Bitcoin and Cryptocurrency Technologies

Arvind Narayanan, Joseph Bonneau, Edward Felten,
Andrew Miller, Steven Goldfeder

Draft — Aug 25, 2015

Feedback welcome! Email bitcoinbook@lists.cs.princeton.edu

Chapter 8: Alternative Mining Puzzles

Mining puzzles are at the very core of Bitcoin because their difficulty limits the ability of any one party to control the consensus process. Because Bitcoin miners earn rewards for the puzzles that they solve, we expect that they'll spend considerable effort trying to find any available shortcuts to solve these puzzles faster or more efficiently, in the hope of increasing their profits. On the other hand, if there's work that helps the network but doesn't directly help them solve puzzles any faster, miners might be incentivized to skip it to minimize their costs. So the design of the puzzle plays an important role in steering and guiding participation in the network.

In this chapter, we're going to discuss a variety of possible alternative puzzle designs, assuming we could modify Bitcoin's puzzle or even design a new puzzle from scratch. A classic design challenge has been to make a puzzle which is ASIC-resistant, leveling the playing field between users with ordinary computing equipment and users with optimized custom hardware. What else could we design the puzzle to achieve? What other kinds of behaviors would we like to encourage or discourage? We'll talk about a few examples with various interesting properties, from decreasing energy consumption to having some socially-useful side effects to discouraging the formation of mining pools. Some of these are already used successfully in practice by altcoins. Others are research ideas that might turn out to be used in the future.

8.1 Essential Puzzle Requirements

We'll start by look at some essential security requirements for mining puzzles. It doesn't do us any good to introduce fancy new features if the puzzle doesn't still satisfy the basic requirements that it needs to keep Bitcoin secure.

There are many possible requirements, some of which we've talked about in Chapters 2 and 5. Mining puzzles need to be quick to verify because every node on the network validates every puzzle solution — even nodes that aren't involved in mining directly, including SPV clients. We would also like to have adjustable difficulty so that the difficulty of the puzzle can be changed over time as new users enter the network with increasing amounts of hash power contributed. This enables the puzzle to be difficult enough that attacks on the blockchain are costly, but puzzle solutions are still found at a fairly steady rate (about once every ten minutes in Bitcoin).

What exactly is Bitcoin's mining puzzle? So far we've just called it "Bitcoin's puzzle." More precisely, we can call it a *partial hash-preimage puzzle*, since the goal is to find preimages for a partially specified hash output — namely, an output below a certain target value. Some other rare property could also work, such as finding a block whose hash has at least k bits set to zero, but comparing the output to a target is probably the simplest.

It's easy to see how Bitcoin's SHA-256 hash-based mining puzzle already satisfies these two requirements. It can be made arbitrarily more difficult by tweaking a single parameter (the *target*). Checking solutions is trivial, requiring just a single SHA-256 computation and a comparison, no matter how difficult the puzzle was to solve.

Progress-freeness. Another central requirement more subtle: the chance of winning a puzzle solution in any unit of time should be roughly proportional to the hash power used. This means that really large miners with very powerful hardware should only have proportional advantage in being the next miner to find a puzzle solution. Even small miners should have some proportional chance of being successful and receiving compensation.

To illustrate this point, let's think about a bad puzzle that doesn't satisfy this requirement. Consider a mining puzzle that takes exactly *n* steps to find a solution. For example, instead of finding a block whose SHA-256 hash is below a certain target, we could require computing *n* consecutive SHA-256 hashes. This wouldn't be efficient to check, but nevermind that for now. The bigger problem here is that since it takes exactly *n* steps to find a solution, then the fastest miner in the network will always be the one who wins the next reward. It would soon become clear which miner was solving every puzzle, and other miners would have no incentive to participate at all.

Again, a good puzzle gives every miner the chance of winning the next puzzle solution in proportion to the amount of hash power they contribute. Imagine throwing a dart at a board randomly, with different size targets which correspond to the mining power held by different miners. If you think about it, this requirement means the odds of solving the puzzle must be independent of how much work you have already spent trying to solve it (because big miners will have always spent more work). This is why a good mining puzzle is called *progress-free*.

From a mathematical perspective, this means that a good mining puzzle must be a *memoryless process* — anything else would inevitably reward miners for past progress in some way. Therefore any feasible puzzle will inherently involve some sort of trial-and-error. The time to find a solution will therefore inevitably form an exponential distribution as we saw in Chapter 2.

Adjustable difficulty, fast verification, and progress-freeness are three crucial properties of Bitcoin mining puzzles. SHA-256-based partial pre-image finding certainly satisfies all three. Some people argue that other properties which Bitcoin's mining puzzle satisfies are also essential, but we'll discuss other potential requirements as they come up while we explore other potential functions.

8.2 ASIC-resistant puzzles

We'll start with the challenge of designing an *ASIC-resistant* puzzle, which has been by far the most widely discussed and sought after alternative mining puzzles. As we discussed in Chapter 5, Bitcoin mining was initially done primarily with ordinary computers, eventually extended to GPUs and customized FPGA devices, and now is almost exclusively done by very powerful optimized ASIC chips.

These ASICs are so much more efficient than general purpose computing equipment that mining with an ordinary computer (or even some early generation ASICs) is no longer worth the price of electricity, even if the hardware is free.

This transition has mean that most individuals participating in the Bitcoin ecosystem (for example customers or merchants transacting using Bitcoin) no longer have any role in the mining process. Some people feel this is a dangerous development, with a smaller group of professional miners controlling the mining process. In Satoshi Nakamoto's original papers on Bitcoin, the phrase "one-CPU-one-vote" was used, which has sometimes been taken to mean Bitcoin should be a democratic system owned by all of its users.

Others feel the rise of ASICs is inevitable and not to the detriment of Bitcoin, and that the desire for ASIC-resistance is simply people wanting to go back to "the good old days." Without taking a side on whether ASIC-resistance is desirable, we can dive into the technical challenges and some of the proposed approaches for achieving this goal.

