Matt Howard and Ajay Patel
CIS 556: Cryptography
University of Pennsylvania
Fall 2015

DriveCoin:
A Proof of Space Cryptocurrency

Abstract

We present an implementation of a cryptocurrency based on the newly-theorized and studied Proof of Space cryptographic primitive. We first discuss the basic elements of a cryptocurrency such as BitCoin. We summarize the idea of a proof of space and then apply the cryptocurrency concepts to our own Proof of Space currency. We explore the subtleties as well as the implications of our implementation. Through discussing our design decisions and the security implications of our system, we contribute a basis on top of which more research into Proof of Space cryptocurrencies can be performed.

1 Introduction

1.1 Bitcoin and other Cryptocurrencies

Bitcoin introduced a ground-breaking application of cryptography to create a decentralized currency. Bitcoin has no central authority. There are no banks, governments, or servers that centralize the currency. Instead, Bitcoin relies upon a peer-to-peer connection system and the network of peers each keeping a copy of a ledger of transactions known as the "blockchain".

In order for a cryptocurrency to be usable, it must, among other properties, have scarce coins, security, and irreversible transactions. Coin scarcity is necessary for the coins to have real monetary value. In mathematical terms, this means with high probability, the number of coins in the system with respect to time is bounded by some function. Security can be defined by the concept of an individual "possessing" some coin as well as the condition that the individual can choose to spend or retain the coin however she chooses. Irreversible transactions, or guarantee against double spending, is the principle that once a coin has been transacted between two parties, the payer can no longer spend that coin. In practice, all three of these properties are met through cryptography in Bitcoin and other cryptocurrencies.

1.2 Proof of Work

In the Bitcoin system, in order to add a block of transactions to the blockchain, you must solve a proof of work problem ("mining"). The incentive for doing so is 1 BTC for a miner who adds a block to the blockchain.The proof of work problem is as follows:

$$\text{SHA256}(Block) < \text{Target Difficulty} \tag{1}$$

In other words, this asserts that the hash of a block must contain a certain number of leading zeros. The expected number of hashes (and the time complexity) of solving the proof of work is exponential ($O(2^n)$) in the number of zeros. Bitcoin adjusts the target difficulty to compensate for the power of the network such that a block is likely to be solved every 10 minutes.

If two miners solve the proof of work, then the largest chain wins in the long run. A proof of work is necessary is NOT to secure transactions. Transactions are unforgeable because they are signed by the RSA private key pair of the sender. Rather, a proof of work, secures the "transaction history". If an attacker were able to remove transactions from the history, he or she could potentially buy an item, and after receiving the item, delete the transaction and spend the bitcoins again ("double spending").

By making the proof of work problem dependent on a scarce resource "computing power", it is unlikely an attacker will be able to outperform the network in adding blocks to the blockchain. In fact, since the attacker must provide a larger valid chain to re-write the transaction history, an attacker's probability of success drops off exponentially as more blocks are added to the block chain.

1.3 The Problem

Bitcoin mining has led to the proliferation mining hardware. Miners require rigs that can perform millions of hashes per minute. Mining rigs range from custom built PCs to FPGA solutions to ASIC solutions. Not only has the advent of FPGA and ASIC mining rigs made mining out of reach for the average person, but it requires constant electric power to run the CPUs and GPUs as well as the constant utilization of dedicated hardware that would not exist otherwise. This poses an environmental problem for Bitcoin.

