

[Mirror] Quadratic Arithmetic Programs: from Zero to Hero

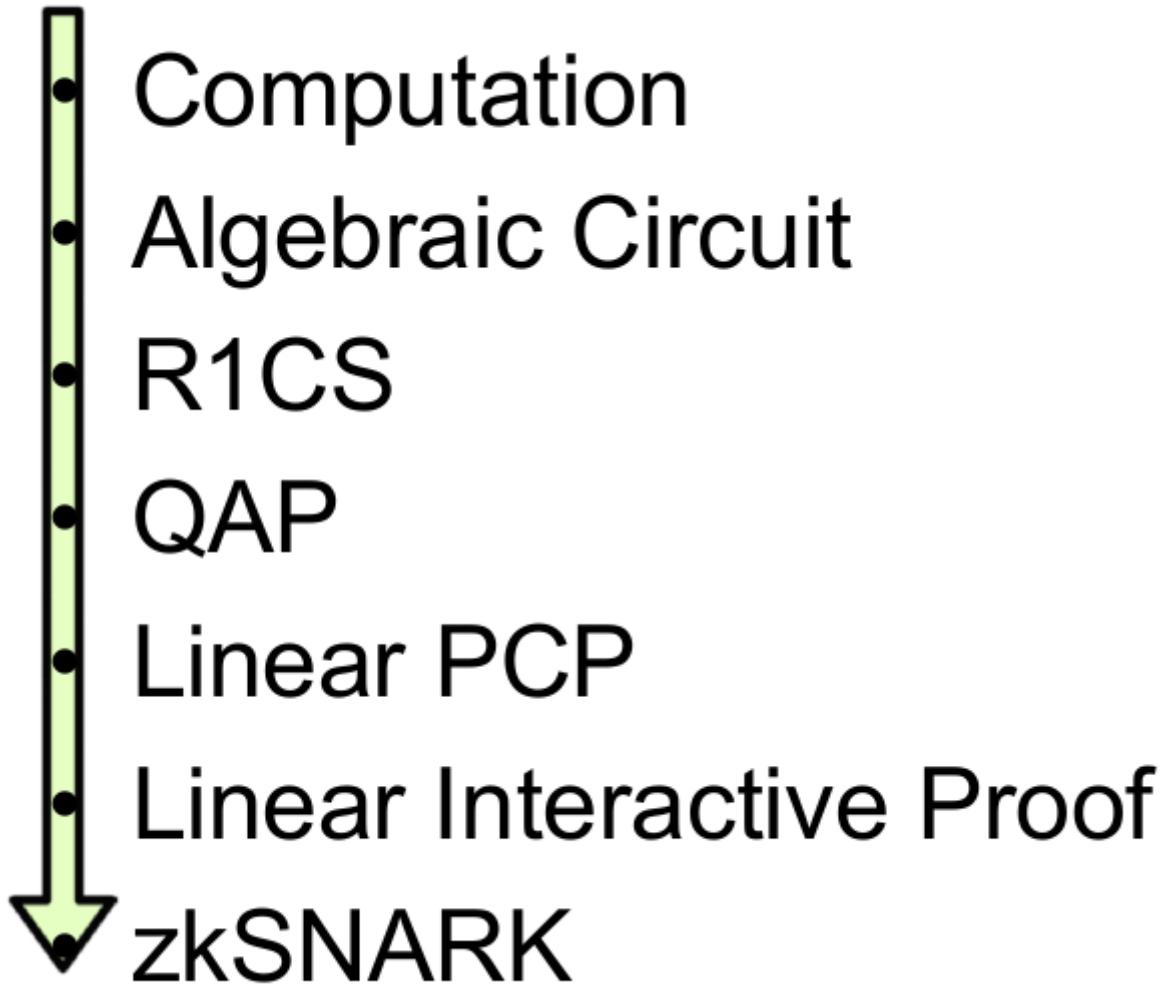
2016 Dec 10

[See all posts \(/\)](#)

This is a mirror of the post at <https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649> (<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>).

There has been a lot of interest lately in the technology behind zk-SNARKs, and people are increasingly [trying to demystify](#) (<https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/>), something that many have come to call "moon math" due to its perceived sheer indecipherable complexity. zk-SNARKs are indeed quite challenging to grasp, especially due to the sheer number of moving parts that need to come together for the whole thing to work, but if we break the technology down piece by piece then comprehending it becomes simpler.

The purpose of this post is not to serve as a full introduction to zk-SNARKs; it assumes as background knowledge that (i) you know what zk-SNARKs are and what they do, and (ii) know enough math to be able to reason about things like polynomials (if the statement $P(x) + Q(x) = (P + Q)(x)$, where P and Q are polynomials, seems natural and obvious to you, then you're at the right level). Rather, the post digs deeper into the machinery behind the technology, and tries to explain as well as possible the first half of the pipeline, as drawn by zk-SNARK researcher Eran Tromer here:



The steps here can be broken up into two halves. First, zk-SNARKs cannot be applied to any computational problem directly; rather, you have to convert the problem into the right "form" for the problem to operate on. The form is called a "quadratic arithmetic program" (QAP), and transforming the code of a function into one of these is itself highly nontrivial. Along with the process for converting the code of a function into a QAP is another process that can be run alongside so that if you have an input to the code you can create a corresponding solution (sometimes called "witness" to the QAP). After this, there is another fairly intricate process for creating the actual "zero knowledge proof" for this witness, and a separate process for verifying a proof that someone else passes along to you, but these are details that are out of scope for this post.

The example that we will choose is a simple one: proving that you know the solution to a cubic equation: $x^3 + x + 5 = 35$ (hint: the answer is 3). This problem is simple enough that the resulting QAP will not be so large as to be intimidating, but nontrivial enough that you can see all of the machinery come into play.

Let us write out our function as follows:

```
def qeval(x):  
    y = x**3  
    return x + y + 5
```

The simple special-purpose programming language that we are using here supports basic arithmetic ($+$, $-$, \cdot , $/$), constant-power exponentiation (x^7 but not x^y) and variable assignment, which is powerful enough that you can theoretically do any computation inside of it (as long as the number of computational steps is bounded; no loops allowed). Note that modulo (%) and comparison operators ($<$, $>$, \leq , \geq) are NOT supported, as there is no efficient way to do modulo or comparison directly in finite cyclic group arithmetic (be thankful for this; if there was a way to do either one, then elliptic curve cryptography would be broken faster than you can say "binary search" and "Chinese remainder theorem").

You can extend the language to modulo and comparisons by providing bit decompositions (eg. $13 = 2^3 + 2^2 + 1$) as auxiliary inputs, proving correctness of those decompositions and doing the math in binary circuits; in finite field arithmetic, doing equality ($==$) checks is also doable and in fact a bit easier, but these are both details we won't get into right now. We can extend the language to support conditionals (eg. if $x < 5 : y = 7$; else: $y = 9$) by converting them to an arithmetic form: $y = 7 \cdot (x < 5) + 9 \cdot (x \geq 5)$ though note that both "paths" of the conditional would need to be executed, and if you have many nested conditionals then this can lead to a large amount of overhead.

Let us now go through this process step by step. If you want to do this yourself for any piece of code, I implemented a compiler here (<https://github.com/ethereum/research/tree/master/zksnark>). (for educational purposes only; not ready for making QAPs for real-world zk-SNARKs quite yet!).

Flattening

The first step is a "flattening" procedure, where we convert the original code, which may contain arbitrarily complex statements and expressions, into a sequence of statements that are of two forms: $x = y$ (where y can be a variable or a number) and $x = y \text{ op } z$ (where op can be $+$, $-$, \cdot , $/$ and y and z can be variables, numbers or themselves sub-expressions). You can think of each of these statements as being kind of like logic gates in a circuit. The result of the flattening process for the above code is as follows:

```
sym_1 = x * x
y = sym_1 * x
sym_2 = y + x
~out = sym_2 + 5
```

If you read the original code and the code here, you can fairly easily see that the two are equivalent.

Gates to R1CS

Now, we convert this into something called a rank-1 constraint system (R1CS). An R1CS is a sequence of groups of three vectors (a, b, c) , and the solution to an R1CS is a vector s , where s must satisfy the equation $s \cdot a \cdot s \cdot b - s \cdot c = 0$, where \cdot represents the dot product - in simpler terms, if we "zip together" a and s , multiplying the two values in the same positions, and then take the sum of these products, then do the same to b and s and then c and s , then the third result equals the product of the first two results. For example, this is a satisfied R1CS:

A	B	C
1	5	
3	0	
35	0	
9	0	
27	0	
30	1	

35	*	1	-	35	=	0
----	---	---	---	----	---	---

But instead of having just one constraint, we are going to have many constraints: one for each logic gate. There is a standard way of converting a logic gate into a (a, b, c) triple depending on what the operation is ($+, -, \cdot$ or $/$) and whether the arguments are variables or numbers. The length of each vector is equal to the total number of variables in the system, including a dummy variable $\sim\text{one}$ at the first index representing the number 1, the input variables, a dummy variable $\sim\text{out}$ representing the output, and then all of the intermediate variables (sym_1 and sym_2 above); the vectors are generally going to be very sparse, only filling in the slots corresponding to the variables that are affected by some particular logic gate.

First, we'll provide the variable mapping that we'll use:

```
'~one', 'x', '~out', 'sym_1', 'y', 'sym_2'
```

The solution vector will consist of assignments for all of these variables, in that order.

Now, we'll give the (a, b, c) triple for the first gate:

```
a = [0, 1, 0, 0, 0, 0]
b = [0, 1, 0, 0, 0, 0]
c = [0, 0, 0, 1, 0, 0]
```

You can see that if the solution vector contains 3 in the second position, and 9 in the fourth position, then regardless of the rest of the contents of the solution vector, the dot product check will boil down to $3 \cdot 3 = 9$, and so it will pass. If the solution vector has -3 in the second position and 9 in the fourth position, the check will also pass; in fact, if the solution vector has 7 in the second position and 49 in the fourth position then that check will still pass — the purpose of this first check is to verify the consistency of the inputs and outputs of the first gate only.

Now, let's go on to the second gate:

```
a = [0, 0, 0, 1, 0, 0]
b = [0, 1, 0, 0, 0, 0]
c = [0, 0, 0, 0, 1, 0]
```

In a similar style to the first dot product check, here we're checking that $\text{sym}_1 \cdot \mathbf{x} = \mathbf{y}$.

Now, the third gate:

```
a = [0, 1, 0, 0, 1, 0]
b = [1, 0, 0, 0, 0, 0]
c = [0, 0, 0, 0, 0, 1]
```

Here, the pattern is somewhat different: it's multiplying the first element in the solution vector by the second element, then by the fifth element, adding the two results, and checking if the sum equals the sixth element. Because the first element in the solution vector is always one, this is just an addition check, checking that the output equals the sum of the two inputs.

Finally, the fourth gate:

```
a = [5, 0, 0, 0, 0, 1]
b = [1, 0, 0, 0, 0, 0]
c = [0, 0, 1, 0, 0, 0]
```

Here, we're evaluating the last check, $\text{~out} = \text{sym}_2 + 5$. The dot product check works by taking the sixth element in the solution vector, adding five times the first element (reminder: the first element is 1, so this effectively means adding 5), and checking it against the third element, which is where we store the output variable.

And there we have our R1CS with four constraints. The witness is simply the assignment to all the variables, including input, output and internal variables:

```
[1, 3, 35, 9, 27, 30]
```

You can compute this for yourself by simply "executing" the flattened code above, starting off with the input variable assignment $\mathbf{x} = 3$, and putting in the values of all the intermediate variables and the output as you compute them.

The complete R1CS put together is:

A
 [0, 1, 0, 0, 0, 0]
 [0, 0, 0, 1, 0, 0]
 [0, 1, 0, 0, 1, 0]
 [5, 0, 0, 0, 0, 1]

B
 [0, 1, 0, 0, 0, 0]
 [0, 1, 0, 0, 0, 0]
 [1, 0, 0, 0, 0, 0]
 [1, 0, 0, 0, 0, 0]

C
 [0, 0, 0, 1, 0, 0]
 [0, 0, 0, 0, 1, 0]
 [0, 0, 0, 0, 0, 1]
 [0, 0, 1, 0, 0, 0]

R1CS to QAP

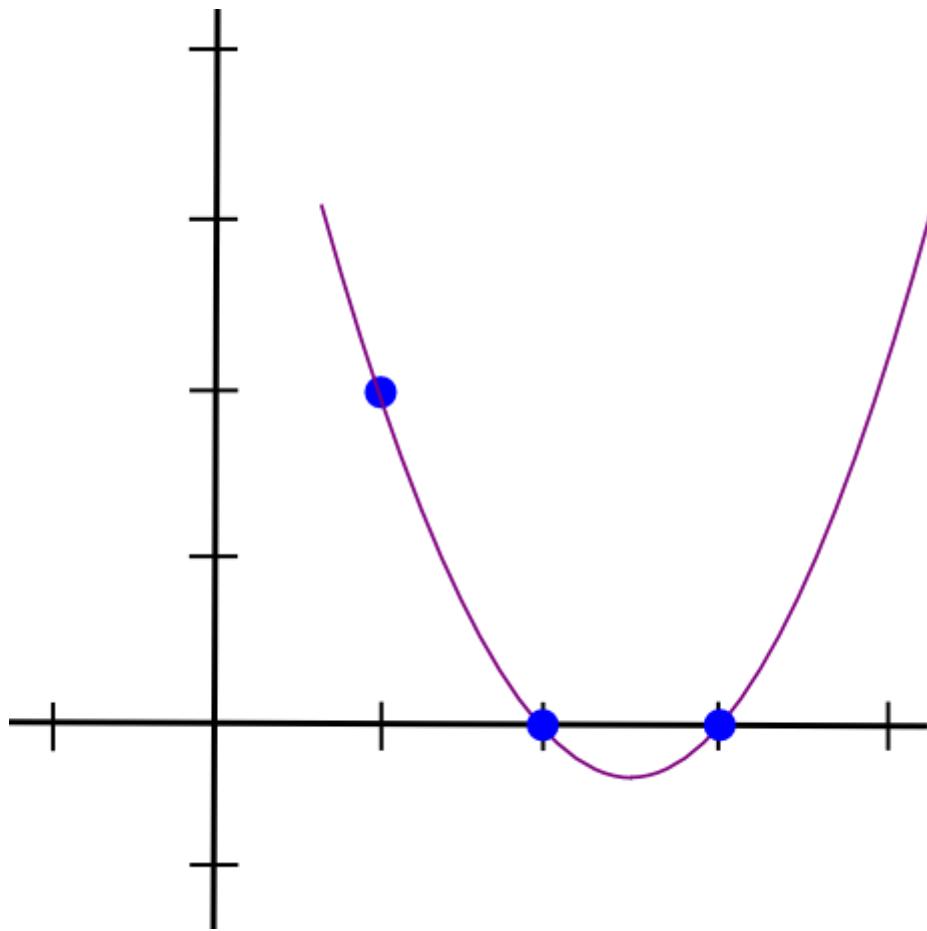
The next step is taking this R1CS and converting it into QAP form, which implements the exact same logic except using polynomials instead of dot products. We do this as follows. We go from four groups of three vectors of length six to six groups of three degree-3 polynomials, where evaluating the polynomials at each x coordinate represents one of the constraints. That is, if we evaluate the polynomials at $x = 1$, then we get our first set of vectors, if we evaluate the polynomials at $x = 2$, then we get our second set of vectors, and so on.

We can make this transformation using something called a *Lagrange interpolation*. The problem that a Lagrange interpolation solves is this: if you have a set of points (ie. (x, y) coordinate pairs), then doing a Lagrange interpolation on those points gives you a polynomial that passes through all of those points. We do this by decomposing the problem: for each x coordinate, we create a polynomial that has the desired y coordinate at that x coordinate and a y coordinate of 0 at all the other x coordinates we are interested in, and then to get the final result we add all of the polynomials together.

Let's do an example. Suppose that we want a polynomial that passes through $(1, 3)$, $(2, 2)$ and $(3, 4)$. We start off by making a polynomial that passes through $(1, 3)$, $(2, 0)$ and $(3, 0)$. As it turns out, making a polynomial that "sticks out" at $x = 1$ and is zero at the other points of interest is easy; we simply do:

$$(x - 2) * (x - 3)$$

Which looks like this:

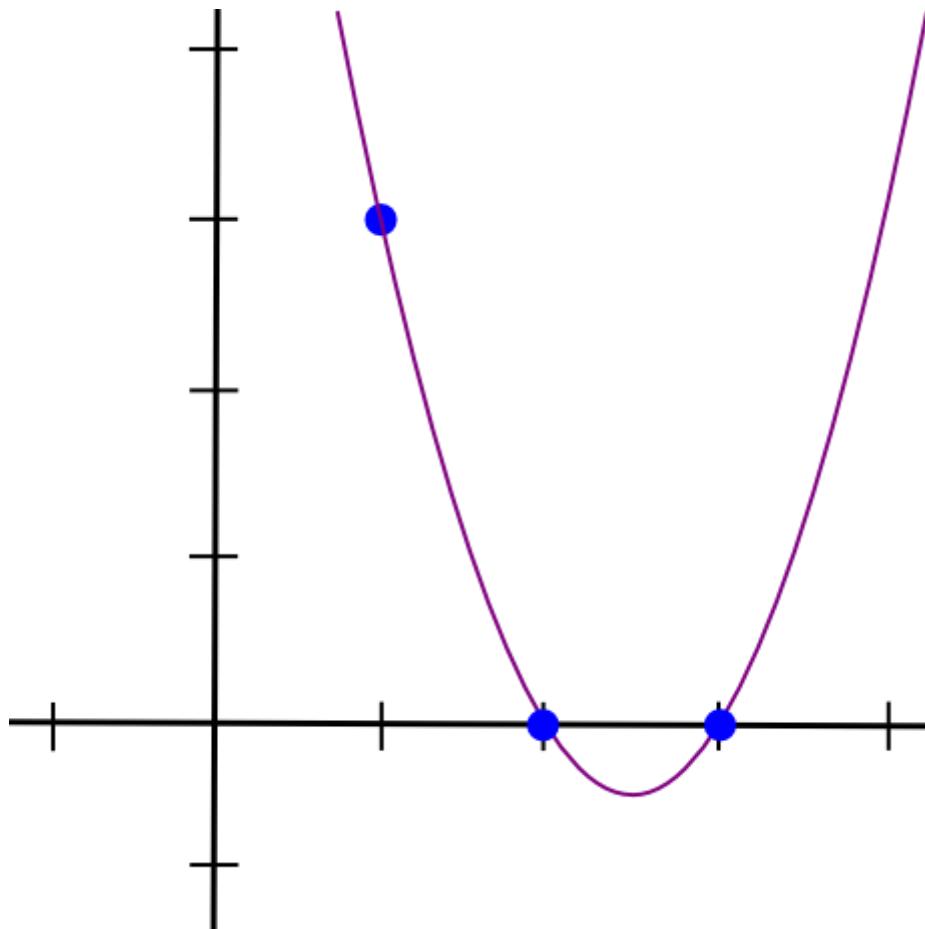


Now, we just need to "rescale" it so that the height at $x=1$ is right:

$$(x - 2) * (x - 3) * 3 / ((1 - 2) * (1 - 3))$$

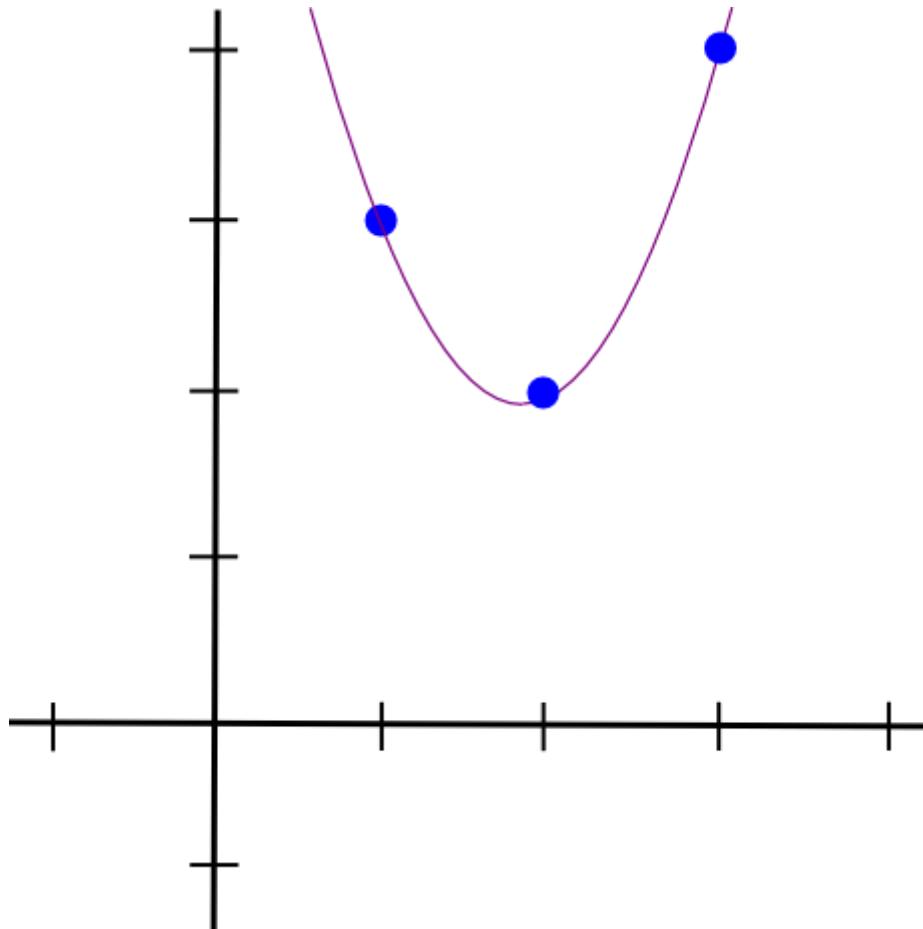
This gives us:

$$1.5 * x**2 - 7.5 * x + 9$$



We then do the same with the other two points, and get two other similar-looking polynomials, except that they "stick out" at $x = 2$ and $x = 3$ instead of $x = 1$. We add all three together and get:

$$1.5 * x**2 - 5.5 * x + 7$$



With exactly the coordinates that we want. The algorithm as described above takes $O(n^3)$ time, as there are n points and each point requires $O(n^2)$ time to multiply the polynomials together; with a little thinking, this can be reduced to $O(n^2)$ time, and with a lot more thinking, using fast Fourier transform algorithms and the like, it can be reduced even further — a crucial optimization when functions that get used in zk-SNARKs in practice often have many thousands of gates.

Now, let's use Lagrange interpolation to transform our R1CS. What we are going to do is take the first value out of every a vector, use Lagrange interpolation to make a polynomial out of that (where evaluating the polynomial at i gets you the first value of the i th a vector), repeat the process for the first value of every b and c vector, and then repeat that process for the second values, the third, values, and so on. For convenience I'll provide the answers right now:

A polynomials

```
[-5.0, 9.166, -5.0, 0.833]
[8.0, -11.333, 5.0, -0.666]
[0.0, 0.0, 0.0, 0.0]
[-6.0, 9.5, -4.0, 0.5]
[4.0, -7.0, 3.5, -0.5]
[-1.0, 1.833, -1.0, 0.166]
```

B polynomials

```
[3.0, -5.166, 2.5, -0.333]
[-2.0, 5.166, -2.5, 0.333]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
```

C polynomials

```
[0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0]
[-1.0, 1.833, -1.0, 0.166]
[4.0, -4.333, 1.5, -0.166]
[-6.0, 9.5, -4.0, 0.5]
[4.0, -7.0, 3.5, -0.5]
```

Coefficients are in ascending order, so the first polynomial above is actually $0.833 \cdot x^3 - 5 \cdot x^2 + 9.166 \cdot x - 5$. This set of polynomials (plus a Z polynomial that I will explain later) makes up the parameters for this particular QAP instance. Note that all of the work up until this point needs to be done only once for every function that you are trying to use zk-SNARKs to verify; once the QAP parameters are generated, they can be reused.

Let's try evaluating all of these polynomials at $x = 1$. Evaluating a polynomial at $x = 1$ simply means adding up all the coefficients (as $1^k = 1$ for all k), so it's not difficult. We get:

```
A results at x=1
```

```
0
1
0
0
0
0
0
```

```
B results at x=1
```

```
0
1
0
0
0
0
0
```

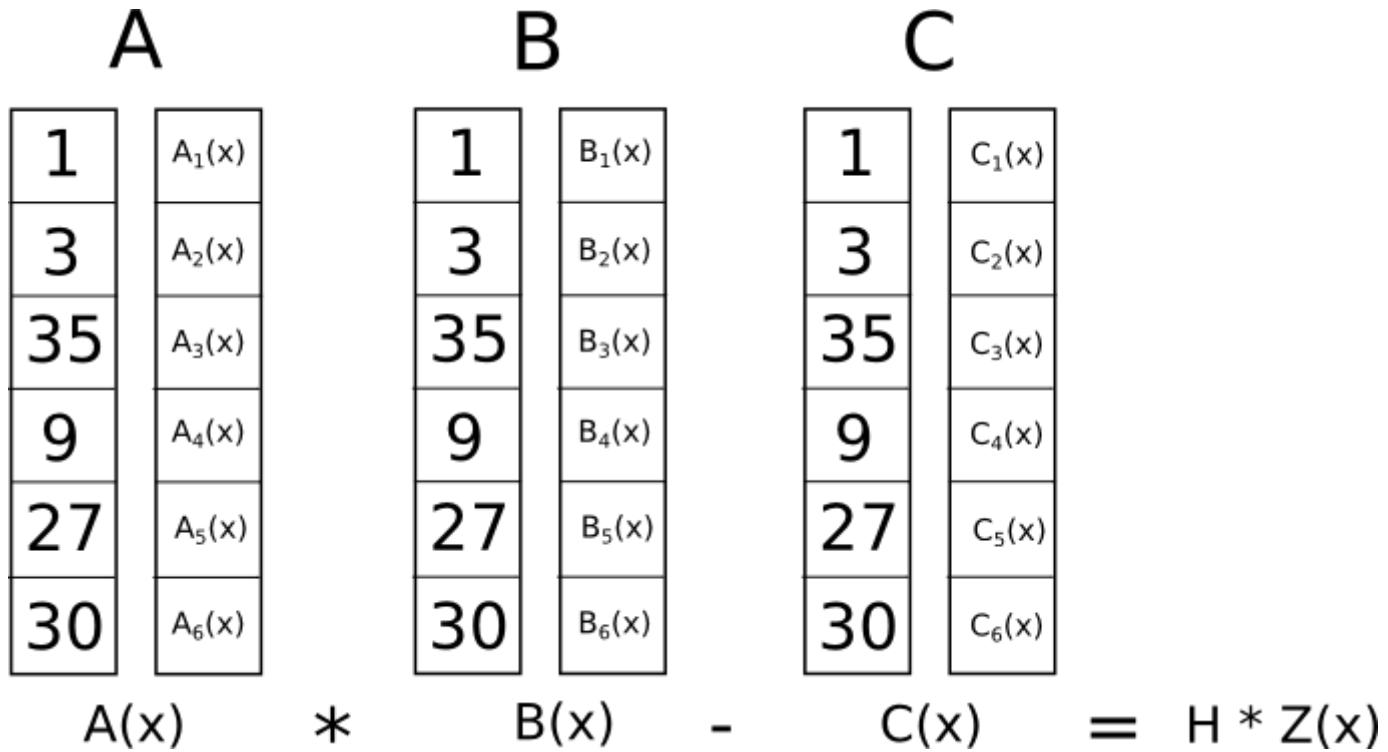
```
C results at x=1
```

```
0
0
0
1
0
0
0
```

And lo and behold, what we have here is exactly the same as the set of three vectors for the first logic gate that we created above.

Checking the QAP

Now what's the point of this crazy transformation? The answer is that instead of checking the constraints in the R1CS individually, we can now check *all of the constraints at the same time* by doing the dot product check *on the polynomials*.



Because in this case the dot product check is a series of additions and multiplications of polynomials, the result is itself going to be a polynomial. If the resulting polynomial, evaluated at every x coordinate that we used above to represent a logic gate, is equal to zero, then that means that all of the checks pass; if the resulting polynomial evaluated at at least one of the x coordinate representing a logic gate gives a nonzero value, then that means that the values going into and out of that logic gate are inconsistent (ie. the gate is $y = x \cdot \text{sym}_1$ but the provided values might be $x = 2, \text{sym}_1 = 2$ and $y = 5$).

Note that the resulting polynomial does not itself have to be zero, and in fact in most cases won't be; it could have any behavior at the points that don't correspond to any logic gates, as long as the result is zero at all the points that *do* correspond to some gate. To check correctness, we don't actually evaluate the polynomial $t = A.s \cdot B.s - C.s$ at every point corresponding to a gate; instead, we divide t by another polynomial, Z , and check that Z evenly divides t - that is, the division t/Z leaves no remainder.

Z is defined as $(x - 1) \cdot (x - 2) \cdot (x - 3) \dots$ - the simplest polynomial that is equal to zero at all points that correspond to logic gates. It is an elementary fact of algebra that *any* polynomial that is equal to zero at all of these points has to be a multiple of this minimal polynomial, and if a polynomial is a multiple of Z then its evaluation at any of those points will be zero; this equivalence makes our job much easier.

Now, let's actually do the dot product check with the polynomials above. First, the intermediate polynomials:

```

A . s = [43.0, -73.333, 38.5, -5.166]
B . s = [-3.0, 10.333, -5.0, 0.666]
C . s = [-41.0, 71.666, -24.5, 2.833]

```

Now, $A.s \cdot B.s - C.s$:

```
t = [-88.0, 592.666, -1063.777, 805.833, -294.777, 51.5, -3.444]
```

Now, the minimal polynomial $Z = (x - 1) \cdot (x - 2) \cdot (x - 3) \cdot (x - 4)$:

$$Z = [24, -50, 35, -10, 1]$$

And if we divide the result above by Z , we get:

$$h = t / Z = [-3.666, 17.055, -3.444]$$

With no remainder.

And so we have the solution for the QAP. If we try to falsify any of the variables in the R1CS solution that we are deriving this QAP solution from — say, set the last one to 31 instead of 30, then we get a t polynomial that fails one of the checks (in that particular case, the result at $x = 3$ would equal -1 instead of 0), and furthermore t would not be a multiple of Z ; rather, dividing t/Z would give a remainder of $[-5.0, 8.833, -4.5, 0.666]$.

Note that the above is a simplification; "in the real world", the addition, multiplication, subtraction and division will happen not with regular numbers, but rather with finite field elements — a spooky kind of arithmetic which is self-consistent, so all the algebraic laws we know and love still hold true, but where all answers are elements of some finite-sized set, usually integers within the range from 0 to $n - 1$ for some n . For example, if $n = 13$, then $1/2 = 7$ (and $7 \cdot 2 = 1$), $3 \cdot 5 = 2$, and so forth. Using finite field arithmetic removes the need to worry about rounding errors and allows the system to work nicely with elliptic curves, which end up being necessary for the rest of the zk-SNARK machinery that makes the zk-SNARK protocol actually secure.

Special thanks to Eran Tromer for helping to explain many details about the inner workings of zk-SNARKs to me.

[Mirror] A Proof of Stake Design Philosophy

2016 Dec 29

[See all posts \(/\)](#)

This is a mirror of the post at <https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51> (<https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51>)

Systems like Ethereum (and Bitcoin, and NXT, and Bitshares, etc) are a fundamentally new class of cryptoeconomic organisms — decentralized, jurisdictionless entities that exist entirely in cyberspace, maintained by a combination of cryptography, economics and social consensus. They are kind of like BitTorrent, but they are also not like BitTorrent, as BitTorrent has no concept of state — a distinction that turns out to be crucially important. They are sometimes described as [decentralized autonomous corporations](#) (<https://letstalkbitcoin.com/is-bitcoin-overpaying-for-false-security>), but they are also not quite corporations — you can't hard fork Microsoft. They are kind of like open source software projects, but they are not quite that either — you can fork a blockchain, but not quite as easily as you can fork OpenOffice.

These cryptoeconomic networks come in many flavors — ASIC-based PoW, GPU-based PoW, naive PoS, delegated PoS, hopefully soon Casper PoS — and each of these flavors inevitably comes with its own underlying philosophy. One well-known example is the maximalist vision of proof of work, where "the" correct blockchain, singular, is defined as the chain that miners have burned the largest amount of economic capital to create. Originally a mere in-protocol fork choice rule, this mechanism has in many cases been elevated to a sacred tenet — see [this Twitter discussion between myself and Chris DeRose](#) (<https://twitter.com/vitalikbuterin/status/687050458301657088>) for an example of someone seriously trying to defend the idea in a pure form, even in the face of hash-algorithm-changing protocol hard forks. Bitshares' [delegated proof of stake](#) (<https://bitshares.org/technology/delegated-proof-of-stake-consensus/>) presents another coherent philosophy, where everything once again flows from a single tenet, but one that can be described even more simply: [shareholders vote](http://docs.bitshares.org/bitshares/dpos.html) (<http://docs.bitshares.org/bitshares/dpos.html>).

Each of these philosophies; Nakamoto consensus, social consensus, shareholder voting consensus, leads to its own set of conclusions and leads to a system of values that makes quite a bit of sense when viewed on its own terms — though they can certainly be criticized when compared against each other. Casper consensus has a philosophical underpinning too, though one that has so far not been as succinctly articulated.

Myself, Vlad, Dominic, Jae and others all have their own views on why proof of stake protocols exist and how to design them, but here I intend to explain where I personally am coming from.

I'll proceed to listing observations and then conclusions directly.

