Bitcoin and Cryptocurrency Technologies

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

Draft — Sep 2, 2015

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

Chapter 9: Bitcoin as a Platform

In earlier chapters we developed the technical underpinnings of Bitcoin and saw how it can be used as a currency. Now we'll look at applications *other than currency* that we can build using Bitcoin as a central component. Some of these rely on Bitcoin as it is today, without any modifications, and many others would require only small modifications.

We've chosen these applications for a combination of practical usefulness and intellectual interest. This list is not in any way exhaustive, but seeing how these applications work (or could work, since many are only ideas or proposals) will give you insight into the many ways in which Bitcoin's functionality can be repurposed.

9.1. Bitcoin as an Append-Only Log

It's helpful to think about Bitcoin as an *append-only log* — a data structure to which we can write new data, and once we've written data, it is tamper-proof and available forever. We also have a secure notion of ordering: we can tell if one piece of data was written to the log before or after another piece. This ordering arises from the block hash pointers, not the block timestamps — a block's timestamp can in fact be a lower (earlier) value than its predecessor. That's because miners can lie about timestamps, miners' clocks may not be synchronized, and there is latency on the network. That said, if a block timestamp appears to be off by more than a few hours, then other miners will reject it, so we can rely on the timestamps being approximately correct. As we'll see, these properties turn out to be quite useful.

Secure timestamping. The append-only log can be used to build a secure timestamping system from Bitcoin. We want to be able to prove that we know some value *x* at some specific time *T*. We might not want to actually *reveal x* at time *T*. Instead we only want to reveal *x* when we actually make the proof, which may be much later than *T* (and of course if we knew it at *T*, we still know it after *T* too). However, once we have made the proof, we want the evidence to be permanent.

Recall from Chapter 1 that we can use hash functions to commit to data. Instead of publishing the data *x* that we want to prove that we know, we can publish just the hash *H(x)* to the block chain. The properties of the hash function guarantee that we can't later find some different value *y* with the same value, $y \neq x$ such that $H(x) = H(y)$. We also rely on the convenient property that the hash of *x* doesn't reveal any information about *x*, as long as *x* is chosen from a distribution with high min-entropy, that is, it is sufficiently unpredictable. If *x* doesn't have this property, then we can pick a random number *r* with high min-entropy and use $H(r \mid x)$ as the commitment, as we saw in Chapter 1.

The main idea is that we can publish just the hash $H(r \mid x)$ at time T, and then at some point later on we can reveal *r* and *x*. Anybody can look at the append-only log and be convinced that we must have must have known *x* at the time we published *H(r | x)*, because there is no other feasible way to have generated that data.

Applications of timestamping. What could we do with this kind of secure timestamping? One idea is to prove prior knowledge of some idea. Suppose we wanted to prove that some invention we filed a patent on was actually in our heads much earlier. We could do this by publishing the hash of a design document or schematic when we first thought of the invention — without revealing to anybody what the idea is. Later on, when we file our patent or when we publicize the idea, we can publish the original documents and information and anybody can look backwards in time and confirm we must have known it earlier when we published the commitment to it.

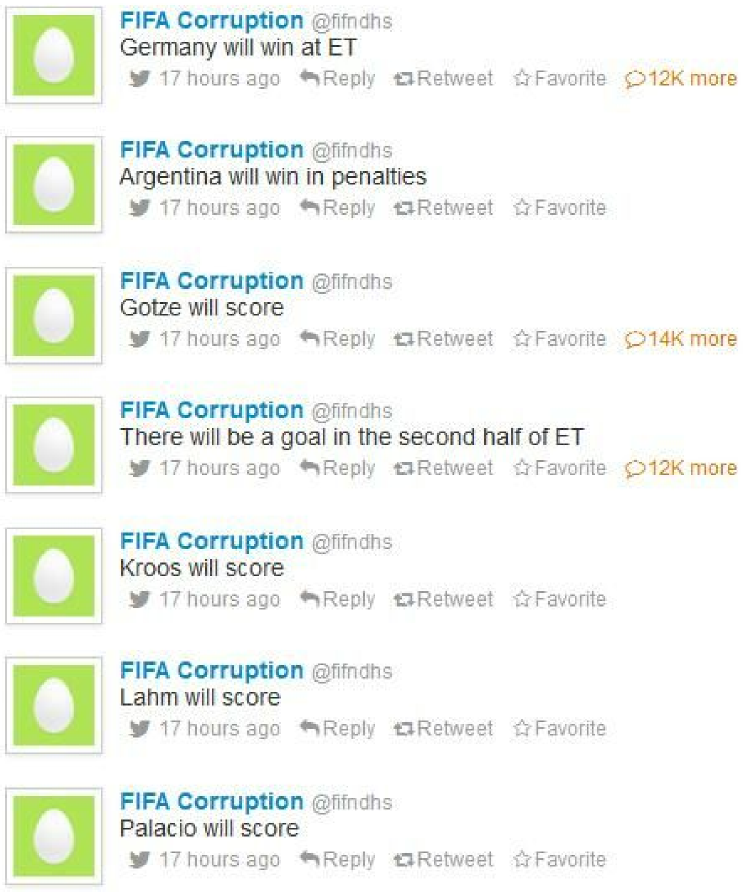
We can also prove that someone has else has received a message we sent them. Suppose Alice hires Bob to perform a programming job; their contract requires Bob to submit his work to Alice by a specific time. Both parties want to make sure that if there is a dispute later about whether Bob submitted the work or whether the code performed to specification, they have proof of what was submitted and when. To ensure this, they can mutually agree to publish a hash of Bob's submitted work signed by both parties. If either party later lies about what was submitted or when, the other party can prove them wrong (say, in a court of arbitration) by revealing the input to the hash.

Many other interesting things can be built just from secure timestamping. There's even an entire public-key signature scheme (called the Guy Fawkes signature scheme) that just uses hash functions and an append-only log. It doesn't require any of the heavy-weight cryptography that's usually used for public-key signatures.

Attacks on Proofs-of-"Clairvoyance". One thing that we can't do with secure timestamping alone — although it would be very nice if we could — is to prove *clairvoyance*, the ability to predict the future. It might seem that that is possible. The idea would be to publish a commitment to a description of an event that's about to occur (such as the outcome of a sporting event or of an election) and then later reveal that information to prove we predicted the event ahead of time. But does this work?

In late 2014, during the final game of the World Cup, someone used this method to "prove" that FIFA, the organization running the World Cup, was corrupt. After the match was over, it turned out that a Twitter account that had tweeted about several events that occurred during the game, timestamped *before the match even began*. For example, it tweeted that Germany would win in extra time and that Mario Götze would score. Seemingly this proves that either the owner of this Twitter account could tell the future or that the match was rigged. But in fact the account essentially tweeted *every possible outcome* before the match started. For every player involved in the match, there was a tweet predicting that he would score; there was a tweet for basically every possible final score of the game; and so on (see Figure 9.1). Before the match ended, they simply *deleted* all of their predictions that turned out to be false. Their Twitter stream was left with only true predictions.

This same basic attack can be performed against any secure timestamping system. You simply commit to a variety of possible outcomes, and then only reveal the commitments that turn out to be true. This means that if you actually *do* have the ability to predict the future and want to prove it, you must prove that you are timestamping *one specific prediction* rather than that multiple predictions. If you are publishing hash-based commitments, this is difficult to do. This is especially true in Bitcoin, since our secure timestamping system does not tie commitments to any individual's public identity. If you don't reveal them, it is easy to publish a large number of commitments and the ones you never reveal cannot easily be traced back to you.


FIFA Corruption @fifndhs
Germany will win at ET
17 hours ago Reply Retweet Favorite 12K more

FIFA Corruption @fifndhs
Argentina will win in penalties
17 hours ago Reply Retweet Favorite

FIFA Corruption @fifndhs
Gotze will score
17 hours ago Reply Retweet Favorite 14K more

FIFA Corruption @fifndhs
There will be a goal in the second half of ET
17 hours ago Reply Retweet Favorite 12K more

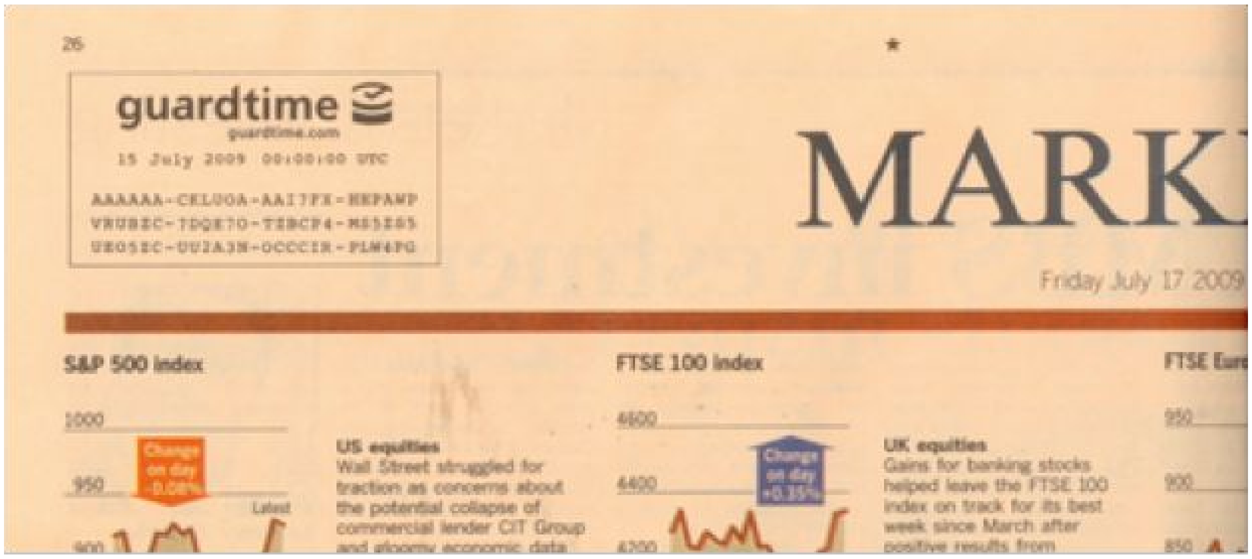
FIFA Corruption @fifndhs
Kroos will score
17 hours ago Reply Retweet Favorite

FIFA Corruption @fifndhs
Lahm will score
17 hours ago Reply Retweet Favorite

FIFA Corruption @fifndhs
Palacio will score
17 hours ago Reply Retweet Favorite

Figure 9.1: A Twitter account which attempted to "prove" that the 2014 FIFA Men's World Cup Final was rigged by "predicting" the outcome of the match. The first, third and fourth tweets ended up being true, the rest were deleted after the match.