1.4 A Solution: Proof of Space

There is an alternative to proof of work called a proof of space scheme introduced by Dziembowski. Proofs of space are a new cryptographic primitive. Rather than an adversary proving to some verifier that some amount of work has been completed, an adversary proves to some verifier that they have allocated some number of bytes of storage. Proof of space makes use of "data storage" as a scarce resource instead of "computational power". Proof of space makes use of hash trees to efficiently allow verification of a challenge without storing the tree. (See 2.2 "The Challenge" for more information)

Proofs of space have two advanteges over proofs of storage. First, they are more energy efficient since on the set-up step requires intensive computation, and the lookup and verify steps are very efficient. This means that a user doens't have to constantly use computation power in order to mine. Only a quick disk look-up is required. Secondly, a proof of space doesn't incentivize dedicated hardware the same way a proof of work does. There is no clear analog to the FPGAs and ASIC hardware used in proof of work; The only medium significantly currently cheaper than hard disk space is tape drive storage, which is impractical for quick look-ups and used today only for archiving. Therefore, the system does not disadvantage the common user and incentive wasteful dedicated resources like a proof of work does.

2 DriveCoin Specification

2.1 Block Format

The **block** is a UTF-8 file of variable length. The first entry is a hash of the previous block. The second entry contains the block number. The third entry includes a list of transactions on the DriveCoin network. The fourth entry contains an RSA public key for a new coin on the network. The last two entries identify a solution to the block's cryptographic challenge (explained in "The Challenge"). Table 1 is an exact specification.

A block if valid if it meets the structural specification and if the PoS is valid (described below).

Description	Standard / Encoding	Length
Hash of Previous Block	SHA-256, hex encoding	64 bytes
Newline	`'0x0A'`	1 byte
Block Number, 0 indexed	Big-endian 64-bit unsigned long, hex encoded	16 bytes
Newline	`'0x0A'`	1 byte
List of Transactions	Described in Next Section	Variable Length
Newline	`'0x0A'`	1 byte
Private Key for 1 new DriveCoin	RSA 2048-bit public key, hex encoded	512 bytes
Newline	`'0x0A'`	1 byte
PoS Hash Tree Root	Hex encoding	64 bytes
Newline	`'0x0A'`	1 byte
PoS Hash Tree Path	64-bit Hex encoded big-endian bit length + hex padded to full octet	Variable length

Table 1: Block Specification

2.1.1 Addresses

DriveCoin addresses are hex-encoded RSA public keys ($N = p * q$ values). Base58 was not used as some of Python's standard library functions could not handle the conversion of a big integer and we did not wish to make the source dependent on Sage.

2.1.2 Transactions

Transactions are a simplified version of Bitcoin's transactions. We omit unnecessary features like multiple recipients and scripting. Each transaction consists of a sender address, a recipient address, and an amount. In order for a transaction to be considered valid, the transaction must be signed by the senders RSA private key and the sender's "balance" (as verified by blockchain transaction history) must be sufficient to cover the amount.

2.1.3 Block Chain

The **block chain** a sequence of blocks with incrementing block numbers starting at 0. A **block chain** is valid if each block is valid and the hash value field matches the preceding block's hash value. The first (number 0) block's previous block hash is defined as the hash of the empty string.

At any point in time, the longest chain on the network is considered to be valid.

2.2 Proof of Space

The challenge scheme consists of a fixed member $\mathcal{A}$ of the network $\mathcal{N}$. The DriveCoin challenge scheme must satisfy the following constraints for an arbitrary N:

1. $\mathcal{A}$ performs a set-up step in $\Omega(N)$ time and $O(N)$ space resulting in $\mathcal{A}$ storing some data $\mathcal{O}$ of size $\Omega(N)$

2. Successive challenges are generated by $\mathcal{N}$ without any member of $\mathcal{N}$ necessarily having knowledge of a solution. The challenge can be stated in $O(1)$ space

3. $\mathcal{A}$ can find a solution using $\mathcal{O}$ in $O(\log N)$ time and space. $\mathcal{A}$ can communicate the solution to $\mathcal{N}$ in $O(1)$ space.

4. A verifier of the solution that any member on $\mathcal{N}$ can run in $O(1)$ time and space without knowledge of $\mathcal{A}$'s $\mathcal{O}$.

Here, N translates to the amount of space $\mathcal{A}$ can dedicate to the challenge.