What does ASIC-resistance mean? Generally speaking, we want to disincentivize the use of custom-built hardware for mining. Interpreting this strictly would mean designing a puzzle for which existing general-purpose computers are already the cheapest and most efficient devices. But this would be impossible. After all, general purpose computers already have special-purpose optimizations. Not all products have the same optimizations and they change with time. For example, in the past decade Intel and AMD have both added support for special instructions (often called "adding hardware support") to compute the AES block cipher more efficiently. So some computers will always be less efficient than others at mining. Besides, it's hard to imagine designing a mining puzzle that would rely on features like the speakers and screen that most individual's personal computers contain. So special-purpose machines stripped of these features would still probably be cheaper and more efficient.

So in reality our goal is a more modest one: coming up with a puzzle that reduces the gap between the most cost-effective customized hardware and what most general-purpose computers can do. ASICs will inevitably be somewhat more efficient, but if we could limit this to an order of magnitude or less it might still be economical for individual users to mine with the computers they already have.

Memory-hard puzzles. The most widely used puzzles which are designed to be ASIC-resistant are called *memory-hard* puzzles — puzzles that require large amount of memory to compute, instead of, or in addition to, a lot of CPU time. A similar but different concept is *memory-bound* puzzles in which the time to access memory dominates the total computation time. A puzzle can be just memory-hard without being memory-bound, or memory-bound without being memory-hard, or both. It's a subtle but important distinction arising from the fact that even if CPU speed is the bottleneck for computation *time*, the *cost* of solving a large number of such puzzles in parallel might still be dominated by the cost of memory, or vice versa.

Why might memory-hard or memory-bound puzzles help ASIC resistance? The logical operations required to compute modern hash functions are only a small part of what goes on in a CPU, meaning that for Bitcoin's puzzle, ASICs get a lot of mileage by not implementing any of the unnecessary functionality. A related factor is that the variation in memory performance (and cost per unit of performance) is much lower than the variation in computing speeds across different type of processors. So if we could design a puzzle that was memory-hard, requiring relatively simple computation but lots of memory to compute, this means that the cost of solving a puzzle would improve a the slower rate of memory cost improvements.

SHA-256 is decidedly not memory-hard, as we've seen, requiring only a tiny 256-bit state which easily fits into CPU registers. But it isn't too difficult to design a memory-hard proof-of-work puzzle.

Scrypt. The most popular memory-hard puzzle is called scrypt. This puzzle is already widely in Litecoin, the second most popular cryptocurrency, and a variety of other Bitcoin alternatives.

Scrypt is a memory-hard hash function, originally designed for hashing passwords in a way that is difficult to brute-force, so the mining puzzle is the same Bitcoin's partial hash-preimage puzzle except with scrypt replacing SHA-256.

The fact that scrypt existed prior to Bitcoin and has been used for password hashing gives some confidence in its security. Password hashing has a similar goal of ASIC-resistance, because for security we want an attacker with customized hardware to not be able to compute password hashes much faster than the legitimate user or server, who presumably have only general-purpose computers.

Scrypt basically works in two steps. The first step involves filling a large buffer of random access memory (RAM) with random data. The second step involves reading from (and updating) this memory in a pseudorandom order, requiring that the entire buffer is stored in RAM.

Figure 8.1: Scrypt pseudocode

```
def scrypt(N, seed):
    V = [0] * N  // initialize memory buffer of length N

  // Fill up memory buffer with pseudorandom data
  V[0] = seed
  for i = 1 to N:
     V[i] = SHA-256(V[i-1])

  // Access memory buffer in a pseudorandom order
  X = SHA-256(V[N-1])
  for i = 1 to N:
     j = X % N  // Choose a random index based on X
     X = SHA-256(X ^ V[j]) // Update X based on this index

  return X
```

Figure 8.1 shows Scrypt pseudocode. It demonstrates the core principles but we've omitted a few details: in reality scrypt works on slightly larger blocks of data and the algorithm for filling up the buffer is slightly more complex.

To see why scrypt is memory-hard, let's imagine trying to compute the same value without using the buffer V. This would certainly be possible — however, in line 9, we would need to recompute the value V[j] on the fly, which would require computing j iterations of SHA-256. Because the value of j during each iteration of the loop will be pseudorandomly chosen between 0 and N-1, this will require about N/2 SHA-256 computations. This means computing the entire function will now take N * N/2 = $N^2/2$ SHA-256 computations, instead of just 2N if a buffer is used! Thus, the use of memory converts scrypt from an O(N) function to an $O(N^2)$. It should be simple to choose N large enough that the $O(N^2)$ is slow enough that using memory is faster.

Time-memory tradeoffs. While it would be much slower to compute scrypt without the help of a large memory buffer, it is still possible to use less memory at the cost of slightly more computation. Suppose that we use a buffer of size N/2 (instead of size N). Now, we could store only the values V[j] if j is even, discarding the values for which j is odd. In the second loop, about half of the time an odd value of j will be chosen, but this is now fairly easy to compute on the fly — we simply compute SHA-256(V[j-1]) since V[j-1] will be in our buffer. Since this happens about half the time, it adds N/2 extra SHA-256 computations.

Thus, halving our memory requirement increases the number of SHA-256 computations by only a quarter (from 2N to 5N/2). In general, we could store every only every kth row of the buffer V, using N/k memory and computing (k+3)N/2 iterations of SHA-256. In the limit, if we set k = N, we're back up to our earlier calculation where the running time becomes $O(N^2)$. These numbers don't apply precisely for scrypt itself, but the asymptotic estimates do.

There are alternate designs that mitigate the ability to trade off memory with time. For example, if the buffer is continually updated in the second loop, it makes the time-memory tradeoff less effective as the updates will have to be stored.

Verification cost. Another limitation of scrypt is that it takes as much memory to verify as it does to compute. In order to make the memory hardness meaningful, N will need to be fairly large. This means that a single computation of scrypt is orders of magnitude more expensive than a single iteration of SHA-256, which is all that is needed to check Bitcoin's simpler mining puzzle.

This has some negative consequences, as every client in the network must repeat this computation in order to check that a claimed new block is valid. This could slow down propagation and acceptance of new blocks and increase the risk of forks. It also means every client (even lightweight SPV clients) must have enough memory to compute the function efficiently. As a result, the amount of memory N which can be used for scrypt in a cryptocurrency is somewhat limited by practical concerns.

Until recently, it wasn't known if it was possible to design a mining puzzle that was memory-hard to compute but fast (and memory-easy) to verify. This property is not useful for password hashing, which had been the primary use case for memory-hard functions before their use in cryptocurrencies.