Secure timestamping the old-fashioned way. Here's a simple low-tech way to do secure timestamping: publish the hash of your data in a newspaper, or some other media which is widely seen by the public, by purchasing an advertisement. Archives of old newspaper issues are maintained at libraries and online. This method provides a high degree of assurance that you knew that data on the day the newspaper was published. Later, when you want to reveal the data you committed, you can even take out a second advertisement to publish the data in the same newspaper.


26

guardtime
guardtime.com
15 July 2009 00:00:00 UTC
AAAAAA-CKLUOA-AAI7FX-HKPAWP
VRUBZC-TDQE7O-TEBCP4-MS5Z65
UE05ZC-UU2A3N-OCCCIR-PLW4PG

MARK

Friday July 17 2009

S&P 500 index
1000
950
900
Change on day -0.08%
Latest

US equities
Wall Street struggled for traction as concerns about the potential collapse of commercial lender CIT Group and gloomy economic data

FTSE 100 index
4600
4400
4200
Change on day +0.35%

UK equities
Gains for banking stocks helped leave the FTSE 100 index on track for its best week since March after positive results from

FTSE Eur
950
900
850

Figure 9.2: **A timestamping service (GuardTime) that publishes hashes in a daily newspaper rather than the Bitcoin block chain.** Customers of the company can pay to have their data included in the timestamp. Recall from Chapter 1 that we can use Merkle trees to encapsulate many pieces of data in a single hash and still efficiently prove that any one of those pieces of data is included in the hash.

Secure timestamping in Bitcoin. If we want to use Bitcoin instead of newspapers for timestamping, where should we place the hash commitment? Somewhere in a transaction? Or directly in a block?

The simplest solution (and the one people came up with first) is instead of sending money to the hash of a public key, just send it to the hash of your data. This "burns" those coins, that is, makes them unspendable and hence lost forever, since you don't know the private key corresponding to that address. To keep your cost down, you'd want to send a very small amount, such as one satoshi (the minimum possible transaction value in Bitcoin).

Although this approach is very simple, the need to burn coins is a disadvantage (although the amount burned is probably negligible compared to the transaction fees incurred). A bigger problem is that Bitcoin miners have no way to know that the transaction output is unspendable, so they must track it forever. The community frowns on this method for this reason.

A more sophisticated approach called CommitCoin allows you to encode your data into the *private* key. Recall that in Chapter 1, we said: "With ECDSA, a good source of randomness is essential because a bad source of randomness will likely leak your key. It makes intuitive sense that if you use bad randomness in generating a key, that the key that you generate will likely not be secure. But it's a quirk of ECDSA that, even if you use bad randomness just in making a signature, using your perfectly good key, that also will leak your private key."

CommitCoin exploits this property. We generate a new private key that encodes our commitment and we derive its corresponding public key. Then we send a tiny transaction (of, say, 2000 satoshi) to that address, and then send it back in two chunks of 1000 Satoshi each. Crucially, when sending it back we use the same randomness both times for signing the transaction. This allows anyone looking at the block chain to compute the private key, which contains the commitment, using the two signatures.

Compared to encoding your commitment in the public key, this CommitCoin avoids the need to burn coins and for miners to track an unspendable output forever. However, it is quite complex.

Unspendable outputs. As of 2014, the preferred way to do Bitcoin timestamping is with an OP_RETURN transaction which results in a provably unspendable output. The OP_RETURN instruction returns immediately with an error so that this script can never be run successfully, and the data you include is ignored. As we saw in Chapter 3, this can be used both as a proof of burn as well as to encode arbitrary data. As of 2015, OP_RETURN allows 80 bytes of data to be pushed, which is more than enough for a hash function output (32 bytes for SHA-256).

```
OP_RETURN <H(data)>
```

Figure 9.3: A provably “unspendable” transaction output script that embeds a data commitment

This method avoids bloat in the unspent transaction output set since miners will prune OP_RETURN outputs. The cost of such a commitment is essentially the cost of one transaction fee. Throughout 2014, a typical transaction fee is less than a penny. The cost can be reduced even further by using a single commitment for multiple values. As of late 2014 there are already several website services that help with this. They collect a bunch of commitments from different users and combine them into a large Merkle tree, publishing one unspendable output containing the Merkle tree root. This acts like a commitment for all the data that users wanted to timestamp that day.

Illicit Content. One downside of being able to write arbitrary data into the block chain is that people might abuse the feature. In most countries, it is illegal to possess and/or distribute some kinds of content, notably child pornography, and penalties can be severe. Copyright laws also restrict the distribution of some content.

Sure enough, several people have tried doing things like this to “grief” (i.e., to harass or annoy) the Bitcoin community. For example, there have been reports of links to pornography published in the Bitcoin block chain. The goal of these griefers is to make it dangerous to download the block chain onto your hard drive and to run a full node, since to do so might mean storing and transmitting material whose possession or dissemination is illegal.

There's no good way to prevent people from writing arbitrary data into the bitcoin block chain. One possible countermeasure is to only accept *Pay-to-Script-Hash* transactions. This would make it a little bit more expensive to write in arbitrary data, but it still wouldn’t prevent it outright.

Fortunately, the law is not an algorithm. It is tempting to try to “hack” the law by technical means to produce unexpected or unintended outcomes, but this not easy. Laws are intended to be interpreted by humans and incorporate factors such as intent. For example, U.S. Code 2252, the section of U.S. federal law that pertains to possession, distribution and receipt of child pornography, uses the wording “*knowingly* possesses, or *knowingly* accesses with intent to view” when describing prohibited activities (emphasis ours).

It is also worth noting that due to the size limitations we discussed above, data such as images (except, perhaps, tiny ones) cannot be directly written into the Bitcoin block chain. They will either have to be hosted externally, with only links written into the block chain, or be encoded in a cumbersome way across multiple transactions. Finally, most Bitcoin clients do not ship with the ability to decode and view data written into transactions, let alone data that’s encoded across multiple transactions.

Overlay Currencies. On the positive side, since we *can* write whatever data we want into Bitcoin, we can also build an entirely new currency system *on top of Bitcoin* without needing to develop a new consensus mechanism. We can simply use Bitcoin as it exists today as an append-only log, and write all of the data that we need for our new currency system directly into the Bitcoin block chain. We call this approach an “overlay currency”. Bitcoin serves as the underlying substrate, and the data of the overlay currency is written into the Bitcoin block chain using unspendable transaction outputs.

Of course, Bitcoin miners will not actually *validate* what you’re writing into the block chain, since they don’t know (and don’t care!) whether the data you write is valid under the rules of your new currency. Anyone can write anything in there who's willing to pay the Bitcoin transaction fees. Instead, you must develop more complicated logic for validating transactions in the new currency, and this logic must reside in each end-user client that participates in sending or receiving this currency.

For example, in an overlay currency miners are no longer able to reject double spends. Instead, every user of the overlay currency has to look at the history of what’s been written in the block chain. If an overlay transaction attempts to spend an overlay coin that has already been spent, then that second transaction should simply be ignored. For this reason, there’s no such thing as a lightweight SPV client for overlay currencies.

Counterparty is a prominent overlay currency. All Counterparty transactions are written into the Bitcoin block chain. During 2014, between 0.5% and 1% of all Bitcoin transactions carried Counterparty data. It also supports a much larger and richer feature set than Bitcoin. The idea is that since Counterparty doesn’t have to develop a new consensus algorithm, and since Bitcoin miners don't need to know about the Counterparty rules, they can instead focus on developing interesting features such as smart contracts, user defined currencies, and more. The Counterparty API can be much larger than the Bitcoin API since Bitcoin miners don't need to understand it or approve of it.

The potential to develop a new currency without having to create a new consensus system is very appealing. You don't even need to encourage new miners to join your system, and you can add new features without needing to change Bitcoin. However, such systems are still reliant on Bitcoin — for example, they are subject to the same fee requirements as other Bitcoin transactions. This approach can also be inefficient, since nodes on the overlay currency may need to process a lot of data, since Bitcoin nodes don't filter these transactions for you.

9.2 Bitcoins as "Smart Property"

Now we'll talk about using bitcoins to represent something other than a unit of currency in the Bitcoin system.

Recall from Chapter 6 that you can trace ownership of value in the Bitcoin system over time, simply by following the transaction graph. Keep in mind the caveat: there's no such thing as a "bitcoin" per se — just unspent transaction outputs, which we refer to as coins. Every bitcoin has a history that anybody can view in the block chain. A coin's history traces all the way back to one or more coinbase transactions in which coins were originally minted. As we said discussed earlier, this is bad for anonymity, since you can often track ownership of coins this way.