- Cryptography is truly special in the 21st century because **cryptography is one of the very few fields where adversarial conflict continues to heavily favor the defender**. Castles are far easier to destroy than build, islands are defendable but can still be attacked, but an average person's ECC keys are secure enough to resist even state-level actors. Cypherpunk philosophy is fundamentally about leveraging this precious asymmetry to create a world that better preserves the autonomy of the individual, and

cryptoeconomics is to some extent an extension of that, except this time protecting the safety and liveness of complex systems of coordination and collaboration, rather than simply the integrity and confidentiality of private messages. **Systems that consider themselves ideological heirs to the cypherpunk spirit should maintain this basic property, and be much more expensive to destroy or disrupt than they are to use and maintain.**

- The "cypherpunk spirit" isn't just about idealism; making systems that are easier to defend than they are to attack is also simply sound engineering.
- **On medium to long time scales, humans are quite good at consensus.** Even if an adversary had access to unlimited hashing power, and came out with a 51% attack of any major blockchain that reverted even the last month of history, convincing the community that this chain is legitimate is much harder than just outrunning the main chain's hashpower. They would need to subvert block explorers, every trusted member in the community, the New York Times, archive.org, and many other sources on the internet; all in all, convincing the world that the new attack chain is the one that came first in the information technology-dense 21st century is about as hard as convincing the world that the US moon landings never happened. **These social considerations are what ultimately protect any blockchain in the long term,** regardless of whether or not the blockchain's community admits it ([note that](https://www.reddit.com/r/bitcoinxt/comments/41pbmf/maxwell_considers_changing_the_pow_algorithm_in/) (https://www.reddit.com/r/bitcoinxt/comments/41pbmf/maxwell_considers_changing_the_pow_algorithm_in/), Bitcoin Core [does admit](#) (https://www.reddit.com/r/Bitcoin/comments/3fg0jw/could_a_cartel_of_pool_operators_collapse_to_ctoat0d/), this primacy of the social layer).
- However, a blockchain protected by social consensus alone would be far too inefficient and slow, and too easy for disagreements to continue without end (though despite all difficulties, [it has happened](#) (<http://www.npr.org/sections/money/2011/02/15/131934618/the-island-of-stone-money/>)); hence, **economic consensus serves an extremely important role in protecting liveness and safety properties in the short term.**
- Because proof of work security can only come from block rewards (in Dominic Williams' terms, it [lacks two of the three Es](#) (https://twitter.com/dominic_w/status/648330685963370496)), and incentives to miners can only come from the risk of them losing their future block rewards, **proof of work necessarily operates on a logic of massive power incentivized into existence by massive rewards.** Recovery from attacks in PoW is very hard: the first time it happens, you can hard fork to change the PoW and thereby render the attacker's ASICs useless, but the second time you no longer have that option, and so the attacker can attack again and again. Hence, the size of the mining network has to be so large that attacks are inconceivable. Attackers of size less than X are discouraged from appearing by having the network constantly spend X every single day. **I reject this logic because (i) it kills trees** (<http://digconomist.net/beci>), **and (ii) it fails to realize the cypherpunk spirit — cost of attack and cost of defense are at a 1:1 ratio, so there is no defender's advantage.**
- **Proof of stake breaks this symmetry by relying not on rewards for security, but rather penalties.** Validators put money ("deposits") at stake, are rewarded slightly to compensate them for locking up their capital and maintaining nodes and taking extra precaution to ensure their private key safety, but the bulk of the cost of reverting transactions comes from penalties that are hundreds or thousands of times larger

than the rewards that they got in the meantime. **The "one-sentence philosophy" of proof of stake is thus not "security comes from burning energy", but rather "security comes from putting up economic value-at-loss".** A given block or state has \$X security if you can prove that achieving an equal level of finalization for any conflicting block or state cannot be accomplished unless malicious nodes complicit in an attempt to make the switch pay \$X worth of in-protocol penalties.

- Theoretically, a majority collusion of validators may take over a proof of stake chain, and start acting maliciously. However, (i) through clever protocol design, their ability to earn extra profits through such manipulation can be limited as much as possible, and more importantly (ii) if they try to prevent new validators from joining, or execute 51% attacks, then the community can simply coordinate a hard fork and delete the offending validators' deposits. **A successful attack may cost \$50 million, but the process of cleaning up the consequences will not be *that* much more onerous than the geth/parity consensus failure of 2016.11.25 (<https://blog.ethereum.org/2016/11/25/security-alert-11242016-consensus-bug-geth-v1-4-19-v1-5-2/>).** Two days later, the blockchain and community are back on track, attackers are \$50 million poorer, and the rest of the community is likely richer since the attack will have caused the value of the token to go *up* due to the ensuing supply crunch. *That's* attack/defense asymmetry for you.
- The above should not be taken to mean that unscheduled hard forks will become a regular occurrence; if desired, the cost of a *single* 51% attack on proof of stake can certainly be set to be as high as the cost of a *permanent* 51% attack on proof of work, and the sheer cost and ineffectiveness of an attack should ensure that it is almost never attempted in practice.
- **Economics is not everything.** Individual actors may be motivated by extra-protocol motives, they may get hacked, they may get kidnapped, or they may simply get drunk and decide to wreck the blockchain one day and to hell with the cost. Furthermore, on the bright side, **individuals' moral forbearances and communication inefficiencies will often raise the cost of an attack to levels much higher than the nominal protocol-defined value-at-loss.** This is an advantage that we cannot rely on, but at the same time it is an advantage that we should not needlessly throw away.
- **Hence, the best protocols are protocols that work well under a variety of models and assumptions** — economic rationality with coordinated choice, economic rationality with individual choice, simple fault tolerance, Byzantine fault tolerance (ideally both the adaptive and non-adaptive adversary variants), Ariely/Kahneman-inspired behavioral economic models (<https://www.amazon.ca/Honest-Truth-About-Dishonesty-Everyone-Especially/dp/0062183613>). ("we all cheat just a little") and ideally any other model that's realistic and practical to reason about. **It is important to have both layers of defense: economic incentives to discourage centralized cartels from acting anti-socially, and anti-centralization incentives to discourage cartels from forming in the first place.**
- **Consensus protocols that work as-fast-as-possible have risks and should be approached very carefully if at all,** because if the *possibility* to be very fast is tied to *incentives* to do so, the combination will reward very high and systemic-risk-inducing levels of **network-level centralization** (eg. all validators running from the same hosting provider). Consensus protocols that don't care too much how fast a validator sends a message, as long as they do so within some acceptably long time interval (eg. 4–8 seconds, as we empirically know that latency in ethereum is usually ~500ms-1s) do not have these

concerns. A possible middle ground is creating protocols that can work very quickly, but where mechanics similar to Ethereum's uncle mechanism ensure that the marginal reward for a node increasing its degree of network connectivity beyond some easily attainable point is fairly low.

From here, there are of course many details and many ways to diverge on the details, but the above are the core principles that at least my version of Casper is based on. From here, we can certainly debate tradeoffs between competing values . Do we give ETH a 1% annual issuance rate and get an \$50 million cost of forcing a remedial hard fork, or a zero annual issuance rate and get a \$5 million cost of forcing a remedial hard fork? When do we increase a protocol's security under the economic model in exchange for decreasing its security under a fault tolerance model? Do we care more about having a predictable level of security or a predictable level of issuance? These are all questions for another post, and the various ways of *implementing* the different tradeoffs between these values are questions for yet more posts. But we'll get to it :)

[Mirror] Exploring Elliptic Curve Pairings

2017 Jan 14

[See all posts \(/\)](#)

This is a mirror of the post at <https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627> (<https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>).

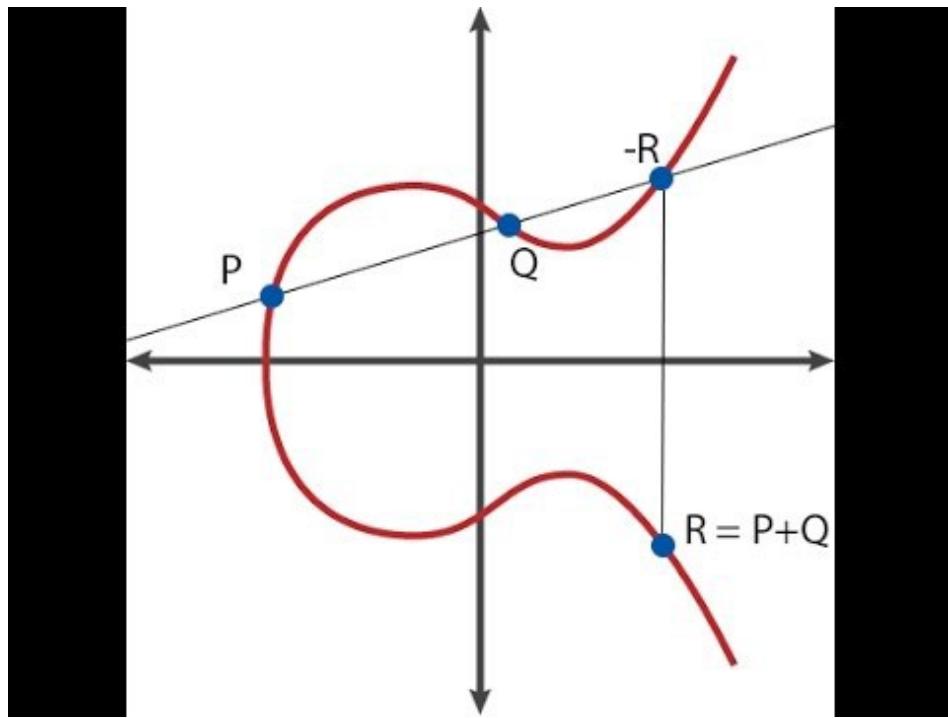
Trigger warning: math.

One of the key cryptographic primitives behind various constructions, including deterministic threshold signatures, zk-SNARKs and other simpler forms of zero-knowledge proofs is the elliptic curve pairing. Elliptic curve pairings (or "bilinear maps") are a recent addition to a 30-year-long history of using elliptic curves for cryptographic applications including encryption and digital signatures; pairings introduce a form of "encrypted multiplication", greatly expanding what elliptic curve-based protocols can do. The purpose of this article will be to go into elliptic curve pairings in detail, and explain a general outline of how they work.

You're not expected to understand everything here the first time you read it, or even the tenth time; this stuff is genuinely hard. But hopefully this article will give you at least a bit of an idea as to what is going on under the hood.

Elliptic curves themselves are very much a nontrivial topic to understand, and this article will generally assume that you know how they work; if you do not, I recommend this article here as a primer:

<https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/> (<https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/>). As a quick summary, elliptic curve cryptography involves mathematical objects called "points" (these are literal two-dimensional points with (x, y) coordinates), with special formulas for adding and subtracting them (ie. for calculating the coordinates of $R = P + Q$), and you can also multiply a point by an integer (ie. $P \cdot n = P + P + \dots + P$, though there's a much faster way to compute it if n is big).



Here's how point addition looks like graphically.

There exists a special point called the "point at infinity" (O), the equivalent of zero in point arithmetic; it's always the case that $P + O = P$. Also, a curve has an "**order**"; there exists a number n such that $P \cdot n = O$ for any P (and of course, $P \cdot (n + 1) = P, P \cdot (7 \cdot n + 5) = P \cdot 5$, and so on). There is also some commonly agreed upon "generator point" G , which is understood to in some sense represent the number 1. Theoretically, any point on a curve (except O) can be G ; all that matters is that G is standardized.

Pairings go a step further in that they allow you to check certain kinds of more complicated equations on elliptic curve points — for example, if $P = G \cdot p, Q = G \cdot q$ and $R = G \cdot r$, you can check whether or not $p \cdot q = r$, having just P, Q and R as inputs. This might seem like the fundamental security guarantees of elliptic curves are being broken, as information about p is leaking from just knowing P , but it turns out that the leakage is highly contained — specifically, the [decisional Diffie Hellman problem](#)

(https://en.wikipedia.org/wiki/Decisional_Diffie%20%93Hellman_assumption) is easy, but the computational Diffie Hellman problem (knowing P and Q in the above example, *computing $R = G \cdot p \cdot q$*) and the [discrete logarithm problem](#) (https://en.wikipedia.org/wiki/Discrete_logarithm) (recovering p from P) remain computationally infeasible (at least, if they were before).

A third way to look at what pairings do, and one that is perhaps most illuminating for most of the use cases that we are about, is that if you view elliptic curve points as one-way encrypted numbers (that is, $\text{encrypt}(p) = p \cdot G = P$), then whereas traditional elliptic curve math lets you check *linear* constraints on the numbers (eg. if $P = G \cdot p, Q = G \cdot q$ and $R = G \cdot r$, checking $5 \cdot P + 7 \cdot Q = 11 \cdot R$ is *really* checking that $5 \cdot p + 7 \cdot q = 11 \cdot r$), pairings let you check *quadratic* constraints (eg. checking $e(P, Q) \cdot e(G, G \cdot 5) = 1$ is *really* checking that $p \cdot q + 5 = 0$). And going up to quadratic is enough to let us work with deterministic threshold signatures, quadratic arithmetic programs and all that other good stuff.

Now, what is this funny $e(P, Q)$ operator that we introduced above? This is the pairing. Mathematicians also sometimes call it a *bilinear map*; the word "bilinear" here basically means that it satisfies the constraints:

$$e(P, Q + R) = e(P, Q) \cdot e(P, R)$$

$$e(P + S, Q) = e(P, Q) \cdot e(S, Q)$$

Note that $+$ and \cdot can be arbitrary operators; when you're creating fancy new kinds of mathematical objects, abstract algebra doesn't care how $+$ and \cdot are *defined*, as long as they are consistent in the usual ways, eg. $a + b = b + a$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ and $(a \cdot c) + (b \cdot c) = (a + b) \cdot c$.

If P , Q , R and S were simple *numbers*, then making a simple pairing is easy: we can do $e(x, y) = 2^{xy}$. Then, we can see:

$$e(3, 4 + 5) = 2^{3 \cdot 9} = 2^{27}$$

$$e(3, 4) \cdot e(3, 5) = 2^{3 \cdot 4} \cdot 2^{3 \cdot 5} = 2^{12} \cdot 2^{15} = 2^{27}$$

It's bilinear!

However, such simple pairings are not suitable for cryptography because the objects that they work on are simple integers and are too easy to analyze; integers make it easy to divide, compute logarithms, and make various other computations; simple integers have no concept of a "public key" or a "one-way function". Additionally, with the pairing described above you can go backwards – knowing x , and knowing $e(x, y)$, you can simply compute a division and a logarithm to determine y . We want mathematical objects that are as close as possible to "black boxes", where you can add, subtract, multiply and divide, *but do nothing else*. This is where elliptic curves and elliptic curve pairings come in.

It turns out that it is possible to make a bilinear map over elliptic curve points — that is, come up with a function $e(P, Q)$ where the inputs P and Q are elliptic curve points, and where the output is what's called an $(F_p)^{12}$ element (at least in the specific case we will cover here; the specifics differ depending on the details of the curve, more on this later), but the math behind doing so is quite complex.

First, let's cover prime fields and extension fields. The pretty elliptic curve in the picture earlier in this post only looks that way if you assume that the curve equation is defined using regular real numbers. However, if we actually use regular real numbers in cryptography, then you can use logarithms to "go backwards", and everything breaks; additionally, the amount of space needed to actually store and represent the numbers may grow arbitrarily. Hence, we instead use numbers in a **prime field**.

A prime field consists of the set of numbers $0, 1, 2, \dots, p - 1$, where p is prime, and the various operations are defined as follows:

$$a + b : (a + b) \% p$$

$$a \cdot b : (a \cdot b) \% p$$

$$a - b : (a - b) \% p$$

$$a/b : (a \cdot b^{p-2}) \% p$$

Basically, all math is done modulo p (see [here](https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic) (<https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/what-is-modular-arithmetic>) for an introduction to modular math). Division is a special case; normally, $\frac{3}{2}$ is not an integer, and here we want to deal only with integers, so we instead try to find the number x such that $x \cdot 2 = 3$, where \cdot of course refers to modular multiplication as defined above. Thanks to [Fermat's little theorem](https://en.wikipedia.org/wiki/Fermat's_little_theorem) (https://en.wikipedia.org/wiki/Fermat's_little_theorem), the exponentiation trick shown above does the job, but there is also a faster way to do it, using the [Extended Euclidean Algorithm](https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm) (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm). Suppose $p = 7$; here are a few examples:

$$2 + 3 = 5 \quad \% 7 = 5$$

$$4 + 6 = 10 \quad \% 7 = 3$$

$$2 - 5 = -3 \quad \% 7 = 4$$

$$6 \cdot 3 = 18 \quad \% 7 = 4$$

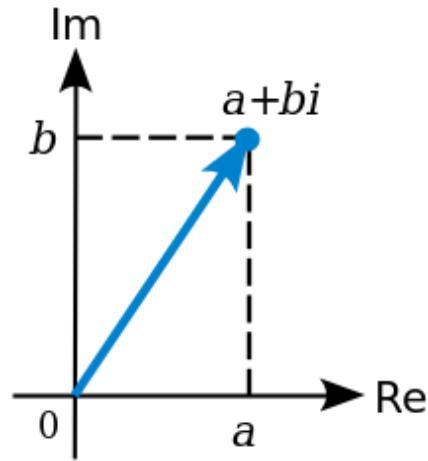
$$3/2 = (3 \cdot 2^5) \% 7 = 5$$

$$5 \cdot 2 = 10 \quad \% 7 = 3$$

If you play around with this kind of math, you'll notice that it's perfectly consistent and satisfies all of the usual rules. The last two examples above show how $(a/b) \cdot b = a$; you can also see that

$(a + b) + c = a + (b + c)$, $(a + b) \cdot c = a \cdot c + b \cdot c$, and all the other high school algebraic identities you know and love continue to hold true as well. In elliptic curves in reality, the points and equations are usually computed in prime fields.

Now, let's talk about **extension fields**. You have probably already seen an extension field before; the most common example that you encounter in math textbooks is the field of complex numbers, where the field of real numbers is "extended" with the additional element $\sqrt{-1} = i$. Basically, extension fields work by taking an existing field, then "inventing" a new element and defining the relationship between that element and existing elements (in this case, $i^2 + 1 = 0$), making sure that this equation does not hold true for any number that is in the original field, and looking at the set of all linear combinations of elements of the original field and the new element that you have just created.



We can do extensions of prime fields too; for example, we can extend the prime field mod 7 that we described above with i , and then we can do:

$$(2 + 3i) + (4 + 2i) = 6 + 5i$$

$$(5 + 2i) + 3 = 1 + 2i$$

$$(6 + 2i) \cdot 2 = 5 + 4i$$

$$4i \cdot (2 + i) = 3 + i$$

That last result may be a bit hard to figure out; what happened there was that we first decompose the product into $4i \cdot 2 + 4i \cdot i$, which gives $8i - 4$, and then because we are working in mod 7 math that becomes $i + 3$. To divide, we do:

$$a/b : (a \cdot b^{(p^2-2)}) \% p$$

Note that the exponent for Fermat's little theorem is now p^2 instead of p , though once again if we want to be more efficient we can also instead extend the Extended Euclidean Algorithm to do the job. Note that $x^{p^2-1} = 1$ for any x in this field, so we call $p^2 - 1$ the "order of the multiplicative group in the field".

With real numbers, the Fundamental Theorem of Algebra

(https://en.wikipedia.org/wiki/Fundamental_theorem_of_algebra) ensures that the quadratic extension that we call the complex numbers is "complete" — you cannot extend it further, because for any mathematical relationship (at least, any mathematical relationship defined by an algebraic formula) that you can come up with between some new element j and the existing complex numbers, it's possible to come up with at least one complex number that already satisfies that relationship. With prime fields, however, we do not have this issue, and so we can go further and make cubic extensions (where the mathematical relationship between some new element w and existing field elements is a cubic equation, so $1, w$ and w^2 are all linearly independent of each other), higher-order extensions, extensions of extensions, etc. And it is these kinds of supercharged modular complex numbers that elliptic curve pairings are built on.

For those interested in seeing the exact math involved in making all of these operations written out in code, prime fields and field extensions are implemented here:

https://github.com/ethereum/py_pairing/blob/master/py_ecc/BN128/BN128_field_elements.py

(https://github.com/ethereum/py_pairing/blob/master/py_ecc/BN128/BN128_field_elements.py).

Now, on to elliptic curve pairings. An elliptic curve pairing (or rather, the specific form of pairing we'll explore here; there are also other types of pairings, though their logic is fairly similar) is a map $G_2 \times G_1 \rightarrow G_t$, where:

- G_1 is an elliptic curve, where points satisfy an equation of the form $y^2 = x^3 + b$, and where both coordinates are elements of F_p (ie. they are simple numbers, except arithmetic is all done modulo some prime number)
- G_2 is an elliptic curve, where points satisfy the same equation as G_1 , except where the coordinates are elements of $(F_p)^{12}$ (ie. they are the supercharged complex numbers we talked about above; we define a new "magic number" w , which is defined by a 12th degree polynomial like $w^{12} - 18 \cdot w^6 + 82 = 0$)
- G_t is the type of object that the result of the elliptic curve goes into. In the curves that we look at, G_t is $(F_p)^{12}$ (the same supercharged complex number as used in G_2)

The main property that it must satisfy is bilinearity, which in this context means that:

- $e(P, Q + R) = e(P, Q) \cdot e(P, R)$
- $e(P + Q, R) = e(P, R) \cdot e(Q, R)$

There are two other important criteria:

- **Efficient computability** (eg. we can make an easy pairing by simply taking the discrete logarithms of all points and multiplying them together, but this is as computationally hard as breaking elliptic curve cryptography in the first place, so it doesn't count)
- **Non-degeneracy** (sure, you could just define $e(P, Q) = 1$, but that's not a particularly useful pairing)

So how do we do this?

The math behind why pairing functions work is quite tricky and involves quite a bit of advanced algebra going even beyond what we've seen so far, but I'll provide an outline. First of all, we need to define the concept of a **divisor**, basically an alternative way of representing functions on elliptic curve points. A divisor of a function basically counts the zeroes and the infinities of the function. To see what this means, let's go through a few examples. Let us fix some point $P = (P_x, P_y)$, and consider the following function:

$$f(x, y) = x - P_x$$

The divisor is $[P] + [-P] - 2 \cdot [O]$ (the square brackets are used to represent the fact that we are referring to *the presence of the point P in the set of zeroes and infinities of the function*, not the point P itself; $[P] + [Q]$ is **not** the same thing as $[P + Q]$). The reasoning is as follows:

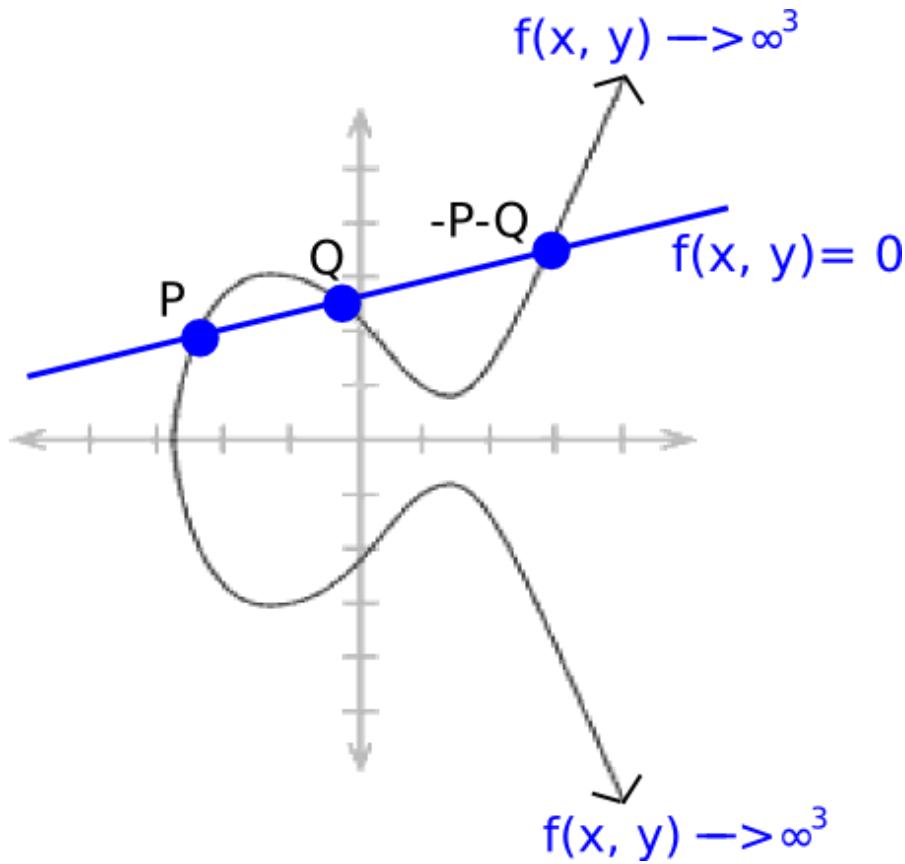
- The function is equal to zero at P , since x is P_x , so $x - P_x = 0$
- The function is equal to zero at $-P$, since $-P$ and P share the same x coordinate
- The function goes to infinity as x goes to infinity, so we say the function is equal to infinity at O . There's a technical reason why this infinity needs to be counted twice, so O gets added with a "multiplicity" of -2 (negative because it's an infinity and not a zero, two because of this double counting).

The technical reason is roughly this: because the equation of the curve is $x^3 = y^2 + b$, y goes to infinity "1.5 times faster" than x does in order for y^2 to keep up with x^3 ; hence, if a linear function includes only x then it is represented as an infinity of multiplicity 2, but if it includes y then it is represented as an infinity of multiplicity 3.

Now, consider a "line function":

$$ax + by + c = 0$$

Where a , b and c are carefully chosen so that the line passes through points P and Q . Because of how elliptic curve addition works (see the diagram at the top), this also means that it passes through $-P - Q$. And it goes up to infinity dependent on both x and y , so the divisor becomes $[P] + [Q] + [-P - Q] - 3 \cdot [O]$.



We know that every "rational function" (ie. a function defined only using a finite number of $+, -, \cdot$ and $/$ operations on the coordinates of the point) uniquely corresponds to some divisor, up to multiplication by a constant (ie. if two functions F and G have the same divisor, then $F = G \cdot k$ for some constant k).

For any two functions F and G , the divisor of $F \cdot G$ is equal to the divisor of F plus the divisor of G (in math textbooks, you'll see $(F \cdot G) = (F) + (G)$), so for example if $f(x, y) = P_x - x$, then $(f^3) = 3 \cdot [P] + 3 \cdot [-P] - 6 \cdot [O]$; P and $-P$ are "triple-counted" to account for the fact that f^3 approaches 0 at those points "three times as quickly" in a certain mathematical sense.

Note that there is a theorem that states that if you "remove the square brackets" from a divisor of a function, the points must add up to $O([P] + [Q] + [-P - Q] - 3 \cdot [O])$ clearly fits, as $P + Q - P - Q - 3 \cdot O = O$, and any divisor that has this property is the divisor of a function.

Now, we're ready to look at Tate pairings. Consider the following functions, defined via their divisors:

- $(F_P) = n \cdot [P] - n \cdot [O]$, where n is the order of G_1 , ie. $n \cdot P = O$ for any P
- $(F_Q) = n \cdot [Q] - n \cdot [O]$
- $(g) = [P + Q] - [P] - [Q] + [O]$

Now, let's look at the product $F_P \cdot F_Q \cdot g^n$. The divisor is:

$$n \cdot [P] - n \cdot [O] + n \cdot [Q] - n \cdot [O] + n \cdot [P + Q] - n \cdot [P] - n \cdot [Q] + n \cdot [O]$$

Which simplifies neatly to:

$$n \cdot [P + Q] - n \cdot [O]$$

Notice that this divisor is of exactly the same format as the divisor for F_P and F_Q above. Hence,
 $F_P \cdot F_Q \cdot g^n = F_{P+Q}$.

Now, we introduce a procedure called the "final exponentiation" step, where we take the result of our functions above (F_P , F_Q , etc.) and raise it to the power $z = (p^{12} - 1)/n$, where $p^{12} - 1$ is the order of the multiplicative group in $(F_p)^{12}$ (ie. for any $x \in (F_p)^{12}$, $x^{(p^{12}-1)} = 1$). Notice that if you apply this exponentiation to any result that has already been raised to the power of n , you get an exponentiation to the power of $p^{12} - 1$, so the result turns into 1. Hence, after final exponentiation, g^n cancels out and we get $F_P^z \cdot F_Q^z = (F_{P+Q})^z$. There's some bilinearity for you.

Now, if you want to make a function that's bilinear in both arguments, you need to go into spookier math, where instead of taking F_P of a value directly, you take F_P of a divisor, and that's where the full "Tate pairing" comes from. To prove some more results you have to deal with notions like "linear equivalence" and "Weil reciprocity", and the rabbit hole goes on from there. You can find more reading material on all of this [here](#)

(http://www.math.ru.nl/~bosma/Students/MScThesis_DennisMeffert.pdf) and [here](#)

(<http://people.cs.nctu.edu.tw/~rjchen/ECC2012S/Elliptic%20Curves%20Number%20Theory%20And%20Cryptography%202n.pdf>).

For an implementation of a modified version of the Tate pairing, called the optimal Ate paring, see [here](#) (https://github.com/ethereum/research/blob/master/zksnark/bn128_pairing.py). The code implements [Miller's algorithm](#) (<https://crypto.stanford.edu/pbc/notes/ep/miller.html>), which is needed to actually compute F_P .

Note that the fact pairings like this are possible is somewhat of a mixed blessing: on the one hand, it means that all the protocols we can do with pairings become possible, but it also means that we have to be more careful about what elliptic curves we use.

Every elliptic curve has a value called an *embedding degree*; essentially, the smallest k such that $p^k - 1$ is a multiple of n (where p is the prime used for the field and n is the curve order). In the fields above, $k = 12$, and in the fields used for traditional ECC (ie. where we don't care about pairings), the embedding degree is often extremely large, to the point that pairings are computationally infeasible to compute; however, if we are not careful then we can generate fields where $k = 4$ or even 1.

If $k = 1$, then the "discrete logarithm" problem for elliptic curves (essentially, recovering p knowing only the point $P = G \cdot p$, the problem that you have to solve to "crack" an elliptic curve private key) can be reduced into a similar math problem over F_p , where the problem becomes much easier (this is called the [MOV attack](#) (<https://crypto.stanford.edu/pbc/notes/elliptic/movattack.html>)); using curves with an embedding degree of 12 or higher ensures that this reduction is either unavailable, or that solving the discrete log problem over pairing results is at least as hard as recovering a private key from a public key "the normal way" (ie. computationally infeasible). Do not worry; all standard curve parameters have been thoroughly checked for this issue.

Stay tuned for a mathematical explanation of how zk-SNARKs work, coming soon.

Special thanks to Christian Reitwiessner, Ariel Gabizon (from Zcash) and Alfred Menezes for reviewing and making corrections.

[Mirror] Zk-SNARKs: Under the Hood

2017 Feb 01

[See all posts \(/\)](#)

This is a mirror of the post at https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6

This is the third part of a series of articles explaining how the technology behind zk-SNARKs works; the previous articles on [quadratic arithmetic programs](#) (<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>) and [elliptic curve pairings](#) (<https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>) are required reading, and this article will assume knowledge of both concepts. Basic knowledge of what zk-SNARKs are and what they do is also assumed. See also [Christian Reitwiesner's article here](#) (<https://blog.ethereum.org/2016/12/05/zksnarks-in-a-nutshell/>) for another technical introduction.

In the previous articles, we introduced the quadratic arithmetic program, a way of representing any computational problem with a polynomial equation that is much more amenable to various forms of mathematical trickery. We also introduced elliptic curve pairings, which allow a very limited form of one-way homomorphic encryption that lets you do equality checking. Now, we are going to start from where we left off, and use elliptic curve pairings, together with a few other mathematical tricks, in order to allow a prover to prove that they know a solution for a particular QAP without revealing anything else about the actual solution.

This article will focus on the [Pinocchio protocol](#) (<https://eprint.iacr.org/2013/279.pdf>), by Parno, Gentry, Howell and Raykova from 2013 (often called PGHR13); there are a few variations on the basic mechanism, so a zk-SNARK scheme implemented in practice may work slightly differently, but the basic principles will in general remain the same.

To start off, let us go into the key cryptographic assumption underlying the security of the mechanism that we are going to use: the [*knowledge-of-exponent](#) (<https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjz1-Ht7-vRAhVEr1QKHvtAKIQFggaMAA&url=https%3A%2F%2Fwww.iacr.org%2Farchive%2Fcrypto2004%2F31520273%2Fbp.pdf&usg=AFQjCNFJk9kCq86ms46ZQVkmMgF>) assumption.

KEA1: For any adversary \mathbf{A} that takes input q, g, g^a and returns (C, Y) with $Y = C^a$, there exists an “extractor” $\tilde{\mathbf{A}}$, which given the same inputs as \mathbf{A} returns c such that $g^c = C$.

Basically, if you get a pair of points P and Q , where $P \cdot k = Q$, and you get a point C , then it is not possible to come up with $C \cdot k$ unless C is “derived” from P in some way that you know. This may seem intuitively obvious, but this assumption actually cannot be derived from any other assumption (eg. discrete log hardness) that we usually use when proving security of elliptic curve-based protocols, and so zk-SNARKs do in fact rest on a somewhat shakier foundation than elliptic curve cryptography more generally — although it’s still sturdy enough that most cryptographers are okay with it.

Now, let’s go into how this can be used. Supposed that a pair of points (P, Q) falls from the sky, where $P \cdot k = Q$, but nobody knows what the value of k is. Now, suppose that I come up with a pair of points (R, S) where $R \cdot k = S$. Then, the KoE assumption implies that the only way I could have made that pair of points was by taking P and Q , and multiplying both by some factor r that I personally know. Note also that thanks to the magic of elliptic curve pairings, checking that $R = k \cdot S$ doesn’t actually require knowing k – instead, you can simply check whether or not $e(R, Q) = e(P, S)$.

Let’s do something more interesting. Suppose that we have ten pairs of points fall from the sky: $(P_1, Q_1), (P_2, Q_2) \dots (P_{10}, Q_{10})$. In all cases, $P_i \cdot k = Q_i$. Suppose that I then provide you with a pair of points (R, S) where $R \cdot k = S$. What do you know now? You know that R is some linear combination $P_1 \cdot i_1 + P_2 \cdot i_2 + \dots + P_{10} \cdot i_{10}$, where I know the coefficients i_1, i_2, \dots, i_{10} . That is, the only way to arrive at such a pair of points (R, S) is to take some multiples of P_1, P_2, \dots, P_{10} and add them together, and make the same calculation with Q_1, Q_2, \dots, Q_{10} .

Note that, given any specific set of $P_1 \dots P_{10}$ points that you might want to check linear combinations for, you can’t actually create the accompanying $Q_1 \dots Q_{10}$ points without knowing what k is, and if you do know what k is then you can create a pair (R, S) where $R \cdot k = S$ for whatever R you want, without bothering to create a linear combination. Hence, for this to work it’s absolutely imperative that whoever creates those points is trustworthy and actually deletes k once they created the ten points. **This is where the concept of a “trusted setup” comes from.**

Remember that the solution to a QAP is a set of polynomials (A, B, C) such that $A(x) \cdot B(x) - C(x) = H(x) \cdot Z(x)$, where:

- A is a linear combination of a set of polynomials $\{A_1 \dots A_m\}$
- B is the linear combination of $\{B_1 \dots B_m\}$ with the same coefficients

- C is a linear combination of $\{C_1 \dots C_m\}$ with the same coefficients

The sets $\{A_1 \dots A_m\}$, $\{B_1 \dots B_m\}$ and $\{C_1 \dots C_m\}$ and the polynomial Z are part of the problem statement.

However, in most real-world cases, A , B and C are extremely large; for something with many thousands of circuit gates like a hash function, the polynomials (and the factors for the linear combinations) may have many thousands of terms. Hence, instead of having the prover provide the linear combinations directly, we are going to use the trick that we introduced above to have the prover prove that they are providing something which is a linear combination, but without revealing anything else.

You might have noticed that the trick above works on elliptic curve points, not polynomials. Hence, what actually happens is that we add the following values to the trusted setup:

- $G \cdot A_1(t), G \cdot A_1(t) \cdot k_a$

- $G \cdot A_2(t), G \cdot A_2(t) \cdot k_a$

- ...

- $G \cdot B_1(t), G \cdot B_1(t) \cdot k_b$

- $G \cdot B_2(t), G \cdot B_2(t) \cdot k_b$

- ...

- $G \cdot C_1(t), G \cdot C_1(t) \cdot k_c$

- $G \cdot C_2(t), G \cdot C_2(t) \cdot k_c$

- ...