2.2.1 Our Challenge

The set-up step consists of constructing a Hash Tree T of height h. Specifically, the Hash tree is defined as a binary tree where every node has an associated 256-bit value and all edges satisfy the following requirement. The value of a left child is the SHA-256 hash of its parent's value. The value of a right child is the SHA-256 hash of its parent's value with every even bit flipped. The tree's root can be any 256-bit value. Each member of the network is expected to generate a random value for its tree's root.

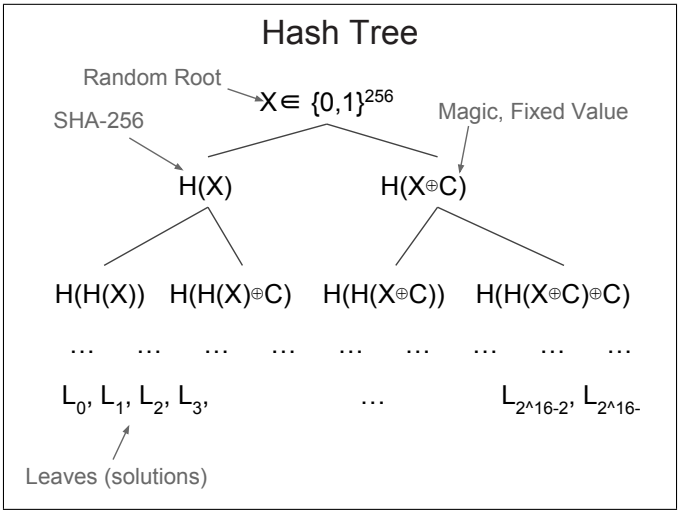

Figure 1: Hash Tree Definition

Each challenge begins with a given 256-bit string s. $\mathcal{A}$ attempts to find a node v with a depth of h that has the least distance to s. We measure distance between two 256-bit values as the absolute value of the difference when interpreting the values as big-endian integers. If A finds a node v with distance d from s, it outputs the value of u, the root of the tree in which v is found, and the path from u to v. We can encode the path from u to v as simply a binary string p of length h where 0 signifies a step to the left child and 1 signifies a step to the right child.

A verifier $\mathcal{V}$ can take in (s, u, p) and then trace the path p down a tree with root u to find the vertex v. V then can compute the distance between v and s and will accept the (s, u, p) duple with the least distance of all known (s, u, p) triples. Therefore, the validity of a (u, p) can change over time as $\mathcal{V}$ sees more duples.

To simplify our theoretical treatment, we assume $\mathcal{A}$ generates one tree's set of leaves of size $N \propto 2^{h-1}$ as

its output $\mathcal{O}$. In practice, we allow $\mathcal{A}$ to choose how much space it dedicates simply by generating multiple hash trees.

2.3 Security Considerations and Preventative Measures

1. Multiple Chains - Because there isn't a disincentive to work on multiple chains at once in proof-of-storage since lookups cheap with respect to time, this scheme may face long multiple chains growing in parallel never leading to consensus. DriveCoin attempts to resolve this by ordering chains with regard to their distance from the target leaf. The chain with the closest average distance per block will be choosen. It is highly unlikely a single attacker will be able to outperform the network in finding a chain such that each block has a smaller distance than a legitimate chain.
2. Grinding - Grinding occurs when a participant who has solved a block is able to manipulate the challenge for the next block such that the participant has a greater probability of solving the next block. A scheme that is vulnerable to this is vulnerable to a double spend attack. In a proof of storage system, the challenge must be selected such that the previous block solver cannot manipulate it. As recommended in the SpaceMint paper, we use an external source of entropy to derive the next challenge. In particular, DriveCoin uses financial market data to derive the next challenge.

2.4 Incentive for Good Behavior

Thinking about DriveCoin from a game theoretical perspective we want to achieve a Nash equilibrium. In DriveCoin, no single individual participant has any incentive to not store the hash tree. By not storing the hash tree they will either not be able to solve the challenge or they will have to brute force hash until they solve the challenge. Because the time complexity of brute force hashing is exponential, while the time complexity of a storage lookup is constant, they are at no advantage to not store the tree.