Fungibility**.** This also leads to an interesting observation: bitcoins aren't *fungible*. In economics, a fungible good is one where all individual units are equivalent and can be substituted for one another. For example, gold is fungible since one ounce of (pure) gold can be substituted for any other ounce of gold. But this isn't always true of Bitcoin because every bitcoin is unique and has a different history.

In many contexts this history may not matter, but if the history is meaningful to someone you want to trade with, it may mean be that your 1.0 bitcoin is not the same as their 1.0 bitcoin. Maybe they wouldn't be willing to exchange theirs with yours because they prefer the history of their coin than that of your coin. For example, just as coin collectors value old coins, some day bitcoin collectors might place special value on coins originating in the genesis block or some other early block in Bitcoin's history.

Smart Property. Could this non-fungibility property be *useful*? We've already seen why it can be bad for privacy because of the potential for deanonymizing users. In this section we'll look at why it can also be useful to give *meaning* to the history of a bitcoin.

Let's think about what it would mean to give meaning to the history of ordinary offline physical currency. Suppose we wanted to add metadata to offline currency. In fact, some people already do this. For example, they like to write various messages on banknotes, often as a joke or a political protest. This generally doesn't affect the value of the banknote, and is just a novelty.

But what if we could have *authenticated* metadata attached to our currency — metadata that cannot easily be duplicated? One way to achieve this is to include a cryptographic signature in the metadata we write, and tie this metadata to the *serial number* of the banknote.

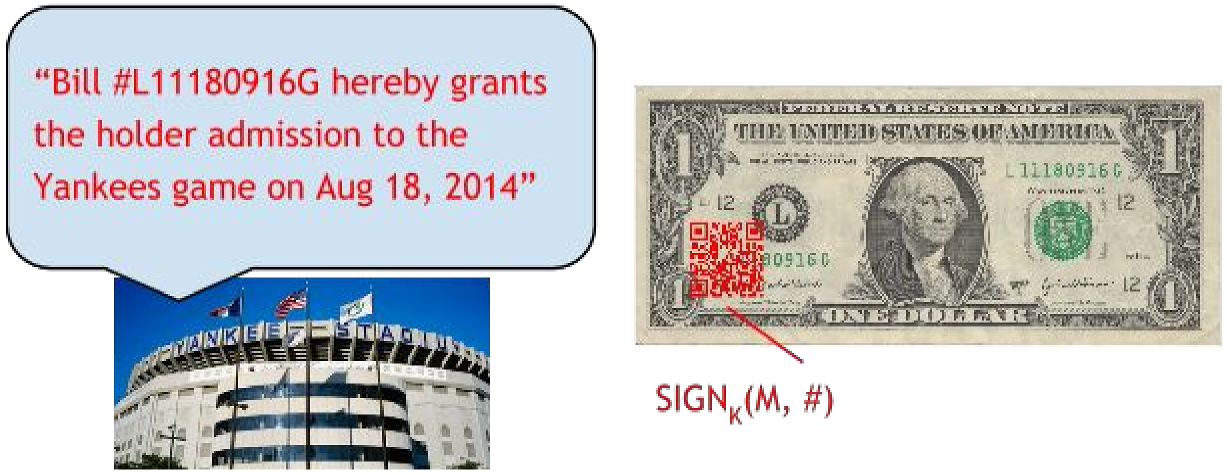

"Bill #L11180916G hereby grants the holder admission to the Yankees game on Aug 18, 2014"
$\mathrm{SIGN}_K(M, \#)$

Figure 9.4: An example of adding useful metadata to ordinary bank notes

What could this be used for? Say a baseball team wants to use dollar bills as tickets. This way they no longer have to go through the hassle of printing their own tickets and making sure that no one can print counterfeit tickets. The New York Yankees could simply assert that the dollar bill with a specific serial number now represents a ticket to a specific game, and in a specific seat. These dollar bills would be distributed in the same ways that paper tickets are normally distributed, such as by being mailed to fans when they buy tickets online. Whoever is holding that note has the right to enter the stadium, sit in the assigned seat, and watch the game, with no other questions asked. The banknote itself is the ticket!

To add authenticity, the Yankees could use digital signatures. They'd sign a message that includes the specific game date, the seat number, and the serial number of the bill — and stamp the message and the signature right on the bill. A 2D barcode would be a convenient form for that data (See Figure 9.4). Alternatively, the stadium could maintain a database that lists the serial numbers and corresponding seat numbers for each game. They could check the database for this information when you tried to enter the gate. This avoids the need to stamp the banknotes.

What does this buy us? Now currency can represent many things. Besides the example of a sports ticket, there are many other applications. We inherit the anti-counterfeiting property that banknotes already have. Law enforcement agencies work very hard to make sure it's difficult to duplicate a bank note! Also, the underlying currency value of the banknote is maintained. After the fan redeems their ticket, the banknote is perfectly usable as regular currency. It may be a problem if everybody wants to physically stamp metadata on currency, but this problem goes away if we use the database approach.

Of course, all of the useful meaning of this new metadata is only as good as our trust in the *issuer* who signed it. Someone must know that there's a specific key used to sign valid Yankees tickets — or download the Yankees' database — in order to recognize its value as a ticket. To anyone else, it would just look like a dollar bill. But that's okay. It's actually a desirable property, since once the ticket has fulfilled its purpose, it can go back into circulation as an ordinary dollar bill.

Colored coins**.** Can we do this digitally on top of Bitcoin? We'd like to keep Bitcoin's nice features such as the ability to transact online, fast transaction settlement, and non-reliance on a bank.

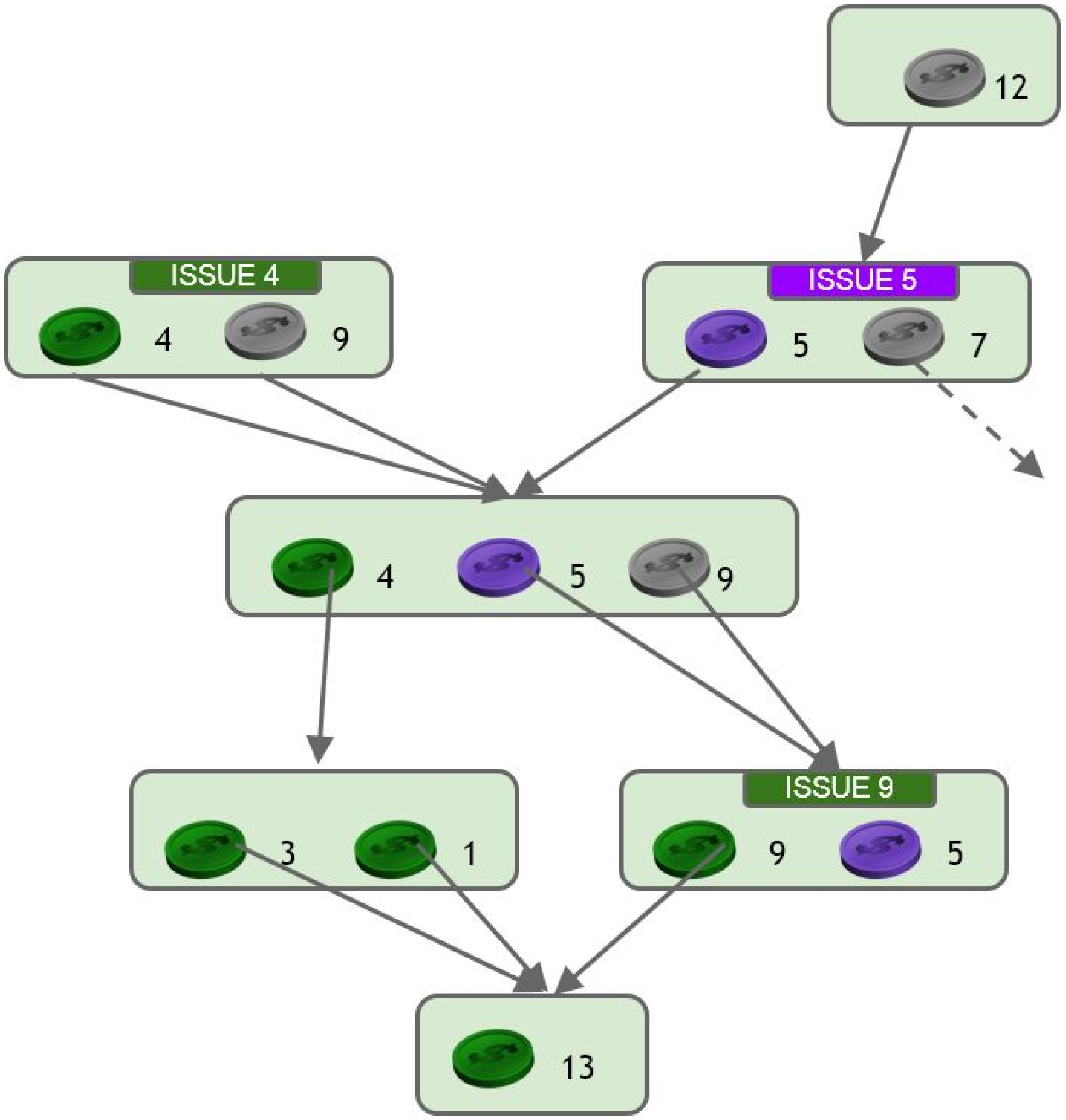


Figure 9.5: Colored coins. The transaction graph shown illustrates issuance and propagation of color

The main idea is to stamp some Bitcoins with a "color", and track that color stamp even as the coin changes hands, just like we are able to stamp metadata onto a physical currency. A bitcoin stamped with a color still functions as a valid bitcoin, but additionally carries this metadata.