You can think of t as a "secret point" at which the polynomial is evaluated. G is a "generator" (some random elliptic curve point that is specified as part of the protocol) and t, k_a, k_b and k_c are "toxic waste", numbers that absolutely must be deleted at all costs, or else whoever has them will be able to make fake proofs. Now, if someone gives you a pair of points P, Q such that $P \cdot k_a = Q$ (reminder: we don't need k_a to check this, as we can do a pairing check), then you know that what they are giving you is a linear combination of A_i polynomials evaluated at t .

Hence, so far the prover must give:

- $a = G \cdot A(t), \pi'_a = G \cdot A(t) \cdot k_a$

- $\pi_b = G \cdot B(t), \pi'_b = G \cdot B(t) \cdot k_b$

- $\pi_c = G \cdot C(t), \pi'_c = G \cdot C(t) \cdot k_c$

Note that the prover doesn't actually need to know (and shouldn't know!) t, k_a, k_b or k_c to compute these values; rather, the prover should be able to compute these values just from the points that we're adding to the trusted setup.

The next step is to make sure that all three linear combinations have the same coefficients. This we can do by adding another set of values to the trusted setup: $G \cdot (A_i(t) + B_i(t) + C_i(t)) \cdot b$, where b is another number that should be considered "toxic waste" and discarded as soon as the trusted setup is completed. We can then have the prover create a linear combination with these values with the same coefficients, and use the same pairing trick as above to verify that this value matches up with the provided $A + B + C$.

Finally, we need to prove that $A \cdot B - C = H \cdot Z$. We do this once again with a pairing check:

$$e(\pi_a, \pi_b)/e(\pi_c, G) = e(\pi_h, G \cdot Z(t))$$

Where $\pi_h = G \cdot H(t)$. If the connection between this equation and $A \cdot B - C = H \cdot Z$ does not make sense to you, go back and read the [article on pairings](https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627) (<https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>).

We saw above how to convert A, B and C into elliptic curve points; G is just the generator (ie. the elliptic curve point equivalent of the number one). We can add $G \cdot Z(t)$ to the trusted setup. H is harder; H is just a polynomial, and we predict very little ahead of time about what its coefficients will be for each individual QAP solution.

Hence, we need to add yet more data to the trusted setup; specifically the sequence:

$G, G \cdot t, G \cdot t^2, G \cdot t^3, G \cdot t^4 \dots$

In the Zcash trusted setup, the sequence here goes up to about 2 million; this is how many powers of t you need to make sure that you will always be able to compute $H(t)$, at least for the specific QAP instance that they care about. And with that, the prover can provide all of the information for the verifier to make the final check.

There is one more detail that we need to discuss. Most of the time we don't just want to prove in the abstract that some solution exists for some specific problem; rather, we want to prove either the correctness of some specific solution (eg. proving that if you take the word "cow" and SHA3 hash it a million times, the final result starts with 0x73064fe5), or that a solution exists if you restrict some of the parameters. For example, in a cryptocurrency instantiation where transaction amounts and account balances are encrypted, you want to prove that you know some decryption key k such that:

1. $\text{decrypt}(\text{old_balance}, k) \geq \text{decrypt}(\text{tx_value}, k)$
2. $\text{decrypt}(\text{old_balance}, k) - \text{decrypt}(\text{tx_value}, k) = \text{decrypt}(\text{new_balance}, k)$

The encrypted `old_balance`, `tx_value` and `new_balance` should be specified publicly, as those are the specific values that we are looking to verify at that particular time; only the decryption key should be hidden. Some slight modifications to the protocol are needed to create a "custom verification key" that corresponds to some specific restriction on the inputs.

Now, let's step back a bit. First of all, here's the verification algorithm in its entirety, courtesy of ben [Sasson](#), [Tromer](#), [Virza](#) and [Chiesa](#) (<https://eprint.iacr.org/2013/879.pdf>):

(c) Verifier V

- INPUTS: verification key vk , input $\vec{x} \in \mathbb{F}_r^n$, and proof π
- OUTPUTS: decision bit

1. Compute $\text{vk}_{\vec{x}} := \text{vk}_{\text{IC},0} + \sum_{i=1}^n x_i \text{vk}_{\text{IC},i} \in \mathbb{G}_1$.
2. Check validity of knowledge commitments for A, B, C :

$$\begin{aligned} e(\pi_A, \text{vk}_A) &= e(\pi'_A, \mathcal{P}_2), e(\text{vk}_B, \pi_B) = e(\pi'_B, \mathcal{P}_2), \\ e(\pi_C, \text{vk}_C) &= e(\pi'_C, \mathcal{P}_2). \end{aligned}$$

3. Check same coefficients were used:

$$e(\pi_K, \text{vk}_\gamma) = e(\text{vk}_{\vec{x}} + \pi_A + \pi_C, \text{vk}_{\beta\gamma}^2) \cdot e(\text{vk}_{\beta\gamma}^1, \pi_B).$$

4. Check QAP divisibility:

$$e(\text{vk}_{\vec{x}} + \pi_A, \pi_B) = e(\pi_H, \text{vk}_Z) \cdot e(\pi_C, \mathcal{P}_2).$$

5. Accept if and only if all the above checks succeeded.

The first line deals with parametrization; essentially, you can think of its function as being to create a "custom verification key" for the specific instance of the problem where some of the arguments are specified. The second line is the linear combination check for A, B and C ; the third line is the check that the linear combinations have the same coefficients, and the fourth line is the product check $A \cdot B - C = H \cdot Z$.

Altogether, the verification process is a few elliptic curve multiplications (one for each "public" input variable), and five pairing checks, one of which includes an additional pairing multiplication. The proof contains eight elliptic curve points: a pair of points each for $A(t), B(t)$ and $C(t)$, a point π_k for $b \cdot (A(t) + B(t) + C(t))$, and a point π_h for $H(t)$. Seven of these points are on the \mathbb{F}_p curve (32 bytes each, as you can compress the y coordinate to a single bit), and in the Zcash implementation one point (π_h) is on the twisted curve in \mathbb{F}_{p^2} (64 bytes), so the total size of the proof is ~288 bytes.

The two computationally hardest parts of creating a proof are:

- Dividing $(A \cdot B - C)/Z$ to get H (algorithms based on the [Fast Fourier transform](#) (https://en.wikipedia.org/wiki/Fast_Fourier_transform) can do this in sub-quadratic time, but it's still quite computationally intensive)
- Making the elliptic curve multiplications and additions to create the $A(t), B(t), C(t)$ and $H(t)$ values and their corresponding pairs

The basic reason why creating a proof is so hard is the fact that what was a single binary logic gate in the original computation turns into an operation that must be cryptographically processed through elliptic curve operations if we are making a zero-knowledge proof out of it. This fact, together with the superlinearity of fast Fourier transforms, means that proof creation takes ~20–40 seconds for a Zcash transaction.

Another very important question is: can we try to make the trusted setup a little... less trust-demanding? Unfortunately we can't make it completely trustless; the KzE assumption itself precludes making independent pairs $(P_i, P_i \cdot k)$ without knowing what k is. However, we can increase security greatly by using N-of-N multiparty computation - that is, constructing the trusted setup between N parties in such a way that as long as at least one of the participants deleted their toxic waste then you're okay.

To get a bit of a feel for how you would do this, here's a simple algorithm for taking an existing set $(G, G \cdot t, G \cdot t^2, G \cdot t^3 \dots)$, and "adding in" your own secret so that you need both your secret and the previous secret (or previous set of secrets) to cheat.

The output set is simply:

$$G, (G \cdot t) \cdot s, (G \cdot t^2) \cdot s^2, (G \cdot t^3) \cdot s^3 \dots$$

Note that you can produce this set knowing only the original set and s , and the new set functions in the same way as the old set, except now using $t \cdot s$ as the "toxic waste" instead of t . As long as you and the person (or people) who created the previous set do not both fail to delete your toxic waste and later collude, the set is "safe".

Doing this for the complete trusted setup is quite a bit harder, as there are several values involved, and the algorithm has to be done between the parties in several rounds. It's an area of active research to see if the multiparty computation algorithm can be simplified further and made to require fewer rounds or made more parallelizable, as the more you can do that the more parties it becomes feasible to include into the trusted setup procedure. It's reasonable to see why a trusted setup between six participants who all know and work with each other might make some people uncomfortable, but a trusted setup with thousands of participants would be nearly indistinguishable from no trust at all - and if you're really paranoid, you can get in and participate in the setup procedure yourself, and be sure that you personally deleted your value.

Another area of active research is the use of other approaches that do not use pairings and the same trusted setup paradigm to achieve the same goal; see [Eli ben Sasson's recent presentation](#) (https://www.youtube.com/watch?v=HJ9K_o-RSY), for one alternative (though be warned, it's at least as mathematically complicated as SNARKs are!)

Special thanks to Ariel Gabizon and Christian Reitwiessner for reviewing.

A Note On Charity Through Marginal Price Discrimination

2017 Mar 11

[See all posts \(/\)](#)

Updated 2018-07-28. See end note.

The following is an interesting idea that I had two years ago that I personally believe has promise and could be easily implemented in the context of a blockchain ecosystem, though if desired it could certainly also be implemented with more traditional technologies (blockchains would help get the scheme network effects by putting the core logic on a more neutral platform).

Suppose that you are a restaurant selling sandwiches, and you ordinarily sell sandwiches for \$7.50. Why did you choose to sell them for \$7.50, and not \$7.75 or \$7.25? It clearly can't be the case that the cost of production is exactly \$7.49999, as in that case you would be making no profit, and would not be able to cover fixed costs; hence, in most normal situations you would still be able to make *some* profit if you sold at \$7.25 or \$7.75, though less. Why less at \$7.25? Because the price is lower. Why less at \$7.75? Because you get fewer customers. It just so happens that \$7.50 is the point at which the balance between those two factors is optimal for you.

Price	Demand	Profit (demand * (price - \$6 production cost))
6.8	6600	5280
6.9	6300	5670
7	6000	6000
7.1	5700	6270
7.2	5400	6480
7.3	5100	6630
7.4	4800	6720
7.5	4500	6750
7.6	4200	6720
7.7	3900	6630
7.8	3600	6480
7.9	3300	6270
8	3000	6000
8.1	2700	5670
8.2	2400	5280



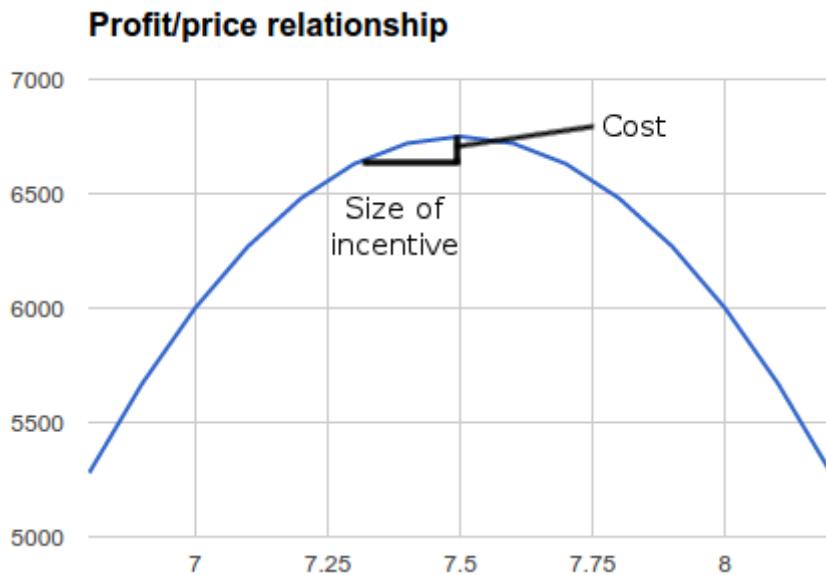
Notice one consequence of this: if you make a *slight* distortion to the optimal price, then even compared to the magnitude of the distortion the losses that you face are minimal. If you raise prices by 1%, from \$7.50 to \$7.575, then your profit declines from \$6750 to \$6733.12—a tiny 0.25% reduction. And that's *profit*—if you had

instead donated 1% of the price of each sandwich, it would have reduced your profit by 5%. The smaller the distortion the more favorable the ratio: raising prices by 0.2% only cuts your profits down by 0.01%.

Now, you could argue that stores are not perfectly rational, and not perfectly informed, and so they may not *actually* be charging at optimal prices, all factors considered. However, if you don't know what direction the deviation is in for any given store, then even still, *in expectation*, the scheme works the same way—except instead of losing \$17 it's more like flipping a coin where half the time you gain \$50 and half the time you lose \$84. Furthermore, in the more complex scheme that we will describe later, we'll be adjusting prices in both directions simultaneously, and so there will not even be any extra risk — no matter how correct or incorrect the original price was, the scheme will give you a predictable small net loss.

Also, the above example was one where marginal costs are high, and customers are picky about prices—in the above model, charging \$9 would have netted you no customers at all. In a situation where marginal costs are much lower (<http://www.thezeromarginalcostsociety.com/>), and customers are less price-sensitive, the losses from raising or lowering prices would be even lower.

So what is the point of all this? Well, suppose that our sandwich shop changes its policy: it sells sandwiches for \$7.55 to the general public, but lowers the prices to \$7.35 for people who volunteered in some charity that maintains some local park (say, this is 25% of the population). The store's new profit is $\$6682.5 \cdot 0.25 + \$6742.5 \cdot 0.75 = \$6727.5$ (that's a \$22.5 loss), but the result is that you are now paying all 4500 of your customers 20 cents each to volunteer at that charity—an incentive size of \$900 (if you just count the customers who actually do volunteer, \$225). So the store loses a bit, but gets a huge amount of leverage, de-facto contributing at least \$225 depending on how you measure it for a cost of only \$22.5.



Now, what we can start to do is build up an ecosystem of "stickers", which are non-transferable digital "tokens" that organizations hand out to people who they think are contributing to worthy causes. Tokens could be organized by category (eg. poverty relief, science research, environmental, local community projects, open source software development, writing good blogs), and merchants would be free to charge marginally lower prices to holders of the tokens that represent whatever causes they personally approve of.

The next stage is to make the scheme recursive — being or working for a merchant that offers lower prices to holders of green stickers is itself enough to merit you a green sticker, albeit one that is of lower potency and gives you a lower discount. This way, if an entire community approves of a particular cause, it may actually be

profit-maximizing to start offering discounts for the associated sticker, and so economic and social pressure will maintain a certain level of spending and participation toward the cause in a stable equilibrium.

As far as implementation goes, this requires:

- A standard for stickers, including wallets where people can hold stickers
- Payment systems that have support for charging lower prices to sticker holders included
- At least a few sticker-issuing organizations (the lowest overhead is likely to be issuing stickers for charity donations, and for easily verifiable online content, eg. open source software and blogs)

So this is something that can certainly be bootstrapped within a small community and user base and then let to grow over time.

Update 2017.03.14: [here](https://github.com/vbuterin/research/blob/master/charity_sim.py) (https://github.com/vbuterin/research/blob/master/charity_sim.py) is an economic model/simulation showing the above implemented as a Python script.

Update 2018.07.28: after discussions with others (Glen Weyl and several Reddit commenters), I realized a few extra things about this mechanism, some encouraging and some worrying:

- The above mechanism could be used not just by charities, but also by centralized corporate actors. For example, a large corporation could offer a bribe of \$40 to any store that offers the 20-cent discount to customers of its products, gaining additional revenue much higher than \$40. So it's empowering but potentially dangerous in the wrong hands... (I have not researched it but I'm sure this kind of technique is used in various kinds of loyalty programs already)
- The above mechanism has the property that a merchant can "donate" $\$x$ to charity at a cost of $\$x^2$ (note: $x^2 < x$ at the scales we're talking about here). This gives it a structure that's economically optimal in certain ways (see [quadratic voting](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2003531) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2003531)), as a merchant that feels twice as strongly about some public good will be inclined to offer twice as large a subsidy, whereas most other social choice mechanisms tend to either undervalue (as in traditional voting) or overvalue (as in buying policies via lobbying) stronger vs weaker preferences.

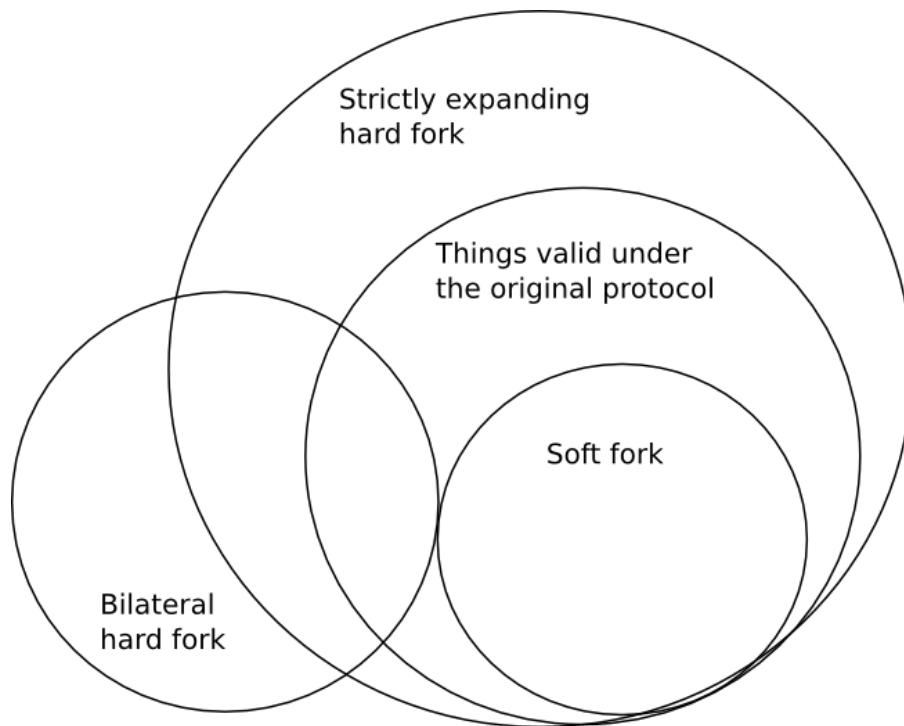
Hard Forks, Soft Forks, Defaults and Coercion

2017 Mar 14

[See all posts \(/\)](#)

One of the important arguments in the blockchain space is that of whether hard forks or soft forks are the preferred protocol upgrade mechanism. The basic difference between the two is that soft forks change the rules of a protocol by *strictly reducing* the set of transactions that is valid, so nodes following the old rules will still get on the new chain (provided that the majority of miners/validators implements the fork), whereas hard forks allow previously invalid transactions and blocks to become valid, so clients must upgrade their clients in order to stay on the hard-forked chain. There are also two sub-types of hard forks: *strictly expanding* hard forks, which strictly expand the set of transactions that is valid, and so effectively the old rules are a soft fork with respect to the new rules, and *bilateral* hard forks, where the two rulesets are incompatible both ways.

Here is a Venn diagram to illustrate the fork types:



The benefits commonly cited for the two are as follows.

- Hard forks allow the developers much more flexibility in making the protocol upgrade, as they do not have to take care to make sure that the new rules "fit into" the old rules
- Soft forks are more convenient for users, as users do not need to upgrade to stay on the chain
- Soft forks are less likely to lead to a chain split
- Soft forks only really require consent from miners/validators (as even if users still use the old rules, if the nodes making the chain use the new rules then only things valid under the new rules will get into the chain)

in any case); hard forks require *opt-in* consent from users

Aside from this, one major criticism often given for hard forks is that hard forks are "coercive". The kind of coercion implied here is not physical force; rather, it's *coercion through network effect*. That is, if the network changes rules from A to B, then even if you personally like A, if most other users like B and switch to B then you have to switch to B despite your personal disapproval of the change in order to be on the same network as everyone else.

Proponents of hard forks are often derided as trying to effect a "hostile take over" of a network, and "force" users to go along with them. Additionally, the risk of chain splits is often used to bill hard forks as "unsafe".

It is my personal viewpoint that these criticisms are wrong, and furthermore in many cases completely backwards. This viewpoint is not specific to Ethereum, or Bitcoin, or any other blockchain; it arises out of general properties of these systems, and is applicable to any of them. Furthermore, the arguments below only apply to controversial changes, where a large portion of at least one constituency (miners/validators and users) disapprove of them; if a change is non-contentious, then it can generally be done safely no matter what the format of the fork is.

First of all, let us discuss the question of coercion. Hard forks and soft forks both change the protocol in ways that some users may not like; *any* protocol change will do this if it has less than exactly 100% support. Furthermore, it is almost inevitable that at least *some* of the dissenters, in any scenario, value the network effect of sticking with the larger group more than they value their own preferences regarding the protocol rules. Hence, both fork types are coercive, in the network-effect sense of the word.

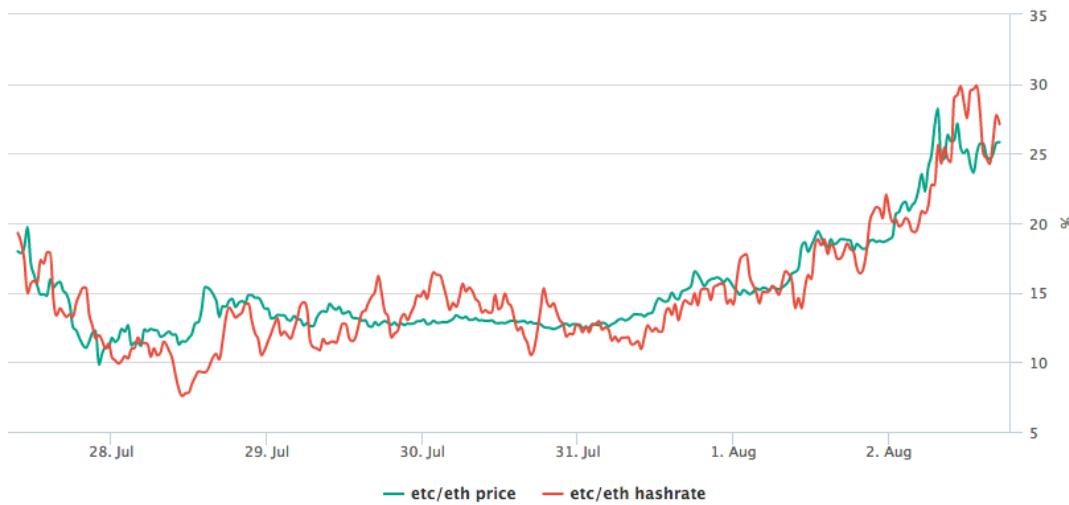
However, there is an essential difference between hard forks and soft forks: *hard forks are opt-in, whereas soft forks allow users no "opting" at all*. In order for a user to join a hard forked chain, they must personally install the software package that implements the fork rules, and the set of users that disagrees with a rule change even more strongly than they value network effects can theoretically simply stay on the old chain - and, practically speaking, such an event has already happened (<https://ethereumclassic.github.io/>).

This is true in the case of both strictly expanding hard forks and bilateral hard forks. In the case of soft forks, however, *if the fork succeeds the unforked chain does not exist*. Hence, **soft forks clearly institutionally favor coercion over secession, whereas hard forks have the opposite bias**. My own moral views lead me to favor secession over coercion, though others may differ (the most common argument raised is that network effects are really really important and it is essential that "one coin rule them all" (<https://blog.ethereum.org/2014/11/20/bitcoin-maximalism-currency-platform-network-effects/>), though more moderate versions of this also exist).

If I had to guess why, despite these arguments, soft forks are often billed as "less coercive" than hard forks, I would say that it is because it feels like a hard fork "forces" the user into installing a software update, whereas with a soft fork users do not "have" to do anything at all. However, this intuition is misguided: what matters is not whether or not individual users have to perform the simple bureaucratic step of clicking a "download" button, but rather whether or not the user is coerced *into accepting a change in protocol rules* that they would rather not accept. And by this metric, as mentioned above, both kinds of forks are ultimately coercive, and it is hard forks that come out as being somewhat better at preserving user freedom.

Now, let's look at highly controversial forks, particularly forks where miner/validator preferences and user preferences conflict. There are three cases here: (i) bilateral hard forks, (ii) strictly expanding hard forks, and (iii) so-called "user-activated soft forks" (UASF). A fourth category is where miners activate a soft fork *without user consent*; we will get to this later.

First, bilateral hard forks. In the best case, the situation is simple. The two coins trade on the market, and traders decide the relative value of the two. From the ETC/ETH case, we have overwhelming evidence that miners are overwhelmingly likely to simply assign their hashrate to coins based on the ratio of prices in order to maximize their profit, regardless of their own ideological views.



Even if some miners profess ideological preferences toward one side or the other, it is overwhelmingly likely that there will be enough miners that are willing to arbitrage any mismatch between price ratio and hashpower ratio, and bring the two into alignment. If a cartel of miners tries to form to not mine on one chain, there are overwhelming incentives to defect.

There are two edge cases here. The first is the possibility that, because of an inefficient difficulty adjustment algorithm, the value of mining the coin goes down because price drops but difficulty does not go down to compensate, making mining very unprofitable, and there are no miners willing to mine at a loss to keep pushing the chain forward until its difficulty comes back into balance. This was not the case with Ethereum, but may well be the case with Bitcoin

([https://www.reddit.com/r/Bitcoin/comments/3axspf/doesnt the lag in difficulty adjustment mean any/](https://www.reddit.com/r/Bitcoin/comments/3axspf/doesnt_the_lag_in_difficulty_adjustment_mean_any/)). Hence, the minority chain may well simply never get off the ground, and so it will die. Note that the normative question of whether or not this is a good thing depends on your views on coercion versus secession; as you can imagine from what I wrote above I personally believe that such minority-chain-hostile difficulty adjustment algorithms are bad.

The second edge case is that if the disparity is very large, the large chain can 51% attack the smaller chain. Even in the case of an ETH/ETC split with a 10:1 ratio, this has not happened; so it is certainly not a given. However, it is always a possibility if miners on the dominant chain prefer coercion to allowing secession and act on these values.

Next, let's look at strictly expanding hard forks. In an SEHF, there is the property that the non-forked chain is valid under the forked rules, and so if the fork has a lower price than the non-forked chain, it will have less hashpower than the non-forked chain, and so the non-forked chain will end up being accepted as the longest chain *by both original-client and forked-client rules* - and so the forked chain "will be annihilated (<https://twitter.com/SatoshiLite/status/839673905627353088>)".

There is an argument that there is thus a strong inherent bias against such a fork succeeding, as the possibility that the forked chain will get annihilated will be baked into the price, pushing the price lower, making it even more likely that the chain will be annihilated... This argument to me seems strong, and so it is a very good reason to make *any* contentious hard fork bilateral rather than strictly expanding.

Bitcoin Unlimited developers suggest dealing with this problem by making the hard fork bilateral manually (<https://medium.com/@g.andrew.stone/what-if-3a48100a6c18#.882uzyyvs>), after it happens, but a better choice would be to make the bilaterality built-in; for example, in the bitcoin case, one can add a rule to ban some unused opcode, and then make a transaction containing that opcode on the non-forked chain, so that under the forked rules the non-forked chain will from then on be considered forever invalid. In the Ethereum case, because of various details about how state calculation works, nearly all hard forks are bilateral almost automatically. Other chains may have different properties depending on their architecture.

The last type of fork that was mentioned above is the user-activated soft fork. In a UASF, users turn on the soft fork rules without bothering to get consensus from miners; miners are expected to simply fall in line out of economic interest. If many users do not go along with the UASF, then there will be a coin split, and this will lead to a scenario identical to the strictly expanding hard fork, except - and this is the really clever and devious part of the concept - *the same "risk of annihilation" pressure that strongly disfavors the forked chain in a strictly expanding hard fork instead strongly favors the forked chain in a UASF*. Even though a UASF is opt-in, it uses economic asymmetry in order to bias itself toward success (though the bias is not absolute; if a UASF is decidedly unpopular then it will not succeed and will simply lead to a chain split).

However, UASFs are a dangerous game. For example, let us suppose that the developers of a project want to make a UASF patch that converts an unused opcode that previously accepted all transactions into an opcode that only accepts transactions that comply with the rules of some cool new feature, though one that is politically or technically controversial and miners dislike. Miners have a clever and devious way to fight back: *they can unilaterally implement a miner-activated soft fork that makes all transactions using the feature created by the soft fork always fail*.

Now, we have three rulesets:

1. The original rules where opcode X is always valid.
2. The rules where opcode X is only valid if the rest of the transaction complies with the new rules
3. The rules where opcode X is always invalid.

Note that (2) is a soft-fork with respect to (1), and (3) is a soft-fork with respect to (2). Now, there is strong economic pressure in favor of (3), and so the soft-fork fails to accomplish its objective.

The conclusion is this. Soft forks are a dangerous game, and they become even more dangerous if they are contentious and miners start fighting back. Strictly expanding hard forks are also a dangerous game. Miner-activated soft forks are coercive; user-activated soft forks are less coercive, though still quite coercive because of the economic pressure, and they also have their dangers. If you really want to make a contentious change, and have decided that the high social costs of doing so are worth it, just do a clean bilateral hard fork, spend some time to add some proper replay protection, and let the market sort it out.

Engineering Security Through Coordination Problems

2017 May 08

[See all posts \(/\)](#)

Recently, there was a small spat between the Core and Unlimited factions of the Bitcoin community, a spat which represents perhaps the fiftieth time the same theme was debated, but which is nonetheless interesting because of how it highlights a very subtle philosophical point about how blockchains work.

ViaBTC, a mining pool that favors Unlimited, [tweeted](https://i.redd.it/x9f7t3rhn4wy.png) ("hashpower is law"), a usual talking point for the Unlimited side, which believes that miners have, and should have, a very large role in the governance of Bitcoin, the usual argument for this being that miners are the one category of users that has a large and illiquid financial incentive in Bitcoin's success. Greg Maxwell (from the Core side) [replied](https://np.reddit.com/r/Bitcoin/comments/69t452/viabtc_comment_to_the_recent_segwit_pool/dh95hat/) that "Bitcoin's security works precisely because hash power is NOT law".

The Core argument is that miners only have a limited role in the Bitcoin system, to secure the ordering of transactions, and they should NOT have the power to determine anything else, including block size limits and other block validity rules. These constraints are enforced by full nodes run by users - if miners start producing blocks according to a set of rules different than the rules that users' nodes enforce, then the users' nodes will simply reject the blocks, regardless of whether 10% or 60% or 99% of the hashpower is behind them. To this, Unlimited often replies with something like "if 90% of the hashpower is behind a new chain that increases the block limit, and the old chain with 10% hashpower is now ten times slower for five months until difficulty readjusts, would you *really* not update your client to accept the new chain?"

Many people often [argue](http://www.ofnumbers.com/2015/07/27/what-is-permissioned-on-permissionless/) against [the use of public blockchains for applications that involve real-world assets or anything with counterparty risk](http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/). The critiques are either total, saying that there is no point in implementing such use cases on public blockchains, or partial, saying that while there may be advantages to storing the *data* on a public chain, the *business logic* should be executed off chain.

The argument usually used is that in such applications, points of trust exist already - there is someone who owns the physical assets that back the on-chain permissioned assets, and that someone could always choose to run away with the assets or be compelled to freeze them by a government or bank, and so managing the digital representations of these assets on a blockchain is like paying for a reinforced steel door for one's house when the window is open. Instead, such systems should use private chains, or even traditional server-based solutions, perhaps adding bits and pieces of cryptography to improve auditability, and thereby save on the inefficiencies and costs of putting everything on a blockchain.

The arguments above are both flawed in their pure forms, and they are flawed in a similar way. While it is *theoretically possible* for miners to switch 99% of their hashpower to a chain with new rules (to make an example where this is uncontroversially bad, suppose that they are increasing the block reward), and even spawn-camp (<https://twitter.com/vitalikbuterin/status/827783678910558208>), the old chain to make it permanently useless, and it is also theoretically possible for a centralized manager of an asset-backed currency to cease honoring one digital token, make a new digital token with the same balances as the old token except with one particular account's balance reduced to zero, and start honoring the new token, in practice *those things are both quite hard to do*.

In the first case, users will have to realize that something is wrong with the existing chain, agree that they should go to the new chain that the miners are now mining on, and download the software that accepts the new rules. In the second case, all clients and applications that depend on the original digital token will break, users will need to update their clients to switch to the new digital token, and smart contracts with no capacity to look to the outside world and see that they need to update will break entirely. In the middle of all this, opponents of the switch can create a fear-uncertainty-and-doubt campaign to try to convince people that maybe they shouldn't update their clients after all, or update their client to some *third* set of rules (eg. changing proof of work), and this makes implementing the switch even more difficult.

Hence, we can say that in both cases, even though there theoretically are centralized or quasi-centralized parties that could force a transition from state A to state B, where state B is disagreeable to users but preferable to the centralized parties, doing so requires **breaking through a hard coordination problem**. Coordination problems are everywhere in society and are often a bad thing - while it would be better for most people if the English language got rid of its highly complex and irregular spelling system and made a phonetic one, or if the United States switched to metric, or if we could immediately drop all prices and wages by ten percent in the event of a recession (<http://www.interfluidity.com/v2/6088.html>), in practice this requires everyone to agree on the switch at the same time, and this is often very very hard.

With blockchain applications, however, we are doing something different: **we are using coordination problems to our advantage**, using the friction that coordination problems create as a bulwark against malfeasance by centralized actors. We can build systems that have property X, and we can guarantee that they will preserve property X to a high degree because changing the rules from X to not-X would require a whole bunch of people to agree to update their software at the same time. Even if there is an actor that could force the change, doing so would be hard. This is the kind of security that you gain from client-side validation of blockchain consensus rules.

Note that this kind of security relies on the decentralization of users specifically. Even if there is only one miner in the world, there is still a difference between a cryptocurrency mined by that miner and a PayPal-like centralized system. In the latter case, the operator can choose to arbitrarily change the rules, freeze people's money, offer bad service, jack up their fees or do a whole host of other things, and the coordination problems are in the operator's favor, as such systems have substantial network effects and so very many users would have to agree at the same time to switch to a better system. In the former case, client-side validation means that many attempts at mischief that the miner might want to engage in are by default rejected, and the coordination problem now works in the users' favor.

Note that the arguments above do NOT, *by themselves*, imply that it is a bad idea for miners to be the principal actors coordinating and deciding the block size (or in Ethereum's case, the gas limit). It may well be the case

that, *in the specific case of the block size/gas limit*, "government by coordinated miners with aligned incentives" is the optimal approach for deciding this one particular policy parameter, perhaps because the risk of miners abusing their power is lower than the risk that any specific chosen hard limit will prove wildly inappropriate for market conditions a decade after the limit is set. However, there is nothing unreasonable about saying that government-by-miners is the best way to decide one policy parameter, and at the same time saying that *for other parameters* (eg. block reward) we want to rely on client-side validation to ensure that miners are constrained. This is the essence of engineering decentralized institutions: it is about strategically using coordination problems to ensure that systems continue to satisfy certain desired properties.