In 2014, a new puzzle called Cuckoo Cycle was proposed by John Tromp. Cuckoo Cycle is based on the difficulty of finding cycles in a graph generated from a cuckoo hash table, a data structure which itself was only proposed in 2001. There isn't any known way to compute it without building up a large hash table, but it can be checked simply by checking that a (relatively small) cycle has been found.

This might make memory-hard or memory-bound proof of work much more practical for use in Bitcoin consensus. Unfortunately, there is no mathematical proof that this function can't be computed efficiently without using memory. Often, new cryptographic algorithms appear secure, but the community is not convinced until they have been around for many years without an attack being found. For this reason, and due to its recent discovery, Cuckoo Cycle has not been used by any cryptocurrency to date.

Scrypt in practice. Scrypt has been used in many cryptocurrencies, including several popular ones such as Litecoin. The results have been somewhat mixed. Scrypt ASICs are already available for the parameters chosen by Litecoin (and copied by many other altcoins). Surprisingly, the performance improvement of these ASICs compared to general purpose computers has been equal to or larger than that for SHA-256! Thus, scrypt was decidedly not ASIC-resistant in the end, as least as used by Litecoin. The developers of Litecoin initially claimed ASIC-resistance was a key advantage over Bitcoin, but have since admitted this is no longer the case.

This may be a result of the relatively low value of N (the memory usage parameter) used by Litecoin, requiring only 128kB to compute (or less if a time-memory tradeoff is used, which was commonly done on GPUs to get the entire buffer to fit into a faster cache). This has made it relatively easy to design lightweight mining ASICs without a complicated memory access bus needed gigabytes of RAM, as general purpose computers have. Litecoin developers didn't choose a value that was much higher (which would make ASICs more difficult to design) because they considered the verification cost impractical.

Other approaches to ASIC-resistance. Recall that our original goal was simply to make it hard to build ASICs with dramatic performance speedups. Memory-hardness is only one approach to this goal, and there are others.

The other approaches, unfortunately, are not very scientific and have not been as rigorously designed or attacked as memory-hard functions. The most well known is called X11, which is simply a combination of eleven different hash functions introduced by an altcoin called Darkcoin and since used by several others. The goal of X11 is to make it considerably more complicated to design an efficient ASIC as all 11 functions must be implemented in hardware. But this is nothing more than an inconvenience for hardware designers. If an ASIC were built for X11, it would surely make CPU and GPU mining obsolete.

Sidebar: where did X11's hash functions come from? From 2007 to 2012, the US National Institute of Standards ran a competition to choose a new hash function family to be the SHA-3 standard. This produced a large number of hash functions which were submitted as candidates, complete with design documents and source code. While many of these candidates were shown not to be cryptographically secure during the competition, 24 survived without any known cryptographic attacks. X11 chose eleven of these, including Keccak, the ultimate competition winner.

Another approach which has been proposed, but not actually implemented, is to have a mining puzzle that's a moving target. That is, the mining puzzle itself would change, just as the difficulty periodically changes in Bitcoin. Ideally, the puzzle would change in such a way that optimized mining hardware for the previous puzzle would no longer be useful for the new puzzle. It's unclear exactly how we would actually change the puzzle once every so often in order to obtain the security requirements we need. If the decision were to be made by the developers of an altcoin, it might be an unacceptable source of centralization. For example, the developers might choose a new puzzle for which they have already developed hardware (or just an optimized FPGA implementation), giving them an early advantage.

Perhaps the sequence of puzzles could be generated automatically, but this seems difficult as well. One idea might be to take a large set of hash functions (say, the 24 SHA-3 candidates which were not broken) and use each for six months to one year, too short of a time for hardware to be developed. Of course, if the schedule were known in advance, then the hardware could simply be designed just in time to ship for the time each function was being used.

The ASIC honeymoon. The lack of ASICs for X11 so far, even though they are clearly possible to build, demonstrates a potentially useful pattern. Because no altcoins using X11 have a particularly high market share, there simply hasn't been a large enough market for anybody to build ASICs for X11 yet. In general, designing ASICs has very high upfront costs (in both time and money) and relatively low marginal costs per unit of hardware produced. Thus, for new and unproven cryptocurrencies, it is not worth making an investment to build hardware if the currency might fail before the new hardware is available for mining. Even when there is a clear market, there is a time delay before hardware units will be ready. It took over a year for the first Bitcoin ASICs to be shipped from when they were first designed, and this was considered to be lightning fast for the hardware industry.

Thus, any new altcoin with a new mining puzzle is likely to experience an *ASIC honeymoon* during which time GPU and FGPA mining (and potentially CPU mining) will be profitable. It may not be possible to stem the tide of ASICs forever, but there is perhaps some value in making it appealing for individuals to participate in mining (and earn some units of the new currency) while it is bootstrapping.

Arguments against ASIC-resistance. We've seen that it may be impossible to achieve ASIC-resistance in the long run. There are also arguments that it is risky to move away from the relatively proven SHA-256 mining puzzle towards a new puzzle that might be weaker cryptographically. Furthermore, SHA-256 mining ASICs are already being designed at close to modern limits on hardware efficiency,

meaning the exponential growth period is probably over and SHA-256 mining will therefore offer the most stability to the network.

Finally, there is an argument that even in the short run ASIC-resistance is a bad feature to have. Recall from Chapter 3 that even if there is a 51% miner, many types of attack aren't rational for them to attempt because it could crash the exchange rate and decimate the value of the miner's investment in hardware since the bitcoins they earn from mining will be worth much less.

With a highly ASIC-resistant puzzle, this security argument might fall apart. For example, an attacker might be able to rent a huge amount of generic computing power temporarily (from a service such as Amazon's EC2), use it to attack, and then suffer no monetary consequences as they no longer need to rent the capacity after the attack. By contrast, with an "ASIC-friendly" puzzle, such an attacker would inherently need to control a large number of ASICs which are useful only for mining the cryptocurrency. Such an attacker would be maximally invested in the future success of the currency. Following this argument to its logical conclusion, to maximize security, perhaps mining puzzles should not only enable efficient mining ASICs to be be built, but be designed such that those ASICs are completely useless outside of the cryptocurrency!