To achieve this, in one transaction, called the "issuing" transaction, we'll insert some extra metadata that declares some of the outputs to have a specific color. An example is illustrated in Figure 9.5. In one transaction, we issue five "purple" bitcoins in one transaction output, while the other output continues to be normal uncolored bitcoins. Someone else, perhaps with a different signing key, issues "green" bitcoins in a different transaction. We call these colors for intuitiveness, but in practice colors are just bit strings. The only property that matters is that coins of the same color and same value are equivalent.

Now we have bitcoins with different colors associated with them. We can still do all the normal things we do with bitcoin transactions. We could have another bitcoin transaction that takes several inputs: some green coins, some purple coins, some uncolored coins, and shuffles them around. It can have some outputs that maintain the color. There may need to be some metadata included in the transaction to determine which color goes to which transaction output. We can split a transaction output of four green coins into two smaller green coins. Later on we could combine multiple green coins into one big green coin.

***OpenAssets*.** As of 2015, the most popular proposal for implementing this in Bitcoin is called OpenAssets. Assets are issued using a special Pay-to-Script-Hash address. If you want to issue colored coins, you first choose a P2SH address to use. Any coin that transfers through that address and comes in without a color will leave with the color designated by that address. For this to be meaningful, you'd have to publicize that address somewhere. There are various exchanges that track which addresses confer which colors onto coins. Since coins can sequentially pass through more than one color-issuing address, they can have more than one color, and that's fine.

Every time you have a transaction that involves colored coins, you have to insert a special marker output. This is a provably unspendable output, similar to what we used for timestamping data commitments. The metadata embedded in the marker output encodes details about how the incoming color value should be divided among the different outputs.

As we noted earlier, this is compatible with Bitcoin. Since it doesn't require changing Bitcoin, the community of miners tends not to discourage or interfere with these schemes. It allows anybody to declare any color they want without having to ask a central authority for the right to issue colored coins. If there are others who understand and abide by the meaning you ascribe to the color you issue, your colored coins may attain additional value beyond their nominal bitcoin value. For example, if the Yankees issue colored coins, these coins will be able to function as tickets to a game provided the stadium operators understand their meaning and let you in based on colored-coin tickets.

One disadvantage of this scheme are that we have to put in the unspendable marker output into every transaction. This adds a bit of overhead, since we must forfeit some money every time we want to trade a colored coin. A second disadvantage is that miners don't check the validity of colored coins, only the underlying bitcoins. To verify that a colored coin you receive is valid, you have to check the *entire transaction history* that the coin was involved in, or trust a third party to do the checking for you. In particular, you can’t use a thin SPV client like you can for regular Bitcoin. That makes it harder to use colored coins on computationally limited devices like mobile phones.

Uses of colored coins and smart property. *Stock in a Company.* A frequently cited motivation for smart property is stock in a company. A company wishing to issue colored coins as stock would publicize its issuing address, and bitcoins that are colored with this address function as shares. One satoshi might represent one share in the company. Shareholders can then trade the stock on the block chain without needing a centralized intermediary like a stock exchange. Of course, shareholders will have to trust that the company will honor the shares. For example, the company may promise to disburse dividends proportionally to each stock or to give shareholders voting power in the company’s decisions. With traditional shares these promises are enforced legally. As of 2015, colored coins or other block chain-based assets don’t have legal recognition in any jurisdiction.

Physical property. Another potential use is that colored coins might represent a claim to some real world property. For example, a colored coin could be associated with a house or a car. Maybe you have a sophisticated car that actually tracks a specific colored coin on the block chain, and automatically starts and drives for anybody who owns that colored coin. Then you could sell your car, or at least transfer control of it, simply by making a single transaction in the block chain. We’ll see in Chapter 11 how this can potentially be implemented technologically as well as the social and legal obstacles to making it happen. But the dream of colored coins and smart property is that any real-world property could be represented in the world of Bitcoin and transferred or traded as easily as bitcoins themselves.

Domain Names. As a final example, consider using colored coins to perform some of the functions of the existing Domain Name System: tracking the ownership and transfer internet domain names as well as the mapping of domain names to IP addresses. The domain name market has a variety of interesting properties: there are a potentially infinite number of names, these names have widely different values based on their memorability and other factors, and the same name might have very different utility to different people. It is possible to use colored coins to handle domain name registration and the functions we listed. However, supporting this application has also been the focus of a prominent altcoin called Namecoin, which we’ll look at in detail in the next chapter. Each approach has benefits: with colored coins you get the security of Bitcoin’s block chain whereas with an altcoin it’s easier to implement the complex logic needed for domain name ownership, transfer, and IP address mapping.

Lecture 9.3: Secure Multi-Party Lotteries in Bitcoin

Now we're going to talk about hosting a “coin flip” game in Bitcoin. Again, we'll start off by describing the offline version of what we're trying to build.

Alice and Bob want to bet five dollars. They both agree to the bet ahead of time and the method for determining the winner. Bob will flip a coin in the air, and while it’s rotating Alice calls out “Heads” or

“Tails”. When the coin lands, they both immediately have a clear understanding of who won the bet, and they both have assurance that the outcome was random and that neither of them was able to influence the outcome.

The sequence of steps in this ceremony as well as the physics of coin flipping play a crucial role in convincing both parties that the game is fair. One shortcoming of this scheme is that both parties have to be present at the same place at the same time. Also, both parties still have to trust that whoever loses will pay up. In the online world, we’d like to be able to have a lottery that is just as “fair”, but also solves the problem of making sure the loser pays.

At first this might seem like a rather peculiar and limited application to be studying in detail. Amusingly, Bitcoin-based betting services such as Satoshi Dice — which rely on a trusted party, unlike the system we’d like to design — have proven very popular, at times representing a large fraction of all Bitcoin transactions on the network.

The real reason we want to study cryptographic coin flipping, however, is that it turns out that if we can design a secure protocol for it, we can use those techniques to build many other interesting and useful protocols. Cryptographers study “secure multiparty computation” where two or more mutually untrusting parties each have some data and want to compute a result that depends on all of their data, but without revealing the data to each other. Think of a sealed-bid auction, but without a trusted auctioneer. Often, these computations need to be randomized, say, to break ties. Finally, we might want the result of the *computation* to determine a *monetary* outcome in an irrevocable way. Maybe we want to ensure that the winning bidder in the auction pays the seller; perhaps we even want to ensure that the seller’s (smart) property being auctioned is automatically transferred to the winning bidder. Alternatively, maybe we want to penalize parties if they deviate from the protocol.

In other words, a secure multi-party lottery is a simple setting in which to study an extraordinarily powerful paradigm: mutually untrusting participants with sensitive inputs jointly executing a program that has the power to manipulate not only bits, but also money.

Coin Flipping Online. The first challenge is replacing the “coin flip” mechanism with some online equivalent. Let’s say we now have three parties, Alice, Bob, and Carol, who all want to select a number 1, 2, or 3 with equal probability. Here’s one attempt at such a protocol. Each of them picks a large random number — Alice chooses x, Bob y, and Carol Z. They tell each other their numbers, and they compute the output as (x + y + z) % 3.

If all of them chose their random numbers independently, this would indeed work. But remember that we’re doing this over the internet, and there’s no way to insist that they all send their numbers “simultaneously”. Alice might wait until she hears Bob’s and Carol’s numbers before broadcasting hers. If she does this, you can see how it’s trivial for her to make the final output whatever she wants. We can’t design the protocol to convince every party that none of the other parties cheated.

To solve this problem we can once again use hash commitments. First, each of them picks a large random number and publishes a hash of this number. Once this is done, each of them reveals the number they picked. The others then check that the revealed numbers hash to the values published in the first step, and compute the final outcome from the three random numbers as shown in Figure 9.6.

Round 1:

Each party picks a large random string — Alice picks x, Bob picks y, and Carol picks z.
The parties publish H(x), H(y), H(z) respectively.
Each party checks that H(x), H(y), H(z) are all distinct values (otherwise aborts the protocol).

Round 2:

The three parties reveal their values, x, y, and z.
Each party checks that the revealed values agree with the hashes published in Round 1.
The outcome is (x + y + z) % 3.

Figure 9.6: Using hash commitments to implement a fair random number generator. This protocol can be easily extended to support any number of parties.

The reason this protocol works is twofold. First, since the hash inputs x, y, and z are large random numbers, no party can predict the others' inputs after the first round. Second, if (say) Alice chooses her input randomly as specified by the protocol, she can be sure that the final output will be random *regardless* of whether or not Bob and Carol choose their inputs randomly.

Fairness**.** What happens if somebody fails to reveal their commitment? In round 2 of the protocol, suppose Carol waits until Alice and Bob have revealed their secrets. Carol, before revealing hers, realizes that she's going to lose if she does. So she might refuse to publish her random number — she can claim to have forgotten it or pretend to go offline. Alice and Bob would likely be suspicious, but they would have no good recourse.

What we'd like is a scheme where whoever makes a commitment is forced to reveal it within some certain time limit. In cryptography, this property is called fairness. Bitcoin provides us with an excellent mechanism for this.

Let's say that Alice wants to make a timed commitment, and Bob is the only other person who is concerned with it. First, Alice puts up a bond, in the form of a Bitcoin transaction output script that specifies that can be spent in one of two ways. One way is with a signed transaction from both Alice and Bob. The other way to spend it is with a signature from just Alice, but only if she also reveals her random number. If Alice's random string is **`x`**, then the scriptPubkey actually contains the value **`H(x)`**.