Furthermore, because our scheme derives the challenge for each block from an external beacon (financial market data), the solver is unable to manipulate the next challenge to give themselves an advantage.

3 Implementation

3.1 Overview

While a cryptocurrency based on a proof of space system has been published in papers before, they all theorized about potential complications and proof of work schemas but did not present a functional cryptocurrentcy. We wanted to get a feasible working proof of concept of a proof of space cryptocurrency. Since we were interested in building a "toy" example, we did not modify the Bitcoin source code. Instead, we built a simple cryptocurrency from ground up. Our goal was not to replicate all of the functionality Bitcoin provides but to provide the bare essentials of a cryptocurrency with a proof of space verification method for the block chain. Our intended contribution is to present a working implementation of a proof of space cryptocurrency. We do not except it to be used in practice, however it may be further studied and analyzed to help understand the implications and issues in using proof of space as a cryptocurrency primitive.

3.2 Technology

In order to speed up development, we used Python as the main language for our cryptocurrency. In addition, the popular packages PyCrypto and Twisted were used to implement cryptography and P2P networking functionality respectively. We used the memory-mapped file library in python to allow efficient and easy access to our novel disk storage format.

3.3 Structure of the Project

The DriveCoin project is seperated into a few different parts, which together build up a working cryptocurrency:

1. Tree - Implementation of our proof of space hash tree construction.
2. Network - Implements peer to peer networking with Twisted over TCP port 8123. This includes listening to requests and broadcasts from peers and sending requests and broadcasts to peers.
3. Transaction - Implements the DriveCoin transaction specification. Is able to verify the signature of the transaction using the sender's address.
4. Block - Implements the DriveCoin block specification.
5. Blockchain - Implements the DriveCoin blockchain specification which is able to maintain the latest blockchain through proof of space verification and use of the network.
6. BalanceManager - Uses the blockchain to calculate balances for every address involved in a transaction in all of DriveCoin's transactions.
7. Wallet - A command line interface for a user to use DriveCoin. A user is able to create wallets (RSA public key addresses), manage them, check balances, and send transactions to other users.
8. Miner - A command line interface for a user to mine new DriveCoins and help add blocks to the blockchain. The miner listens on the network for pending transactions, adds them to a block, and attempts to add the block to the blockchain by solving a proof of space challenge.
9. Beacon - Component of the network responsible for broadcasting the basis of new proof of space challenges

3.4 Implementation of Proof of Space System

Our proof of storage system consists of three main components: the set-up step, the look-up procedure, and the verification procedure. They work together to realize the challenge presented in Section 2.2.

The set up step allows the user to store a file of arbitrary size filled with potential solutions to our challenge. The ratio of the user's total storage dedicated to DriveCoin to the total storage dedicated to DriveCoin across the network must be proportional to the user's chance of winning a challenge. Therefore, the number of potential solutions must be linear to storage space and look-ups must be fast (linear, or at least logarithmic). We chose to store the solutions as nodes in a binary search tree. Solutions correspond to leaves in a hash tree. Each node is a struct of the leaf's value, the root value of its original hash tree, the binary path down

the tree to the leaf, and pointers to the node's left and right children. The solution set file consists of an 8-bit integer storing the number of nodes followed by 82-bit blocks storing the nodes, with the root of the binary search tree always first in the file. The binary search tree is unbalanced which may create non-optimal height but since the node values are uniformly randomly distributed, heuristically this still creates a tree of height $O(\log n)$. The set-up step involved creating hash trees until the file is of the user's specified size. Building the hash trees relies on a recursive depth-first search algorithm which allows the entire tree to be generated while in $O(\log(N))$ memory space, where N is the size of the tree. Each hash tree's leaves are added to the file by inserting them in the binary search tree and storing their struct in the next available 82 Byte block.