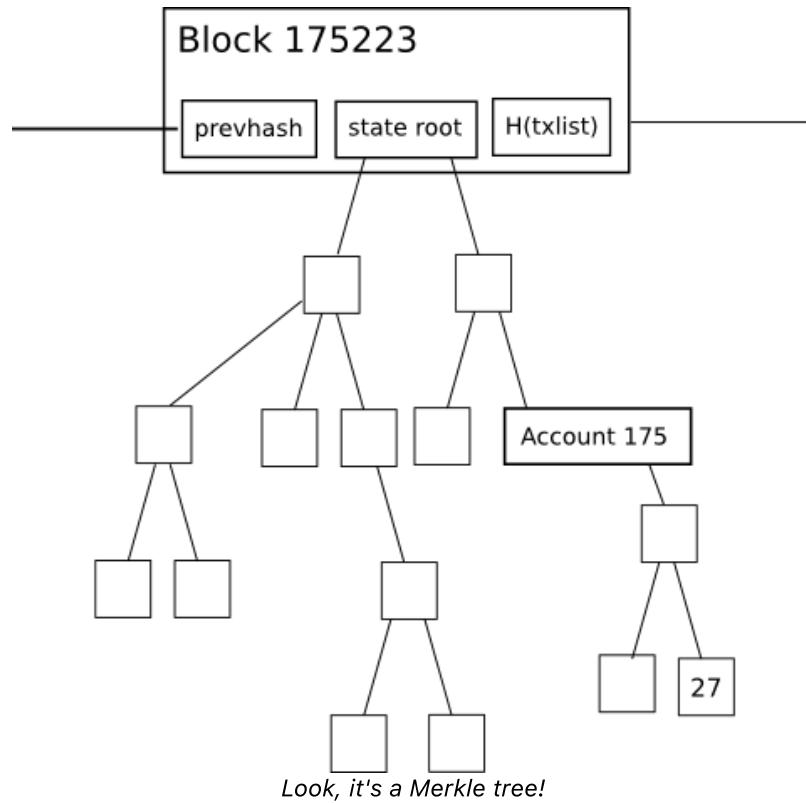
The arguments above also do not imply that it is always optimal to try to put everything onto a blockchain even for services that are trust-requiring. There generally are at least some gains to be made by running more business logic on a blockchain, but they are often much smaller than the losses to efficiency or privacy. And this is ok; the blockchain is not the best tool for every task. What the arguments above *do* imply, though, is that if you are building a blockchain-based application that contains many centralized components out of necessity, then you can make substantial further gains in trust-minimization by giving users a way to access your application through a regular blockchain client (eg. in the case of Ethereum, this might be Mist, Parity, Metamask or Status), instead of getting them to use a web interface that you personally control.

Theoretically, the benefits of user-side validation are optimized if literally every user runs an independent "ideal full node" - a node that accepts all blocks that follow the protocol rules that everyone agreed to when creating the system, and rejects all blocks that do not. In practice, however, this involves asking every user to process every transaction run by everyone in the network, which is clearly untenable, especially keeping in mind the rapid growth of smartphone users in the developing world.

There are two ways out here. The first is that we can realize that while it is *optimal* from the point of view of the above arguments that everyone runs a full node, it is certainly not *required*. Arguably, any major blockchain running at full capacity will have already reached the point where it will not make sense for "the common people" to expend a fifth of their hard drive space to run a full node, and so the remaining users are hobbyists and businesses. As long as there is a fairly large number of them, and they come from diverse backgrounds, the coordination problem of getting these users to collude will still be very hard.

Second, we can rely on **strong light client technology**.

There are two levels of "light clients" that are generally possible in blockchain systems. The first, weaker, kind of light client simply convinces the user, with some degree of economic assurance, that they are on the chain that is supported by the majority of the network. This can be done much more cheaply than verifying the entire chain, as all clients need to do is in proof of work schemes verify nonces or in proof stake schemes verify signed certificates that state "either the root hash of the state is what I say it is, or you can publish this certificate into the main chain to delete a large amount of my money". Once the light client verifies a root hash, they can use Merkle trees to verify any specific piece of data that they might want to verify.



The second level is a "nearly fully verifying" light client. This kind of client doesn't just try to follow the chain that the majority follows; rather, it also tries to follow only chains that follow all the rules. This is done by a combination of strategies; the simplest to explain is that a light client can work together with specialized nodes (credit to Gavin Wood for coming up with the name "fishermen") whose purpose is to look for blocks that are invalid and generate "fraud proofs", short messages that essentially say "Look! This block has a flaw over here!". Light clients can then verify that specific part of a block and check if it's actually invalid.

If a block is found to be invalid, it is discarded; if a light client does not hear any fraud proofs for a given block for a few minutes, then it assumes that the block is probably legitimate. There's a bit more complexity (<https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding>) involved in handling the case where the problem is not data that is *bad*, but rather data that is *missing*, but in general it is possible to get quite close to catching all possible ways that miners or validators can violate the rules of the protocol.

Note that in order for a light client to be able to efficiently validate a set of application rules, those rules must be executed inside of consensus - that is, they must be either part of the protocol or part of a mechanism executing inside the protocol (like a smart contract). This is a key argument in favor of using the blockchain for both data storage and business logic execution, as opposed to just data storage.

These light client techniques are imperfect, in that they do rely on assumptions about network connectivity and the number of other light clients and fishermen that are in the network. But it is actually not crucial for them to work 100% of the time for 100% of validators. Rather, all that we want is to create a situation where any attempt by a hostile cartel of miners/validators to push invalid blocks without user consent will cause a large amount of headaches for lots of people and ultimately require everyone to update their software if they want to continue to synchronize with the invalid chain. As long as this is satisfied, we have achieved the goal of security through coordination frictions.

Analyzing Token Sale Models

2017 Jun 09

[See all posts \(/\)](#)

Note: I mention the names of various projects below only to compare and contrast their token sale mechanisms; this should NOT be taken as an endorsement or criticism of any specific project as a whole. It's entirely possible for any given project to be total trash as a whole and yet still have an awesome token sale model.

The last few months have seen an increasing amount of innovation in token sale models. Two years ago, the space was simple: there were capped sales, which sold a fixed number of tokens at a fixed price and hence fixed valuation and would often quickly sell out, and there were uncapped sales, which sold as many tokens as people were willing to buy. Now, we have been seeing a surge of interest, both in terms of theoretical investigation and in many cases real-world implementation, of hybrid capped sales, reverse dutch auctions, Vickrey auctions, proportional refunds, and many other mechanisms.

Many of these mechanisms have arisen as responses to perceived failures in previous designs. Nearly every significant sale, including Brave's Basic Attention Tokens, Gnosis, upcoming sales such as Bancor, and older ones such as Maidsafe and even the Ethereum sale itself, has been met with a substantial amount of criticism, all of which points to a simple fact: so far, we have still not yet discovered a mechanism that has all, or even most, of the properties that we would like.

Let us review a few examples.

Maidsafe



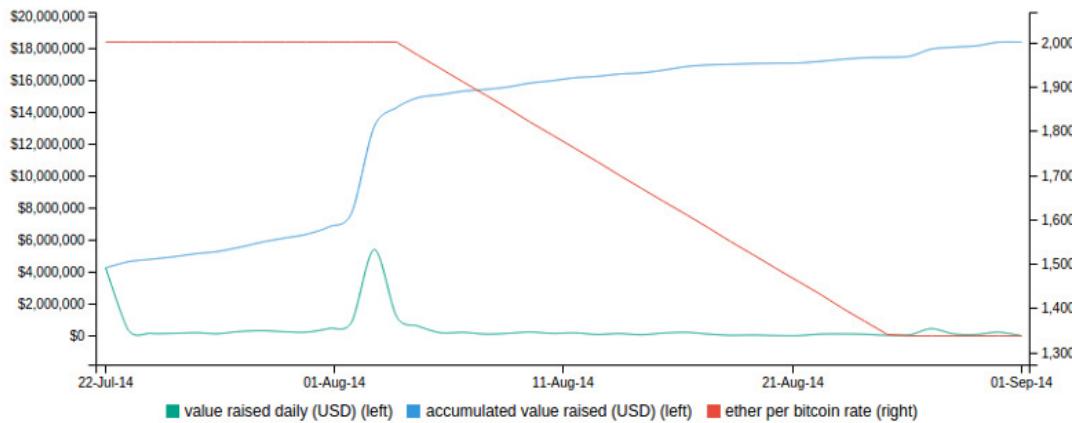
The decentralized internet platform (<http://maidsafe.net/>) raised \$7m in five hours (<https://www.forbes.com/sites/kashmirhill/2014/06/03/mastercoin-maidsafe-crowdsale/#7fda1c71207d>). However, they made the mistake of accepting payment in two currencies (BTC and MSC), and giving a favorable rate to MSC buyers. This led to (<https://www.cryptocoinsnews.com/maidsafe-embroiled-safecoin-presale-mastercoin-pump-dump/>), a temporary ~2x appreciation in the MSC price, as users rushed in to buy MSC to participate in the sale at the

more favorable rate, but then the price saw a similarly steep drop after the sale ended. Many users converted their BTC to MSC to participate in the sale, but then the sale closed too quickly for them, leading to them being stuck with a ~30% loss.

This sale, and several others after it (cough cough [WeTrust](https://blog.wetrust.io/eth-and-btc-contribution-dcd16876bf28) (<https://blog.wetrust.io/eth-and-btc-contribution-dcd16876bf28>), [TokenCard](https://www.cryptocoinsnews.com/ethereum-based-debit-card-tokencard-ico-raises-12-7-million-30-minutes/) (<https://www.cryptocoinsnews.com/ethereum-based-debit-card-tokencard-ico-raises-12-7-million-30-minutes/>)), showed a lesson that should hopefully by now be uncontroversial: running a sale that accepts multiple currencies at a fixed exchange rate is dangerous and bad. Don't do it.

Ethereum

The Ethereum sale was uncapped, and ran for 42 days. The sale price was 2000 ETH for 1 BTC for the first 14 days, and then started increasing linearly, finishing at 1337 ETH for 1 BTC.



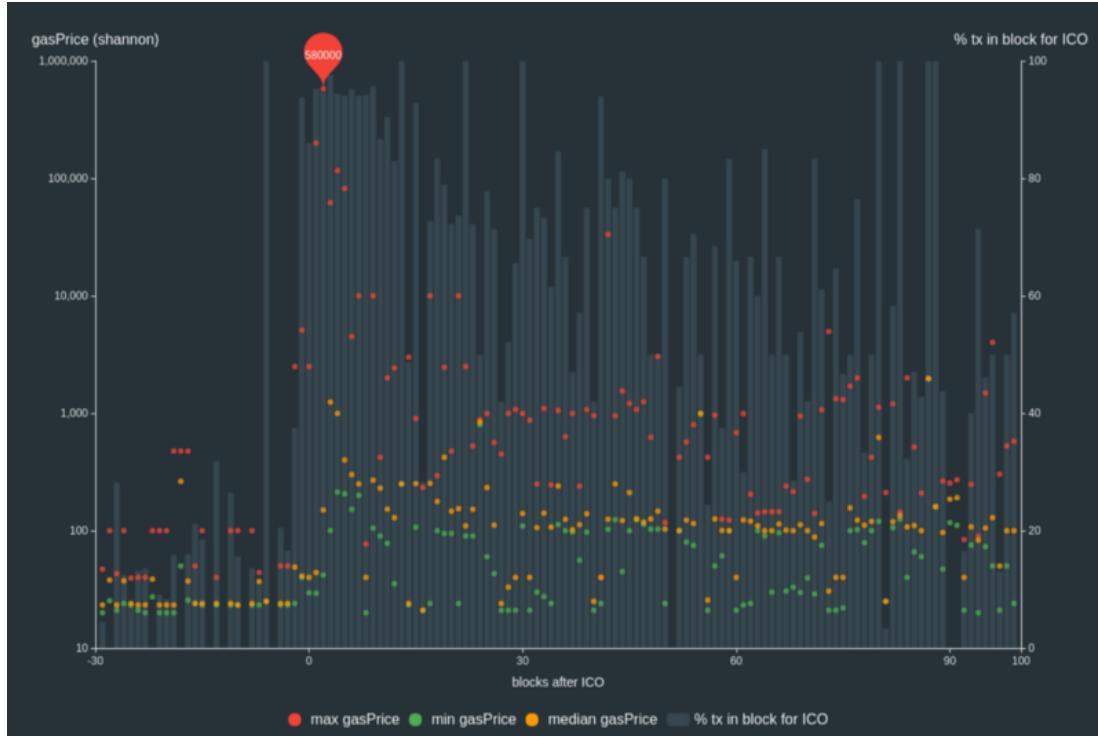
Nearly every uncapped sale is criticized for being "greedy" (a criticism I have significant reservations about, but we'll get back to this later), though there is also another more interesting criticism of these sales: they give participants *high uncertainty about the valuation* that they are buying at. To use a not-yet-started sale as an example, there are likely many people who would be willing to pay \$10,000 for a pile of Bancor tokens if they knew for a fact that this pile represented 1% of all Bancor tokens in existence, but many of them would become quite apprehensive if they were buying a pile of, say, 5000 Bancor tokens, and they had no idea whether the total supply would be 50000, 500000 or 500 million.

In the Ethereum sale, buyers who really cared about predictability of valuation generally bought on the 14th day, reasoning that this was the last day of the full discount period and so on this day they had maximum predictability together with the full discount, but the pattern above is hardly economically optimal behavior; the equilibrium would be something like everyone buying in on the last hour of the 14th day, making a private tradeoff between certainty of valuation and taking the 1.5% hit (or, if certainty was really important, purchases could spill over into the 15th, 16th and later days). Hence, the model certainly has some rather weird economic properties that we would really like to avoid if there is a convenient way to do so.

BAT

Throughout 2016 and early 2017, the capped sale design was most popular. Capped sales have the property that it is very likely that interest is oversubscribed, and so there is a large incentive to getting in first. Initially, sales took a few hours to finish. However, soon the speed began to accelerate. First Blood made a lot of news by finishing their \$5.5m sale in [two minutes](http://themerkle.com/ethereum-based-esports-platform-firstblood-raises-us5-5m-in-mere-minutes/) (<http://themerkle.com/ethereum-based-esports-platform-firstblood-raises-us5-5m-in-mere-minutes/>) - while [active denial-of-service attacks](https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/) (<https://blog.ethereum.org/2016/09/22/ethereum-network-currently-undergoing-dos-attack/>) on the Ethereum

blockchain were taking place. However, the apotheosis of this race-to-the-Nash-equilibrium did not come until the BAT sale last month, when a \$35m sale was completed within 30 seconds (<https://techcrunch.com/2017/06/01/brave-ico-35-million-30-seconds-brendan-eich/>), due to the large amount of interest in the project.



Not only did the sale finish within two blocks, but also:

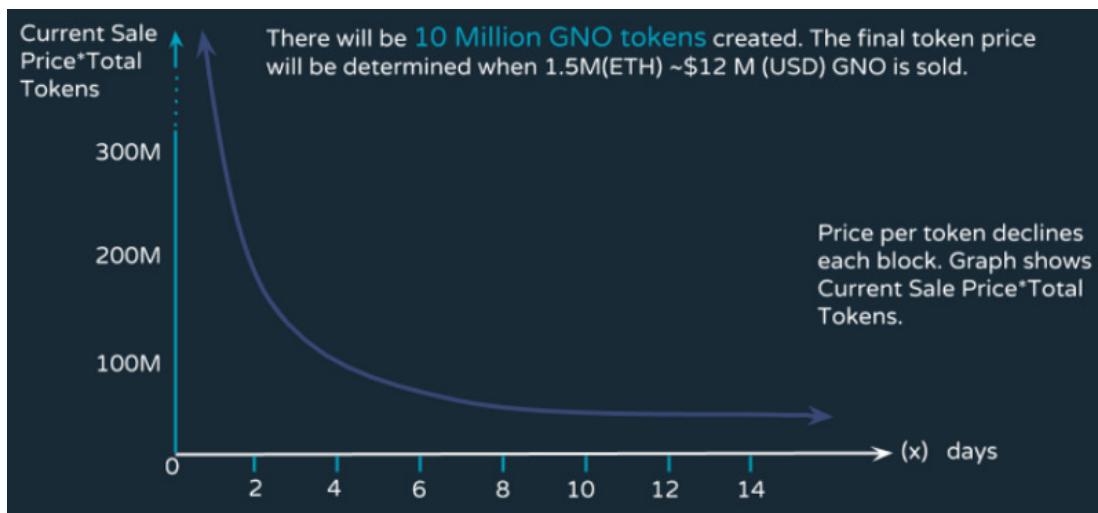
- The total transaction fees paid were 70.15 ETH (<https://medium.com/the-bitcoin-podcast-blog/a-look-at-the-bat-token-distribution-bb3bcb92748f>) (>\$15,000), with the highest single fee being ~\$6,600
- 185 purchases were successful, and over 10,000 failed
- The Ethereum blockchain's capacity was full for 3 hours after the sale started

Thus, we are starting to see capped sales approach their natural equilibrium: people trying to outbid each other's transaction fees, to the point where potentially millions of dollars of surplus would be burned into the hands of miners. And that's before the next stage starts: large mining pools butting into the start of the line and just buying up all of the tokens themselves before anyone else can.

Gnosis

The Gnosis sale attempted to alleviate these issues with a novel mechanism: the reverse dutch auction. The terms, in simplified form, are as follows. There was a capped sale, with a cap of \$12.5 million USD. However, the portion of tokens that would actually be given to purchasers depended on how long the sale took to finish. If it finished on the first day, then only ~5% of tokens would be distributed among purchasers, and the rest held by the Gnosis team; if it finished on the second day, it would be ~10%, and so forth.

The purpose of this is to create a scheme where, if you buy at time T, then you are guaranteed to buy in at a valuation which is at most $\frac{1}{T}$.



The goal is to create a mechanism where the optimal strategy is simple. First, you personally decide what is the highest valuation you would be willing to buy at (call it V). Then, when the sale starts, you don't buy in immediately; rather, you wait until the valuation drops to below that level, and then send your transaction.

There are two possible outcomes:

1. The sale closes before the valuation drops to below V . Then, you are happy because you stayed out of what you thought is a bad deal.
2. The sale closes after the valuation drops to below V . Then, you sent your transaction, and you are happy because you got into what you thought is a good deal.

However, many people predicted that because of "fear of missing out" (FOMO), many people would just "irrationally" buy in at the first day, without even looking at the valuation. And this is exactly what happened: the sale finished in a few hours, with the result that the sale reached its cap of \$12.5 million when it was only selling about 5% of all tokens that would be in existence - an implied valuation of over \$300 million (<http://www.trustnodes.com/2017/04/24/ethereum-based-gnosis-ico-sells-10-minutes-300-million-evaluation>).

All of this would of course be an excellent piece of confirming evidence for the narrative that markets are totally irrational, people don't think clearly before throwing in large quantities of money (and often, as a subtext, that the entire space needs to be somehow suppressed to prevent further exuberance) if it weren't for one inconvenient fact: ***the traders who bought into the sale were right.***



Even in ETH terms, despite the massive ETH price rise, the price of 1 GNO has increased from ~0.6 ETH to ~0.8 ETH.

What happened? A couple of weeks before the sale started, facing public criticism that if they end up holding the majority of the coins they would act like a central bank with the ability to heavily manipulate GNO prices, the Gnosis team agreed to hold 90% of the coins that were not sold for a year. From a trader's point of view, coins that are locked up for a long time are coins that cannot affect the market, and so in a short term analysis, might as well not exist. This is what initially propped up Steem to such a high valuation last year in July (<http://coinmarketcap.com/currencies/steem/#charts>), as well as Zcash in the very early moments when the price of each coin was over \$1,000 (<http://www.coindesk.com/what-is-the-value-zcash-market-searches-answers/>).

Now, one year is not *that* long a time, and locking up coins for a year is nowhere close to the same thing as locking them up forever. However, the reasoning goes further. Even after the one year holding period expires, you can argue that it is in the Gnosis team's interest to only release the locked coins if they believe that doing so will make the price go up, and so if you trust the Gnosis team's judgement this means that they are going to do something *which is at least as good for the GNO price as simply locking up the coins forever*. Hence, in reality, the GNO sale was really much more like a capped sale with a cap of \$12.5 million and a valuation of \$37.5 million. And the traders who participated in the sale reacted exactly as they should have, leaving scores of internet commentators wondering what just happened.

There is certainly a weird bubbliness about crypto-assets, with various no-name assets (<http://coinmarketcap.com/>) attaining market caps of \$1-100 million (including BitBean (<http://coinmarketcap.com/currencies/bitbean/>), as of the time of this writing at \$12m, PotCoin (<http://coinmarketcap.com/currencies/potcoin/>), at \$22m, PepeCash (<http://coinmarketcap.com/assets/pepe-cash/>), at \$13m and SmileyCoin (<http://coinmarketcap.com/currencies/smileycoin/>), at \$14.7m) just because. However, there's a strong case to be made that the participants *at the sale stage* are in many cases doing nothing wrong, at least for themselves; rather, traders who buy in sales are simply (correctly) predicting the existence of an ongoing bubble has been brewing since the start of 2015 (and arguably, since the start of 2010).

More importantly though, bubble behavior aside, there is another legitimate criticism of the Gnosis sale: despite their 1-year no-selling promise, eventually they will have access to the entirety of their coins, and they **will** to a limited extent be able to act like a central bank with the ability to heavily manipulate GNO prices, and traders will have to deal with all of the monetary policy uncertainty that that entails.

Specifying the problem

So what would a **good** token sale mechanism look like? One way that we can start off is by looking through the criticisms of existing sale models that we have seen and coming up with a list of desired properties.

Let's do that. Some natural properties include:

1. **Certainty of valuation** - if you participate in a sale, you should have certainty over at least a ceiling on the valuation (or, in other words, a floor on the percentage of all tokens you are getting).
2. **Certainty of participation** - if you try to participate in a sale, you should be able to generally count on succeeding.
3. **Capping the amount raised** - to avoid being perceived as greedy (or possibly to mitigate risk of regulatory attention), the sale should have a limit on the amount of money it is collecting.
4. **No central banking** - the token sale issuer should not be able to end up with an unexpectedly very large percentage of the tokens that would give them control over the market.

5. Efficiency - the sale should not lead to substantial economic inefficiencies or deadweight losses.

Sounds reasonable?

Well, here's the not-so-fun part.

- (1) and (2) cannot be fully satisfied simultaneously.
- At least without resorting to very clever tricks, (3), (4) and (5) cannot be satisfied simultaneously.

These can be cited as "the first token sale dilemma" and "the second token sale trilemma".

The proof for the first dilemma is easy: suppose you have a sale where you provide users with certainty of a \$100 million valuation. Now, suppose that users try to throw \$101 million into the sale. At least some will fail. The proof for the second trilemma is a simple supply-and-demand argument. If you satisfy (4), then you are selling all, or some fixed large percentage, of the tokens, and so the valuation you are selling at is proportional to the price you are selling at. If you satisfy (3), then you are putting a cap on the price. However, this implies the possibility that the equilibrium price at the quantity you are selling exceeds the price cap that you set, and so you get a shortage, which inevitably leads to either (i) the digital equivalent of standing in line for 4 hours at a very popular restaurant, or (ii) the digital equivalent of ticket scalping - both large deadweight losses, contradicting (5).

The first dilemma cannot be overcome; some valuation uncertainty or participation uncertainty is inescapable, though when the choice exists it seems better to try to choose participation uncertainty rather than valuation uncertainty. The closest that we can come is compromising on *full participation to guarantee partial participation*. This can be done with a proportional refund (eg. if \$101 million buy in at a \$100 million valuation, then everyone gets a 1% refund). We can also think of this mechanism as being an uncapped sale where part of the payment comes in the form of *locking up* capital rather than spending it; from this viewpoint, however, it becomes clear that the requirement to lock up capital is an efficiency loss, and so such a mechanism fails to satisfy (5). If ether holdings are not well-distributed then it arguably harms fairness by favoring wealthy stakeholders.

The second dilemma is difficult to overcome, and many attempts to overcome it can easily fail or backfire. For example, the Bancor sale is considering limiting the transaction gas price for purchases to 50 shannon (~12x the normal gasprice). However, this now means that the optimal strategy for a buyer is to set up a large number of accounts, and from each of those accounts send a transaction that triggers a contract, which then attempts to buy in (the indirection is there to make it impossible for the buyer to accidentally buy in more than they wanted, and to reduce capital requirements). The more accounts a buyer sets up, the more likely they are to get in. Hence, in equilibrium, this could lead to even *more* clogging of the Ethereum blockchain than a BAT-style sale, where at least the \$6600 fees were spent on a single transaction and not an entire denial-of-service attack on the network. Furthermore, any kind of on-chain transaction spam contest severely harms fairness, because the cost of participating in the contest is constant, whereas the reward is proportional to how much money you have, and so the result disproportionately favors wealthy stakeholders.

Moving forward

There are three more clever things that you can do. First, you can do a reverse dutch auction just like Gnosis, but with one change: instead of holding the unsold tokens, put them toward some kind of public good. Simple examples include: (i) airdrop (ie. redistributing to all ETH holders), (ii) donating to the Ethereum Foundation (<https://ethereum.org/donate>), (iii) donating to Parity (<http://parity.io>), Brainbot (<http://www.brainbot.com/>), Smartpool

(<http://smartpool.io/>) or other companies and individuals independently building infrastructure for the Ethereum space, or (iv) some combination of all three, possibly with the ratios somehow being voted on by the token buyers.

Second, you can keep the unsold tokens, but solve the "central banking" problem by committing to a fully automated plan for how they would be spent. The reasoning here is similar to that for why many economists are interested in [rules-based monetary policy](https://www.mercatus.org/system/files/Salter-Monetary-PolicyRules.pdf) (<https://www.mercatus.org/system/files/Salter-Monetary-PolicyRules.pdf>): even if a centralized entity has a large amount of control over a powerful resource, much of the political uncertainty that results can be mitigated if the entity credibly commits to following a set of programmatic rules for how they apply it. For example, the unsold tokens can be put into a market maker that is tasked with preserving the tokens' price stability.

Third, you can do a capped sale, where you limit the amount that can be bought by each person. Doing this effectively requires a KYC process, but the nice thing is a KYC entity can do this once, whitelisting users' addresses after they verify that the address represents a unique individual, and this can then be reused for every token sale, alongside other applications that can benefit from per-person sybil resistance like [Akasha's](http://akasha.world) (<http://akasha.world>), [quadratic voting](http://ericposner.com/quadratic-voting/) (<http://ericposner.com/quadratic-voting/>). There is still deadweight loss (ie. inefficiency) here, because this will lead to individuals with no personal interest in tokens participating in sales because they know they will be able to quickly flip them on the market for a profit. However, this is arguably not that bad: it creates a kind of [crypto universal basic income](https://www.reddit.com/r/CryptoUBI/) (<https://www.reddit.com/r/CryptoUBI/>), and if behavioral economics assumptions like the [endowment effect](https://en.wikipedia.org/wiki/Endowment_effect) (https://en.wikipedia.org/wiki/Endowment_effect) are even slightly true it will also succeed at the goal of ensuring widely distributed ownership.

Are single round sales even good?

Let us get back to the topic of "greed". I would claim that not many people are, in principle, opposed to the idea of development teams that are capable of spending \$500 million to create a really great project getting \$500 million. Rather, what people are opposed to is (i) the idea of completely new and untested development teams getting \$50 million all at once, and (ii) even more importantly, the *time mismatch between developers' rewards and token buyers' interests*. In a single-round sale, the developers have only one chance to get money to build the project, and that is near the start of the development process. There is no feedback mechanism where teams are first given a small amount of money to prove themselves, and then given access to more and more capital over time as they prove themselves to be reliable and successful. During the sale, there is comparatively little information to filter between good development teams and bad ones, and once the sale is completed, the incentive to developers to keep working is relatively low compared to traditional companies. The "greed" isn't about getting lots of money, it's about getting lots of money without working hard to show you're capable of spending it wisely.

If we want to strike at the heart of this problem, how would we solve it? I would say the answer is simple: start moving to mechanisms other than single round sales.

I can offer several examples as inspiration:

- [Angelshares](https://bitsharestalk.org/index.php?topic=1631.0) (<https://bitsharestalk.org/index.php?topic=1631.0>) - this project ran a sale in 2014 where it sold off a fixed percentage of all AGS every day for a period of several months. During each day, people could contribute an unlimited amount to the crowdsale, and the AGS allocation for that day would be split among all contributors. Basically, this is like having a hundred "micro-rounds" of uncapped sales over the course of most of a year; I would claim that the duration of the sales could be stretched even further.
- [Mysterium](http://mysterium.network) (<http://mysterium.network>), which held a little-noticed [micro-sale](https://medium.com/mysterium-network/mysterium-network-presale-early-access-5x-reward-b292d423f96) (<https://medium.com/mysterium-network/mysterium-network-presale-early-access-5x-reward-b292d423f96>) six months before the big one.

- Bancor (<http://bancor.network>), which recently agreed (<https://blog.bancor.network/the-community-of-the-currency-9770087fde17>) to put all funds raised over a cap into a market maker which will maintain price stability along with maintaining a price floor of 0.01 ETH. These funds cannot be removed from the market maker for two years.

It seems hard to see the relationship between Bancor's strategy and solving time mismatch incentives, but an element of a solution is there. To see why, consider two scenarios. As a first case, suppose the sale raises \$30 million, the cap is \$10 million, but then after one year everyone agrees that the project is a flop. In this case, the price would try to drop below 0.01 ETH, and the market maker would lose all of its money trying to maintain the price floor, and so the team would only have \$10 million to work with. As a second case, suppose the sale raises \$30 million, the cap is \$10 million, and after two years everyone is happy with the project. In this case, the market maker will not have been triggered, and the team would have access to the entire \$30 million.

A related proposal is Vlad Zamfir's "safe token sale mechanism (https://medium.com/@Vlad_Zamfir/a-safe-token-sale-mechanism-8d73c430ddd1)". The concept is a very broad one that could be parametrized in many ways, but one way to parametrize it is to sell coins at a price ceiling and then have a price floor slightly below that ceiling, and then allow the two to diverge over time, freeing up capital for development over time if the price maintains itself.

Arguably, none of the above three are sufficient; we want sales that are spread out over an even longer period of time, giving us much more time to see which development teams are the most worthwhile before giving them the bulk of their capital. But nevertheless, this seems like the most productive direction to explore in.

Coming out of the Dilemmas

From the above, it should hopefully be clear that while there is no way to counteract the dilemma and trilemma head on, there are ways to chip away at the edges by thinking outside the box and compromising on variables that are not apparent from a simplistic view of the problem. We can compromise on guarantee of participation slightly, mitigating the impact by using time as a third dimension: if you don't get in during round N, you can just wait until round N + 1 which will be in a week and where the price probably will not be that different.

We can have a sale which is uncapped as a whole, but which consists of a variable number of periods, where the sale within each period is capped; this way teams would not be asking for very large amounts of money without proving their ability to handle smaller rounds first. We can sell small portions of the token supply at a time, removing the political uncertainty that this entails by putting the remaining supply into a contract that continues to sell it automatically according to a prespecified formula.

Here are a few possible mechanisms that follow some of the spirit of the above ideas:

- Host a Gnosis-style reverse dutch auction with a low cap (say, \$1 million). If the auction sells less than 100% of the token supply, automatically put the remaining funds into another auction two months later with a 30% higher cap. Repeat until the entire token supply is sold.
- Sell an unlimited number of tokens at a price of \$X and put 90% of the proceeds into a smart contract that guarantees a price floor of $\$0.9 \cdot X$. Have the price ceiling go up hyperbolically toward infinity, and the price floor go down linearly toward zero, over a five-year period.
- Do the exact same thing AngelShares did, though stretch it out over 5 years instead of a few months.
- Host a Gnosis-style reverse dutch auction. If the auction sells less than 100% of the token supply, put the remaining funds into an automated market maker that attempts to ensure the token's price stability (note that if the price continues going up anyway, then the market maker would be selling tokens, and some of these earnings could be given to the development team).

- Immediately put all tokens into a market maker with parameters+variables X (minimum price), s (fraction of all tokens already sold), t (time since sale started), T (intended duration of sale, say 5 years), that sells tokens at a price of $\frac{k}{(\frac{t}{T-s})}$ (this one is weird and may need to be economically studied more).

Note that there are other mechanisms that should be tried to solve other problems with token sales; for example, revenues going into a multisig of curators, which only hand out funds if milestones are being met, is one very interesting idea that should be done more. However, the design space is highly multidimensional, and there are a lot more things that could be tried.

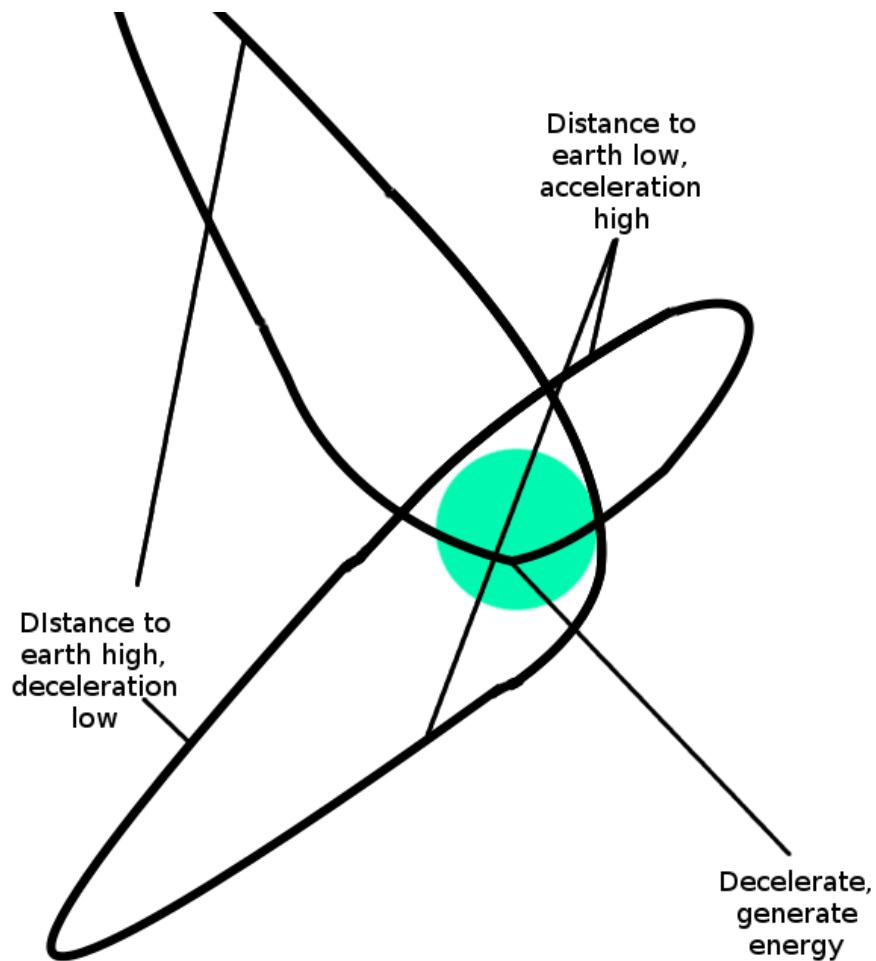
On Path Independence

2017 Jun 22

[See all posts \(/\)](#)

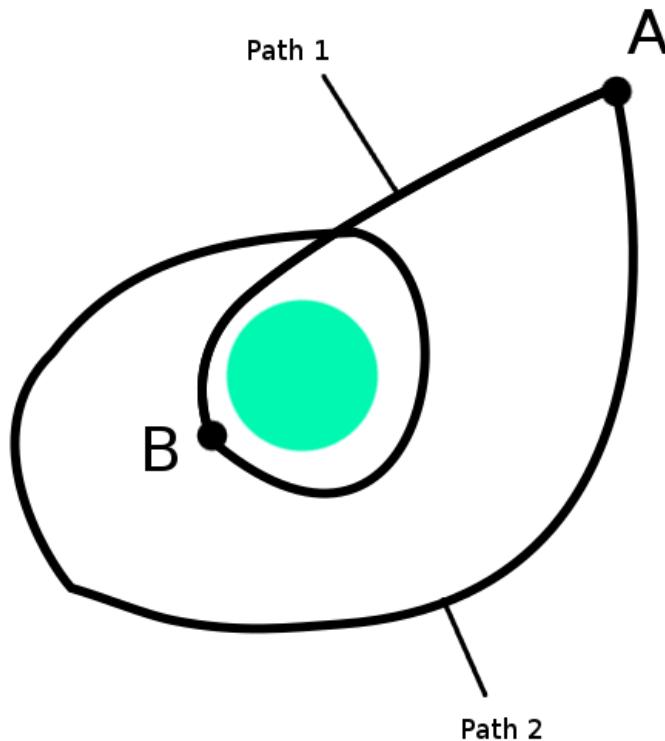
Suppose that someone walks up to you and starts exclaiming to you that he thinks he has figured out how to create a source of unlimited free energy. His scheme looks as follows. First, you get a spaceship up to low Earth orbit. There, Earth's gravity is fairly high, and so the spaceship will start to accelerate heavily toward the earth. The spaceship puts itself into a trajectory so that it barely brushes past the Earth's atmosphere, and then keeps hurtling far into space. Further in space, the gravity is lower, and so the spaceship can go higher before it starts once again coming down. When it comes down, it takes a curved path toward the Earth, so as to maximize its time in low orbit, maximizing the acceleration it gets from the high gravity there, so that after it passes by the Earth it goes even higher. After it goes high enough, it flies through the Earth's atmosphere, slowing itself down but using the waste heat to power a thermal reactor. Then, it would go back to step one and keep going.