Next, Alice and Bob both sign a transaction that pays the bond to Bob (which is one of the two ways it can be spent). Why would Alice agree to this? The transaction carries an ***nLockTime*** value that guarantees Bob can't claim the bond before some time **t**. Since Alice plans to reveal her committed value before then and recover the bond, it is safe for her sign this transaction.

Now if Alice leaves without revealing her value, Bob can claim the bond at time **t**. This doesn't *force* Alice to reveal her commitment but she *will* lose the entire bond that she put up. So the guarantee that she'll reveal her secret value depends on the amount of money she's willing to put in the bond.

```
scriptPubKey:
       OP_IF
              <AlicePubKey> OP_CHECKSIGVERIFY <BobPubKey> OP_CHECKSIG
       OP_ELSE
              <AlicePubKey> OP_CHECKSIGVERIFY OP_HASH <H(x)> OP_EQUAL
       OP_ENDIF

scriptSig for Case 1:
       <BobSignature> <AliceSignature> 0
scriptSig for Case 2:
       x <AliceSignature> 1
```

Figure 9.7: The transaction output scriptPubkey and scriptSigs used in a timed hash commitment.

How can we use this timed hash commitment to implement our secure lottery? We'll have almost the exact same structure as before, except instead of using the simple hash commitments, we'll used these timed commitments. Whoever does not reveal their random value before the deadline will forfeit a security deposit that's used to compensate the other two players. Revealing the random value is now simply a matter of recovering the bond by providing the correct secret input **x**.

This lottery scheme can be implemented on top of Bitcoin. But it's a bit complicated and the timed hash commitments require multiple non-standard transactions. When there are **n** parties in the lottery, $\mathbf{n^2}$ commitments are needed since each party needs to put up a bond for each other party. The players have to escrow more money in total than they are even betting. But it is reasonable for a small number of participants, and there are variants with better efficiency. Most importantly, it serves as an existence proof that seemingly impossible protocols such as flipping a virtual coin over the internet and penalizing a party for aborting the protocol are possible in the Bitcoin world.

Lecture 9.4: Bitcoin as Public Randomness Source

In the last section, we showed how a group of people can jointly choose a fair random value. In this section we'll talk about using Bitcoin to generate random values that are fair to *anyone* in the public.

Why would we want this? Let's discuss a few examples of applications that already rely on public sources of random values.

NBA draft lottery. One example that occurs every spring in the U.S. is the NBA draft lottery. All 30 teams in the NBA get together and randomly choose — with some weighting based on how each team performed in the previous season — the order in which teams get to select the top amateur players in the country who are ready to turn professional. This was first done in 1985. The lottery was conducted over live television, and involved picking envelopes after they're shuffled in a transparent spinning drum. This lottery generated a bit of controversy then, because the New York Knicks won in the first year and were able to draft the highly sought-after center Patrick Ewing (an eventual member of the Basketball Hall of Fame). Since the lottery was filmed in New York City, some fans of other teams alleged that the process was rigged in favor of the Knicks.

There are numerous conspiracy theories for how the NBA might have this process, such as the famous "bent corner" theory suggesting that the Knicks' envelope had its corner bent so the commissioner could distinguish it from the others by touch. Another theory suggests the Knicks' envelope was kept in a freezer and the commissioner simply grabbed the one cold envelope. These theories show why it is very hard to hold a drawing like this and prove that it was fair — there are many plausible avenues to cheat. Just think of what professional sleight-of-hand magicians can appear to do! Even today, this lottery occurs every year and each time it leads to a variety of conspiracy theories and rumors that the lottery isn't a fair random drawing.

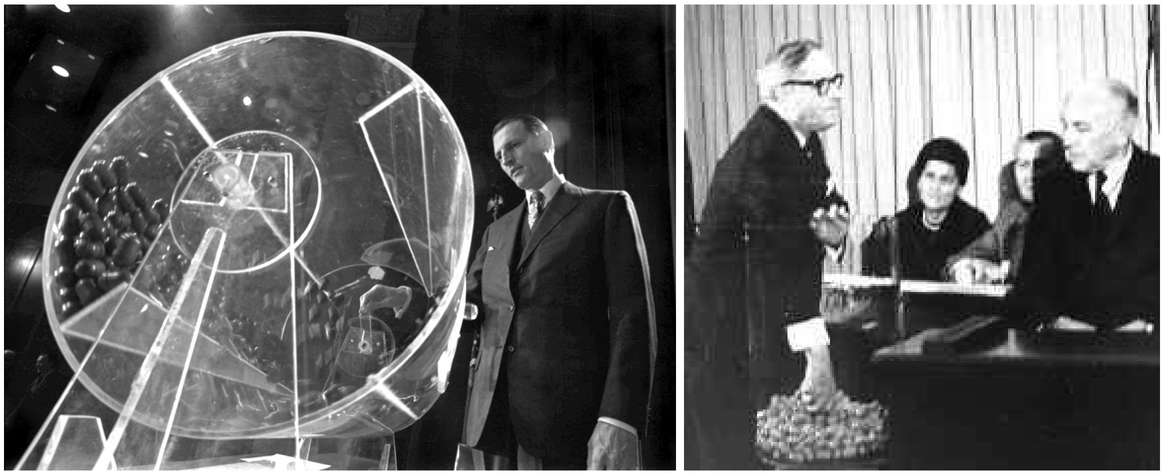


Figure 9.8: Images from the 1969 (Vietnam war) military draft lottery.

U.S. military draft lottery. A more serious example comes from 1969, when there was conscription lottery in the United States to determine which young men would be required to join the armed services. Most of them were sent to fight in the Vietnam War. A procedure similar to the NBA lottery was used, carried out by several representatives from the US Congress and broadcast on live television (Figure 9.8). They dumped small capsules labeled with each day of the year into a large plastic drum, and then took turns reaching in with their hands to pull the numbers out. Men eligible to be drafted were given a priority number based on the day of the year their birthday fell on. The priority number determined the order in which they would be drafted.

This 1969 draft was the first time this lottery procedure was used on a national scale. The goal was to make the process more fair (by taking it out of the hands of thousands of local draft boards) and to demonstrate to the public that it was a random process. Unfortunately, the lottery was botched. Within a week, statisticians looking at the data noticed an anomalous pattern (illustrated in Figure 9.9). Days late in the year received low draft numbers. Though the deviation is very subtle, it is statistically significant and very unlikely to have happened due to chance. When they went back to review the tapes, it turned out that they rotated the drum exactly an even number of times, such that the capsules that started out on top tended to still be on the top. There wasn't sufficient mixing to make it a statistically random draw.

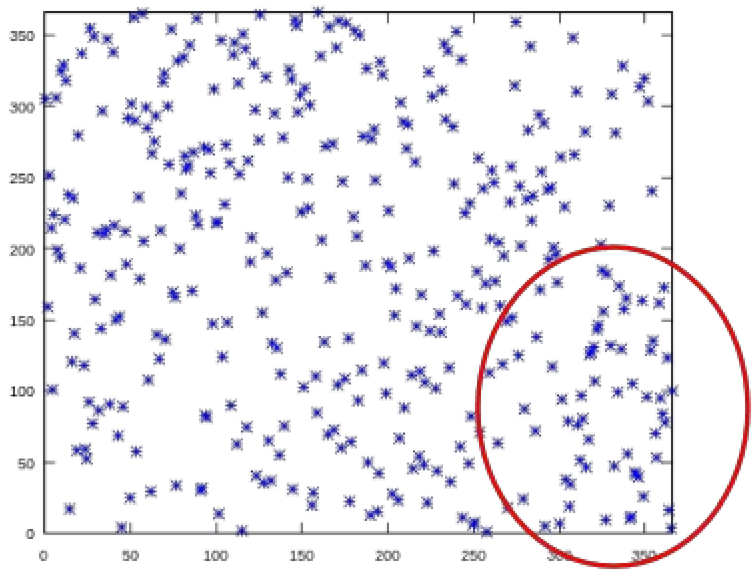


Figure 9.9: Statistical bias of the 1969 draft lottery. Day of the year (x-axis) versus lottery number (y-axis).

What both of those examples show is that it's very hard to generate public randomness and convince the public that you've actually done a good job. There's a risk that the process might not be truly random and free of influence. There's also a risk that even if the process *is* random, that the public won't believe you.

Cryptographic "Beacons". The public displays of randomness using a wheel, flipping coins, rolling a dice, and so on have been so popular throughout history because they're cheap and easy to understand. But they don't scale so well to large-scale scenarios because it's very hard for people to audit. Even if the video of the procedure appears legitimate, people may reasonably be suspicious that the lottery conductor has performed some sleight of hand to rig the process.

Could we do better cryptographically? Let’s use the term “cryptographic beacon” to refer to a service that provides a public source of randomness. The idea is that the beacon will continuously publish new random data at a regular rate that nobody can predict in advance. Hopefully everybody agrees that there’s no way for anyone to predict what the beacon will output next, so everybody can rely on it as a fair random value.

If a perfect cryptographic beacon existed, then it could be used for any of the public lotteries we mentioned. Even if you just wanted to play bingo at your local social club, you wouldn’t need to use a large drum of numbers. If everybody trusted the beacon, you would save a lot of effort from having to have these small physical displays of randomness.

Cryptographers have proposed many other applications of public randomness, including voting systems, zero-knowledge proofs, and cut-and-choose protocols. Many of these can be done much more simply and efficiently if you have a perfect cryptographic beacon. Unfortunately, we haven't found a perfect way to implement such a beacon yet.