The file format is designed so that the user can add additional hash trees to the solution set at a later time if she so pleases. This is useful if more hard disk space becomes available. We could add functionality to remove hash trees from the solution set, but this is a bit more complicated to do efficiently. In essence, we would like to "prune" the hash tree as to make the tree more balanced, and we decided to leave the design of such an algorithm for further work.

We chose to use hash trees of height 16 since this is a manageable recursion depth. This means our hash trees contain $2^{16-1} = 32768$ leaves each, and each tree requires 32768 leaves $\cdot$ 82 bytes/leaf $= 2.56$MB to store on disk. If we want to look at the space density of our storage mechanism, we might ask what the user gets cryptographically for 1 TB. This translates to 2^{40} B /(82 bytes/leaf) $\approx 2^{33.64}$. This means that given any uniform random 256-bit value, we can expect to find a hash tree for which one of its leaves matches the first 34 bits of our value.

The look-up procedure consists of searching for a given challenge value. This is $\log(N)$ time in the amount of space used, which still makes lookups fast even if terabytes of data are stored. Since the challenge value is the result of a SHA-256 hash and our node values are the results of other SHA-256 hashes, we don't expect to find a perfect match based on the security of SHA-256. Instead, we want to find the closest node value. This means traversing as one would in a regular binary search tree search until you reach a leaf. The leaf is either the least upper bound or the greatest lower bound of the given value in the tree depending if the leaf's parent is to the left or the right of the leaf. To find the closest node value as measured by the absolute value of the difference, we simply consider the leaf node along with the leaf node's ancestor that serves as the closest bound in the opposite direction. We calculate the distances to the challenge value and pick the closer one. Then, we use the hash tree parent value and the binary tree path stored in the struct as our proof of space solution.

The verification step consists of taking hash tree root value, a binary tree path, and a leaf value and determining if the leaf value is derived from the other two values. Algorithmically, this consists of tracing from the root value down the hash tree until reaching the leaf, following the definition of our hash tree. This operation takes place in $O(1)$ time since it reduces to computing 16 SHA-256 hashes.

3.4.1 Proof of Security

We claim our proof of space implementation satisfies the definition in Section 2.2.

The set-up step consists of building out one hash tree in $\Theta(2^h) = \Theta(2^{15}) = \Theta(1)$ time and $\Theta(h) = \Theta(16) = \Theta(1)$ space, assuming we perform a depth-first search and can output the leaves as we generate them. The output $\mathcal{O}$ is then proportional to the number of leaves $2^{h-1} = 2^{15}$. We allow the user to build arbitrarily

many hash trees, so let n be the number of hash trees generated. Then, the set-up runs in $\Theta(n)$ space and $\Theta(n)$ time, generating a $\Theta(n)$ output. This satisfies condition #1.

A challenge is generated based off of information from the network and is independent of the amount of space stored. Thus the challenge uses $\Theta(1)$ space and time. $\mathcal{A}$ must find a hash tree leaf value as close to the challenge as possible, which runs in $O(\log n)$ time and results in a hash tree root and a binary path to the leaf. Thus, the solution can be sent to the network in $\Theta(1)$ space.

The verification step consists of tracing the path down the hash tree to the leaf and verifying its hash, which runs in $\Theta(h) = \Theta(1)$ time and $O(1)$ space. Therefore, our system meets the proof of space requirement.

3.5 Beacon

Our scheme requires a random beacon to generate the basis for new Proof of Space challenges regularly on time intervals. This concept comes from Park, who suggests using market prices. A random becon must be unpredictable and free from artificial manipulation. We considered following Park's advice and using NASDAQ data, but this was problematic for a few reasons. First, NASDAQ is only open 9:30 a.m. to 4:00 p.m. EST, outside of which prices don't change and we need a constantly changing data source. Second, while the market may exhibit the behavior of a random walk and has a high amount of entropy overall, it is far from a completely random source of information, and a bad player could exploit this vulnerability to precompute blocks.