Something like this:



Now, if you know anything about Newtonian dynamics, chances are you'll immediately recognize that this scheme is total bollocks. But how do you know? You could make an appeal to symmetry, saying "look, for every slice of the orbital path where you say gravity gives you high acceleration, there's a corresponding slice of the orbital path where gravity gives you just as high deceleration, so I don't see where the net gains are coming from". But then, suppose the man presses you. "Ah," he says, "but in that slice where there is high acceleration your initial velocity is low, and so you spend a lot of time inside of it, whereas in the corresponding slice, your incoming velocity is high, and so you have less time to decelerate". How do you really, conclusively, prove him wrong?

One approach is to dig deeply into the math, calculate the integrals, and show that the supposed net gains are in fact exactly equal to zero. But there is also a simple approach: recognize that **energy is path-independent**. That is, when the spaceship moves from point A to point B, where point B is closer to the earth, its kinetic energy certainly goes up because its speed increases. But because total energy (kinetic plus potential) is conserved (https://en.wikipedia.org/wiki/Conservation_of_energy), and potential energy is only dependent on the spaceship's position, and not how it got there, we know that regardless of what path from point A to point B the spaceship takes, once it gets to point B *the total change in kinetic energy will be exactly the same*.



Different paths, same change in energy

Furthermore, we know that the kinetic energy gain from going *from point A to point A* is also independent of the path you take along the way: in all cases it's exactly zero.

One concern (<https://steemit.com/cryptocurrency/@hms10/bancor-is-flawed>), sometimes cited (https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/d8bw7iw/),

against on-chain market makers (that is, fully automated on-chain mechanisms that act as always-available counterparties for people who wish to trade one type of token for another) is that they are invariably easy to exploit.

As an example, let me quote a [recent post](http://hackingdistributed.com/2017/06/19/bancor-is-flawed/) (<http://hackingdistributed.com/2017/06/19/bancor-is-flawed/>) discussing this issue in the context of Bancor:

The prices that Bancor offers for tokens have nothing to do with the actual market equilibrium. Bancor will always trail the market, and in doing so, will bleed its reserves. A simple thought experiment suffices to illustrate the problem.

Suppose that market panic sets around X. Unfounded news about your system overtake social media. Let's suppose that people got convinced that your CEO has absconded to a remote island with no extradition treaty, that your CFO has been embezzling money, and your CTO was buying drugs from the darknet markets and shipping them to his work address to make a Scarface-like mound of white powder on his desk.

Worse, let's suppose that you know these allegations to be false. They were spread by a troll army wielded by a company with no products, whose business plan is to block everyone's coin stream.

Bancor would offer ever decreasing prices for X coins during a bank run, until it has no reserves left. You'd watch the market panic take hold and eat away your reserves. Recall that people are convinced that the true value of X is 0 in this scenario, and the Bancor formula is guaranteed to offer a price above that. So your entire reserve would be gone.

The post discusses many issues around the Bancor protocol, including details such as code quality, and I will not touch on any of those; instead, I will focus purely on the topic of on-chain market maker efficiency and exploitability, using Bancor (along with MKR) purely as examples and not seeing to make any judgements on the quality of either project as a whole.

For many classes of naively designed on-chain market makers, the comment above about exploitability and trailing markets applies verbatim, and quite seriously so. However, there are also classes of on-chain market makers that are definitely not suspect to their entire reserve being drained due to some kind of money-pumping attack. To take a simple example, consider the market maker selling MKR for ETH whose internal state consists of a current price, p , and which is willing to buy or sell an infinitesimal amount of MKR at each price level. For example, suppose that $p = 5$, and you wanted to buy 2 MKR. The market would sell you:

- 0.00...01 MKR at a price of 5 ETH/MKR
- 0.00...01 MKR at a price of 5.00...01 ETH/MKR
- 0.00...01 MKR at a price of 5.00...02 ETH/MKR
-
- 0.00...01 MKR at a price of 6.99...98 ETH/MKR
- 0.00...01 MKR at a price of 6.99...99 ETH/MKR

Altogether, it's selling you 2 MKR at an average price of 6 ETH/MKR (ie. total cost 12 ETH), and at the end of the operation p has increased to 7. If someone *then* wanted to sell 1 MKR, they would be spending 6.5 ETH, and at the end of *that* operation p would drop back down to 6.

Now, suppose that I told you that such a market maker started off at a price of $p = 5$, and after an unspecified series of events p is now 4. Two questions:

1. How much MKR did the market maker gain or lose?
2. How much ETH did the market maker gain or lose?

The answers are: it gained 1 MKR, and lost 4.5 ETH. Notice that this result is totally independent of the path that p took. Those answers are correct if p went from 5 to 4 directly with one buyer, they're correct if there was first one buyer that took p from 5 to 4.7 and a second buyer that took p the rest of the way to 4, and they're even correct if p first dropped to 2, then increased to 9.818, then dropped again to 0.53, then finally rose again to 4.

Why is this the case? The simplest way to see this is to see that if p drops below 4 and then comes back up to 4, the sells on the way down are exactly counterbalanced by buys on the way up; each sell has a corresponding buy of the same magnitude at the exact same price. But we can also see this by viewing the market maker's core mechanism differently. Define the market maker as having a *single-dimensional* internal state p , and having MKR and ETH balances defined by the following formulas:

$$\text{mkr_balance}(p) = 10 - p$$

$$\text{eth_balance}(p) = p^2/2$$

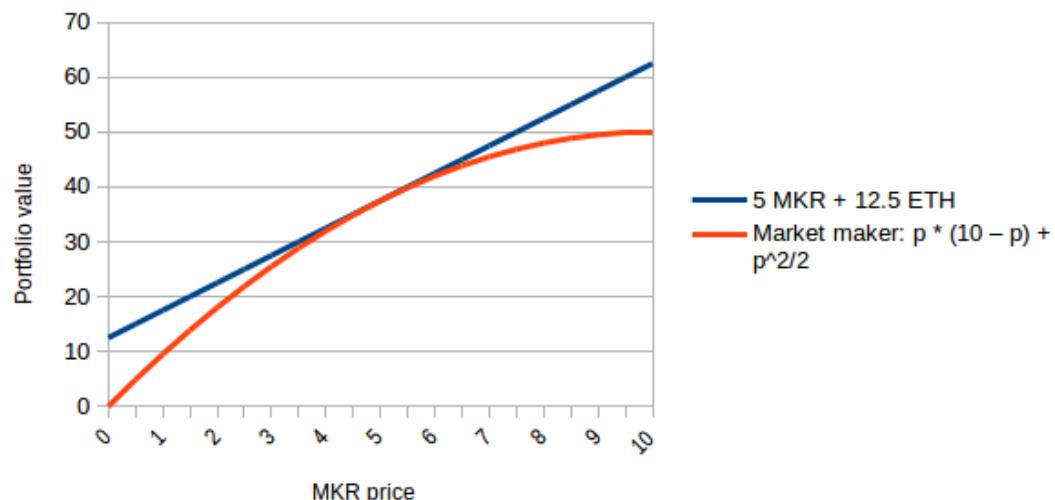
Anyone has the power to "edit" p (though only to values between 0 and 10), but they can only do so by supplying the right amount of MKR or ETH, and getting the right amount of MKR and ETH back, so that the balances still match up; that is, so that the amount of MKR and ETH held by the market maker after the operation is the amount that they are supposed to hold according to the above formulas, with the new value for p that was set. Any edit to p that does not come with MKR and ETH transactions that make the balances match up automatically fails.

Now, the fact that any series of events that drops p from 5 to 4 also raises the market maker's MKR balance by 1 and drops its ETH balance by 4.5, regardless of what series of events it was, should look elementary:
 $\text{mkr_balance}(4) - \text{mkr_balance}(5) = 1$ and $\text{eth_balance}(4) - \text{eth_balance}(5) = -4.5$.

What this means is that a "reserve bleeding" attack on a market maker that preserves this kind of path independence property is impossible. Even if some trolls successfully create a market panic that drops prices to near-zero, when the panic subsides, and prices return to their original levels, the market maker's position will be unchanged - even if both the price, and the market maker's balances, made a bunch of crazy moves in the meantime.

Now, this does not mean that market makers cannot lose money, compared to other holding strategies. If, when you start off, 1 MKR = 5 ETH, and then the MKR price moves, and we compare the performance of holding 5 MKR and 12.5 ETH in the market maker versus the performance of just holding the assets, the result looks like this:

Return of market maker versus balanced portfolio



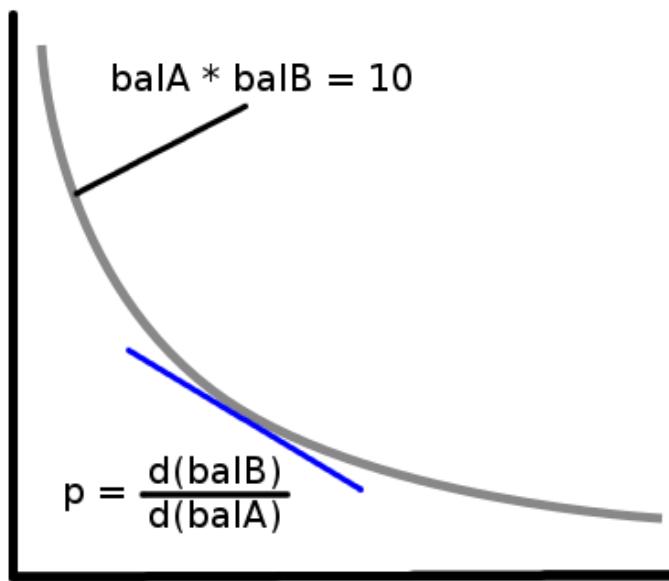
Holding a balanced portfolio always wins, except in the case where prices stay exactly the same, in which case the returns of the market maker and the balanced portfolio are equal. Hence, the purpose of a market maker of this type is to subsidize guaranteed liquidity as a public good for users, serving as trader of last resort, and not to earn revenue. However, we certainly can modify the market maker to earn revenue, and quite simply: we have it charge a spread. That is, the market maker might charge $1.005 \cdot p$ for buys and offer only $0.995 \cdot p$ for sells. Now, being the beneficiary of a market maker becomes a bet: if, in the long run, prices tend to move in one direction, then the market maker loses, at least relative to what they could have gained if they had a balanced portfolio. If, on the other hand, prices tend to bounce around wildly but ultimately come back to the same point, then the market maker can earn a nice profit. This sacrifices the "path independence" property, but in such a way that any deviations from path independence are always in the market maker's favor.

There are many designs that path-independent market makers could take; if you are willing to create a token that can issue an unlimited quantity of units, then the "constant reserve ratio" mechanism (where for some constant ratio $0 \leq r \leq 1$, the token supply is $p^{1/r-1}$ and the reserve size is $r \cdot p^{1/r}$ also counts as one, provided that it is implemented correctly and path independence is not compromised by bounds and rounding errors.

If you want to make a market maker for existing tokens without a price cap, my favorite (credit to Martin Koppelman) mechanism is that which maintains the invariant $\text{tokenA_balance}(p) \cdot \text{tokenB_balance}(p) = k$ for some constant k . So the formulas would be:

$$\text{tokenA_balance}(p) = \sqrt{k \cdot p}$$

$$\text{tokenB_balance}(p) = \sqrt{k/p}$$



Where p is the price of tokenB denominated in tokenA. In general, you can make a path-independent market maker by defining any (monotonic) relation between `tokenA_balance` and `tokenB_balance` and calculating its derivative at any point to give the price.

The above only discusses the role of path independence in preventing one particular type of issue: that where an attacker somehow makes a series of transactions in the context of a series of price movements in order to repeatedly drain the market maker of money. With a path independent market maker, such "money pump" vulnerabilities are impossible. However, there certainly are *other* kinds of inefficiencies that may exist. If the price of MKR drops from 5 ETH to 1 ETH, then the market maker used in the example above will have lost 28 ETH worth of value, whereas a balanced portfolio would only have lost 20 ETH. Where did that 8 ETH go?

In the best case, the price (that is to say, the "real" price, the price level where supply and demand among all users and traders matches up) drops quickly, and some lucky trader snaps up the deal, claiming an 8 ETH profit minus negligible transaction fees. But what if there are multiple traders? Then, if the price between block n and block $n + 1$ differs, the fact that traders can bid against each other by setting transaction fees creates an all-pay auction, with revenues going to the miner. As a consequence of the [revenue equivalence theorem](https://en.wikipedia.org/wiki/Revenue_equivalence_theorem) (https://en.wikipedia.org/wiki/Revenue_equivalence), we can deduce that we can expect that the transaction fees that traders send into this mechanism will keep going up until they are roughly equal to the size of the profit earned (at least initially; the *real* equilibrium is for miners to just snap up the money themselves). Hence, either way schemes like this are ultimately a gift to the miners.

One way to increase social welfare in such a design is to make it possible to create purchase transactions that are only worthwhile for miners to include if they actually make the purchase. That is, if the "real" price of MKR falls from 5 to 4.9, and there are 50 traders racing to arbitrage the market maker, and only the first one of those 50 will make the trade, then only that one should pay the miner a transaction fee. This way, the other 49 failed trades will not clog up the blockchain. [EIP 86](http://github.com/ethereum/EIPs/issues/208) (<http://github.com/ethereum/EIPs/issues/208>), slated for Metropolis, opens up a path toward standardizing such a conditional transaction fee mechanism (another good side effect is that this can also make token sales more unobtrusive, as similar all-pay-auction mechanics apply in many token sales).

Additionally, there are other inefficiencies if the market maker is the *only* available trading venue for tokens. For example, if two traders want to exchange a large amount, then they would need to do so via a long series of small buy and sell transactions, needlessly clogging up the blockchain. To mitigate such inefficiencies, an on-chain market maker should only be *one* of the trading venues available, and not the only one. However, this is arguably not a large concern for protocol developers; if there ends up being a demand for a venue for facilitating large-scale trades, then someone else will likely provide it.

Furthermore, the arguments here only talk about path independence of the market maker *assuming* a given starting price and ending price. However, because of various psychological effects, as well as multi-equilibrium effects, the ending price is plausibly a function not just of the starting price and recent events that affect the "fundamental" value of the asset, but also of the pattern of trades that happens in response to those events. If a price-dropping event takes place, and because of poor liquidity the price of the asset drops quickly, it may end up recovering to a lower point than if more liquidity had been present in the first place. That said, this may actually be an argument in favor of subsidized market makers: if such multiplier effects exist, then they will have a positive impact on price stability that goes beyond the first-order effect of the liquidity that the market maker itself provides.

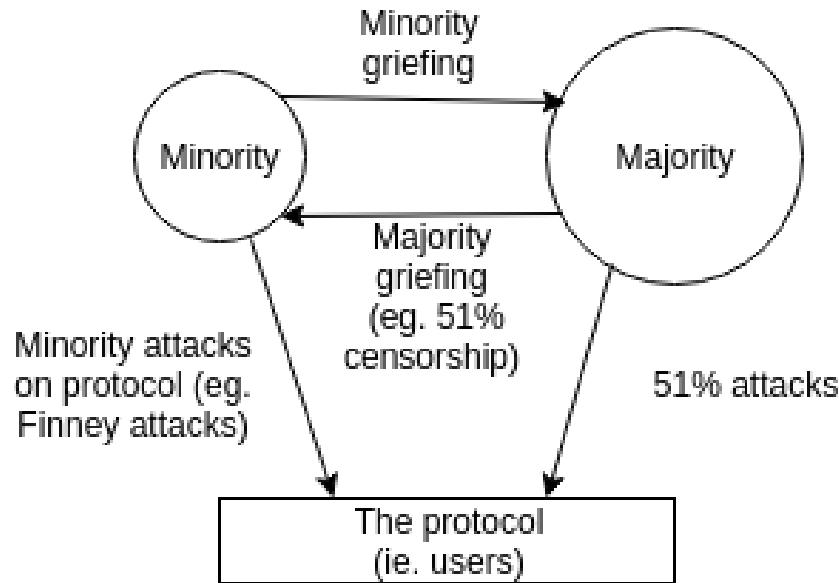
There is likely a lot of research to be done in determining exactly which path-independent market maker is optimal. There is also the possibility of hybrid semi-automated market makers that have the same guaranteed-liquidity properties, but which include some element of asynchrony, as well as the ability for the operator to "cut in line" and collect the profits in cases where large amounts of capital would otherwise be lost to miners. There is also not yet a coherent theory of just how much (if any) on-chain automated guaranteed liquidity is optimal for various objectives, and to what extent, and by whom, these market makers should be subsidized. All in all, the on-chain mechanism design space is still in its early days, and it's certainly worth much more broadly researching and exploring various options.

The Triangle of Harm

2017 Jul 16

[See all posts \(/\)](#)

The following is a diagram from a slide that I made in one of my presentations at Cornell this week:



If there was one diagram that could capture the core principle of Casper's incentivization philosophy, this might be it. Hence, it warrants some further explanation.

The diagram shows three constituencies - the minority, the majority and the protocol (ie. users), and four arrows representing possible adversarial actions: the minority attacking the protocol, the minority attacking the majority, the majority attacking the protocol, and the majority attacking the minority. Examples of each include:

- **Minority attacking the protocol** - [Finney attacks](https://bitcoin.stackexchange.com/questions/4942/what-is-a-finney-attack) (<https://bitcoin.stackexchange.com/questions/4942/what-is-a-finney-attack>) (an attack done by a miner on a proof of work blockchain where the miner double-spends unconfirmed, or possibly single-confirmed, transactions)
- **Minority attacking the majority** - [feather forking](https://bitcointalk.org/index.php?topic=312668.0) (<https://bitcointalk.org/index.php?topic=312668.0>) (a minority in a proof of work chain attempting to revert any block that contains some undesired transactions, though giving up if the block gets two confirmations)
- **Majority attacking the protocol** - traditional 51% attacks
- **Majority attacking the minority** - a 51% censorship attack, where a cartel refuses to accept any blocks from miners (or validators) outside the cartel

The essence of Casper's philosophy is this: **for all four categories of attack**, we want to put an upper bound on the ratio between the amount of harm suffered by the victims of the attack and the cost to the attacker. In some ways, every design decision in Casper flows out of this principle.

This differs greatly from the usual proof of work incentivization school of thought in that in the proof of work view, the last two attacks are left undefended against. The first two attacks, Finney attacks and feather forking, are costly because the attacker risks their blocks not getting included into the chain and so loses revenue. If the attacker is a majority, however, the attack is costless, because the attacker can always guarantee that their chain will be the main chain. In the long run, difficulty adjustment ensures that the total revenue of all miners is exactly the same no matter what, and this further means that if an attack causes some victims to lose money, then the attacker *gains* money.

This property of proof of work arises because traditional Nakamoto proof of work fundamentally *punishes dissent* - if you as a miner make a block that aligns with the consensus, you get rewarded, and if you make a block that does not align with the consensus you get penalized (the penalty is not in the protocol; rather, it comes from the fact that such a miner expends electricity and capital to produce the block and gets no reward).

Casper, on the other hand, works primarily by *punishing equivocation* - if you send two messages that conflict with each other, then you get very heavily penalized, even if one of those messages aligns with the consensus (read more on this in the [blog post on "minimal slashing conditions"](#) (<https://medium.com/@VitalikButerin/minimal-slashing-conditions-20f0b500fc6c>)). Hence, in the event of a finality reversion attack, those who caused the reversion event are penalized, and everyone else is left untouched; the majority can attack the protocol only at heavy cost, and the majority cannot cause the minority to lose money.

It gets more challenging when we move to talking about two other kinds of attacks - liveness faults, and censorship. A liveness fault is one where a large portion of Casper validators go offline, preventing the consensus from reaching finality, and a censorship fault is one where a majority of Casper validators refuse to accept some transactions, or refuse to accept consensus messages from other Casper validators (the victims) in order to deprive them of rewards.

This touches on a fundamental dichotomy: **speaker/listener fault equivalence**.



Suppose that person B says that they did not receive a message from person A. There are two possible explanations: (i) person A did not send the message, (ii) person B pretended not to hear the message. Given just the evidence of B's claim, *there is no way to tell which of the two explanations is correct*. The relation to blockchain protocol incentivization is this: if you see a protocol execution where 70% of validators' messages are included in the chain and 30% are not, and see nothing else (and this is what the blockchain sees), then there is no way to tell whether the problem is that 30% are offline or 70% are censoring. If we want to make both kinds of attacks expensive, there is only one thing that we can do: **penalize both sides**.

Penalizing both sides allows either side to "grief" the other, by going offline if they are a minority and censoring if they are a majority. However, we can establish bounds on how easy this griefing is, through the technique of **griefing factor analysis**. The griefing factor of a strategy is essentially the amount of money lost by the victims divided by the amount of money lost by the attackers, and the griefing factor of a protocol is the highest griefing factor that it allows. For example, if a protocol allows me to cause you to lose \$3 at a cost of \$1 to myself, then the griefing factor is 3. If there are no ways to cause others to lose money, the griefing factor is zero, and if you can cause others to lose money at no cost to yourself (or at a benefit to yourself), the griefing factor is infinity.

In general, **wherever a speaker/listener dichotomy exists, the griefing factor cannot be globally bounded above by any value below 1**. The reason is simple: either side can grief the other, so if side A can grief side B with a factor of x then side B can grief side A with a factor of $\frac{1}{x}$; x and $\frac{1}{x}$ cannot both be below 1 simultaneously. We can play around with the factors; for example, it may be considered okay to allow griefing factors of 2 for majority attackers in exchange for keeping the griefing factor at 0.5 for minorities, with the reasoning that minority attackers are more likely. We can also allow griefing factors of 1 for small-scale attacks, but specifically for large-scale attacks force a chain split where on one chain one side is penalized and on another chain another side is penalized, trusting the market to pick the chain where attackers are not favored. Hence there is a lot of room for compromise and making tradeoffs between different concerns within this framework.

Penalizing both sides has another benefit: it ensures that if the protocol is harmed, the attacker is penalized. This ensures that whoever the attacker is, they have an incentive to avoid attacking that is commensurate with the amount of harm done to the protocol. However, if we want to bound the ratio of harm to the protocol over cost to attackers, we need a formalized way of measuring how much harm to the protocol was done.

This introduces the concept of the **protocol utility function** - a formula that tells us how well the protocol is doing, that should ideally be calculable from inside the blockchain. In the case of a proof of work chain, this could be the percentage of all blocks produced that are in the main chain. In Casper, protocol utility is zero for a perfect execution where every epoch is finalized and no safety failures ever take place, with some penalty for every epoch that is not finalized, and a very large penalty for every safety failure. If a protocol utility function can be formalized, then penalties for faults can be set as close to the loss of protocol utility resulting from those faults as possible.

A Note on Metcalfe's Law, Externalities and Ecosystem Splits

2017 Jul 27

[See all posts \(/\)](#)

Looks like it's blockchain split season again (<http://bitcoincash.org/>). For background of various people discussing the topic, and whether such splits are good or bad:

- Power laws and network effects (arguing the BTC/BCC split may destroy value due to network effect loss): <https://medium.com/crypto-fundamental/power-laws-and-network-effects-why-bitcoincash-is-not-a-free-lunch-5adb579972aa> (<https://medium.com/crypto-fundamental/power-laws-and-network-effects-why-bitcoincash-is-not-a-free-lunch-5adb579972aa>).
- Brian Armstrong on the Ethereum Hard Fork (last year): <https://blog.coinbase.com/on-the-ethereum-hard-fork-780f1577e986> (<https://blog.coinbase.com/on-the-ethereum-hard-fork-780f1577e986>).
- Phil Daian on the ETH/ETC split: <http://pdaian.com/blog/stop-worrying-love-etc/> (<http://pdaian.com/blog/stop-worrying-love-etc/>).

Given that ecosystem splits are not going away, and we may well see more of them in the crypto industry over the next decade, it seems useful to inform the discussion with some simple economic modeling. With that in mind, let's get right to it.

Suppose that there exist two projects A and B, and a set of users of total size N , where A has N_a users and B has N_b users. Both projects benefit from network effects, so they have a utility that increases with the number of users. However, users also have their own differing taste preferences, and this may lead them to choose the smaller platform over the bigger platform if it suits them better.

We can model each individual's private utility in one of four ways:

$1. U(A) = p + N_a$	$U(B) = q + N_b$
$2. U(A) = p \cdot N_a$	$U(B) = q \cdot N_b$
$3. U(A) = p + \ln N_a$	$U(B) = q + \ln N_b$
$4. U(A) = p \cdot \ln N_a$	$U(B) = q \cdot \ln N_b$

p and q are private per-user parameters that you can think of as corresponding to users' distinct preferences. The difference between the first two approaches and the last two reflects differences between interpretations of Metcalfe's law, or more broadly the idea that the per-user value of a system grows with the number of users. The [original formulation](https://en.wikipedia.org/wiki/Metcalfe%27s_law) (https://en.wikipedia.org/wiki/Metcalfe%27s_law) suggested a per-user value of N

(that is, a total network value of N^2), but other analysis (see [here](#) (<http://spectrum.ieee.org/computing/networks/metcalfe-s-law-is-wrong>)) suggests that above very small scales $N \log N$ usually dominates; there is a controversy over which model is correct. The difference between the first and second (and between the third and fourth) is the extent to which utility from a system's intrinsic quality and utility from network effects are complementary - that is, are the two things good in completely separate ways that do not interact with each other, like social media and coconuts, or are network effects an important part of letting the intrinsic quality of a system shine?

We can now analyze each case in turn by looking at a situation where N_a users choose A and N_b users choose B, and analyze each individual's decision to choose one or the other from the perspective of economic externalities - that is, does a user's choice to switch from A to B have a positive net effect on others' utility or a negative one? If switching has a positive externality, then it is virtuous and should be socially nudged or encouraged, and if it has a negative externality then it should be discouraged. We model an "ecosystem split" as a game where to start off $N_a = N$ and $N_b = 0$ and users are deciding for themselves whether or not to join the split, that is, to move from A to B, possibly causing N_a to fall and N_b to rise.

Switching (or not switching) from A to B has externalities because A and B both have network effects; switching from A to B has the negative externality of reducing A's network effect, and so hurting all remaining A users, but it also has the positive externality of increasing B's network effect, and so benefiting all B users.

Case 1

Switching from A to B gives N_a users a negative externality of one, so a total loss of N_a , and it gives N_b users a positive externality of one, so a total gain of N_b . Hence, the total externality is of size $N_b - N_a$; that is, switching from the smaller to the larger platform has positive externalities, and switching from the larger platform to the smaller platform has negative externalities.

Case 2

Suppose P_a is the sum of p values of N_a users, and Q_b is the sum of q values of N_b users. The total negative externality is P_a and the total positive externality is Q_b . Hence, switching from the smaller platform to the larger has positive social externalities if the two platforms have equal intrinsic quality to their users (ie. users of A intrinsically enjoy A as much as users of B intrinsically enjoy B, so p and q values are evenly distributed), but if it is the case that A is bigger but B is better, then there are positive externalities in switching to B.

Furthermore, notice that if a user is making a switch from a larger A to a smaller B, then this itself is revealed-preference evidence that, for that user, and for all existing users of B, $\frac{q}{p} > \frac{N_a}{N_b}$. However, if the split stays as a split, and does not proceed to become a full-scale migration, then that means that users of A hold different views, though this could be for two reasons: (i) they intrinsically dislike A but not by enough to justify the switch, (ii) they intrinsically like A more than B. This could arise because (a) A users have a higher opinion of A than B users, or (b) A users have a lower opinion of B than B users. In general, we see that moving from a system that makes its average user less happy to a system that makes its average user more happy has positive externalities, and in other situations it's difficult to say.

Case 3

The derivative of $\ln x$ is $\frac{1}{x}$. Hence, switching from A to B gives N_a users a negative externality of $\frac{1}{N_a}$, and it gives N_b users a positive externality of $\frac{1}{N_b}$. Hence, the negative and positive externalities are both of total size one, and thus cancel out. Hence, switching from one platform to the other imposes no social externalities, and

it is socially optimal if all users switch from A to B if and only if they think that it is a good idea for them personally to do so.

Case 4

Let P_a and Q_b are before. The negative externality is of total size $\frac{P_a}{N_a}$ and the positive externality is of total size $\frac{Q_b}{N_b}$. Hence, if the two systems have equal intrinsic quality, the externality is of size zero, but if one system has higher intrinsic quality, then it is virtuous to switch to it. Note that as in case 2, if users are switching from a larger system to a smaller system, then that means that they find the smaller system to have higher intrinsic quality, although, also as in case 2, if the split remains a split and does not become a full-scale migration, then that means other users see the intrinsic quality of the larger system as higher, or at least not lower by enough to be worth the network effects.

The existence of users switching to B suggests that for them, $\frac{q}{p} \geq \frac{\log N_a}{\log N_b}$, so for the $\frac{Q_b}{N_b} > \frac{P_a}{N_a}$ condition to not hold (ie. for a move from a larger system to a smaller system not to have positive externalities) it would need to be the case that users of A have similarly high values of p - an approximate heuristic is, the users of A would need to love A so much that if *they* were the ones in the minority that would be willing to split off and move to (or stay with) the smaller system. In general, it thus seems that moves from larger systems to smaller systems that actually do happen will have positive externalities, but it is far from ironclad that this is the case.

Hence, if the first model is true, then to maximize social welfare we should be trying to nudge people to switch to (or stay with) larger systems over smaller systems, and splits should be discouraged. If the fourth model is true, then we should be at least slightly trying to nudge people to switch to smaller systems over larger systems, and splits should be slightly encouraged. If the third model is true, then people will choose the socially optimal thing all by themselves, and if the second model is true, it's a toss-up.

It is my personal view that the truth lies somewhere between the third and fourth models, and the first and second greatly overstate network effects above small scales. The first and second model (the N^2 form of Metcalfe's law) essentially state that a system growing from 990 million to 1 billion users gives the same increase in per-user utility as growing from 100,000 to 10.1 million users, which seems very unrealistic, whereas the $N \log N$ model (growing from 100 million to 1 billion users gives the same increase in per-user utility as growing from 100,000 to 10 million users) intuitively seems much more correct.

And the third model says: if you see people splitting off from a larger system to create a smaller system because they want something that more closely matches their personal values, then the fact that these people have already shown that they value this switch enough to give up the comforts of the original system's network effects is by itself enough evidence to show that the split is socially beneficial. Hence, unless I can be convinced that the first model is true, or that the second model is true and the specific distributions of p and q values make splits make negative negative externalities, I maintain my existing view that those splits that actually do happen (though likely not *hypothetical* splits that end up not happening due to lack of interest) are in the long term socially beneficial, value-generating events.

A Prehistory of the Ethereum Protocol

2017 Sep 14

[See all posts \(/\)](#)

Although the ideas behind the current Ethereum protocol have largely been stable for two years, Ethereum did not emerge all at once, in its current conception and fully formed. Before the blockchain has launched, the protocol went through a number of significant evolutions and design decisions. The purpose of this article will be to go through the various evolutions that the protocol went through from start to launch; the countless work that was done on the implementations of the protocol such as Geth, cppethereum, pyethereum, and EthereumJ, as well as the history of applications and businesses in the Ethereum ecosystem, is deliberately out of scope.

Also out of scope is the history of Casper and sharding research. While we can certainly make more blog posts talking about all of the various ideas Vlad, Gavin, myself and others came up with, and discarded, including "proof of proof of work", hub-and-spoke chains, "[hypercubes](https://blog.ethereum.org/2014/10/21/scalability-part-2-hypercubes)", "[shadow chains](https://blog.ethereum.org/2014/09/17/scalability-part-1-building-top)", "[Plasma](http://plasma.io/)", "[chain fibers](https://github.com/ethereum/wiki/wiki/Chain-Fibers-Redux)", and [various iterations of Casper](https://blog.ethereum.org/2016/12/06/history-casper-chapter-1), as well as Vlad's rapidly evolving thoughts on reasoning about incentives of actors in consensus protocols and properties thereof, this would also be far too complex a story to go through in one post, so we will leave it out for now.

Let us first begin with the very earliest version of what would eventually become Ethereum, back when it was not even called Ethereum. When I was visiting Israel in October 2013, I spent quite a bit of time with the Mastercoin team, and even suggested a few features for them. After spending a couple of times thinking about what they were doing, I sent the team a proposal to make their protocol more generalized and support more types of contracts without adding an equally large and complex set of features:

<https://web.archive.org/web/20150627031414/http://vbuterin.com/ultimatescripting.html>
<https://web.archive.org/web/20150627031414/http://vbuterin.com/ultimatescripting.html>

Ultimate Scripting: A Platform for Generalized Financial Contracts on Mastercoin

0.1. Introduction

Perhaps the key advantage of Mastercoin over the raw Bitcoin protocol is the potential to include much more advanced transaction types, including transactions that specify behavior based on future information well off into the future. For example, Mastercoin joins Ripple in being one of the only two major cryptocurrency networks that include the ability for users to make binding exchange offers as a type of transaction. From there, the Mastercoin Foundation intends to integrate even more complex contracts, including bets, contracts for difference and on-blockchain dice rolls. However, up until this point Mastercoin has been taking a relatively unstructured process in developing these ideas, essentially treating each one as a separate "feature" with its own transaction code and rules. This document outlines an alternative way of specifying Mastercoin contracts which follows an open-ended philosophy, specifying only the basic data and arithmetic building blocks and allowing anyone to craft arbitrarily complex Mastercoin contracts to suit their own needs, including needs which we may not even anticipate.