***The NIST beacon*.** The National Institute of Standards and Technology (NIST) has, since 2011, run its own beacon service. They claim to generate their random numbers through a complicated laboratory setup involving two entangled protons. The idea is to provide strong guarantees that the numbers are random because by generating them from a quantum mechanical phenomenon. If you accept the Heisenberg Uncertainty Principle and other widely-believed laws of physics, then this should be truly random and unpredictable. The service is set up so that it produces new random data every sixty seconds along with a digital signature over the data. The NIST beacon provides a convenient interface for programmatic applications: the numbers can simply be read out from a web feed.

This quantum mechanical procedure is in some sense the “limit” for physical displays of randomness. But it does nothing to alleviate the essential problem of trust — you have to trust that NIST is in fact carrying out the procedure as they claim. You have to trust that somewhere in a building in Maryland NIST has their actual laboratory that produces these numbers and that they aren’t simply staging the procedure. You also have to believe that they aren’t reserving the ability to deliberately overwrite some of the random values before they publish them.

Other potential ways to build a beacon: natural phenomena. What about an alternate approach where we use some natural phenomenon that everybody can observe? Perhaps we could use details about the weather, such as what temperature it's going to be tomorrow at a specific place, or how strong the wind will be, or whether or not it will rain. Of course, we have some ability to predict the weather ahead of time, but not precisely, so perhaps we can use the “least significant bits” of the measured values. The limitation here is that all participants need to be at the same place to get the same measurements.

To avoid this we could turn to sun spots, which are bursts of activity on the surface of the sun. Yet another example is cosmic background radiation, which is noise that you can listen to with a radio antenna from any point on the planet; everybody should be able to read the same value. These are phenomena that happen at such a large scale that it's easy to convince yourself that nobody will succeed in rigging the process. It's far fetched to imagine that somebody would fly a spacecraft towards the surface of the Sun in order to somehow tamper with it just to rig some lottery back on Earth. So these approaches have several good properties: public observability, security against manipulation, and, in most cases, an acceptable level of unpredictability.

One problem with these approaches is that they're fairly slow. For example, if your random signal is the daily high temperature, then you only get one reading per day. The surface of the sun doesn't change too often. In many cryptographic applications, random bits are used as input to a *pseudorandom generator* (PRG). For the PRG to be secure, the input needs to be 80 bits (or more) in length. It might take a while for 80 bits of randomness to accumulate with sources based on weather and astronomy.

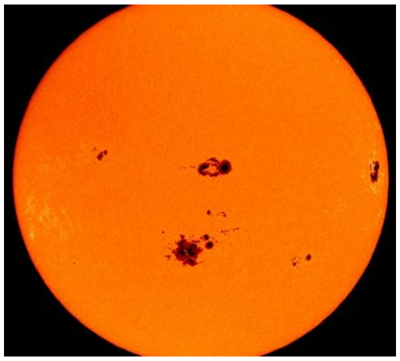


Figure 9.10: NASA image of sun spots.

Besides, it requires expertise to measure sunspots, so you'd effectively need to rely on some trusted observer to publish the measurements. However, there could be many trusted observers, and we can hope that they'd "keep each other honest". Applications that consume beacons, or users of such applications, could choose which of the observers they'd like to rely on. They can also easily switch observers at any time. This property is called "trust agility" and is arguably superior to having a single entity such as NIST that produces the beacon.

There's a deeper problem, one that at first sight might seem trivial. How do we turn a real-world observation — a temperature, a photograph of sunspots — into a string of bits in such a way that every observer will end up with the same bit string? We could try quantizing the measurement: for example, we could express the temperature in Fahrenheit and use the first decimal digit as the beacon output. But unless every observer's thermometer is unrealistically precise, there will be times when some observers will read the temperature as (say) 62.7 and others will read it as 62.8. It seems that no matter which natural phenomenon we pick and what protocol we use, there will always be "corner cases". For a cryptographic beacon, even a small probability of inconsistent measurements may be unacceptable because it will cause the random bits output by a PRG to be completely different.

***Financial data*.** A similar idea is to use feeds of financial data such as stock market prices. Again, these are publicly observable values. Unlike natural phenomena, they are reported as digital values, so the problem of inconsistent observations goes away. There's a strong to believe that it's very hard to predict the low-level fluctuation of stock prices: if you could predict within a penny what the final price of a specific stock will be on the New York stock exchange tomorrow, you could make a lot of profit as a day trader. Someone could could try to influence the price by buying or selling the stock to drive it to a specific value, but that has a real cost that you can't avoid.

However, this approach also has the problem of relying on a trusted party, namely the stock exchange. Even though the stock exchange has a strong incentive to establish that it's honest and acting in good faith, there could still be the suspicion that they might try to change the price of a stock by a penny (for example, by inserting their own order into the order book) if it would let them rig a valuable lottery.

With all the approaches we've looked at, it seems hard to avoid having some trusted party who has influence over some crucial part of the process.

***Using Bitcoin as a Beacon*.** Fortunately, a theme so far of this entire book has been that Bitcoin is a promising technology for removing centralized trust from protocols in ways we didn't think previously think were possible. Can we use Bitcoin as a random beacon? We'd like to extract random data from the Bitcoin block chain while keeping all decentralized properties that make Bitcoin itself so attractive.

Recall that miners must compute lots of random hash values while they're attempting to find a winning block. Perhaps this means that no one can predict or influence what the next block hash will be without actually doing the work of mining. Of course the first several bits of any block hash will be zero, but it turns out that under suitable assumptions, the only way to predict the remaining bits would be to influence them by finding a winning block and selectively discarding it.

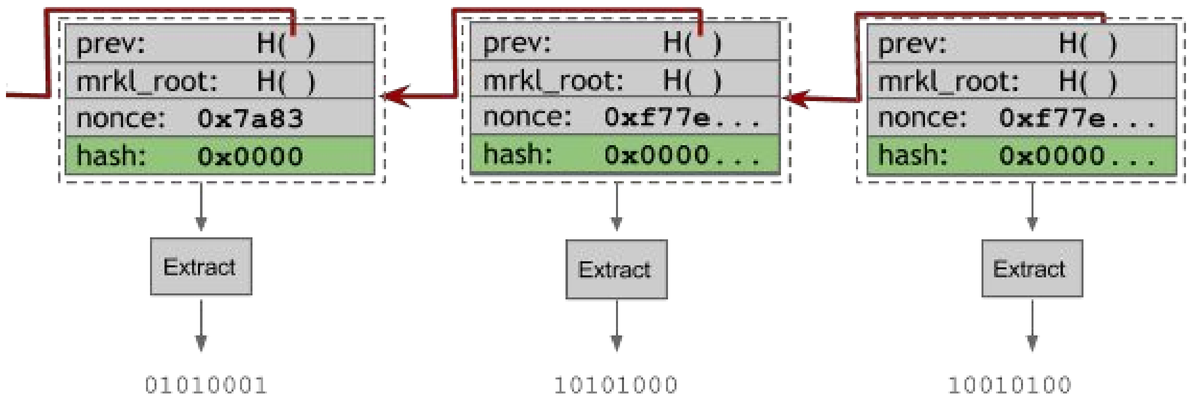


Figure 9.11: Extracting public randomness from the hashes of blocks in the block chain.

That makes it simple to turn the block chain into a randomness beacon. For every block in the chain, we apply a "randomness extractor" to the value of the block header. A randomness extractor, roughly speaking, is like a hash function that is designed to squeeze all of the random entropy of the input into the one uniformly random string. Every time a block is published, we have new beacon output.

Evaluating the security of a Bitcoin beacon. Let's say a miner is participating in a lottery whose outcome is determined by the output of the Bitcoin beacon for some pre-specified future block B in the block chain. Let's further suppose that the miner gets lucky and finds a hash puzzle solution for block B. The miner now has the choice of whether or not to publish the block. If he doesn't like the lottery outcome that would result from his publishing the block he found, he can simply discard it and let the lottery be determined by whoever else publishes block B. However, the miner would forfeit the revenue earned from that block.

Suppose you're a miner, and you're participating in a lottery where you're one of N players, with each player betting B bitcoins. Let's calculate how big B needs to be for you to find the selective withholding strategy worthwhile. Say you successfully find a block at the block height that determines the lottery, and realize that if you publish it, you will definitely lose the lottery, whereas if you discard the block you'll still have a 1/N chance of winning B * N bitcoins. That means it will be rational to discard the block if your expected payout of 1/N * B * N bitcoins (i.e., B bitcoins) is greater than the reward for mining a block (roughly 25 bitcoins in 2015, ignoring transaction fees). So the attack is profitable if B > 25. In mid 2015, 25 bitcoins is roughly 4,000 US dollars. So if the bet per player is under $4,000, the lottery will be secure against this attack if the players are rational.

One of the advantages of this is scheme is that it's a fully decentralized beacon, with no central point of trust. Compared to some other beacon proposals, it is fairly fast. It can create an output roughly every ten minutes. It's also useful to be able to estimate the cost to an attacker to manipulate the beacon outputs using our simple model above.

A downside downside of using Bitcoin as a beacon is that its timing is somewhat imprecise. Say we want to read the value of the beacon tomorrow at noon. We don't know exactly which block will be the latest block at that time. Although on average a block will be published within 10 minutes before or after noon, there is some variance. We also have to plan to tolerate a bit more delay if we want to reduce the likelihood of the block we look at being lost in a short fork. As is usual in Bitcoin, we'd want to wait for roughly six blocks to arrive before we believe that the beacon value has truly settled.

Another downside is that the cost of manipulating the beacon value may be too low for some applications we care about. If we were actually running the NBA draft, where there are tens of millions of dollars at stake, it may suddenly look worthwhile for one of the teams to start bribing Bitcoin miners to manipulate this process. It remains an open question if we can extend this construction to make it secure in a case like that where millions of dollars are on the line.