8.3 Proof-Of-Useful-Work

In Chapter 5 we discussed how the energy consumed (some would say wasted) by Bitcoin mining, referred to as *negative externalities* by economists, is a potential concern. We estimated that Bitcoin mining consumes several hundred megawatts of power. The obvious question is whether there is some puzzle for which the work done to solve it provides some other benefit to society. This would amount to a form of recycling and could help increase political support for cryptocurrencies. Of course, this puzzle would still need to satisfy several basic requirements to make it suitable for use in a consensus protocol.

Previous distributed computing projects. The idea of using idle computers (or "spare cycles") for good is much older than Bitcoin. Table 8.3 lists a few of the most popular volunteer computing projects. All these projects have a property that might make them suitable for use as a computational puzzle: specifically, they involve some sort of a "needle in a haystack" problem where there is a large space of potential solutions and small portions of the search space can be checked relatively quickly and in parallel. For example, in SETI@home volunteers are given small portions of observed radio signals to scan for potential patterns, while in distributed.net volunteers are given a small range of potential secret keys to test.

Volunteer computing projects have succeeded by assigning small portions of the solution space to individuals for checking. In fact, this paradigm is so common that a specific library called BOINC (Berkeley Open Infrastructure for Network Computing) was developed to make it easy to parcel out small pieces of work for individuals to finish.

In these applications, volunteers were motivated mainly by interest in the underlying problem, though these projects also often use leaderboards for volunteers to show off how much computation they have contributed. This has led to some attempts to game the leaderboards by reporting work that wasn't actually finished, requiring some projects to resort to sending a small amount of redundant work to detect cheating. For use in a cryptocurrency, of course, the motivation is primarily monetary and we can expect participants to attempt to cheat as much as technically possible.

Project	Founded	Goal	Impact
Great Internet Mersenne Prime Search	1996	Finding large Mersenne primes	Found the new "largest prime number" twelve straight times, including $2^{57885161} - 1$
distributed.net	1997	Cryptographic brute-force demos	First successful public brute-force of a 64-bit cryptographic key
SETI@home	1999	Identifying signs of extraterrestrial life	Largest project to date with over 5 million participants
Folding@home	2000	Atomic-level simulations of protein folding	Greatest computing capacity of any volunteer computing project. More than 118 scientific papers.

Table 8.3: Popular "Volunteer computing" projects

Challenges in adapting useful-proof-of-work. Given the success of these projects, we might attempt to simply use these problems directly. For example, in the case of SETI@Home, where volunteers are given a segments of radio observations which they test for statistical anomalies, we might decide that statistical anomalies which are rarer than some threshold are considered "winning" solutions to the puzzle and allow any miner who finds one to create a block.

There are a few problems with this idea. First, note that potential solutions are not all equally likely to be a winning solution. Participants might realize that certain segments are more likely to produce anomalies than others. With a centralized project, participants are assigned work so all segments can be analyzed eventually (perhaps with more promising segments given priority). For mining, however, any miner can attempt any segment, meaning miners might flock to try the most likely segments first. This could mean the puzzle is not entirely progress-free, if faster miners know they can test the most promising segments first. Compare this to Bitcoin's puzzle, in which any nonce is equally likely to any other to produce a valid block, so all miners are incentivized to choose random nonces to try. The problem here demonstrates a key property of Bitcoin's puzzle that we previously took for granted, that of an *equiprobable solution space.*

Next, consider the problem that SETI@home has a fixed amount of data to analyze based on observations taken by radio telescopes. It's possible that with as mining power increased, there would be no more raw data to analyze. Compare this again to Bitcoin, in which an effectively infinite number of SHA-256 puzzles can be created. This reveals another important requirement: an *inexhaustible puzzle space* is needed.

Finally, consider that SETI@home uses a trusted, centralized set of administrators to curate the new radio data and determine what participants should be looking for. Again, since we are using our puzzle to build a consensus algorithm we can't assume a centralized party to manage the puzzle. Thus, we need a puzzle that can be *algorithmically generated*.

Which volunteer computing projects might be suitable as puzzles?. Returning to Figure 8.3, we can see that SETI@home and Folding@home clearly won't work for decentralized consensus protocol. Both probably lack all three properties we've now added to our list. The cryptographic brute-force problems taken on by distributed.net could work, although they are typically chosen in response to specific decryption challenges that have been set by companies looking to evaluate the security of certain algorithms. These can't be algorithmically generated. We can algorithmically generate decryption challenges to be broken by brute forcing, but in a sense this exactly what SHA-256 partial pre-image finding already does and it serves no beneficial function.

This leaves the Great Internet Mersenne Prime Search, which turns out to be close to workable. The challenges can be algorithmically generated (find a prime larger than the previous one) and the puzzle space is inexhaustible. In fact, its infinite, since it has been proven that there are infinite number of prime numbers (and an infinite number of Mersenne Primes in particular).

The only real drawback is that large Mersenne Primes take a long time to find and are very rare. In fact, the Great Internet Mersenne Prime Search has found only 14 Mersenne primes in over 18 years! It clearly wouldn't work to add less than one block per year to a blockchain. This specific problem appears to lack the adjustable difficulty property that we stated was essential in Section 8.1. It turns out, however, that a similar problem involving finding prime numbers appears workable as a computational puzzle.

Primecoin. As of this writing, the only proof-of-useful-work system deployed in practice is called Primecoin. The challenge in Primecoin is to find a *Cunningham chain* of prime numbers. A Cunningham chain is a sequence of k prime numbers $p_1, p_2, \ldots p_k$ such that $p_i = 2p_{i-1} + 1$ for each number in the chain. That is, you take a prime number, double it and add one to get another prime number, and continue until you get a composite number. The sequence 2, 5, 11, 23, 47 is a Cunningham chain of length 5. The potential sixth number in the chain, 95, is not prime ($95 = 5 \cdot 19$). The longest known Cunningham chain is of length 19 (starting at 7991019772166787018701610 1). It is conjectured and widely believed, but not proven, that there exist Cunningham chains of length k for any k.