0.2. Specification

The underlying idea behind this specification is to allow anyone to create a contract which pays out according to an arbitrary formula. The formula will be defined in a Bitcoin-like stack-based scripting language, consisting of numbers and opcodes.

The evaluation algorithm is as follows:

```
dataStack = []
opStack = script
while len(opStack) > 0:
    varOp = opStack.pop()
    if type(varOp) == 'opcode':
        eval(dataStack,op)
    else:
        dataStack.push(varOp)
return dataStack.pop()
```

Where eval is defined for each opcode below. Any error (eg. division by zero) will make the script return FAIL, and result in the entire transaction being treated as invalid by the Mastercoin network. All variables will be signed 64-bit integers, and all arithmetic operations wrap around (that is, if the underlying arithmetic operation returns R, the value pushed is $((R + 2^{63}) \% 2^{64}) - 2^{63}$.

Notice that this is very far from the later and more expansive vision of Ethereum: it specialized purely in what Mastercoin was trying to specialize in already, namely two-party contracts where parties A and B would both put in money, and then they would later get money out according to some formula specified in the contract (eg. a bet would say "if X happens then give all the money to A, otherwise give all the money to B"). The scripting language was not Turing-complete.

The Mastercoin team was impressed, but they were not interested in dropping everything they were doing to go in this direction, which I was increasingly convinced is the correct choice. So here comes version 2, circa December:

<https://web.archive.org/web/20131219030753/http://vitalik.ca/ethereum.html>
https://lh6.googleusercontent.com/soPo_aa2YSpV8DvGGZbGjAkZehtiqJEa8dPzOM4ZSzAvZcAfNbnVqErQL1JlG8lcmgpQyXmb3cO9m2iasJQKZZmTXGQsI_0tvT

Ethereum: The Ultimate Smart Contract and Autonomous Corporation Platform on the Blockchain

In the last few months, there has been a great amount of interest into the area of using the Bitcoin blockchain, the mechanism that allows for the entire world to agree on the state of a public ownership database, for more than just money. Perhaps the first, and oldest, such alternative application is colored coins, which is a protocol that allows users to label specific bitcoins and treat them as assets representing some real world value - whether company shares, collectibles or even existing currencies like gold and USD. A more independent alternative, Ripple, also includes the ability to create custom currencies and assets, but adds a decentralized exchange. More recently, Mastercoin has started to go even further, allowing more complex financial contracts such as hedging, trust-free dice rolls, binary options and self-stabilizing currencies - essentially, almost any common financial instrument imaginable. Taken together, all of these projects can be thought of as initial efforts toward a sort of "cryptocurrency 2.0" - they are to Bitcoin what Web 2.0 was to the World Wide Web circa 1995.

At the same time, there has been significant interest in "[decentralized autonomous corporations](#)" - autonomous entities that operate on the blockchain in a completely transparent and publicly managed way without any central control whatsoever. Rather than the relationships of the investors, owners and employees of the corporation being managed by a legal contract or a set of organizational bylaws, the funds and corporate resources are managed directly on the blockchain. However, decentralized autonomous corporations are difficult to implement today, simply because the scripting systems of Bitcoin, and even proto-cryptocurrency 2.0 alternatives like Ripple and Mastercoin, are far too limited to allow the kind of arbitrarily complex computation that DACs require. Although these platforms have begun to offer increasingly complex contracts such as financial derivatives, order matching and trust-free bets, the way that the protocols are set up is inherently limited and closed-ended: each of these use cases is treated as a specific transaction type, not allowing any way for users to build contracts that the developers have not specifically chosen to include.

What this project intends to do is take cryptocurrency 2.0, and generalize it - create a fully-fledged, Turing-complete (but heavily fee-regulated) cryptographic ledger that allows participants to encode arbitrarily complex contracts, autonomous agents and relationships that will be mediated entirely by the blockchain. On-chain currencies, futures contracts, prediction markets, Namecoin-style domain name systems and even provably fair gambling sites will become trivial to implement, existing as simple, hundred-line-of-code contracts on the chain.

Basic Building Blocks

Here you can see the results of a substantial rearchitecting, largely a result of a long walk through San Francisco I took in November once I realized that smart contracts could potentially be fully generalized. Instead of the scripting language being simply a way of describing the terms of relations between two parties, contracts were themselves fully-fledged accounts, and had the ability to hold, send and receive assets, and even maintain a permanent storage (back then, the permanent storage was called "memory", and the only temporary "memory" was the 256 registers). The language switched from being a stack-based machine to being a register-based one on my own volition; I had little argument for this other than that it seemed more sophisticated.

Additionally, notice that there is now a built-in fee mechanism:

Whenever ether is sent to a script, the following happens:

1. The ether's endowment increases by the amount sent
2. All registers are reset to zero.
3. The sender is placed into R0.
4. The value sent is placed into R1.
5. The fee is placed into R2.
6. The index pointer is set to zero, and STEPCOUNT = 0
7. Repeat forever:
 - o set TOTALFEE = 0
 - o set STEPCOUNT <- STEPCOUNT + 1
 - o if STEPCOUNT > 16, set TOTALFEE <- TOTALFEE + STEPFee
 - o see if the command at the index pointer is a valid command and not STOP. If it is invalid or STOP, HALT and break out of the loop
 - o see if the command will do any modifications to the contract. If so, set TOTALFEE <- TOTALFEE + DATAFee
 - o see if the command will fill up a previously zero memory field. If so, set TOTALFEE <- TOTALFEE + MEMORYFee
 - o see if the command will zero a previously used memory field. If so, set TOTALFEE <- TOTALFEE - MEMORYFee
 - o see if the command is EXTRD or BALANCE. If so, set TOTALFEE <- TOTALFEE + EXTRDfee
 - o see if the command is MKTX or RAWTX. If so, set TOTALFEE <- TOTALFEE + (transaction's value plus transaction's fee)
 - o **if TOTALFEE > contract's endowment, HALT and break out of the loop**
 - o **else, subtract TOTALFEE from contract's endowment. Note that TOTALFEE may be negative in some cases, in which case the endowment would actually increase**
 - o run the command

At this point, ether literally was gas; after every single computational step, the balance of the contract that a transaction was calling would drop a little bit, and if the contract ran out of money execution would halt. Note that this "receiver pays" mechanism meant that the contract itself had to require the sender to pay the contract a fee, and immediately exit if this fee is not present; the protocol allocated an allowance of 16 free execution steps to allow contracts to reject non-fee-paying transactions.

This was the time when the Ethereum protocol was entirely my own creation. From here on, however, new participants started to join the fold. By far the most prominent on the protocol side was Gavin Wood, who reached out to me in an about.me message in December 2013:

Gav Wood sent you a message on about.me

1 message

i@gawwood.com <i@gawwood.com>
 Reply-To: i@gawwood.com
 To: vbuterin@gmail.com

Thu, Dec 19, 2013 at 11:53 AM



Hi Vitalik!

[View Dashboard](#)**Gav Wood sent you a message**

“ Johnny gave me the heads up - I can do C++ (e.g. github/gavofyork). How far are you with ethereum?

[REPLY TO GAV](#)

This email was sent to you by [about.me/gawwood](#), and is not an official communication from about.me.

Cheers,
 The about.me team

[Don't want these emails? One Click Unsubscribe](#)
[Terms of Service](#) | [Privacy Policy](#)
 about.me 2601 Mission St San Francisco, CA 94110

Jeffrey Wilcke, lead developer of the Go client (back then called "ethereal") also reached out and started coding around the same time, though his contributions were much more on the side of client development rather than protocol research.

Ethereum Inbox X

 **Jeffrey Wilcke** <stygeo@gmail.com>
 to me ▼

12/20/13 ★ ↶ ↷

Hi there,

I was reading over the Ethereum spec and implementing some of it's future as the protocol seems rather interesting. However I came across a few errors on this page <http://vitalik.ca/ethereum.html>

Basic Building Block, Transactions: you mention `[0 ... 2^256 - 1]` this would give a rather odd number (https://www.google.com/search?q=rls=en_NL&q=2^256&ie=UTF-8&oe=UTF-8#q=2^256-1&rls=en_NL&safe=off). I suppose you meant 256^2 ? Also right after you mention 32 byte integers, that should probably be 32 bit integers or 4 bytes (also probably unsigned integers).

I also had a question about the contracts. You mention that stack is non-persistent but memory is. Now I suppose that you serialize the memory and store it in database X after each run, or how would that go? Are contracts which are persisted mutable in that way? (I could have missed this part)

As for in and outputs, you mention one input and one output per transaction. How would you deal with "change"? Say for example I would like to send you 2.3. I have one inbound Tx of 5. Now how would I go about sending you 2.3? I know BTC creates a Tx of 5 with 2 outputs. 2.3 to whatever address I specified and 2.7 to a change address so I don't end up sending you too much.

I've implemented several opcodes of the E-VM. It currently has a 256^2 registers and each contract currently holds a maximum of 256 (ints). I've successfully implemented the following op codes:
 STOP, ADD, SUB, LT, LD, SET, JMP and JMPI. And got your currency as a contract sample working up instruction 12.

Just wanted to let you know and wish you all the luck with the further development of Ethereum. It looks promising :-)

Regards,
 Jeff

 **Vitalik Buterin** <vbuterin@gmail.com>
 to Jeffrey ▼

12/21/13 ★ ↶ ↷

Hey Jeremy,

Glad to see you're interested in Ethereum. My answers:

1. Yes, I do mean 32-byte numbers in the range `[0 ... 2^256 - 1]`. The idea is that they have to be this big to store addresses, hashes, private keys,

"Hey Jeremy, glad to see you're interested in Ethereum..."

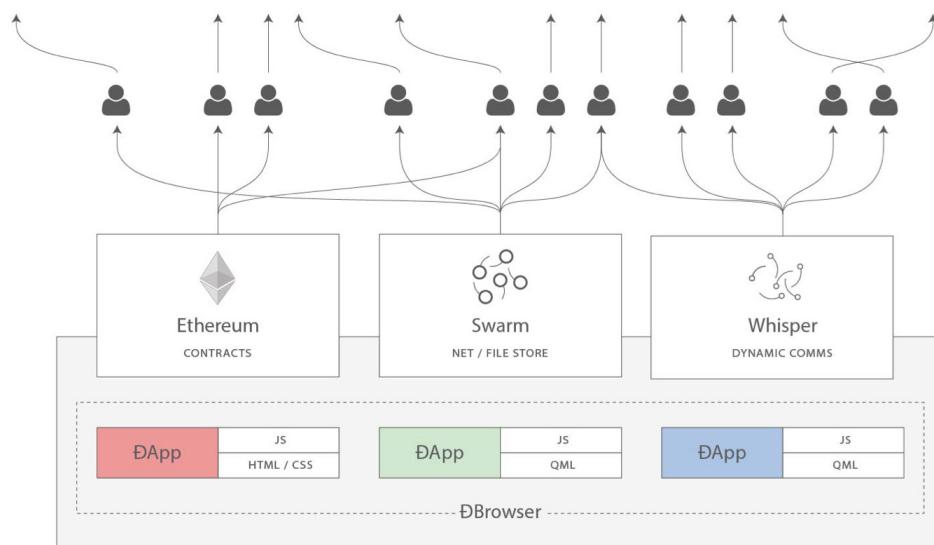
Gavin's initial contributions were two-fold. First, you might notice that the contract calling model in the initial design was an asynchronous one: although contract A could create an "internal transaction" to contract B ("internal transaction" is Etherscan's lingo; initially they were just called "transactions" and then later "message calls" or "calls"), the internal transaction's execution would not start until the execution of the first transaction completely finished. This meant that transactions could not use internal transactions as a way of getting information from other contracts; the only way to do that was the EXTRO opcode (kind of like an SLOAD that you could use to read other contracts' storage), and this too was later removed with the support of Gavin and others.

When implementing my initial spec, Gavin naturally implemented internal transactions synchronously without even realizing that the intent was different - that is to say, in Gavin's implementation, when a contract calls another contract, the internal transaction gets executed immediately, and once that execution finishes, the VM

returns back to the contract that created the internal transaction and proceeds to the next opcode. This approach seemed to both of us to be superior, so we decided to make it part of the spec.

Second, a discussion between him and myself (during a walk in San Francisco, so the exact details will be forever lost to the winds of history and possibly a copy or two in the deep archives of the NSA) led to a re-factoring of the transaction fee model, moving away from the "contract pays" approach to a "sender pays" approach, and also switching to the "gas" architecture. Instead of each individual transaction step immediately taking away a bit of ether, the transaction sender pays for and is allocated some "gas" (roughly, a counter of computational steps), and computational steps drew from this allowance of gas. If a transaction runs out of gas, the gas would still be forfeit, but the entire execution would be reverted; this seemed like the safest thing to do, as it removed an entire class of "partial execution" attacks that contracts previously had to worry about. When a transaction execution finishes, the fee for any unused gas is refunded.

Gavin can also be largely credited for the subtle change in vision from viewing Ethereum as a platform for building programmable money, with blockchain-based contracts that can hold digital assets and transfer them according to pre-set rules, to a general-purpose computing platform. This started with subtle changes in emphasis and terminology, and later this influence became stronger with the increasing emphasis on the "Web 3" ensemble, which saw Ethereum as being one piece of a suite of decentralized technologies, the other two being Whisper and Swarm.



There were also changes made around the start of 2014 that were suggested by others. We ended up moving back to a stack-based architecture after the idea was suggested by Andrew Miller and others.

On 12/19/2013 03:40 PM, [Andrew Miller](#) wrote:
 > Hi Vitalik,
 > I'd really like to talk with you more about this. I'm really
 > interested in extending the functionality of Bitcoin beyond trivial
 > money-shoving financial transactions and up to user-customizable
 > contracts, and I'm pretty stoked that you're taking a shot at this as
 > a serious project.
 > Here are some specific concerns/questions about your transaction language:
 >
 > 1. (Note: this is my most superficial criticism) Why did you design
 > your own register language? What's wrong with a stack based language
 > similar to Bitcoin? You can have a turing-complete and higher order
 > stack language (look at Joy, Factor or Forth). If anything I'd
 > recommend a lambda-calculus based language. From Stack-based to
 > Register-based is such a superficial change and there's absolutely no
 > motivation for it, yet most of your document is about minutiae related
 > to this. When you present your contract examples, you're writing in
 > pseudocode that isn't really any closer to ASM than to stack-based or
 > functional anyway. You might also look at E, a language based on
 > javascript that was explicitly designed for the purpose of writing
 > smart contracts. <http://www.erights.org/elang/>

Charles Hoskinson suggested the switch from Bitcoin's SHA256 to the newer SHA3 (or, more accurately, keccak256). Although there was some controversy for a while, discussions with Gavin, Andrew and others led to establishing that the size of values on the stack should be limited to 32 bytes; the other alternative being considered, unlimited-size integers, had the problem that it was too difficult to figure out how much gas to charge for additions, multiplications and other operations.

The initial mining algorithm that we had in mind, back in January 2014, was a contraption called Dagger:

<https://github.com/ethereum/wiki/blob/master/Dagger.md> (<https://github.com/ethereum/wiki/blob/master/Dagger.md>)

Algorithm specification:

Essentially, the Dagger algorithm works by creating a directed acyclic graph (the technical term for a tree where each node is allowed to have multiple parents) with a total of $2^{23} - 1$ nodes in sequence. Each node depends on 3-15 randomly selected nodes before it. If the miner finds a node between index 2^{22} and 2^{23} such that this resulting hash is below 2^{256} divided by the difficulty parameter, the result is a valid proof of work.

Let d be the underlying data (eg. in Bitcoin's case the block header), n be the nonce and || be the string concatenation operator (ie. `'foo' || 'bar' == 'foobar'`). The entire code for the algorithm is as follows:

```
D(data,xn,0) = sha3(data)
D(data,xn,n) =
    with v = sha3(data + xn + n)
        L = 2 if n < 2^21 else 11 if n < 2^22 else 3
        a[k] = floor(v/n^k) mod n for 0 <= k < 2
        a[k] = floor(v/n^k) mod 2^22 for 2 <= k < L
    sha3(v ++ D(data,xn,a[0]) ++ D(data,xn,a[1]) ++ ... ++ D(data,xn,a[L-1]))
```

Properties:

Objective: find xn , n such that $n > 2^{22}$ and $D(data,xn,n) \leq 2^{256} / \text{diff}$

Dagger was named after the "directed acyclic graph" (DAG), the mathematical structure that is used in the algorithm. The idea is that every N blocks, a new DAG would be pseudorandomly generated from a seed, and the bottom layer of the DAG would be a collection of nodes that takes several gigabytes to store. However, generating any individual value in the DAG would require calculating only a few thousand entries. A "Dagger computation" involved getting some number of values in random positions in this bottom-level dataset and hashing them together. This meant that there was a fast way to make a Dagger calculation - already having the data in memory, and a slow, but not memory intensive way - regenerating each value from the DAG that you need to get from scratch.

The intention of this algorithm was to have the same "memory-hardness" properties as algorithms that were popular at the time, like Scrypt, but still be light-client friendly. Miners would use the fast way, and so their mining would be constrained by memory bandwidth (the theory is that consumer-grade RAM is already very heavily optimized, and so it would be hard to further optimize it with ASICs), but light clients could use the memory-free but slower version for verification. The fast way might take a few microseconds and the slow but memory-free way a few milliseconds, so it would still be very viable for light clients.

From here, the algorithm would change several times over the course of Ethereum development. The next idea that we went through is "adaptive proof of work"; here, the proof of work would involve executing randomly selected Ethereum contracts, and there is a clever reason why this is expected to be ASIC-resistant: if an ASIC was developed, competing miners would have the incentive to create and publish many contracts that that ASIC was not good at executing. There is no such thing as an ASIC for general computation, the story goes, as that is just a CPU, so we could instead use this kind of adversarial incentive mechanism to make a proof of work that essentially was executing general computation.

This fell apart for one simple reason: [long-range attacks](https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/) (<https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/>). An attacker could start a chain from block 1, fill it up with only simple contracts that they can create specialized hardware for, and rapidly overtake the main chain. So... back to the drawing board.

The next algorithm was something called Random Circuit, described in this google doc [here](https://docs.google.com/document/d/19c0L71neWpTN-jyWW-87mzrTTmS2h3lAYxXpRAvPfo) (<https://docs.google.com/document/d/19c0L71neWpTN-jyWW-87mzrTTmS2h3lAYxXpRAvPfo>), proposed by myself and Vlad Zamfir, and [analyzed by Matthew Wampler-Doty](https://nbviewer.jupyter.org/gist/anonymous/cb53d06b837be97ebe32) (<https://nbviewer.jupyter.org/gist/anonymous/cb53d06b837be97ebe32>), and others. The idea here was also to simulate general-purpose computation inside a mining algorithm, this time by executing randomly generated circuits. There's no hard proof that something based on these principles could not work, but the computer hardware experts that we reached out to in 2014 tended to be fairly pessimistic on it. Matthew Wampler-Doty himself suggested a proof of work based on SAT solving, but this too was ultimately rejected.

Finally, we came full circle with an algorithm called "Dagger Hashimoto". "Dashimoto", as it was sometimes called in short, borrowed many ideas from [Hashimoto](https://pdfs.semanticscholar.org/3b23/7cc60c1b9650e260318d33bec471b8202d5e.pdf) (<https://pdfs.semanticscholar.org/3b23/7cc60c1b9650e260318d33bec471b8202d5e.pdf>), a proof of work algorithm by Thaddeus Dryja that pioneered the notion of "I/O bound proof of work", where the dominant limiting factor in mining speed was not hashes per second, but rather megabytes per second of RAM access. However, it combined this with Dagger's notion of light-client-friendly DAG-generated datasets. After many rounds of tweaking by myself, Matthew, Tim and others, the ideas finally converged into the algorithm we now call [Ethash](https://github.com/ethereum/wiki/wiki/Ethash) (<https://github.com/ethereum/wiki/wiki/Ethash>).

```

def hashimoto(header, nonce, full_size, dataset_lookup):
    n = full_size / HASH_BYTES
    w = MIX_BYTES // WORD_BYTES
    mixhashes = MIX_BYTES / HASH_BYTES
    # combine header+nonce into a 64 byte seed
    s = sha3_512(header + nonce[::-1])
    # start the mix with replicated s
    mix = []
    for _ in range(MIX_BYTES / HASH_BYTES):
        mix.extend(s)
    # mix in random dataset nodes
    for i in range(ACCESSES):
        p = fnv(i ^ s[0], mix[i % w]) % (n // mixhashes) * mixhashes
        newdata = []
        for j in range(MIX_BYTES / HASH_BYTES):
            newdata.extend(dataset_lookup(p + j))
        mix = map(fnv, mix, newdata)
    # compress mix
    cmix = []
    for i in range(0, len(mix), 4):
        cmix.append(fnv(fnv(fnv(mix[i], mix[i+1]), mix[i+2]), mix[i+3]))
    return {
        "mix digest": serialize_hash(cmix),
        "result": serialize_hash(sha3_256(s+cmix))
    }

def hashimoto_light(full_size, cache, header, nonce):
    return hashimoto(header, nonce, full_size, lambda x: calc_dataset_item(cache, x))

def hashimoto_full(full_size, dataset, header, nonce):
    return hashimoto(header, nonce, full_size, lambda x: dataset[x])

```

By the summer of 2014, the protocol had considerably stabilized, with the major exception of the proof of work algorithm which would not reach the Ehash phase until around the beginning of 2015, and a semi-formal specification existed in the form of Gavin's [yellow paper](http://gavwood.com/Paper.pdf) (<http://gavwood.com/Paper.pdf>).

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER

EIP-150 REVISION

DR. GAVIN WOOD
FOUNDER, ETHEREUM & ETHCORE
GAVIN@ETHCORE.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, not least Bitcoin. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

1. INTRODUCTION

With ubiquitous internet connections in most places of the world, global information transmission has become incredibly cheap. Technology-rooted movements like Bitcoin have demonstrated, through the power of the default, consensus mechanisms and voluntary respect of the social contract that it is possible to use the internet to make

information is often lacking, and plain old prejudices are difficult to shake.

Overall, I wish to provide a system such that users can be guaranteed that no matter with which other individuals, systems or organisations they interact, they can do so with absolute confidence in the possible outcomes and how those outcomes might come about.

In August 2014, I developed and introduced the [uncle mechanism](https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/) (<https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/>), which allows Ethereum's blockchain to have a shorter block time and higher capacity while mitigating centralization risks. This was introduced as part of PoC6.

Discussions with the Bitshares team led us to consider [adding heaps](https://blog.ethereum.org/2014/08/27/state-ethereum-august-edition/) (<https://blog.ethereum.org/2014/08/27/state-ethereum-august-edition/>) as a first-class data structure, though we ended up not doing this due to lack of time, and later security audits and DoS attacks will show that it is actually much harder than we had thought at the time to do this safely.

In September, Gavin and I planned out the next two major changes to the protocol design. First, alongside the state tree and transaction tree, every block would also contain a "receipt tree". The receipt tree would include hashes of the logs created by a transaction, along with intermediate state roots. Logs would allow transactions to

create "outputs" that are saved in the blockchain, and are accessible to light clients, but that are not accessible to future state calculations. This could be used to allow decentralized applications to easily query for events, such as token transfers, purchases, exchange orders being created and filled, auctions being started, and so forth.

There were other ideas that were considered, like making a Merkle tree out of the entire execution trace of a transaction to allow anything to be proven; logs were chosen because they were a compromise between simplicity and completeness.

The second was the idea of "precompiles", solving the problem of allowing complex cryptographic computations to be usable in the EVM without having to deal with EVM overhead. We had also gone through many more ambitious ideas about "[native contracts \(<https://blog.ethereum.org/2014/08/27/state-ethereum-august-edition/>\)](https://blog.ethereum.org/2014/08/27/state-ethereum-august-edition/)", where if miners have an optimized implementation of some contracts they could "vote" the gasprice of those contracts down, so contracts that most miners could execute much more quickly would naturally have a lower gas price; however, all of these ideas were rejected because we could not come up with a cryptoeconomically safe way to implement such a thing. An attacker could always create a contract which executes some trapdoored cryptographic operation, distribute the trapdoor to themselves and their friends to allow them to execute this contract much faster, then vote the gasprice down and use this to DoS the network. Instead we opted for the much less ambitious approach of having a smaller number of precompiles that are simply specified in the protocol, for common operations such as hashes and signature schemes.

Gavin was also a key initial voice in developing the idea of "[protocol abstraction \(<https://blog.ethereum.org/2015/07/05/on-abstraction/>\)](https://blog.ethereum.org/2015/07/05/on-abstraction/)" - moving as many parts of the protocol such as ether balances, transaction signing algorithms, nonces, etc into the protocol itself as contracts, with a theoretical final goal of reaching a situation where the entire ethereum protocol could be described as making a function call into a virtual machine that has some pre-initialized state. There was not enough time for these ideas to get into the initial Frontier release, but the principles are expected to start slowly getting integrated through some of the Constantinople changes, the Casper contract and the sharding specification.

This was all implemented in PoC7; after PoC7, the protocol did not really change much, with the exception of minor, though in some cases important, details that would come out through security audits...

In early 2015, came the pre-launch security audits organized by Jutta Steiner and others, which included both software code audits and academic audits. The software audits were primarily on the C++ and Go implementations, which were led by Gavin Wood and Jeffrey Wilcke, respectively, though there was also a smaller audit on my pyethereum implementation. Of the two academic audits, one was performed by Ittay Eyal (of "selfish mining" fame), and the other by Andrew Miller and others from Least Authority. The Eyal audit led to a minor protocol change: the total difficulty of a chain would not include uncles. The [Least Authority audit \(\[https://leastauthority.com/blog/least_authority_performs_incentive_analysis_for_ethereum/\]\(https://leastauthority.com/blog/least_authority_performs_incentive_analysis_for_ethereum/\)\)](https://leastauthority.com/blog/least_authority_performs_incentive_analysis_for_ethereum/) was more focused on smart contract and gas economics, as well as the Patricia tree. This audit led to several protocol changes. One small one is the use of sha3(addr) and sha3(key) as trie keys instead of the address and key directly; this would make it harder to perform a worst-case attack on the trie.

There are useful parallels between this refund loop and the publish-subscribe function illustrated in Miller's thesis. He demonstrates several hazards that are present when the `publish` callbacks are run synchronously:

- exceptions raised during the callback would prevent execution of later callbacks
- reentrancy hazards if the callback itself executes `publish()`, `subscribe()`, or `unsubscribe()`: repeated actions, missing actions, and inconsistent delivery of messages

Some analogous issues in the crowdfund example are:

- delivering a contribution, after the funding deadline, with just enough gas to allow some refunds to go through, but not all: the contract could be left in a state where it was unable to refund the remaining contributions
- if the refund was triggered by a contract at the end of a long call stack, the `send` instructions will fail. However the example appears to ignore the return value of the `send`, so execution will continue. All records will be cleared, and the funds can never be recovered.
- **the refund callback could make a new donation, triggering another refund cycle, potentially double-refunding the earlier contributions, or failing to refund later ones**

And a warning that was perhaps a bit too far ahead of its time...

Another significant thing that we discussed was the gas limit voting mechanism. At the time, we were already concerned by perceived lack of progress in the bitcoin block size debate, and wanted to have a more flexible design in Ethereum that could adjust over time as needed. But the challenge is: what is the optimal limit? My initial thought had been to make a dynamic limit, targeting 1.5· the long-term exponential moving average of the actual gas usage, so that in the long run on average blocks would be $\frac{2}{3}$ full. However, Andrew showed that this was exploitable in some ways - specifically, miners who wanted to raise the limit would simply include

transactions in their own blocks that consume a very large amount of gas, but take very little time to process, and thereby always create full blocks at no cost to themselves. The security model was thus, at least in the upward direction, equivalent to simply having miners vote on the gas limit.

We did not manage to come up with a gas limit strategy that was less likely to break, and so Andrew's recommended solution was to simply have miners vote on the gas limit explicitly, and have the default strategy for voting be the 1.5- EMA rule. The reasoning was that we were still very far from knowing the right approach for setting maximum gas limits, and the risk of any specific approach failing seemed greater than the risk of miners abusing their voting power. Hence, we might as well simply let miners vote on the gas limit, and accept the risk that the limit will go too high or too low, in exchange for the benefit of flexibility, and the ability for miners to work together to very quickly adjust the limit upwards or downwards as needed.



After a mini-hackathon between Gavin, Jeff and myself, PoC9 was launched in March, and was intended to be the final proof of concept release. A testnet, Olympic, ran for four months, using the protocol that was intended to be used in the livenet, and Ethereum's long-term plan was established. Vinay Gupta wrote a blog post, "[The Ethereum Launch Process](#) (<https://blog.ethereum.org/2015/03/03/ethereum-launch-process/>)", that described the four expected stages of Ethereum livenet development, and gave them their current names: Frontier, Homestead, Metropolis and Serenity.

Olympic ran for four months. In the first two months, many bugs were found in the various implementations, consensus failures happened, among other issues, but around June the network noticeably stabilized. In July a decision was made to make a code-freeze, followed by a release, and on July 30 the release took place.



On Medium-of-Exchange Token Valuations

2017 Oct 17

[See all posts \(/\)](#)

One kind of token model that has become popular among many recent token sale projects is the "network medium of exchange token". The general pitch for this kind of token goes as follows. We, the developers, build a network, and this network allows you to do new cool stuff. This network is a sharing-economy-style system: it consists purely of a set of sellers, that provide resources within some protocol, and buyers that purchase the services, where both buyers and sellers come from the community. But the purchase and sale of things within this network *must* be done with the new token that we're selling, and this is why the token will have value.

If it were the developers themselves that were acting as the seller, then this would be a very reasonable and normal arrangement, very similar in nature to a Kickstarter-style product sale. The token actually would, in a meaningful economic sense, be backed by the services that are provided by the developers.

We can see this in more detail by describing what is going on in a simple economic model. Suppose that N people value a product that a developer wants to release at $\$x$, and believe the developer will give them the product. The developer does a sale, and raises N units for $\$w < x$ each, thus raising a total revenue of $\$Nw$. The developer builds the product, and gives it to each of the buyers. At the end of the day, the buyers are happy, and the developer is happy. Nobody feels like they made an avoidable mistake in participating, and everyone's expectations have been met. This kind of economic model is clearly stable.

Now, let's look at the story with a "medium of exchange" token. N people value a product that will exist in a decentralized network at $\$x$; the product will be sold at a price of $\$w < x$. They each buy $\$w$ of tokens in the sale. The developer builds the network. Some sellers come in, and offer the product inside the network for $\$w$. The buyers use their tokens to purchase this product, spending $\$w$ of tokens and getting $\$x$ of value. The sellers spend $\$v < w$ of resources and effort producing this product, and they now have $\$w$ worth of tokens.

Notice that here, the cycle is not complete, and in fact it never will be; there needs to be an ongoing stream of buyers and sellers for the token to continue having its value. The stream does not strictly speaking *have to be* endless; if in every round there is a chance of at least $\frac{v}{w}$ that there will be a next round, then the model still works, as even though *someone* will eventually be cheated, the risk of any individual participant becoming that person is lower than the benefit that they get from participating. It's also totally possible that the token would depreciate in each round, with its value multiplying by some factor f where $\frac{v}{w} < f < 1$, until it eventually reaches a price of zero, and it would still be on net in everyone's interest to participate. Hence, the model is theoretically feasible, but you can see how this model is more complex and more tenuous than the simple "developers as seller" model.

Traditional macroeconomics has a [simple equation](https://en.wikipedia.org/wiki/Equation_of_exchange) (https://en.wikipedia.org/wiki/Equation_of_exchange) to try to value a medium of exchange:

$$MV = PT$$

Here:

- M is the total money supply; that is, the total number of coins
- V is the "velocity of money"; that is, the number of times that an average coin changes hands every day
- P is the "price level". This is the price of goods and services *in terms of* the token; so it is actually the *inverse* of the currency's price
- T is the transaction volume: the economic value of transactions per day

The proof for this is a trivial equality: if there are N coins, and each changes hands M times per day, then this is $M \cdot N$ coins' worth of economic value transacted per day. If this represents $\$T$ worth of economic value, then the price of each coin is $\frac{T}{M \cdot N}$, so the "price level" is the inverse of this, $\frac{M \cdot N}{T}$.

For easier analysis, we can recast two variables:

- We refer to $\frac{1}{V}$ with H , the time that a user holds a coin before using it to make a transaction
- We refer to $\frac{1}{P}$ with C , the price of the currency (think $C = \text{cost}$)

Now, we have:

$$\frac{M}{H} = \frac{T}{C}$$

$$MC = TH$$

The left term is quite simply the market cap. The right term is the economic value transacted per day, multiplied by the amount of time that a user holds a coin before using it to transact.

This is a steady-state model, assuming that the same quantity of users will also be there. In reality, however, the quantity of users may change, and so the price may change. The time that users hold a coin may change, and this may cause the price to change as well.

Let us now look once again at the economic effect on the users. *What do users lose* by using an application with a built-in appcoin rather than plain old ether (or bitcoin, or USD)? The simplest way to express this is as follows: the "implicit cost" imposed by such a system on users *the cost to the user of holding those coins for that period of time, instead of holding that value in the currency that they would otherwise have preferred to hold*.

There are many factors involved in this cost: cognitive costs, exchange costs and spreads, transaction fees, and many smaller items. One particular significant factor of this implicit cost is expected return. If a user expects the appcoin to only grow in value by 1% per year, while their other available alternatives grow 3% per year, and they hold \$20 of the currency for five days, then that is an expected loss of roughly $\$20 \cdot 2$.

One immediate conclusion from this particular insight is that **appcoins are very much a multi-equilibrium game**. If the appcoin grows at 2% per year, then the fee drops to \$0.0027, and this essentially makes the "de-facto fee" of the application (or at least a large component of it) 2x cheaper, attracting more users and

growing its value more. If the appcoin starts falling at 10% per year, however, then the "de-facto fee" grows to \$0.035, driving many users away and accelerating its growth.

This leads to increased opportunities for market manipulation, as a manipulator would not just be wasting their money fighting against a single equilibrium, but may in fact successfully nudge a given currency from one equilibrium into another, and profit from successfully "predicting" (ie. causing) this shift. It also means there is a large amount of path dependency, and established brands matter a lot; witness the epic battles over which fork of the bitcoin blockchain can be called Bitcoin for one particular high-profile example.

Another, and perhaps even more important, conclusion is that the market cap of an appcoin **depends crucially on the holding time** H . If someone creates a very efficient exchange, which allows users to purchase an appcoin in real time and then immediately use it in the application, then allowing sellers to immediately cash out, then the market cap would drop precipitously. If a currency is stable or prospects are looking optimistic, then this may not matter because users actually see no disadvantage from holding the token instead of holding something else (ie. zero "de-facto fee"), but if prospects start to turn sour then such a well-functioning exchange can accelerate its demise.