Finally, our security evaluation ignores some real-life factors. For example, a miner who is part of a mining pool doesn't lose much by discarding a block, since they're rewarded on the basis of shares rather than blocks. For now, Bitcoin beacons are an interesting but unproven idea.

Scripting support for beacons. What if we extended Bitcoin's scripting language with a special opcode to read beacon values? Currently there's no way to have any randomness in Bitcoin scripts. That's by design, because miners have to verify scripts and they all want to agree on whether a script is valid or not. But if we use the beacon value, it's a public source of verifiable randomness. We could use the beacon to add randomness into transaction scripts that every miner could agree on.

Suppose we had one opcode that would make a random decision based on the beacon output of the previous block. We could replace the entire complicated lottery protocol just one script that reads the beacon value and assigns the output to one of **n** keys. It wouldn't require a multi-round protocol, security deposits, or timed hash commitments.

One drawback of this idea is that it would now be possible for miners to manipulate the lottery simply by delaying the lottery transaction until a later block if they find that including the transaction in the block they're mining would cause them to lose the lottery. It no longer requires forfeiting block rewards. It's possible to make a variation of the beacon opcode that avoids this attack. Instead of referring to the previous block, you specify to use the beacon value at a particular block height.

Lecture 9.5: Prediction Markets and Real World Data Feeds

For the final topic of this chapter, we'll look how to implement a prediction market in a decentralized way using cryptocurrencies and the related topic bringing real-world data into Bitcoin.

A prediction market allows people to come together and make bets on future events such as a sports game or an election. Participants in a prediction market buy, sell, and trade "shares" in specific outcomes of such events.

Team	Germany	Argentina	Brazil	United States	England	Netherlands
Pre-tournament	0.12	0.09	0.22	0.01	0.05	0.03
After group stage	0.18	0.15	0.31	0.06	0.00	0.05
Before semifinals	0.26	0.21	0.45	0.00	0.00	0.08
Before finals	0.64	0.36	0.00	0.00	0.00	0.00
Final	1	0	0	0	0	0.00

Table 9.12: Prices in a hypothetical prediction market for a selection of teams during the 2014 World Cup. The price of a share betting on the US team to win the cup rose from 1 cent to 6 cents after US performed well in the group stage. A share in Brazil rose progressively to 45 cents as Brazil advanced into the semifinals and then lost its entire value after Brazil lost its semifinal match. After the tournament only shares in Germany (who won the tournament) had any value.

Let’s walk through an example that should make the concepts behind prediction markets more clear. The 2014 World Cup was held in Brazil. Suppose there was a market where you could buy and sell shares associated with each team, and the shares for the team that wins will ultimately be worth 1.0 and all the other shares are worth 0. Going into the tournament, every team would start out with some nonzero price, based on what the market believes their chances of winning are. Examples are shown in Table 9.12 for five different teams.

In the pre-tournament phase, Germany shares are trading for about 12 cents, which means that the market roughly believes Germany has a 12% chance of winning. As the tournament progresses, these prices will fluctuate, reflecting how the market participants adjust their beliefs of how likely each team is to win.

In our example, England was initially trading at five cents but went to zero after the group stage. That’s because England was knocked in the group stage. There's no longer any way for them to win, and the price reflects that; their shares are now worthless. On the other hand, the U.S. team that was initially thought to have a very poor chance of surviving the group stage turned out to do very well. If you had thought to buy U.S. shares in the beginning when they were very cheap (one cent), you could sell them immediately after the group stage for six cents. You’d get back six times the money you bet. You wouldn't have to wait until after the end of the tournament to make a profit. Even though the US team didn’t end up winning the tournament, you’d be able to profit from the fact that you anticipated a change in beliefs about their chances of winning after their strong performance in the group stage.

When we get to the semifinals, there are only four teams left. U.S. and England were knocked out so their share prices have already gone to zero. Now every remaining team has a relatively high price, and their share prices should add up to 1.0. Brazil in particular was favored to win, and thus had the highest price. In fact, Brazil lost in the semifinals and their share price went to zero. Within the span of a couple of hours, the market’s beliefs changed dramatically. You would have been able to profit in a very short time frame if you were confident going in to the match that Brazil was overrated; you could take a “short position” on Brazil and/or bet on the other teams.

Going into the finals there are only two teams left and their shares again add up to 1.0. At the very end of the tournament, of course, the only shares that finally have any value are those of the German team since they ended up winning.

Obviously, one way you could have made a profit would have been to buy shares in Germany at the beginning for 12 cents and hold them all the way to to the end. This is basically how traditional sports betting works — you place a bet before the tournament starts and collect the payout after the end of the tournament. However, in a prediction market, there are many other ways to play and to profit. You can invest in any team at any time, and you can profit solely on the ability to predict that people's beliefs will change, regardless of the final outcome.

Here's another example, this time from a real prediction market. Before the 2008 US Presidential election, the Iowa Electronic Markets allowed people to buy shares for whether Barack Obama or or John McCain would win. In Figure 9.13, the price of Barack Obama shares is shown in blue and McCain shown in red. You can see that as the months of the campaigning unfolded, people's beliefs about who would win fluctuated. But by the day before the election, Obama was given a 90% chance to win. The market was well aware that the outcome was basically settled before votes were cast.

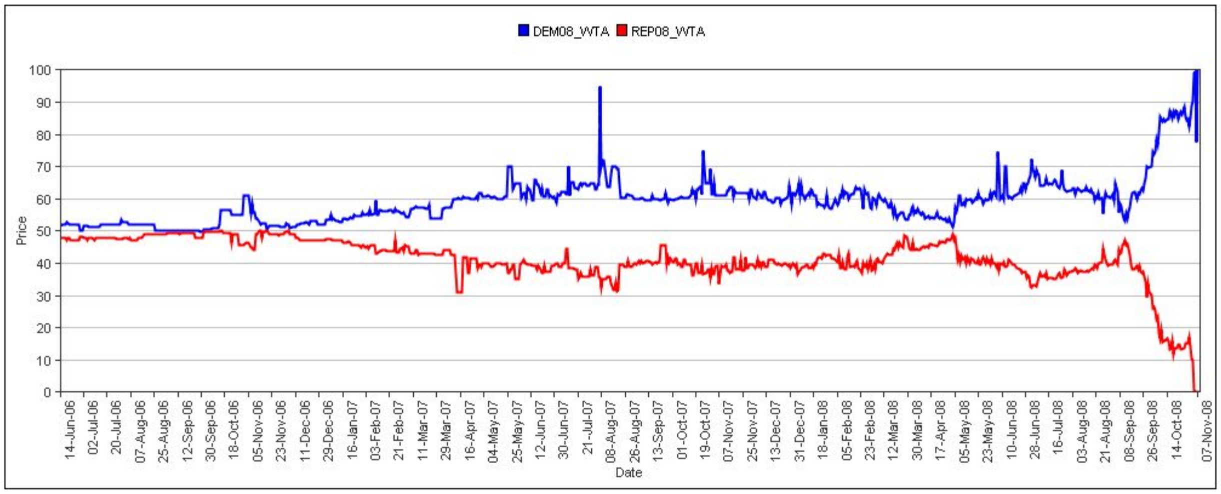


Figure 9.13: The price of prediction market shares about the 2008 US Presidential election, from Iowa Electronic Markets.

Sidebar: The Power of Prediction Markets. Economists tend to be enthusiastic about prediction markets. Information that's relevant to forecasting future events is often widely dispersed, and prediction markets are an excellent mechanism to aggregate that information by giving participants a way to profit from their knowledge. Under suitable economics models, the market price of shares can be interpreted as the probability of the outcome, although there are concerns that real prediction markets suffer from biases. Empirically, prediction markets have held up very well against other forecasting methods such as polling and expert panels.

However, prediction markets face many regulatory uncertainties and hurdles. Intrade was the most popular prediction market on the internet before it ran into regulatory compliance issues in the U.S. and shut down in 2013. Many economists were disappointed by this because they felt we lost a valuable social tool that revealed useful information about the future.

Decentralized prediction markets. What would it take to build a *decentralized* prediction market? There are a couple of tasks that we'll have to decentralize in order to make a prediction market work. We need a way of accepting money and disbursing payouts, and we need a way of enforcing that the correct amounts are paid out according to the outcome. We'll especially need decentralized arbitration. Arbitration is the process of asserting which outcomes actually happened. Most of the time, in the case of a national election or a sports match, it's pretty obvious who won and who lost. But there are also many gray areas. We'll also need to decentralize the order book, which is a way for people to find counterparties to trade shares with. We'll go through each of these challenges in order.

Payment and settlement. Let's design a hypothetical altcoin called "Futurecoin" that has explicit support for prediction markets. We'd need a few new transaction types that perform functions specific to prediction markets. It might look something like Figure 9.14.

CreateMarket allows any user to create a prediction market for any event by specifying an arbitrator (in terms of a public key) who is authorized to declare the outcome of that event, as well as a deadline (in terms of block height) when the market closes. The event_id is an arbitrary string that ties together the different transactions that refer to the same market. Futurecoin doesn't care about what real-world event the event_id refers, and there is no way to specify this within the system. We'll discuss different options for arbitration shortly.