Now, to turn this into computational puzzle, we need three parameters *m*, *n*, and *k* which we will explain momentarily. For a given challenge *x* (the hash of the previous block), we take the first *m* bits of *x* and consider any chain of length *k* or greater in which the first prime in the chain is an *n*-bit prime and has the same *m* leading bits as *x* to be a valid solution. Note that we can adjust *n* and *k* to make the puzzle harder. Increasing *k* (the required chain length) makes the problem exponentially harder, while increasing *n* (the size of the starting prime) makes it linearly harder. This provides fine-tuning of the difficulty. The value of *m* just needs to be large enough that trying to pre-compute solutions before seeing the value of the previous block is infeasible.

All of the other properties we have discussed appear to be provided: solutions are relatively quick to verify, the problem is progress-free, the problem space is infinite (assuming some well-studied mathematical conjectures about the distribution of prime numbers are true), and puzzles can be algorithmically generated. Indeed, this puzzle has been in use for Primecoin for almost two years and this has produced the largest-known primes in Cunningham chains for many values of *k*.

This provides strong evidence that it is possible to make proof-of-useful-work practical in some limited circumstances. Of course, it's debatable the extent to which finding large Cunningham chains is useful. It's possible that they may have some applied purpose in the future and they certainly stand as a small contribution to our collective mathematical knowledge. Currently, however, they have no known practical applications.

Permacoin and proof-of-storage. A different approach to proof-of-useful work is *proof-of-storage* (also sometimes called *proof-of-retrievability*). Rather than requiring a solely computational puzzle, what if we could design a puzzle that required storing a large amount of data to compute? If this data were useful, then miners' investment in mining hardware would effectively be contributing to a widely distributed and replicated archival storage system.

We'll take a look at *Permacoin*, the first proposal for proof-of-storage for use in consensus. We begin with a large file which we'll call *F*. For now, let's assume everybody agrees on the value of *F* and the file will not change. For example, *F* might be chosen by a trusted dealer when a cryptocurrency is launched, much as any new currency needs to agree on a genesis block to get going. This would ideally be a file of public value. For example, experimental data collected from the Large Hadron Collider already consists of several hundred petabytes (PB). Providing a free backup to this data would be quite useful.

Of course, since *F* is a huge file most participants will not be able to store the entire file. But we already know how to use cryptographic hash functions to ensure everybody agrees on *F* without knowing the entire thing. The simplest approach would be for everybody to agree on *H(F)*, but a better approach is to represent *F* using a large Merkle tree and have all participants agree on the value of the root. Now, everybody can agree on the value of *F* and it is efficient to prove that any portion of *F* is correct.

In Permacoin, each miner M stores a random subset $F_M \subseteq F$. To achieve this, when the miner generates a public key K_M which they will use to receive funds, they hash their public key to generate a pseudorandom set of blocks F_M which they must store to be able to mine. This subset will be of some fixed number of blocks k_1. We have to assume here that there is some way for them to fetch those blocks when they start mining — perhaps downloading them from a canonical source.

Once the miner has stored F_M locally, the puzzle is fairly similar to conventional SHA-256 mining. Given a previous block hash x, the miner chooses a random nonce value n and hashes this to generate a pseudorandom subset $F_{M,n} \subseteq F_M$ consisting of $k_2 < k_1$ blocks. Note that this subset depends both on the nonce they have chosen and their public key. Finally, the miner computers a SHA-256 hash of n and the blocks in F_k. If the value of this hash is below a target difficulty, they have found a valid solution.

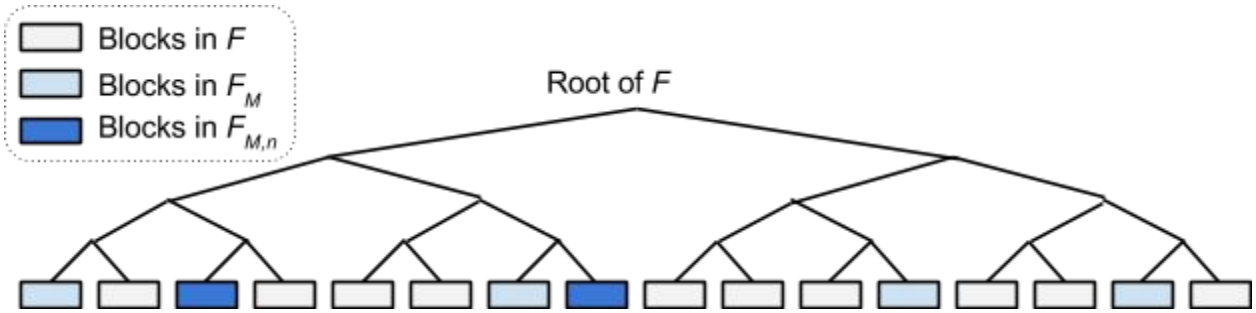

Figure 8.4: Choosing random blocks in a file in Permacoin.
In this example k_1=6 and k_2=2. In a real implementation these parameters would be much larger.

Verifying a solution requires the following steps:

- Verify that $F_{M,n}$ was correctly generated from the miner's public key K_M and nonce n
- Verify that each block of of $F_{M,n}$ is correct by verifying their path in the Merkle tree to the globally-agreed upon root of F.
- Verify that H($F_{M,n}$ // n) is less than the target difficulty.

It should be easy to see why solving the puzzle requires the miner to store all of $F_{M,n}$ locally. For each nonce, the miner needs to test the hash of a random subset of blocks of $F_{M,n}$, which would be prohibitively slow to fetch over the network from remote storage.

Unlike the case with scrypt, there are no reasonable time/memory trade-offs provided that k_2 is big enough. If a miner stored only half of F_M locally, and k_2=20, they'd have to try a million nonces before they found one that didn't require any blocks to be fetched over the network. So decreasing their storage burden by a constant factor increases their computational burden exponentially. Of course, setting k_2 to be too large will not be very efficient, since k_2 Merkle tree paths must be transmitted and verified in any valid solution.

There is also a trade-off in setting k_1. The smaller k_1 is, the less storage is needed to function as a miner and hence mining is more democratic. However, this also means larger miners have no incentive to store more than k_1 blocks of *F*, even if they have the ability to store more.

As usual, this is a slight simplification of the full Permacoin proposal, but this is enough to understand the key design components. The biggest practical challenge, of course, is finding a suitably large file that is important, public and in need of additional replication. There are also significant complexities if the file *F* changes over time, as well as with adjusting the mining difficulty over time.