In the interest of simplicity, we opted for a centralized, random beacon that publishes a random 256-bit value every 10 minutes. This consists of a simply python script and webserver to generate and serve these random values as hex-encoded text files over HTTP. We use PyCrypto to generate cryptographically-secure pseudorandom values. This is an obvious simplification of our otherwise decentralized currency, but the source of our beacon is independent of the operation of our currency.

4 Future Work

The DriveCoin project is in no way complete and ready to be used in the wild as a cryptocurrency. There a number of tasks that should or could be looked at for future development:

1. Networking and Peer-to-Peer Management - While we have a basic system for connecting peers together, the system could be more robust. Currently, all users are connected to each other (a complete connected graph). This would not scale to the size of Bitcoin. Instead some smarter processes could be developed allowing the creation of a graph that is strongly connected that would still allow broadcasts to flow through the network without having all users connect to each other.
2. Beacon - We used a beacon to solve the problem of grinding. In reality, this beacon may be data from financial markets, which if the Random Market Hypothesis is correct, should introduce additional entropy into the target value for the proof of space challenge. However, in our DriveCoin implementation, the beacon is simulated by a webserver that takes in a timestamp (in the past) and returns random bits associated with that timestamp. It may even be possible to solve this problem without having to depend on an external beacon.

3. Bitcoin features - Ideally, all of the features Bitcoin has would be ported over to DriveCoin. This would include the scripting language in transactions and optimizations made to calculate account balances and lookup transactions quickly.
4. Self-adjusting network parameters - Bitcoin automatically adjusts the network's parameters to reach a certain difficulty (1 block solved every 10 minutes). DriveCoin does not currently attempt to change any network parameters at the moment.
5. Mining incentives and minting cap - DriveCoin does not currently set a limit to the number of coins that can be minted. Bitcoin, on the other hand, will stop minting coins after it reaches 21 million. In order to still provide miners an incentive to add new blocks to the blockchain, Bitcoin plans to have transactions offer a transaction fee which goes to the miner who adds the transaction to a block in the blockchain. DriveCoin also has not yet implemented a form of transaction fees.
6. Transmission of Data - Since we had a limited scope of time to implement the specification, the data being sent over the network is in the form of a Python pickle string (python's built in serializing format) instead of encoded in a strict format and checked for that format. This can lead to executable code being sent over the network if a malicious party were to modify the source code.
7. Further Security Audits - The DriveCoin specification needs to be further verified that it cannot be taken advantage of by an attacker. While grinding and multiple chains are two possible problems that arise with a proof of space cryptocurrency, there may be other problems we were unaware of when developing the specification.

5 Results

The resulting cryptocurrency we built is able to function across the network to send and receive transactions, agree upon a proof of space challenge through an external beacon, verify proof of space challenge responses from peers, and achieve consensus on a proof of space blockchain.

It is a lightweight cryptocurrency implementation that does not rely on the computation of millions of hashes to secure the network. The passive use of hard disk storage allows average users of DriveCoin to participate in the mining process without having to buy dedicated hardware and overall should be a more energy efficient process for validating a blockchain.

6 Conclusion

We were able to create a simple cryptocurrency implementing a proof of space system. By working upon two papers, one focused on proofs of space, and another focused on their application to a cryptocurrency, we were able to create a novel new cryptocurrency which we believe to be the first implementation of a proof of space based cryptocurrency.

7 References

1. DriveCoin Source Code - https://github.com/moward/DriveCoin
2. Bitcoin Paper - https://bitcoin.org/bitcoin.pdf
3. Proofs of Space - https://eprint.iacr.org/2013/796.pdf
4. Proof of Space 2 - https://bitcointalk.org/index.php?topic=310323.0
5. SpaceCoin - https://eprint.iacr.org/2015/528.pdf
6. BasicCoin - https://github.com/zack-bitcoin/basiccoin