BuyPortfolio lets you purchase a portfolio of shares of some event. For the price of one futurecoin, you can buy one share in *every* possible outcome of the event. Suppose we're betting on the 2014 World Cup. There are 32 teams that could win. For one coin, you could buy 32 shares, one for each team — this is clearly "worth" exactly one coin since exactly one of the teams will ultimately win. Any user can unilaterally create a BuyPortfolio without needing a counterparty. The transaction essentially destroys one futurecoin provided as input by the user and creates one new share in every outcome. There is also an transaction type to sell a portfolio, which lets you sell (or burn) a share in every outcome to get one futurecoin back. For one futurecoin, you can buy a share in every outcome and then you can turn a share in every outcome back into a futurecoin.

```
CreateMarket(event_id, arbitrator_key, deadline)
        create a new prediction market, specifying the arbitrator and parameters
BuyPortfolio(event_id)
        purchase one share in every outcome for 1 futurecoin
TradeShares(...)
        transfer shares in exchange for futurecoins
SellPortfolio(event_id)
        redeem one share in every outcome for 1 futurecoin
```

Figure 9.14: New transaction types in Futurecoin, a hypothetical altcoin that implements a decentralized prediction market

You can also trade shares for futurecoins, or one kind of share for another kind of share, as long as you can find someone to trade with. This case is much more interesting. You could spend a futurecoin to buy a share in every outcome, and then sell off the shares in outcomes you don't think are likely to occur. For the teams you don't want to bet on, you could sell those shares to someone else who does want to bet on that team. Once you do this, you no longer have a balanced portfolio on every team, and you can no longer automatically redeem your portfolio for one futurecoin. Instead you have to wait until the bet ends in order to redeem your shares — and if the team(s) you bet on didn't win, you might not be able to redeem them for anything at all. On the other hand, you can also profit directly by trading. You could buy a balanced portfolio, wait for prices to change, and then sell all of the shares directly for futurecoins, which you could then trade for Bitcoin or any other currency of your choice.

***Prediction market arbitration*.** How can we do arbitration in a decentralized way? How can we make assertions about who actually won so we can let people redeem their winning shares at the end? The simplest system is to have a trusted arbitrator, which is what CreateMarket above does. Any user can launch a market where they are the arbitrator (or designate someone else as the arbitrator). They can create a transaction and announce that they are opening a market on the World Cup outcomes. They will decide who won in the end, and if you trust them then you should be willing to accept their signature as evidence of the outcome.

Like in many other markets, we imagine that over time some entities will build reputations as reliable arbitrators. Then they would have some incentive to arbitrate correctly in order to maintain their valuable reputations. But there's always the risk that they could steal a lot of money — more than their reputation is worth — by rigging a bet. This would be very dangerous in a prediction market. For example, in the World Cup market, the arbitrator could assert that Argentina won, even though they actually lost. If the arbitrator had bet heavily on Argentina themselves, then they might be able to profit enough from it to justify ruining their reputation.

Could we have a more decentralized arbitration system? One option is to designate multiple arbitrators, with the outcome being decided based on the majority. There are also ideas based on voting — either by all users who hold shares in the market, or by miners of the cryptocurrency. Proposals along these lines often propose penalizing voters for voting against the majority. But there are many potential problems with these approaches and we simply don't know how well they will work in practice.

A further wrinkle is that sometimes reality is complicated. In addition to the problem of arbitrators lying, there might be a legitimate dispute over the outcome of the event. Our favorite example is from the 2014 Super Bowl. There's a tradition at the Super Bowl of the winning team dumping a bucket of Gatorade on their head coach. People like to bet on the color of the Gatorade that the winning team uses for this celebration, and this betting has happened for two or three decades. In 2014, there were bets taken on Yellow, Orange, and all the other colors of Gatorade. But that year, an unprecedented outcome made it hard to settle the bet. When the Seahawks won, they dumped orange Gatorade on their head coach, Pete Carroll. Then a little bit later, a few other players decided to do it again and dump *another* bucket of Gatorade on him. The first bucket contained orange Gatorade, and the second bucket contained yellow Gatorade.

If you were running a prediction market where people had bet on the color of the Gatorade, how would you handle this scenario? It's not clear if orange, yellow, or both should win. What happened in practice with several sports betting services is that they decided it was better to lose some money in order to maintain their reputations. As a show of good faith to their customers, they paid out winnings to anyone who bet on *either* orange or yellow.

Of course, in a decentralized prediction market this isn't so easy, because you can't just create money out of thin air to pay both sets of parties. Instead the arbitrator could split the winnings equally among both orange and yellow. Instead of closing at a value of 1.0, both shares would close at a value of 0.5. You could define the contract carefully to avoid this confusion, but you can't be sure you've anticipated every possibility. The lesson here is that arbitration is partly a social problem and no technical solution is going to be perfect.

***Data feeds*.** The idea of arbitration leads to a more general concept: extending cryptocurrencies with a mechanism to assert facts about the real world. We call such a mechanism a data feed. A fact might be about typical prediction-market events like who won an election, or the price of a stock or commodity on a certain day, or any other real world data of importance. If we had such facts available in Bitcoin, the scripting language would be able to use them as inputs. For example, a script might be able to load the current price of copper onto the stack and make decisions based on the value.

If trusted data feeds existed, we could place — and automatically settle — bets on sports matches or the future price of commodities. A prediction market is only one application that this would enable. You could hedge risks in your investment portfolio by making bets against the price of stocks you own. And you could derive a variety of financial instruments like forwards and futures that are ordinarily traded in financial markets. Wouldn't it be great if we could do all of this within Bitcoin?

We can separate the technical question of how to *represent* real-world facts in Bitcoin (or an altcoin) from the socio-technical question of how to improve our confidence in the correctness of the feed. We've already looked at the former question when discussing options for arbitration.

A clever way to encode data feeds into ordinary Bitcoin is called Reality Keys. In this system, the arbitrator creates a pair of signing keys for every outcome of every event they are interested in — one key pair for "Yes", and one key pair for "No". They publish the public keys when the event is first registered, and later publish exactly one of the two *private* keys when the outcome is settled. If Alice were betting against Bob that the outcome would occur, they could send their wagers to a Bitcoin output that can either be claimed by Alice using a signature from Alice and from the "Yes" key, or claimed by Bob using a signature from Bob and from the "No" key. This falls well short of the ideal goal of being able to use data feed values as script inputs in arbitrary ways, but it allows simple applications like the wager described above. Note that the arbitrator doesn't need to know about or get involved in the specific wager between Alice and Bob.

***Order books*.** The final piece of a prediction market a decentralized order book. Once again this is a pretty general concept, and realizing it would allow many other applications. What's an order book? In real prediction markets, or most financial markets, there isn't a single market price. Instead there are *bids* and *asks* which are listed in the *order book*. A bid is the highest price that anyone is willing to buy a share for, and the *ask* is the lowest price that anyone is willing to sell the share for. Typically the ask is greater than the bid (otherwise there would be two participants who would be matched up, a trade would occur, and at least one of the orders would no longer remain in the order book). A participant who wants to buy a share right away can do so at the ask price and a participant who wants to sell right away can do so at the bid price. These are called “market orders” since they execute at market price, as opposed to the “limit orders” that are recorded in the order book that execute at the specified limit price (or better).

Traditionally this has been done in a centralized way with a single order book service (typically an exchange) that collects all the orders. The problem, as is typical of centralized services, is that a dishonest exchange might profit at the expense of the participants. If the exchange receives a market buy order, they might themselves buy from the best ask before placing the order they received, then turn around and sell the shares they just bought at a higher price, pocketing the difference. This practice is called frontrunning. It shows up in a variety of financial settings, and it's considered a crime. Centralized order books require legal enforcement in order to discourage frontrunning and ensure confidence in the integrity of the system.

In a decentralized order book, we can’t rely on strong legal enforcement. But there's a clever solution, which is to simply forget about frontrunning. Instead of declaring it a crime and defending against it, we’ll call it a feature. The idea is that anybody can submit limit orders to miners by broadcasting transactions, and miners can match any two orders as long as the bid is *greater* than or equal to the ask. The miner simply gets to keep the difference as a form of transaction fee. Now miners now have no incentive to frontrun because frontrunning an order will never be more profitable than simply fulfilling it and capturing the surplus.

This is an elegant way to build a decentralized order book. The main downside is the miner fees that traders must pay. To avoid paying that fee, people might submit much more conservative orders and may not be willing to reveal up front the best price at which they are willing to trade. This might make the market less efficient. We don’t yet know how this kind of order book with miners matching orders will function in practice, but it seems to be a promising idea.

In conclusion, Bitcoin as it is today can act as a platform for a variety of applications. But for some applications Bitcoin only takes us so far. It doesn't have all the features we need for a secure decentralized prediction market or a decentralized order book.

But what if we could start from scratch and forget about soft forks, hard forks, and other challenges in bolting new features on to Bitcoin? We’ve learned a lot since 2008 when Bitcoin first came out. Why not design a new cryptocurrency from scratch and make everything better?

In the next chapter, we’ll look at altcoins, which are attempts to do just that. We’ll talk about all the promising ideas and the challenges to face in starting a new cryptocurrency.

Further Reading

Project pages / specifications of two of the overlay protocols we looked at:

The Counterparty protocol specification

The OpenAssets protocol

The secure multiparty lottery protocol we described is from the following paper, which is not for the faint of heart!

Andrychowicz, Marcin, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on Bitcoin**. In IEEE Security and Privacy 2014.**

Papers by economists on the power of prediction markets:

Wolfers, Justin, and Eric Zitzewitz. Prediction markets**. No. w10504. National Bureau of Economic Research, 2004.**

Arrow, Kenneth J. et al. The promise of prediction markets**. 2008.**

The prediction market design we described is from this paper, co-authored by several of the present authors:

Clark, Jeremy, Joseph Bonneau, Edward W. Felten, Joshua A. Kroll, Andrew Miller, and Arvind Narayanan. On Decentralizing Prediction Markets and Order Books**. In Workshop on the Economics of Information Security, 2014.**