You might think that exchanges are inherently inefficient, requiring users to create an account, login, deposit coins, wait for 36 confirmations, trade and logout, but in fact hyper-efficient exchanges are around the corner. Here (https://www.reddit.com/r/ethereum/comments/55m04x/lets_run_onchain_decentralized_exchanges_the_way/) is a thread discussing designs for fully autonomous synchronous on-chain transactions, which can convert token A into token B, and possibly even then use token B to do something, *within a single transaction*. Many other platforms are being developed as well.

What this all serves to show is that relying purely on the medium-of-exchange argument to support a token value, while attractive because of its seeming ability to print money out of thin air, is ultimately quite brittle. Protocol tokens using this model may well be sustained for some time due to irrationality and temporary equilibria where the implicit cost of holding the token is zero, but it is a kind of model which always has an unavoidable risk of collapsing at any time.

So what is the alternative? One simple alternative is the etherdelta approach, where an application simply collects fees in the interface. One common criticism is: but can't someone fork the interface to take out the fees? A counter-retort is: someone can also fork the interface to replace your protocol token with ETH, BTC, DOGE or whatever else users would prefer to use. One can make a more sophisticated argument that this is hard because the "pirate" version would have to compete with the "official" version for network effect, but one can just as easily create an official fee-paying client that refuses to interact with non-fee-paying clients as well; this kind of network effect-based enforcement is similar to how value-added-taxes are typically enforced in Europe and other places. Official-client buyers would not interact with non-official-client sellers, and official-client sellers would not interact with non-official-client buyers, so a large group of users would need to switch to the "pirate" client at the same time to successfully dodge fees. This is not perfectly robust, but it is certainly as good as the approach of creating a new protocol token.

If developers want to front-load revenue to fund initial development, then they can sell a token, with the property that all fees paid are used to buy back some of the token and burn it; this would make the token backed by the future expected value of upcoming fees spent inside the system. One can transform this design into a more direct utility token by requiring users to use the utility token to pay fees, and having the interface use an exchange to automatically purchase tokens if the user does not have tokens already.

The important thing is that for the token to have a stable value, it is highly beneficial for the token supply to have **sinks** - places where tokens actually disappear and so the total token quantity decreases over time. This way, there is a more transparent and explicit fee paid by users, instead of the highly variable and difficult to calculate "de-facto fee", and there is also a more transparent and explicit way to figure out what the value of protocol tokens should be.

STARKs, Part I: Proofs with Polynomials

2017 Nov 09

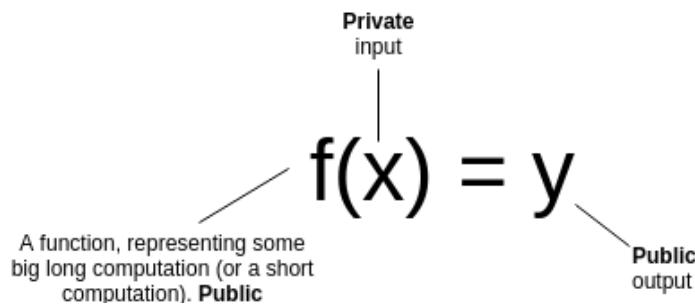
[See all posts \(/\)](#)

Special thanks to Eli Ben-Sasson for ongoing help, explanations and review, coming up with some of the examples used in this post, and most crucially of all inventing a lot of this stuff; thanks to Hsiao-wei Wang for reviewing

Hopefully many people by now have heard of [ZK-SNARKs](https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6), the general-purpose succinct zero knowledge proof technology that can be used for all sorts of usecases ranging from verifiable computation to privacy-preserving cryptocurrency. What you might not know is that ZK-SNARKs have a newer, shinier cousin: ZK-STARKs. With the T standing for "transparent", ZK-STARKs resolve one of the primary weaknesses of ZK-SNARKs, its reliance on a "trusted setup". They also come with much simpler cryptographic assumptions, avoiding the need for elliptic curves, pairings and the knowledge-of-exponent assumption and instead relying purely on hashes and information theory; this also means that they are secure even against attackers with quantum computers.

However, this comes at a cost: the size of a proof goes up from 288 bytes to a few hundred kilobytes. Sometimes the cost will not be worth it, but at other times, particularly in the context of public blockchain applications where the need for trust minimization is high, it may well be. And if elliptic curves break or quantum computers do come around, it definitely will be.

So how does this other kind of zero knowledge proof work? First of all, let us review what a general-purpose succinct ZKP does. Suppose that you have a (public) function f , a (private) input x and a (public) output y . You want to prove that you know an x such that $f(x) = y$, without revealing what x is. Furthermore, for the proof to be *succinct*, you want it to be verifiable much more quickly than computing f itself.



Let's go through a few examples:

- f is a computation that takes two weeks to run on a regular computer, but two hours on a data center. You send the data center the computation (ie. the code to run f), the data center runs it, and gives back the answer y with a proof. You verify the proof in a few milliseconds, and are convinced that y actually is the answer.
- You have an encrypted transaction, of the form " X_1 was my old balance. X_2 was your old balance. X_3 is my new balance. X_4 is your new balance". You want to create a proof that this transaction is valid

(specifically, old and new balances are non-negative, and the decrease in my balance cancels out the increase in your balance). x can be the *pair of encryption keys*, and f can be a function which contains as a built-in public input the transaction, takes as input the keys, decrypts the transaction, performs the check, and returns 1 if it passes and 0 if it does not. y would of course be 1.

- You have a blockchain like Ethereum, and you download the most recent block. You want a proof that this block is valid, and that this block is at the tip of a chain where every block in the chain is valid. You ask an existing full node to provide such a proof. x is the entire blockchain (yes, all ?? gigabytes of it), f is a function that processes it block by block, verifies the validity and outputs the hash of the last block, and y is the hash of the block you just downloaded.

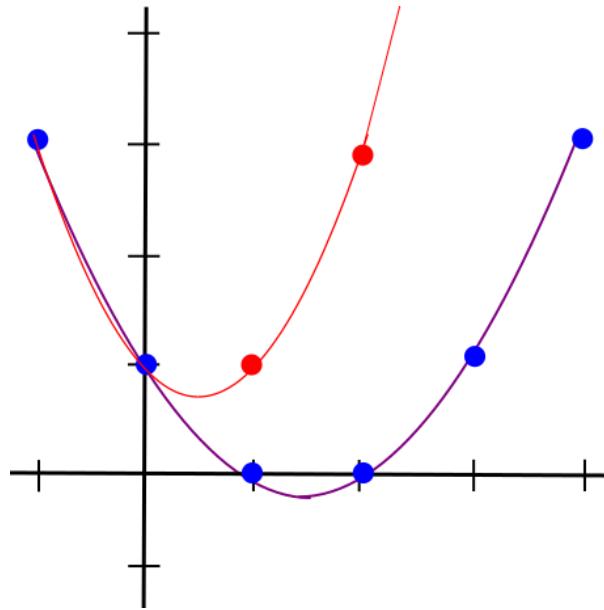


So what's so hard about all this? As it turns out, the *zero knowledge* (ie. privacy) guarantee is (relatively!) easy to provide; there are a bunch of ways to convert any computation into an instance of something like the three color graph problem, where a three-coloring of the graph corresponds to a solution of the original problem, and then use a traditional zero knowledge proof protocol to prove that you have a valid graph coloring without revealing what it is. This [excellent post by Matthew Green from 2014](#) (<https://blog.cryptographyengineering.com/2014/11/27/zero-knowledge-proofs-illustrated-primer/>) describes this in some detail.

The much harder thing to provide is *succinctness*. Intuitively speaking, proving things about computation succinctly is hard because computation is *incredibly fragile*. If you have a long and complex computation, and you as an evil genie have the ability to flip a 0 to a 1 anywhere in the middle of the computation, then in many cases even one flipped bit will be enough to make the computation give a completely different result. Hence, it's hard to see how you can do something like randomly sampling a computation trace in order to gauge its correctness, as it's just too easy to miss that "one evil bit". However, with some fancy math, it turns out that you can.

The general very high level intuition is that the protocols that accomplish this use similar math to what is used in [erasure coding](#) (https://en.wikipedia.org/wiki/Erasure_coding), which is frequently used to make *data fault-tolerant*. If you have a piece of data, and you encode the data as a line, then you can pick out four points on

the line. Any two of those four points are enough to reconstruct the original line, and therefore also give you the other two points. Furthermore, if you make even the slightest change to the data, then it is guaranteed at least three of those four points. You can also encode the data as a degree-1,000,000 polynomial, and pick out 2,000,000 points on the polynomial; any 1,000,001 of those points will recover the original data and therefore the other points, and any deviation in the original data will change at least 1,000,000 points. The algorithms shown here will make heavy use of polynomials in this way for *error amplification*.

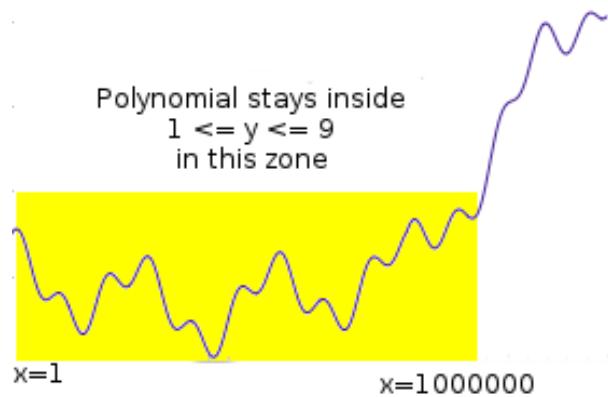


Changing even one point in the original data will lead to large changes in a polynomial's trajectory

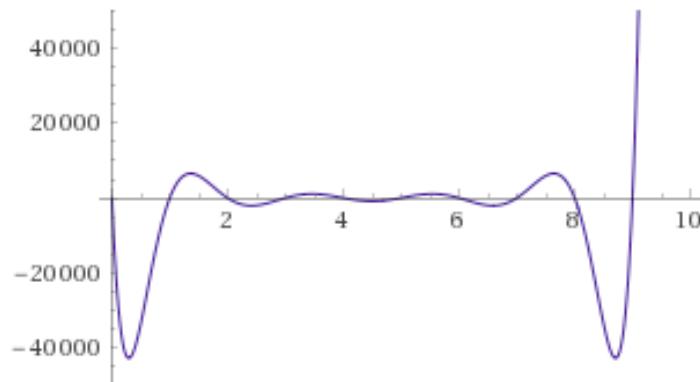
A Somewhat Simple Example

Suppose that you want to prove that you have a polynomial P such that $P(x)$ is an integer with $0 \leq P(x) \leq 9$ for all x from 1 to 1 million. This is a simple instance of the fairly common task of "range checking"; you might imagine this kind of check being used to verify, for example, that a set of account balances is still positive after applying some set of transactions. If it were $1 \leq P(x) \leq 9$, this could be part of checking that the values form a correct Sudoku solution.

The "traditional" way to prove this would be to just show all 1,000,000 points, and verify it by checking the values. However, we want to see if we can make a proof that can be verified in less than 1,000,000 steps. Simply randomly checking evaluations of P won't do; there's always the possibility that a malicious prover came up with a P which satisfies the constraint in 999,999 places but does not satisfy it in the last one, and random sampling only a few values will almost always miss that value. So what *can* we do?

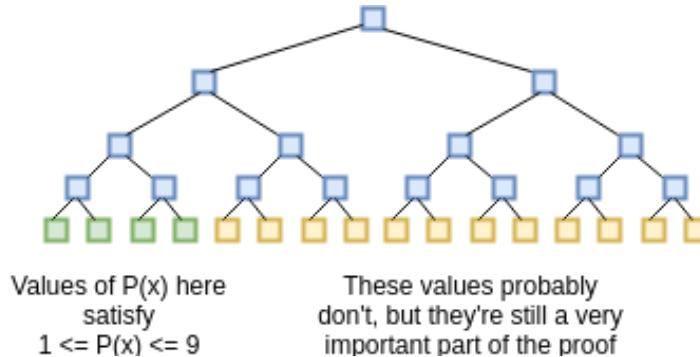


Let's mathematically transform the problem somewhat. Let $C(x)$ be a *constraint checking polynomial*; $C(x) = 0$ if $0 \leq x \leq 9$ and is nonzero otherwise. There's a simple way to construct $C(x)$: $x \cdot (x - 1) \cdot (x - 2) \cdots (x - 9)$ (we'll assume all of our polynomials and other values use exclusively integers, so we don't need to worry about numbers in between).

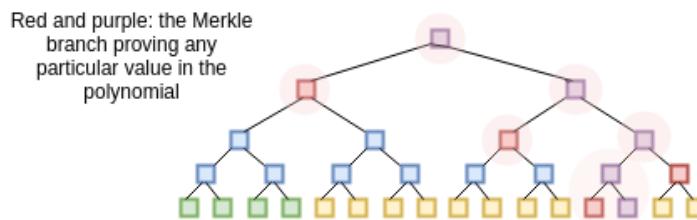


Now, the problem becomes: prove that you know P such that $C(P(x)) = 0$ for all x from 1 to 1,000,000. Let $Z(x) = (x - 1) \cdot (x - 2) \cdots (x - 1000000)$. It's a known mathematical fact that *any* polynomial which equals zero at all x from 1 to 1,000,000 is a multiple of $Z(x)$. Hence, the problem can now be transformed again: prove that you know P and D such that $C(P(x)) = Z(x) \cdot D(x)$ for all x (note that if you know a suitable $C(P(x))$ then dividing it by $Z(x)$ to compute $D(x)$ is not too difficult; you can use [long polynomial division](#) (<http://www.purplemath.com/modules/polydiv2.htm>) or more realistically a faster algorithm based on [FFTs](#) (https://en.wikipedia.org/wiki/Fast_Fourier_transform)). Now, we've converted our original statement into something that looks mathematically clean and possibly quite provable.

So how does one prove this claim? We can imagine the proof process as a three-step communication between a prover and a verifier: the prover sends some information, then the verifier sends some requests, then the prover sends some more information. First, the prover commits to (ie. makes a Merkle tree and sends the verifier the root hash of) the evaluations of $P(x)$ and $D(x)$ for all x from 1 to 1 billion (yes, billion). This includes the 1 million points where $0 \leq P(x) \leq 9$ as well as the 999 million points where that (probably) is not the case.



We assume the verifier already knows the evaluation of $Z(x)$ at all of these points; the $Z(x)$ is like a "public verification key" for this scheme that everyone must know ahead of time (clients that do not have the space to store $Z(x)$ in its entirety can simply store the Merkle root of $Z(x)$ and require the prover to also provide branches for every $Z(x)$ value that the verifier needs to query; alternatively, there are some number fields over which $Z(x)$ for certain x is very easy to calculate). After receiving the commitment (ie. Merkle root) the verifier then selects a random 16 x values between 1 and 1 billion, and asks the prover to provide the Merkle branches for $P(x)$ and $D(x)$ there. The prover provides these values, and the verifier checks that (i) the branches match the Merkle root that was provided earlier, and (ii) $C(P(x))$ actually equals $Z(x) \cdot D(x)$ in all 16 cases.



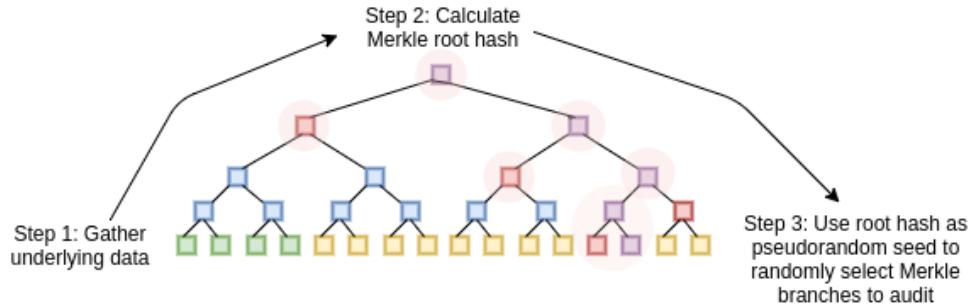
We know that this proof *perfect completeness* - if you actually know a suitable $P(x)$, then if you calculate $D(x)$ and construct the proof correctly it will always pass all 16 checks. But what about *soundness* - that is, if a malicious prover provides a bad $P(x)$, what is the minimum probability that they will get caught? We can analyze as follows. Because $C(P(x))$ is a degree-10 polynomial composed with a degree-1,000,000 polynomial, its degree will be at most 10,000,000. In general, we know that two different degree- N polynomials agree on at most N points; hence, a degree-10,000,000 polynomial which is not equal to any polynomial which always equals $Z(x) \cdot D(x)$ for some x will necessarily disagree with them all at least 990,000,000 points. Hence, the probability that a bad $P(x)$ will get caught in even one round is already 99%; with 16 checks, the probability of getting caught goes up to $1 - 10^{-32}$; that is to say, the scheme is about as hard to spoof as it is to compute a hash collision.

So... what did we just do? We used polynomials to "boost" the error in any bad solution, so that any incorrect solution to the original problem, which would have required a million checks to find directly, turns into a solution to the verification protocol that can get flagged as erroneous at 99% of the time with even a single check.

We can convert this three-step mechanism into a *non-interactive proof*, which can be broadcasted by a single prover once and then verified by anyone, using the Fiat-Shamir heuristic (https://en.wikipedia.org/wiki/Fiat-Shamir_heuristic). The prover first builds up a Merkle tree of the $P(x)$ and $D(x)$ values, and computes the root hash of the tree. The root itself is then used as the source of entropy that determines what branches of the tree the prover needs to provide. The prover then broadcasts the Merkle

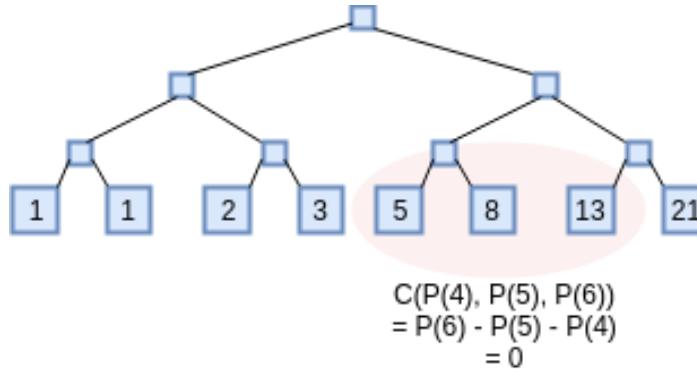
root and the branches together as the proof. The computation is all done on the prover side; the process of computing the Merkle root from the data, and then using that to select the branches that get audited, effectively substitutes the need for an interactive verifier.

The only thing a malicious prover without a valid $P(x)$ can do is try to make a valid proof over and over again until eventually they get *extremely* lucky with the branches that a Merkle root that they compute selects, but with a soundness of $1 - 10^{-32}$ (ie. probability of at least $1 - 10^{-32}$ that a given attempted fake proof will fail the check) it would take a malicious prover billions of years to make a passable proof.



Going Further

To illustrate the power of this technique, let's use it to do something a little less trivial: prove that you know the millionth Fibonacci number. To accomplish this, we'll prove that you have knowledge of a polynomial which represents a computation tape, with $P(x)$ representing the x th Fibonacci number. The constraint checking polynomial will now hop across three x -coordinates: $C(x_1, x_2, x_3) = x_3 - x_2 - x_1$ (notice how if $C(P(x), P(x + 1), P(x + 2)) = 0$ for all x then $P(x)$ represents a Fibonacci sequence).



The translated problem becomes: prove that you know P and D such that $C(P(x), P(x + 1), P(x + 2)) = Z(x) \cdot D(x)$. For each of the 16 indices that the proof audits, the prover will need to provide Merkle branches for $P(x)$, $P(x + 1)$, $P(x + 2)$ and $D(x)$. The prover will additionally need to provide Merkle branches to show that $P(0) = P(1) = 1$. Otherwise, the entire process is the same.

Now, to accomplish this in reality there are two problems that need to be resolved. The first problem is that if we actually try to work with regular numbers the solution would not be efficient *in practice*, because the numbers themselves very easily get extremely large. The millionth Fibonacci number, for example, has 208988 digits. If we actually want to achieve succinctness in practice, instead of doing these polynomials with regular numbers, we need to use finite fields - number systems that still follow the same arithmetic laws we know and

love, like $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$ and $(a^2 - b^2) = (a - b) \cdot (a + b)$, but where each number is guaranteed to take up a constant amount of space. Proving claims about the millionth Fibonacci number would then require a more complicated design that implements big-number arithmetic *on top of* this finite field math.

The simplest possible finite field is modular arithmetic; that is, replace every instance of $a + b$ with $a + b \bmod N$ for some prime N , do the same for subtraction and multiplication, and for division use modular inverses (https://en.wikipedia.org/wiki/Modular_multiplicative_inverse) (eg. if $N = 7$, then $3 + 4 = 0$, $2 + 6 = 1$, $3 \cdot 4 = 5$, $4/2 = 2$ and $5/2 = 6$). You can learn more about these kinds of number systems from my description on prime fields here (<https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>) (search "prime field" in the page) or this Wikipedia article (https://en.wikipedia.org/wiki/Modular_arithmetic) on modular arithmetic (the articles that you'll find by searching directly for "finite fields" and "prime fields" unfortunately tend to be very complicated and go straight into abstract algebra, don't bother with those).

Second, you might have noticed that in my above proof sketch for soundness I neglected to cover one kind of attack: what if, instead of a plausible degree-1,000,000 $P(x)$ and degree-9,000,000 $D(x)$, the attacker commits to some values that are not on *any* such relatively-low-degree polynomial? Then, the argument that an invalid $C(P(x))$ must differ from any valid $C(P(x))$ on at least 990 million points does not apply, and so different and much more effective kinds of attacks are possible. For example, an attacker could generate a random value p for every x , then compute $d = C(p)/Z(x)$ and commit to these values in place of $P(x)$ and $D(x)$. These values would not be on any kind of low-degree polynomial, but they *would* pass the test.

It turns out that this possibility can be effectively defended against, though the tools for doing so are fairly complex, and so you can quite legitimately say that they make up the bulk of the mathematical innovation in STARKs. Also, the solution has a limitation: you can weed out commitments to data that are *very* far from any degree-1,000,000 polynomial (eg. you would need to change 20% of all the values to make it a degree-1,000,000 polynomial), but you cannot weed out commitments to data that only differ from a polynomial in only one or two coordinates. Hence, what these tools will provide is *proof of proximity* - proof that *most* of the points on P and D correspond to the right kind of polynomial.

As it turns out, this is sufficient to make a proof, though there are two "catches". First, the verifier needs to check a few more indices to make up for the additional room for error that this limitation introduces. Second, if we are doing "boundary constraint checking" (eg. verifying $P(0) = P(1) = 1$ in the Fibonacci example above), then we need to extend the proof of proximity to not only prove that most points are on the same polynomial, but also prove that *those two specific points* (or whatever other number of specific points you want to check) are on that polynomial.

In the next part of this series, I will describe the solution to proximity checking in much more detail, and in the third part I will describe how more complex constraint functions can be constructed to check not just Fibonacci numbers and ranges, but also arbitrary computation.

STARKs, Part II: Thank Goodness It's FRI-day

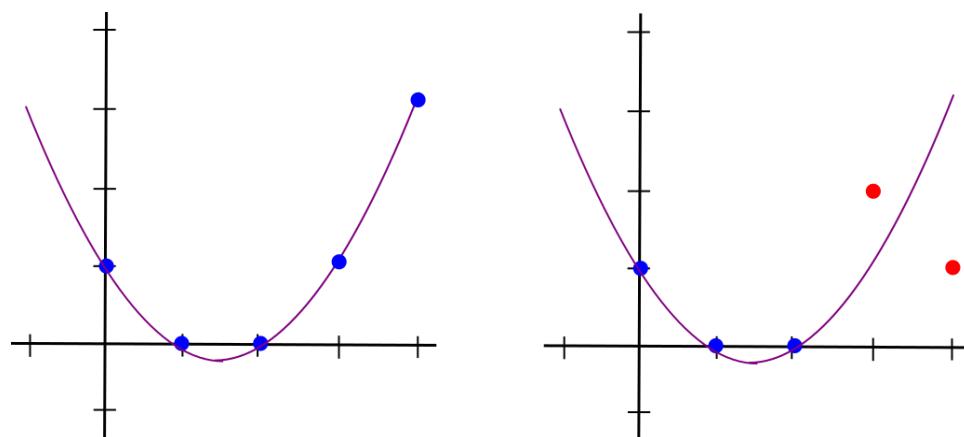
2017 Nov 22

[See all posts \(1\)](#)

Special thanks to Eli Ben-Sasson for ongoing help and explanations, and Justin Drake for reviewing

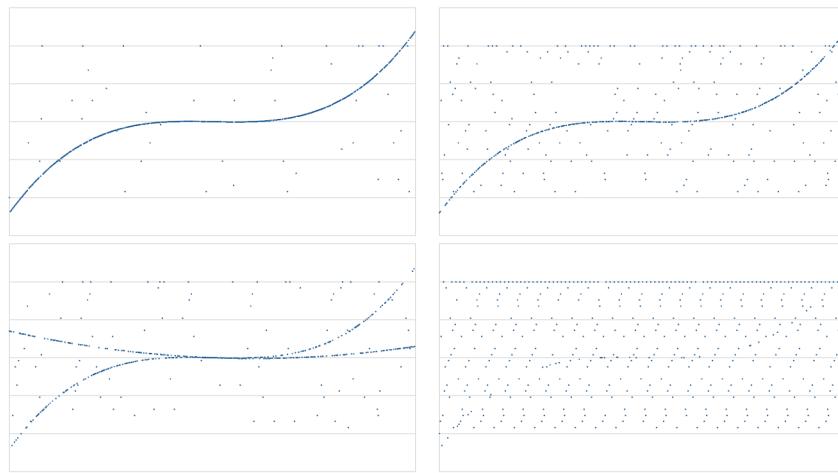
In the last part of this series, we talked about how you can make some pretty interesting succinct proofs of computation, such as proving that you have computed the millionth Fibonacci number, using a technique involving polynomial composition and division. However, it rested on one critical ingredient: the ability to prove that at least the great majority of a given large set of points are on the same low-degree polynomial. This problem, called "low-degree testing", is perhaps the single most complex part of the protocol.

We'll start off by once again re-stating the problem. Suppose that you have a set of points, and you claim that they are all on the same polynomial, with degree less than D (ie. $\deg < 2$ means they're on the same line, $\deg < 3$ means they're on the same line or parabola, etc). You want to create a succinct probabilistic proof that this is actually true.



Left: points all on the same $\deg < 3$ polynomial. Right: points not on the same $\deg < 3$ polynomial

If you want to verify that the points are *all* on the same degree $< D$ polynomial, you would have to actually check every point, as if you fail to check even one point there is always some chance that that point will not be on the polynomial even if all the others are. But what you *can* do is *probabilistically check* that at least *some fraction* (eg. 90%) of all the points are on the same polynomial.

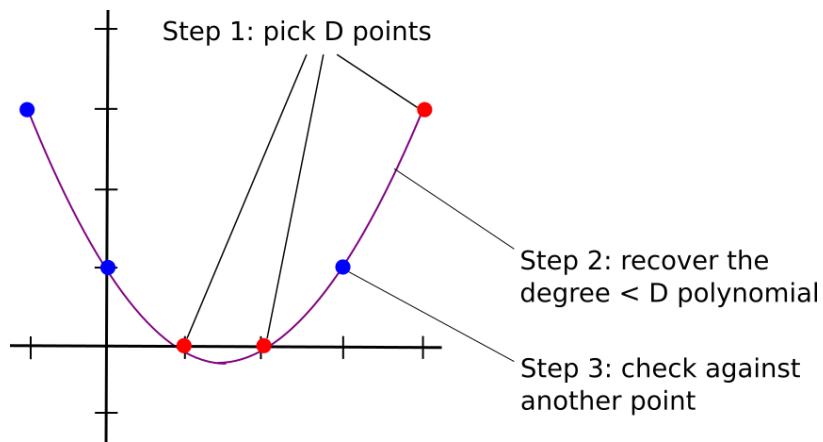


Top left: possibly close enough to a polynomial. Top right: not close enough to a polynomial. Bottom left: somewhat close to two polynomials, but not close enough to either one. Bottom right: definitely not close enough to a polynomial.

If you have the ability to look at every point on the polynomial, then the problem is easy. But what if you can only look at a few points - that is, you can ask for whatever specific point you want, and the prover is obligated to give you the data for that point as part of the protocol, but the total number of queries is limited? Then the question becomes, how many points do you need to check to be able to tell with some given degree of certainty?

Clearly, D points is **not** enough. D points are exactly what you need to uniquely define a degree $< D$ polynomial, so *any* set of points that you receive will correspond to *some* degree $< D$ polynomial. As we see in the figure above, however, $D + 1$ points or more *do* give some indication.

The algorithm to check if a given set of values is on the same degree $< D$ polynomial with $D + 1$ queries is not too complex. First, select a random subset of D points, and use something like Lagrange interpolation (search for "Lagrange interpolation" [here](https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649) (<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>) for a more detailed description) to recover the unique degree $< D$ polynomial that passes through all of them. Then, randomly sample one more point, and check that it is on the same polynomial.



Note that this is only a proximity test, because there's always the possibility that most points are on the same low-degree polynomial, but a few are not, and the $D + 1$ sample missed those points entirely. However, we can derive the result that if less than 90% of the points are on the same degree $< D$ polynomial, then the test will fail with high probability. Specifically, if you make $D + k$ queries, and if at least some portion p of the points are not on the same polynomial as the rest of the points, then the test will only pass with probability $(1 - p)^k$.

But what if, as in the examples from the previous article, D is very high, and you want to verify a polynomial's degree with less than D queries? This is, of course, impossible to do directly, because of the simple argument made above (namely, that *any* $k \leq D$ points are all on at least one degree $< D$ polynomial). However, it's quite possible to do this indirectly *by providing auxiliary data*, and achieve massive efficiency gains by doing so. And this is exactly what new protocols like [FRI](https://eccc.weizmann.ac.il/report/2017/134/) (<https://eccc.weizmann.ac.il/report/2017/134/>) ("Fast RS IOPP", RS = "[Reed-Solomon](#) (https://en.wikipedia.org/wiki/Reed%20%20Solomon_error_correction#Constructions.)", IOPP = "Interactive Oracle Proofs of Proximity"), and similar earlier designs called probabilistically checkable proofs of proximity (PCPPs), try to achieve.

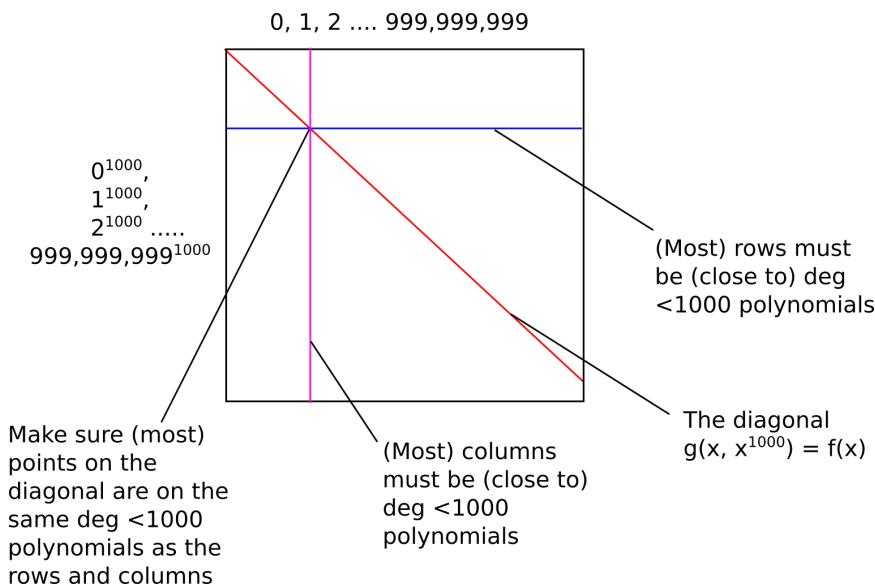
A First Look at Sublinearity

To prove that this is at all possible, we'll start off with a relatively simple protocol, with fairly poor tradeoffs, but that still achieves the goal of sublinear verification complexity - that is, you can prove proximity to a degree $< D$ polynomial with less than D queries (and, for that matter, less than $O(D)$ computation to verify the proof).

The idea is as follows. Suppose there are N points (we'll say $N = 1$ billion), and they are all on a degree $< 1,000,000$ polynomial $f(x)$. We find a bivariate polynomial (ie. an expression like $1 + x + xy + x^5 \cdot y^3 + x^{12} + x \cdot y^{11}$), which we will denote $g(x, y)$, such that $g(x, x^{1000}) = f(x)$. This can be done as follows: for the k th degree term in $f(x)$ (eg. $1744 \cdot x^{185423}$), we decompose it into $x^{k \% 1000} \cdot y^{\text{floor}(k/1000)}$ (in this case, $1744 \cdot x^{423} \cdot y^{185}$). You can see that if $y = x^{1000}$, then $1744 \cdot x^{423} \cdot y^{185}$ equals $1744 \cdot x^{185423}$.

In the first stage of the proof, the prover commits to (ie. makes a Merkle tree of) the evaluation of $g(x, y)$ over the *entire* square $[1\dots N]x\{x^{1000} : 1 \leq x \leq N\}$ - that is, all 1 billion x coordinates for the columns, and all 1 billion corresponding *thousandth powers* for the y coordinates of the rows. The diagonal of the square represents the values of $g(x, y)$ that are of the form $g(x, x^{1000})$, and thus correspond to values of $f(x)$.

The verifier then randomly picks perhaps a few dozen rows and columns (possibly using [the Merkle root of the square as a source of pseudorandomness](#) (https://en.wikipedia.org/wiki/Fiat%20%20Shamir_heuristic) if we want a non-interactive proof), and for each row or column that it picks the verifier asks for a sample of, say, 1010 points on the row and column, making sure in each case that one of the points demanded is on the diagonal. The prover must reply back with those points, along with Merkle branches proving that they are part of the original data committed to by the prover. The verifier checks that the Merkle branches match up, and that the points that the prover provides actually do correspond to a degree-1000 polynomial.



This gives the verifier a statistical proof that (i) most rows are populated mostly by points on degree < 1000 polynomials, (ii) most columns are populated mostly by points on degree < 1000 polynomials, and (iii) the diagonal line is mostly on these polynomials. This thus convinces the verifier that most points on the diagonal actually do correspond to a degree $< 1,000,000$ polynomial.

If we pick thirty rows and thirty columns, the verifier needs to access a total of $1010 \text{ points} \cdot 60 \text{ rows+cols} = 60600$ points, less than the original $1,000,000$, but not by that much. As far as computation time goes, interpolating the degree < 1000 polynomials will have its own overhead, though since polynomial interpolation can be made subquadratic the algorithm as a whole is still sublinear to verify. The *prover* complexity is higher: the prover needs to calculate and commit to the entire $N \cdot N$ rectangle, which is a total of 10^{18} computational effort (actually a bit more because polynomial evaluation is still superlinear). In all of these algorithms, it will be the case that proving a computation is substantially more complex than just running it; but as we will see the overhead does not have to be *that* high.