Long-term challenges and economics. To summarize this section, proof-of-useful-work is a very natural goal, but it is quite challenging to achieve it given the other requirements of a good computational puzzle for a consensus protocol. Although at least two examples are known which are technically feasible, Primecoin and Permacoin, both carry some technical drawbacks (primarily longer verification time of purported solutions). Furthermore, both provide fairly minor public benefits compared to the scale of effort we've seen levied at Bitcoin mining with millions of dollars worth of capital and megawatts of electricity consumed.

There is an interesting economic argument that the benefit of any proof-of-useful-work should be a pure *public good*. In economics, a public good is one that is non-excludable, meaning nobody can be prevented from using it, and non-rivalrous, meaning the good's use by others does not affect its value. The classic example is a lighthouse.

Some of the examples we discussed here, such as protein folding, might not be a pure public good because some firms (such as large pharmaceutical corporations) may benefit more from increased knowledge about protein folding than others. Essentially, mining would be cheaper for these parties since they are gaining more benefit from the public benefits than others would be.

8.4 Nonoutsourceable Puzzles

Let's turn to another potential design goal for alternative mining puzzles: preventing the formation of mining pools. As we discussed in Chapter 5 and elsewhere, most Bitcoin miners mine as part of a pool rather than independently. This has resulted in a few large pools which together represent most of the mining power. Since each pool is operated by a central pool administrator, some feel this is a dangerous trend away from Bitcoin's core design principle of decentralization and can compromise its security.

While a mining pool with a majority share is an obvious problem, any large centrally managed pool might implement a non-default mining strategy and attack the network. Such pools are also a juicy target for hackers to try and compromise to immediately control a large amount of mining power. The pool operators might collude to censor transactions or enforce high transaction fees. At the very least, having most miners in pools also means that most miners aren't running a fully validating node.

Interestingly, these concerns have an analogy in the realm of voting. It's illegal in the United States and many other nations, for individuals to sell their vote. Arguably participating in a pool controlled by someone else is akin to selling your vote in the Bitcoin consensus protocol.

Technical requirements for pools. Recall that mining pools appear to be an emergent phenomenon. There's no evidence that Satoshi was thinking of mining pools at the time of Bitcoin's original design. It wasn't apparent for a few years that efficient pools could be run between many individuals who don't know or trust each other.

As we saw in Chapter 5, mining pools work by designating a typically a pool operator with a well-known public key. Each of the participating miners mines as usual but sends in shares to the pool operator. These shares are "near misses" or "partial solutions" which would be valid solutions at a lower difficulty level. This shows the pool operator how much work the miner is performing. Whenever one of the pool participants finds a valid block, the pool operator then distributes the rewards amongst the pool participants based on the number of shares they have submitted. As we discussed in Chapter 5, there are many formulas for dividing the revenue up, but all mining pools follow this basic structure.

The existence of pools thus relies on at least two technical properties of Bitcoin. The first is that it's easy for a miner to prove (probabilistically) how much work they are doing by submitting shares. By choosing a low enough threshold for shares, miners can easily prove how much work they are performing with arbitrary precision regardless of the actual difficulty of finding an valid block. This facet of mining puzzles appears difficult to change, given that we need a puzzle that can be created with arbitrary difficulty.

Second, pool members can easily prove to the pool operator that they're following the rules and working to find valid blocks which would reward the pool as a whole. This works because the pool's public key is committed to in the coinbase transaction included in the block's Merkle tree of transactions. Once a miner finds a block or even a share, they can't change which public key is the recipient of the newly minted coins.

Block discarding attacks. There is one weakness in this scheme for implementing mining pools: there is nothing to to enforce that participating miners actually submit valid blocks to the pool manager in the event that they find them. Suppose that there's a pool member that's upset with a large mining pool. They can participate in the pool by mining and submitting shares just like normal, but in the event that they actually find a valid block that would reward the pool they simply discard it and don't tell the pool operator about it.

This attack reduces the pool's overall mining power as none of the attacker's work is contributing towards finding valid blocks. However the the attacker will still be rewarded as they appear to be submitting valid shares and simply getting unlucky to not find any valid blocks. If the mining pool is designed to be revenue-neutral (that is, all mining rewards are redistributed back to participants) then this attack can cause the pool to run at a loss.

This attack is sometimes called a *vigilante* or *sabotage* attack and is considered a form of vandalism because the attack appears to be costly for both the attacker and the pool. The attacker loses money because every block they discard would have led to some proportion of the block rewards being returned to them. Of course, the attacker still gets rewards for other puzzle solutions that are found.

It appears that a rational attacker wouldn't employ this strategy, since they would lose money without gaining anything tangible. It turns out (quite surprisingly) that there are cases where this strategy can be profitable, as discussed in the box below. But in any case, we want to design an entirely new mining puzzle formulation that ensures this strategy is always profitable.

Sidebar: block discarding attacks between pools. People assumed for years that it can't be profitable for a participant to discard valid blocks found on behalf of the pool. It turns out this strategy can be profitable if one mining pool uses it to attack another. This was proposed apocryphally many times and first thoroughly analyzed in a paper by Ittay Eyal in 2015.

Let's consider a simple case: suppose two mining pools, A and B, each have 50% of the total mining capacity. Now suppose B uses half of its mining power (25% of the total capacity) to mine as a member in pool A, but discards all blocks found. We can show, in a simplified model, that B will now earns 5/9 of the total rewards, greater than the 50% it would earn by mining normally. In this simple case, dedicating half of its mining power to attacking can be shown to the optimal strategy for pool B.
The situation grows more complicated with multiple pools. Block discarding has not been observed in practice on a large scale as of this writing. But it remains possible that in the long run, attacks like this one will throw the viability of large mining pools into question.

Rewarding sabotage. Our design goal is to make it so that miners are incentivized to mine in a pool but not submit valid blocks to the pool manager. Currently, only the pool manager can collect the mining rewards because the manager requires all participants to include a specific public key in the coinbase transaction of blocks they are mining. Proper inclusion can be easily checked in submitted partial solutions. The pool manager is the only party that knows the private key and hence can determine where the newly minted coins go.

