber-style testing into the codebase. We then used it to implement a very useful set of testing scenarios as

behaviors:

Match the mining and Foundation rewards with the whitepaper

- Add hardcoded coinbase
- Add coinbase to each mined block
- Refuse a Foundation output invalid
- Check a policy sequence of Cuckatoo using deterministic Feijoada
- Check a policy sequence of Cuckatoo and RandomX using deterministic Feijoada
- Check if blocks added in a blockchain match the policy
- Check if accept multi policies
- Refuse blocks that were not mined with a desired algorithm
- Mine empty chain
- Mine genesis reward chain
- Mine Cuckatoo genesis reward chain
- Mine forks
- Mine losing forks
- Longer fork
- Spend in fork and compact
- Output header mappings
- Mine md5 genesis reward chain
- Accept valid md5 PoW
- Refuse invalid md5 PoW
- Mine RandomX genesis reward chain
- Accept valid RandomX PoW
- Refuse invalid RandomX PoW
- Mine ProgPow genesis reward chain
- Accept valid ProgPow
- Refuse invalid ProgPow PoW

This approach has allowed us to generate good test coverage for our new features, improve coverage for existing functionality, communicate our test coverage more easily, and open the door for more people to contribute to our codebase.

4 Challenges

We experienced several challenges during the project.

Difficulty adjustment was difficult to tune across the different ciphers. We initially thought several of these difficulties were caused by the difficulty adjustment algorithm, but they were actually caused by subtle issues in how the algorithms score difficulty and how the algorithms were used.

We had difficulty with the drivers for GPU hardware, both OpenCL and CUDA, each of which has its own peculiarities. The particular hardware configuration on the build machine could impact the resulting software build in unforeseen ways. We continue to help novice users fully utilize the GPU hardware on their machines and expect continued improvement in this area.

5 Future Directions

In the short term, we plan to add support for Windows, Macintosh, and mining pools, while enhancing support for GPU hardware.

Longer term, we intend to add new cryptographic ciphers and evolve our approach to supporting and using multiple ciphers on the blockchain.

We want to support Epic Cash transactions better to facilitate its use in commerce.

And we want to have significantly more testing and a higher-assurance codebase.

We think all of this can be done in the months to come.

6 Conclusions

We described our goals and strategy, the technical details of our implementation, our experiences with the implementation, and future directions for development.

Our code is available on GitLab and is free and open-source software. We are an open development community and welcome involvement from anyone interested in supporting the project.

Date: 23 July 2019

Author: Todd Lewis <tlewis@brickabode.com>

Created: 2019-08-13 Tue 12:04

Validate