But what if we required that all participants also knew the private key (and hence could redirect the funds after mining a block?). To do this, we need a puzzle in which each solution attempt requires knowledge of the private key in the coinbase transaction. We can change the puzzle from "find a block whose hash is below a certain target" to "find a block for which the hash of *a signature* on the block is below a certain target." This signature must be computed using the same public key in the coinbase transaction.

Such a puzzle leaves would-be pool operators with two untenable choices. They might distribute the private key to all pool participants, in which case any of them can steal all of the funds. Alternately, they can perform the signatures on behalf of pool participants. Computing a signature is orders of

magnitude more expensive than computing a hash, however, so in this case the pool manager would be doing the majority of the heavy lifting. It would be better for the pool manager to simply be a solo miner.

The pros and cons non-outsourceable mining. Since this puzzle can't effectively be outsourced to an untrusted participant, it makes it much more challenging, if not outright impossible, to form a mining pool with untrusted participants. It effectively prevents *all* pools, even efforts like P2Pool to make a decentralized pool without no pool manager.

There's an argument that deploying such a puzzle might perversely lead to *more* centralization, not less, because it would discourage small miners from participating due to the high variance they would face. This would leave only large mining operations. Currently, while pools may nominally control a large amount of mining power, it isn't clear that they can use this to launch an attack without seeing many of their members defect. It remains an open question which risk is worse — that of large mining pools, or of limiting mining to operators large enough to live with a high variance.

The holy grail would be to design a consensus protocol which is "naturally" low-variance by rewarding miners a small amount for lower-difficulty puzzles. This would mean miners don't need to form pools and yet small miners may still participate. Simply decreasing the average time between blocks won't work — it would need to be decreased by a factor of 1,000 or more for the resulting variance to be equivalent to today's large mining pools. But then the delay between blocks would be less than a second and the number of stale blocks would be chaotically high. It remains an open question if there is an alternate version of the consensus protocol which would enable easier mining puzzles without requiring near-instantaneous broadcast of all solutions.

8.5 Proof-of-Stake and Virtual Mining

To wrap up this chapter, let's look at the idea of replacing computational puzzles with *virtual mining*. This term refers to a disparate set of approaches but they all have in common that they require only a small expenditure of computational resources by participating miners.

Closing the loop on mining. As a thought experiment, suppose Bitcoin or another cryptocurrency becomes the dominant form of payment globally. Miners would start with some initial holding of cryptocurrency, use it to purchase mining equipment and electricity, consume these resources, and in the process, acquire new cryptocurrency in the form of mining rewards. This process continually burns

energy and raw materials.

Figure 8.5: The cycle of Bitcoin mining

Once mining hardware becomes a commodity and electricity is a commodity (as it generally already is), no miner would have a significant advantage over any other miner in terms of how efficiently they could convert their initial cryptocurrency holdings into mining rewards. Barring minor variations in efficiency, whoever invests the most into mining will receive the most rewards.

The basic question motivating virtual mining is: what would happen if we removed the step of spending money on power and equipment? After all, this process is primarily used to prove who has invested the most in mining. Why not simply allocate mining "power" directly to all currency holders in proportion to how much currency they actually hold?

Recall that the original goal of Bitcoin mining was to enable a form of voting on the state of the blockchain, with miners with more computing power gaining more votes. We could instead design our "voting" system so that votes are determined by how much currency one currently holds.

Advantages of virtual mining. The primary advantage of this approach is obvious: it removes the wasteful right half of the mining cycle from Figure 8.5, leaving us with a "closed" system as shown in Figure 8.6.

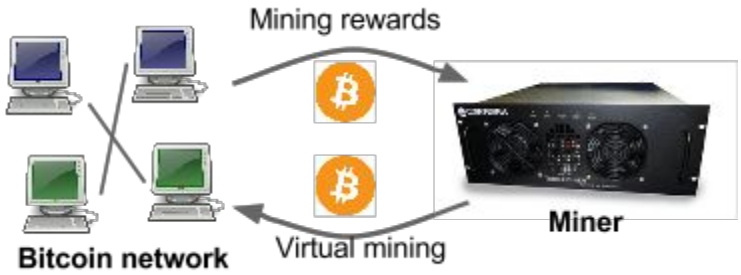

Figure 8.6: The virtual mining cycle

In addition to simplicity, this approach would dramatically reduce Bitcoin's environmental footprint. It wouldn't reduce energy consumption to zero, because miners will always have to expend some

computational resources to communicate with the network and validate. Some virtual mining schemes also require a small amount of computational mining as well. But in either case, the vast majority of the mining work performed in Bitcoin can potentially be eliminated.

Virtual mining may also reduce the trend towards centralization. Because there is no mining hardware involved there is no concern about an ASIC advantage; all miners are able to mine as "efficiently" as all others. Any virtual mining puzzle achieves all of the goals of ASIC-resistant puzzles.

Perhaps most importantly, virtual mining might solve the problem which we discussed in the context of ASIC-resistant puzzles, namely that miners may not be invested in the long-term health of the currency. Anybody who holds any bitcoins is effectively a stakeholder in the currency, and a powerful virtual miner (such as one who holds 51% or more of all currency) is a very large stakeholder. They have an incentive to do things that would benefit the system as a whole because it increases the value of the coins that they hold. This argument is even stronger than the argument that a miner sitting on a large stock of mining equipment whose value depends on the future of the currency will not behave maliciously.

This is where the term *proof-of-stake* comes from. Even more than eliminating mining and saving energy, perhaps the most fundamental motivation for virtual mining is to ensure that mining is done by stakeholders in the currency who have the strongest incentives to be good stewards of the system.

Implementing virtual mining: Peercoin. There are many variations of virtual mining of which we'll describe a few of the most common ideas. We should emphasize that these ideas have not yet been studied in a scientific and rigorous way, nor have they undergone the level of practical testing that proof-of-work has due to Bitcoin's popularity.

To start with, we'll consider the approach taken by Peercoin, which was launched in 2012 as the first altcoin using proof-of-stake. Peercoin is a hybrid proof-of-work/proof-of-stake algorithm in which "stake" is denominated by "coin-age." The coin-age of a specific unspent transaction output is the product of the amount held by that output and the number of blocks that output has remained unspent. Now, to mine a block in Peercoin a miner must solve a SHA-256 based computational puzzle just like in Bitcoin. However, the difficulty of this puzzle is adjusted down based on how much coin-age they are willing to consume. To do this, the block includes a special "coinstake" transaction in which some transactions are spent simply to reset their coin-age to zero. The sum of the coin-ages consumed in the coinstake transaction decides how difficult the proof-of-work puzzle is to make a given block valid.

It is possible for miners to mine with very little stake and a large amount of computational power, but the difficulty formula is chosen to make it dramatically easier to find a block if some coin-age is consumed. The effect of the computational puzzle is mainly to ensure that the process is randomized if two miners attempt to consume a similar quantity of coin-age.

Many virtual mining altcoins have adopted slightly different designs, including Nxt, BitShares, BlackCoin and Reddcoin. In each of these, some amount of stake is used to make a computational puzzle vastly easier, purportedly to the point that the computational puzzle is no longer the main challenge in mining.

Alternate forms of stake. A couple of alternatives to this hybrid model that are worth discussing:

- **Proof-of-stake.** The purest form of proof-of-stake is simply to make mining easier for those who can show they control a large amount of currency. This is similar to Peercoin's proof-of-coin-age, only with age not taken into account. The downside of this approach is that unlike coin-age which resets after successful mining, the richest participants are always given the easiest mining puzzle.
- **Proof-of-deposit**. In this formulation, when coins are used by a miner to mint a block, they become frozen for a set number of blocks. This can be thought of as a mirror of coin-age: instead of rewarding a miner for holding coins which have been unspent for a long time in the past, this system rewards miners who are willing to have coins go unmoved for a long time into the future. In both approaches, miners' stake effectively comes from the opportunity cost of not being able to use the coins to perform other actions.

The nothing-at-stake problem. Virtual mining is an active area of ongoing research and there are large open problems. While a few cryptocurrencies have launched and survived using virtual mining, they have faced the same pressure as Bitcoin to withstand motivated attackers.

The generic vulnerability of virtual mining schemes is what's often called the *nothing-at-stake* problem or *stake-grinding attacks.* Suppose an attacker with a proportion $\alpha < 50\%$ of the stake is attempting to create a fork of *k* blocks. As we've discussed previously, this attack will fail with high probability that is exponentially increasing in *k*. In traditional mining, a failed attack has a significant opportunity cost, because that miner could have been earning mining rewards during the mining process instead of wasting mining resources on its failed attack.

With virtual mining, this opportunity cost doesn't exist. A miner can use their stake to mine in the current longest chain while simultaneously attempting to create a fork. If their fork succeeds, it will have consumed a large amount of their stake. If it fails, the record of it failing will not be reflected on the eventual longest chain.

Thus, rational miners might be constantly attempting to fork the chain. Various attempts have been made to address this issue. Most virtual mining schemes have been much more aggressive about using checkpointing to prevent long forks, but as discussed previously, this is a bit of an end-run around a decentralized consensus protocol.

For Ethereum (an altcoin launched in mid 2015 that we will discuss later), a proposal called Slasher allows punishment for miners who attempt to fork the chain. In Slasher, using stake to mine requires signing the current block with the private key corresponding to the transactions making up the miner's stake. If a miner ever uses the same stake to sign two inconsistent chains (neither of which is

a prefix of the other), Slasher allows miners to enter these two signatures later on in the blockchain as proof of misbehavior and collect a portion of this stake as a bounty. While this appears to provide an effective solution, the details of the protocol are quite complicated and it has yet to be deployed successfully.

A final countermeasure that may exist is that, as we've seen for traditional mining schemes, miners may simply not have a strong incentive to attack because this would damage the system and undermine their stake, even if the attack is successful.

Other drawbacks of virtual mining. Two other drawbacks are worth quickly mentioning. The first is that in some forms of virtual mining, even in the absence of stake-grinding, might make some types of attacks easier because it is possible to "save up" for a burst of mining power. For example, a large amount of coin-stake can be pooled to enable a dramatic surge of mining to, perhaps, introduce a fork. This is possible even if a system like Slasher is used to discourage mining on two chains at once. To discourage this type of attack, Peercoin limits the age parameter to 90 days when computing coin-age.

A second issue is that if a miner in a virtual mining system obtains 51% of the available stake, they can maintain it forever by only mining on top of their own blocks, essentially taking control of the block chain. Even if new stake emerges from mining rewards and transaction fees, the 51% miner will obtain this new stake and their share of the total stake will slowly approach 100%. In traditional mining, even if a 51% miner exists it is always possible that some new miner will emerge with more mining equipment and energy and reduce the majority miner. With virtual mining, it is much more difficult to avoid this problem.

Can virtual mining actually work? Virtual mining remains somewhat controversial in the mainstream Bitcoin community. There is an argument that security fundamentally requires burning real resources, requiring real computational hardware and expending real electrical power in order to find puzzle solutions. If this argument is believed, then the apparent waste of the proof of work system can be interpreted as the cost of the security that you get. But this argument hasn't been proven, just as the security of virtual mining hasn't been proven.

Further Reading

The paper that defines memory-hard functions and proposes scrypt:

Percival, Colin. Stronger key derivation via sequential memory-hard functions. Self-published, 2009.

Earlier papers on memory-bound functions:

Abadi, Martin, Mike Burrows, Mark Manasse, and Ted Wobber. Moderately hard, memory-bound functions. ACM Transactions on Internet Technology, 2005.

Dwork, Cynthia, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In Advances in Cryptology—Crypto, 2003.

The Cuckoo Cycle proposal:

Tromp, John. Cuckoo Cycle: a memory-hard proof-of-work system. IACR Cryptology ePrint Archive 2014.

The Permacoin proposal:

Miller, Andrew, Ari Juels, Elaine Shi, Bryan Parno, and Justin Katz. Permacoin: Repurposing bitcoin work for data preservation. In IEEE Security and Privacy, 2014.

This paper discusses different hash function designs and the SHA-3 contest:

Preneel, Bart. The first 30 years of cryptographic hash functions and the NIST SHA-3 competition. In Topics in Cryptology — CT-RSA, 2010.

The proposal for non-outsourceable puzzles:

Miller, Andrew, Elaine Shi, Ahmed Kosba, and Jonathan Katz. Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions. Pre-print 2015.