A Modular Math Interlude

Before we go into our more complex protocols, we will need to take a bit of a digression into the world of modular arithmetic. Usually, when we work with algebraic expressions and polynomials, we are working with regular numbers, and the arithmetic, using the operators $+$, $-$, \cdot , $/$ (and exponentiation, which is just repeated multiplication), is done in the usual way that we have all been taught since school: $2 + 2 = 4$, $72/5 = 14.4$, $1001 \cdot 1001 = 1002001$, etc. However, what mathematicians have realized is that these ways of defining addition, multiplication, subtraction and division are not the *only* self-consistent ways of defining those operators.

The simplest example of an alternate way to define these operators is modular arithmetic, defined as follows. The $\%$ operator means "take the remainder of": 15, 53, etc (note that the answer is always non-negative, so for example -1). For any specific prime number p , we can redefine:

$$x + y \Rightarrow (x + y) \% p$$

$$x \cdot y \Rightarrow (x \cdot y) \% p$$

$$x^y \Rightarrow (x^y) \% p$$

$$x - y \Rightarrow (x - y) \% p$$

$$x/y \Rightarrow (x \cdot y^{p-2}) \% p$$

The above rules are all self-consistent. For example, if $p = 7$, then:

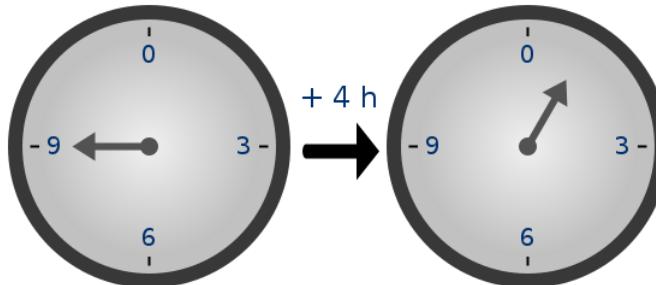
- $5 + 3 = 1$ (as $8 \% 7 = 1$)
- $1 - 3 = 5$ (as $-2 \% 7 = 5$)
- $2 \cdot 5 = 3$
- $3/5 = 2$ (as $(3 \cdot 5^5) \% 7 = 9375 \% 7 = 2$)

More complex identities such as the distributive law also hold: $(2 + 4) \cdot 3$ and $2 \cdot 3 + 4 \cdot 3$ both evaluate to 4. Even formulas like $(a^2 - b^2) = (a - b) \cdot (a + b)$ are still true in this new kind of arithmetic. Division is the hardest part; we can't use regular division because we want the values to always remain integers, and regular division often gives non-integer results (as in the case of $3/5$). The funny $p - 2$ exponent in the division formula above is a consequence of getting around this problem using Fermat's little theorem

(https://en.wikipedia.org/wiki/Fermat%27s_little_theorem), which states that for any nonzero $x < p$, it holds that $x^{p-1} \% p = 1$. This implies that x^{p-2} gives a number which, if multiplied by x one more time, gives 1, and so we can say that x^{p-2} (which is an integer) equals $\frac{1}{x}$. A somewhat more complicated but faster way to evaluate this modular

division operator is the [extended Euclidean algorithm](https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm) (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm), implemented in python [here](#)

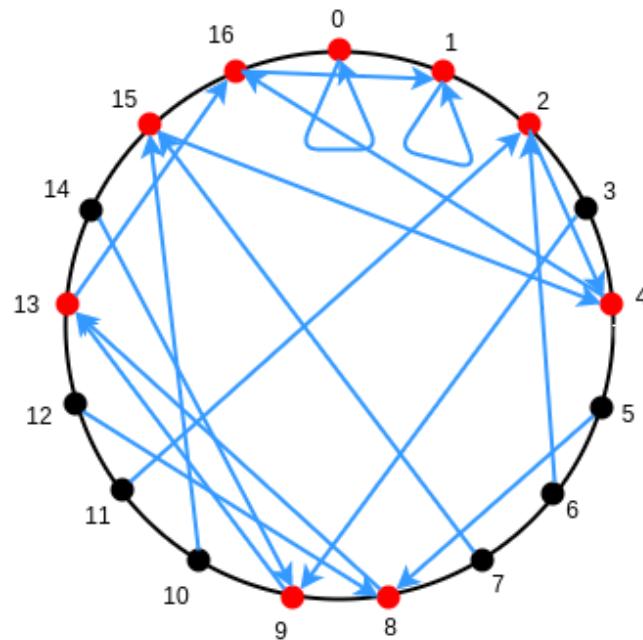
(https://github.com/ethereum/py_ecc/blob/b036cf5cb37e9b89622788ec714a7da9cdb2e635/py_ecc/secp256k1/secp256k1.py#L34)



Because of how the numbers "wrap around", modular arithmetic is sometimes called "clock math"

With modular math we've created an entirely new system of arithmetic, and because it's self-consistent in all the same ways traditional arithmetic is self-consistent we can talk about all of the same kinds of structures over this field, including polynomials, that we talk about in "regular math". Cryptographers love working in modular math (or, more generally, "finite fields") because there is a bound on the size of a number that can arise as a result of any modular math calculation - no matter what you do, the values will not "escape" the set $\{0, 1, 2 \dots p - 1\}$.

Fermat's little theorem also has another interesting consequence. If $p - 1$ is a multiple of some number k , then the function $x \rightarrow x^k$ has a small "image" - that is, the function can only give $\frac{p-1}{k} + 1$ possible results. For example, $x \rightarrow x^2$ with $p = 17$ has only 9 possible results.



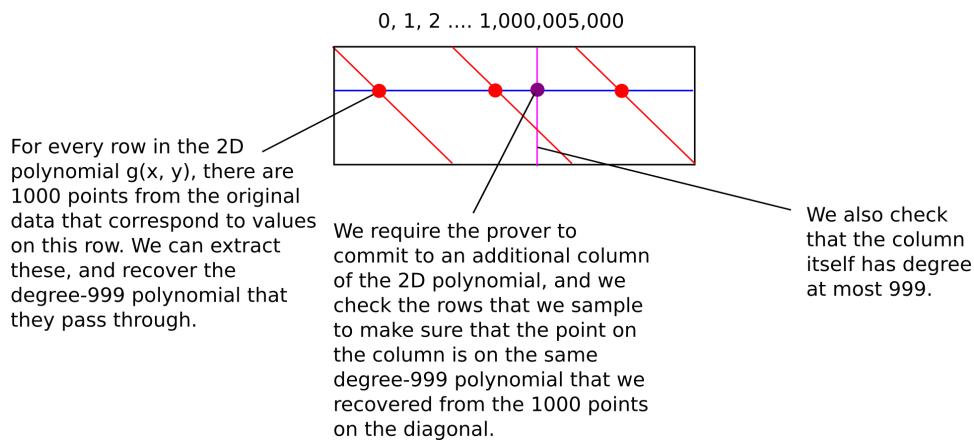
With higher exponents the results are more striking: for example, $x \rightarrow x^8$ with $p = 17$ has only 3 possible results. And of course, $x \rightarrow x^{16}$ with $p = 17$ has only 2 possible results: for 0 it returns 0, and for everything else it returns 1.

Now A Bit More Efficiency

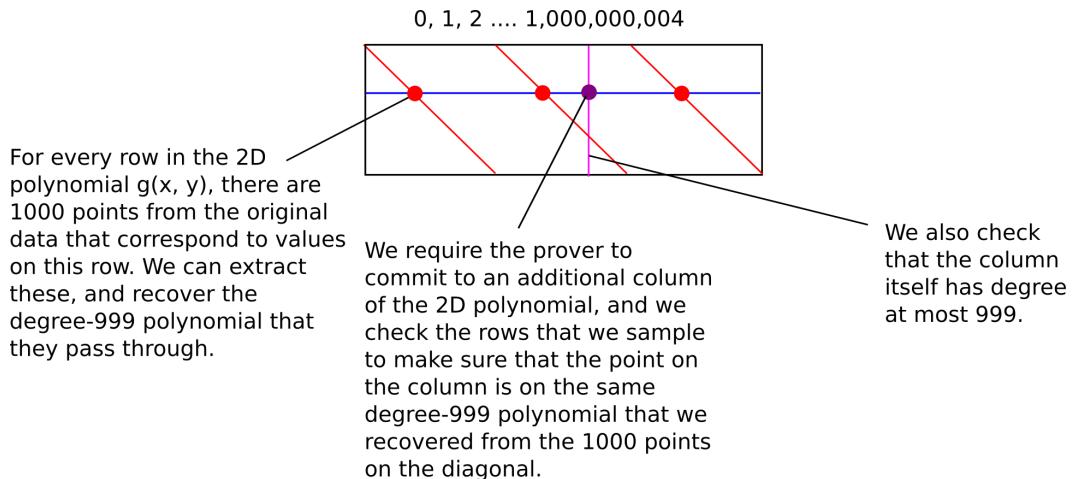
Let us now move on to a slightly more complicated version of the protocol, which has the modest goal of reducing the prover complexity from 10^{18} to 10^{15} , and then 10^9 . First, instead of operating over regular numbers, we are going to be checking proximity to polynomials *as evaluated with modular math*. As we saw in the previous article, we need to do this to prevent numbers in our STARKs from growing to 200,000 digits anyway. Here, however, we are going to use the "small image" property of certain modular exponentiations as a side effect to make our protocols far more efficient.

Specifically, we will work with $p = 1,000,005,001$. We pick this modulus because (i) it's greater than 1 billion, and we need it to be at least 1 billion so we can check 1 billion points, (ii) it's prime, and (iii) $p - 1$ is an even multiple of 1000. The exponentiation x^{1000} will have an image of size 1,000,006 - that is, the exponentiation can only give 1,000,006 possible results.

This means that the "diagonal" (x, x^{1000}) now becomes a diagonal with a wraparound; as x^{1000} can only take on 1,000,006 possible values, we only need 1,000,006 rows. And so, the full evaluation of $g(x, x^{1000})$ now has only $\sim 10^{15}$ elements.



As it turns out, we can go further: we can have the prover only commit to the evaluation of g on a single column. The key trick is that the original data itself already contains 1000 points that are on any given row, so we can simply sample those, derive the degree < 1000 polynomial that they are on, and then check that the corresponding point on the column is on the same polynomial. We then check that the column itself is a < 1000 polynomial.



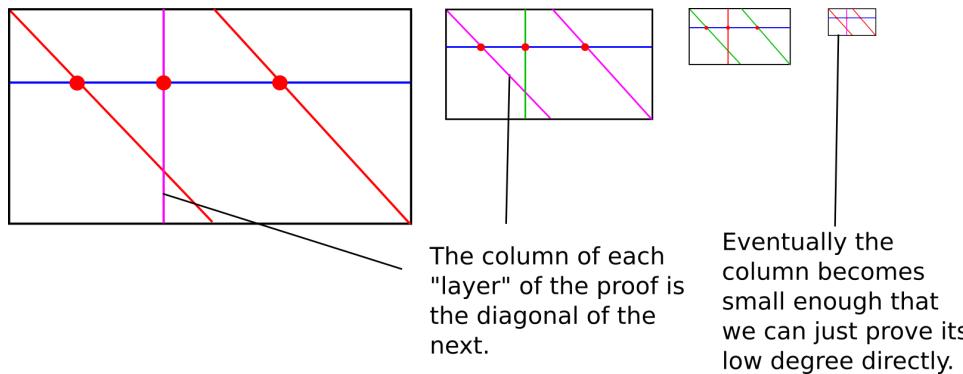
The verifier complexity is still sublinear, but the prover complexity has now decreased to 10^9 , making it linear in the number of queries (though it's still superlinear in practice because of polynomial evaluation overhead).

And Even More Efficiency

The prover complexity is now basically as low as it can be. But we can still knock the verifier complexity down further, from quadratic to logarithmic. And the way we do *that* is by making the algorithm recursive. We start off with the last protocol above, but instead of trying to embed a polynomial into a 2D polynomial where the degrees in x and y are equal, we embed the polynomial into a 2D polynomial where the degree bound in x is a small constant value; for simplicity, we can even say this must be 2. That is, we express $f(x) = g(x, x^2)$, so that the row check always requires only checking 3 points on each row that we sample (2 from the diagonal plus one from the column).

If the original polynomial has degree $< n$, then the rows have degree < 2 (ie. the rows are straight lines), and the column has degree $< \frac{n}{2}$. Hence, what we now have is a linear-time process for converting a problem of proving proximity to a polynomial of degree $< n$ into a problem of proving proximity to a polynomial of degree $< \frac{n}{2}$.

Furthermore, the number of points that need to be committed to, and thus the prover's computational complexity, goes down by a factor of 2 each time (Eli Ben-Sasson likes to compare this aspect of FRI to [fast fourier transforms](#) (https://en.wikipedia.org/wiki/Fast_Fourier_transform), with the key difference that unlike with FFTs, each step of recursion only introduces one new sub-problem instead of branching out into two). Hence, we can simply keep using the protocol on the column created in the previous round of the protocol, until the column becomes so small that we can simply check it directly; the total complexity is something like $n + \frac{n}{2} + \frac{n}{4} + \dots \approx 2n$.



In reality, the protocol will need to be repeated several times, because there is still a significant probability that an attacker will cheat one round of the protocol. However, even still the proofs are not too large; the verification complexity is logarithmic in the degree, though it goes up to $\log^2 n$ if you count the size of the Merkle proofs.

The "real" FRI protocol also has some other modifications; for example, it uses a binary [Galois field](#) (https://en.wikipedia.org/wiki/Finite_field#Explicit_construction_of_finite_fields) (another weird kind of finite field; basically, the same thing as the 12th degree extension fields I talk about [here](#) (<https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>), but with the prime modulus being 2). The exponent used for the row is also typically 4 and not 2. These modifications increase efficiency and make the system friendlier to building STARKs on top of it. However, these modifications are not essential to understanding how the algorithm works, and if you really wanted to, you could definitely make STARKs with the simple modular math-based FRI described here too.

Soundness

I will warn that *calculating soundness* - that is, determining just how low the probability is that an optimally generated fake proof will pass the test for a given number of checks - is still somewhat of a "here be dragons" area in this space. For the simple test where you take 1,000,000 + k points, there is a simple lower bound: if a given dataset has the property that, for any polynomial, at least portion p of the dataset is not on the polynomial, then a test on that dataset will pass with at most $(1 - p)^k$ probability. However, even that is a very pessimistic

lower bound - for example, it's not possible to be much more than 50% close to two low-degree polynomials at the same time, and the probability that the first points you select will be the one with the most points on it is quite low. For full-blown FRI, there are also complexities involving various specific kinds of attacks.

Here (<https://eccc.weizmann.ac.il/report/2016/149/>) is a recent article by Ben-Sasson et al describing soundness properties of FRI in the context of the entire STARK scheme. In general, the "good news" is that it seems likely that in order to pass the $D(x) \cdot Z(x) = C(P(x))$ check on the STARK, the $D(x)$ values for an invalid solution would need to be "worst case" in a certain sense - they would need to be maximally far from *any* valid polynomial. This implies that we don't need to check for *that* much proximity. There are proven lower bounds, but these bounds would imply that an actual STARK need to be ~1-3 megabytes in size; conjectured but not proven stronger bounds reduce the required number of checks by a factor of 4.

The third part of this series will deal with the last major part of the challenge in building STARKs: how we actually construct constraint checking polynomials so that we can prove statements about arbitrary computation, and not just a few Fibonacci numbers.

A Quick Gasprice Market Analysis

2017 Dec 14

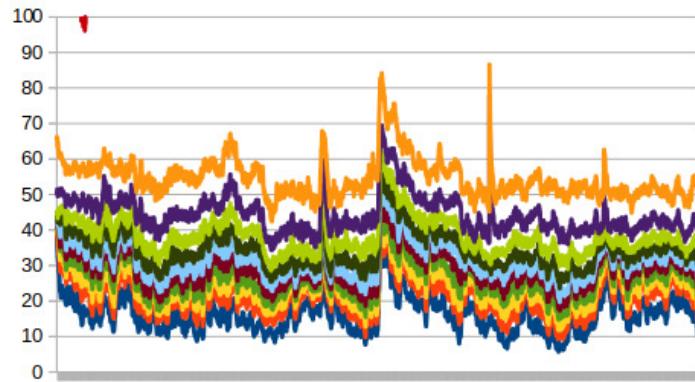
[See all posts \(/\)](#)

Here is [a file \(/images/gas-analysis-files/gas_analysis.json\)](#) that contains data, extracted from geth, about transaction fees in every block between 4710000 and 4730000. For each block, it contains an object of the form:

```
{
  "block":4710000,
  "coinbase":"0x829bd824b016326a401d083b33d092293333a830",
  "deciles":[40,40.1,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001,44.100030001],
  "free":10248,
  "timedelta":8
}
```

The "deciles" variable contains 11 values, where the lowest is the lowest gasprice in each block, the next is the gasprice that only 10% of other transaction gasprices are lower than, and so forth; the last is the highest gasprice in each block. This gives us a convenient summary of the distribution of transaction fees that each block contains. We can use this data to perform some interesting analyses.

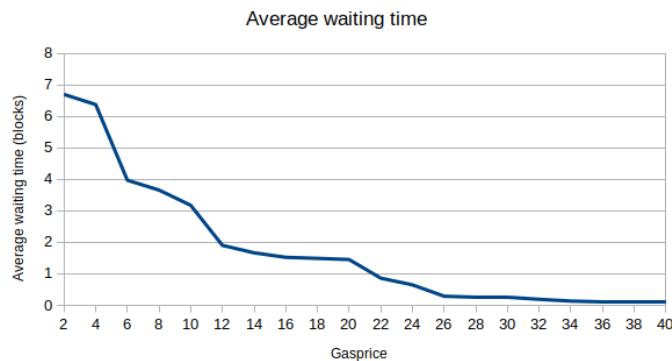
First, a chart of the deciles, taking 50-block moving averages to smooth it out:



What we see is a gasprice market that seems to actually stay reasonably stable over the course of more than three days. There are a few occasional spikes, most notably the one around block 4720000, but otherwise the deciles all stay within the same band all the way through. The only exception is the highest gasprice transaction (that red squiggle at the top left), which fluctuates wildly because it can be pushed upward by a single very-high-gasprice transaction.

We can try to interpret the data in another way: by calculating, for each gasprice level, the average number of blocks that you need to wait until you see a block where the lowest gasprice included is lower than that gasprice. Assuming that miners are rational and all have the same view (implying that if the lowest gasprice in

a block is X, then that means there are no more transactions with gasprices above X waiting to be included), this might be a good proxy for the average amount of time that a transaction sender needs to wait to get included if they use that gasprice. The stats are:



There is clear clustering going on at the 4, 10 and 20 levels; it seems to be an underexploited strategy to send transactions with fees slightly above these levels, getting in before the crowd of transactions right at the level but only paying a little more.

However, there is quite a bit of evidence that miners **do not** have the same view; that is, some miners see a very different set of transactions from other miners. First of all, we can filter blocks by miner address, and check what the deciles of each miner are. Here is the output of this data, splitting by 2000-block ranges so we can spot behavior that is consistent across the entire period, and filtering out miners that mine less than 10 blocks in any period, as well as filtering out blocks with more 21000 free gas (high levels of free gas may signify an abnormally high minimum gas price policy, like for example 0x6a7a43be33ba930fe58f34e07d0ad6ba7adb9b1f at ~40 gwei and 0xb75d1e62b10e4ba91315c4aa3facc536f8a922f5 at ~10 gwei). We get:

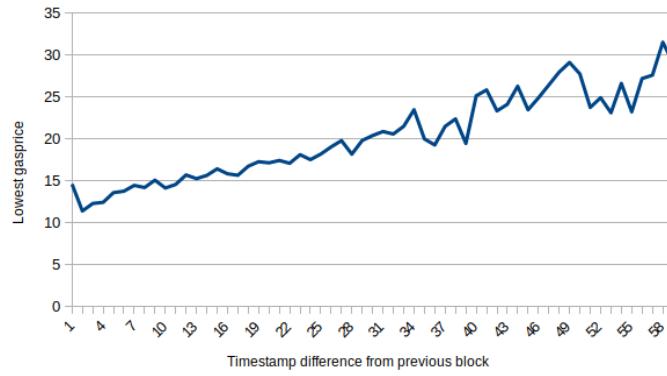
```

0x829bd824b016326a401d083b33d092293333a830 [30, 28, 27, 21, 28, 34, 23, 24, 32, 32]
0xea674fdde714fd979de3edf0f56aa9716b898ec8 [17, 11, 10, 15, 17, 23, 17, 13, 16, 17]
0x5a0b54d5dc17e0aadc383d2db43b0a0d3e029c4c [31, 17, 20, 18, 16, 27, 21, 15, 21, 21]
0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5 [20, 16, 19, 14, 17, 18, 17, 14, 15, 15]
0xb2930b35844a230f00e51431acae96fe543a0347 [21, 17, 19, 17, 17, 25, 17, 16, 19, 19]
0x180ba8f73897c0cb26d76265fc7868cf936e617 [13, 13, 15, 18, 12, 26, 16, 13, 20, 20]
0xf3b9d2c81f2b24b0fa0acaaa865b7d9ced5fc2fb [26, 25, 23, 21, 22, 28, 25, 24, 26, 25]
0x4bb96091ee9d802ed039c4d1a5f6216f90f81b01 [17, 21, 17, 14, 21, 32, 14, 14, 19, 23]
0x2a65aca4d5fc5b5c859090a6c34d164135398226 [26, 24, 20, 16, 22, 33, 20, 18, 24, 24]

```

The first miner is consistently higher than the others; the last is also higher than average, and the second is consistently among the lowest.

Another thing we can look at is timestamp differences - the difference between a block's timestamp and its parent. There is a clear correlation between timestamp difference and lowest gasprice:



This makes a lot of sense, as a block that comes right after another block should be cleaning up only the transactions that are too low in gasprice for the parent block to have included, and a block that comes a long time after its predecessor would have many more not-yet-included transactions to choose from. The differences are large, suggesting that a single block is enough to bite off a very substantial chunk of the unconfirmed transaction pool, adding to the evidence that most transactions are included quite quickly.

However, if we look at the data in more detail, we see very many instances of blocks with low timestamp differences that contain many transactions with higher gasprices than their parents. Sometimes we do see blocks that actually look like they clean up what their parents could not, like this:

```
{"block":4710093,"coinbase":"0x5a0b54d5dc17e0aadcc383d2db43b0a0d3e029c4c","deciles":[25,40,40,
{"block":4710094,"coinbase":"0xea674fdde714fd979de3edf0f56aa9716b898ec8","deciles":[4,16,20,2
```

But sometimes we see this:

```
{"block":4710372,"coinbase":"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5","deciles":[1,30,35,4
{"block":4710373,"coinbase":"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5","deciles":[1,32,32,4
```

And sometimes we see miners suddenly including many 1-gwei transactions:

```
{"block":4710379,"coinbase":"0x5a0b54d5dc17e0aadcc383d2db43b0a0d3e029c4c","deciles":[21,25,31,
{"block":4710380,"coinbase":"0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5","deciles":[1,1,1,1,1
```

This strongly suggests that a miner including transactions with gasprice X should NOT be taken as evidence that there are not still many transactions with gasprice higher than X left to process. This is likely because of imperfections in network propagation.

In general, however, what we see seems to be a rather well-functioning fee market, though there is still room to improve in fee estimation and, most importantly of all, continuing to work hard to improve base-chain scalability so that more transactions can get included in the first place.

Notes on Blockchain Governance

2017 Dec 17

[See all posts \(/\)](#)

In which I argue that "tightly coupled" on-chain voting is overrated, the status quo of "informal governance" as practiced by Bitcoin, Bitcoin Cash, Ethereum, Zcash and similar systems is much less bad than commonly thought, that people who think that the purpose of blockchains is to completely expunge soft mushy human intuitions and feelings in favor of completely algorithmic governance (emphasis on "completely") are absolutely crazy, and loosely coupled voting as done by Carbonvotes and similar systems is underrated, as well as describe what framework should be used when thinking about blockchain governance in the first place.

See also: https://medium.com/@Vlad_Zamfir/against-on-chain-governance-a4ceacd040ca
[\(https://medium.com/@Vlad_Zamfir/against-on-chain-governance-a4ceacd040ca\)](https://medium.com/@Vlad_Zamfir/against-on-chain-governance-a4ceacd040ca).

One of the more interesting recent trends in blockchain governance is the resurgence of on-chain coin-holder voting as a multi-purpose decision mechanism. Votes by coin holders are sometimes used in order to decide who operates the super-nodes that run a network (eg. DPOS in EOS, NEO, Lisk and other systems), sometimes to vote on protocol parameters (eg. the Ethereum gas limit) and sometimes to vote on and directly implement protocol upgrades wholesale (eg. [Tezos](http://tezos.com/) (<http://tezos.com/>)). In all of these cases, the votes are automatic - the protocol itself contains all of the logic needed to change the validator set or to update its own rules, and does this automatically in response to the result of votes.

Explicit on-chain governance is typically touted as having several major advantages. First, unlike the highly conservative philosophy espoused by Bitcoin, it can evolve rapidly and accept needed technical improvements. Second, by creating an *explicit* decentralized framework, it avoids the perceived pitfalls of *informal* governance, which is viewed to either be too unstable and prone to chain splits, or prone to becoming too de-facto centralized - the latter being the same argument made in the famous 1972 essay "[Tyranny of Structurelessness](http://www.jofreeman.com/joreen/tyranny.htm) (<http://www.jofreeman.com/joreen/tyranny.htm>)".

Quoting [Tezos documentation](https://www.tezos.com/governance) (<https://www.tezos.com/governance>):

While all blockchains offer financial incentives for maintaining consensus on their ledgers, no blockchain has a robust on-chain mechanism that seamlessly amends the rules governing its protocol and rewards protocol development. As a result, first-generation blockchains empower de facto, centralized core development teams or miners to formulate design choices.

And (<https://twitter.com/tez0s/status/884528964194238464>):

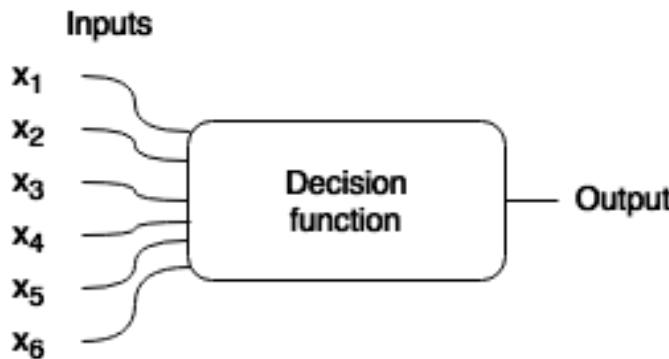
Yes, but why would you want to make [a minority chain split] easier? Splits destroy network effects.

On-chain governance used to select validators also has the benefit that it allows for networks that impose high computational performance requirements on validators without introducing economic centralization risks and other traps of the kind that appear in public blockchains (eg. [the validator's dilemma](https://eprint.iacr.org/2015/702.pdf) (<https://eprint.iacr.org/2015/702.pdf>)).

So far, all in all, on-chain governance seems like a very good bargain.... so what's wrong with it?

What is Blockchain Governance?

To start off, we need to describe more clearly what the process of "blockchain governance" *is*. Generally speaking, there are two informal models of governance, that I will call the "decision function" view of governance and the "coordination" view of governance. The decision function view treats governance as a function $f(x_1, x_2 \dots x_n) \rightarrow y$, where the inputs are the wishes of various legitimate stakeholders (senators, the president, property owners, shareholders, voters, etc) and the output is the decision.



The decision function view is often useful as an approximation, but it clearly frays very easily around the edges: people often can and do break the law and get away with it, sometimes rules are ambiguous, and sometimes revolutions happen - and all three of these possibilities are, at least sometimes, *a good thing*. And often even behavior inside the system is shaped by incentives created by *the possibility* of acting outside the system, and this once again is at least sometimes a good thing.

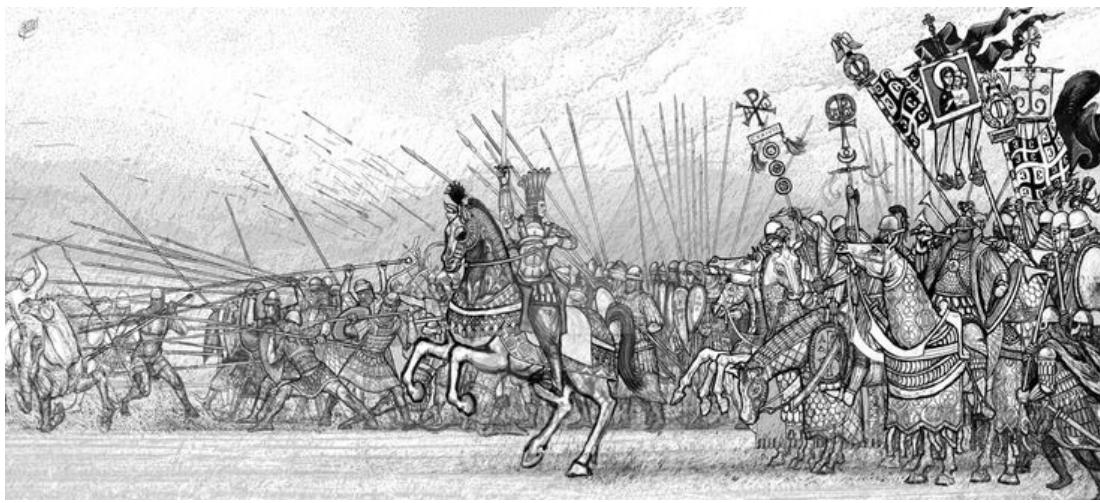
The coordination model of governance, in contrast, sees governance as something that exists in layers. The bottom layer is, in the real world, the laws of physics themselves (as a geopolitical realist would say, guns and bombs), and in the blockchain space we can abstract a bit further and say that it is each individual's ability to run whatever software they want in their capacity as a user, miner, stakeholder, validator or whatever other kind of agent a blockchain protocol allows them to be. The bottom layer is always the ultimate deciding layer; if, for example, all Bitcoin users wake up one day and decides to edit their clients' source code and replace the entire code with an Ethereum client that listens to balances of a particular ERC20 token contract, then that means that that ERC20 token *is* bitcoin. The bottom layer's ultimate governing power cannot be stopped, but the actions that people take on this layer can be *influenced* by the layers above it.

The second (and crucially important) layer is coordination institutions. The purpose of a coordination institution is to create focal points around how and when individuals should act in order to better coordinate behavior. There are many situations, both in blockchain governance and in real life, where if you act in a certain way alone, you are likely to get nowhere (or worse), but if everyone acts together a desired result can be achieved.

	A	B
A	(5, 5)	(0, 0)
B	(0, 0)	(5, 5)

An abstract coordination game. You benefit heavily from making the same move as everyone else.

In these cases, it's in your interest to go if everyone else is going, and stop if everyone else is stopping. You can think of coordination institutions as putting up green or red flags in the air saying "go" or "stop", *with an established culture* that everyone watches these flags and (usually) does what they say. Why do people have the incentive to follow these flags? Because *everyone else* is already following these flags, and you have the incentive to do the same thing as what everyone else is doing.



A Byzantine general rallying his troops forward. The purpose of this isn't just to make the soldiers feel brave and excited, but also to reassure them that *everyone else* feels brave and excited and will charge forward as well, so an individual soldier is not just committing suicide by charging forward alone.

Strong claim: this concept of coordination flags encompasses *all* that we mean by "governance"; in scenarios where coordination games (or more generally, multi-equilibrium games) do not exist, the concept of governance is meaningless.

In the real world, military orders from a general function as a flag, and in the blockchain world, the simplest example of such a flag is the mechanism that tells people whether or not a hard fork "is happening". Coordination institutions can be very formal, or they can be informal, and often give suggestions that are ambiguous. Flags would ideally always be either red or green, but sometimes a flag might be yellow, or even holographic, appearing green to some participants and yellow or red to others. Sometimes there are also multiple flags that conflict with each other.

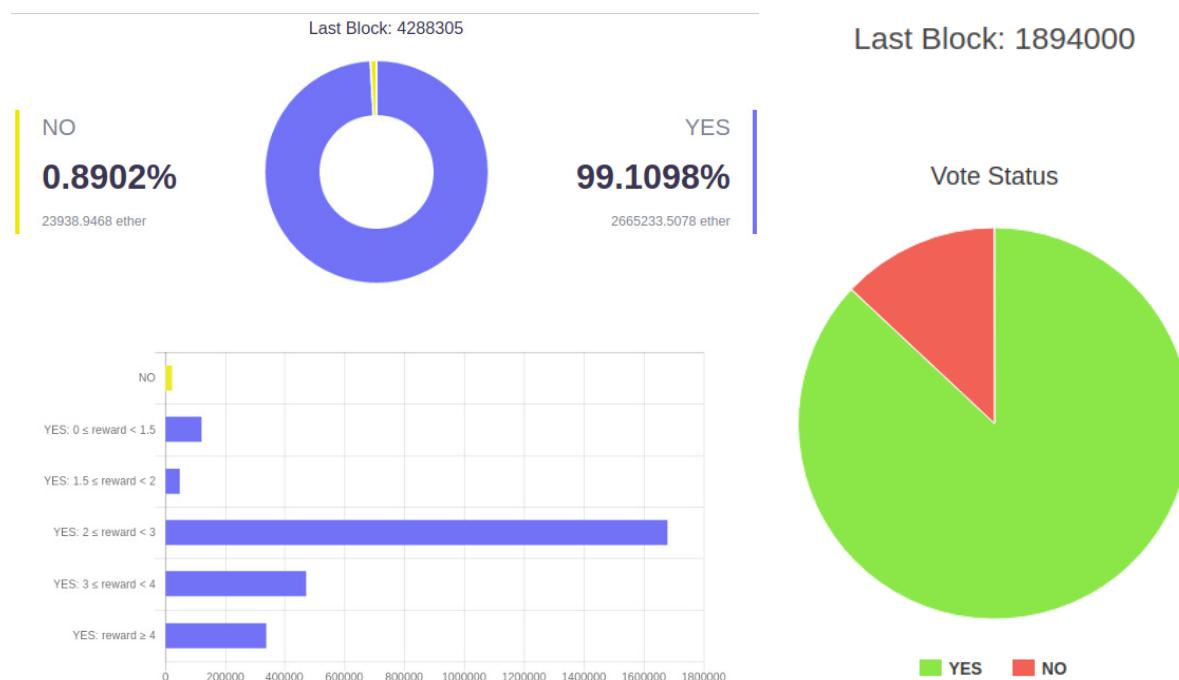
The key questions of governance thus become:

- What should layer 1 be? That is, what features should be set up in the initial protocol itself, and how does this influence the ability to make formulaic (ie. decision-function-like) protocol changes, as well as the level of power of different kinds of agents to act in different ways?
- What should layer 2 be? That is, what coordination institutions should people be encouraged to care about?

The Role of Coin Voting

Ethereum also has a history with coin voting, including:

- **DAO proposal votes:** <https://daostats.github.io/proposals.html>
- **The DAO Carbonvote:** <http://v1.carbonvote.com/>
- **The EIP 186/649/669 Carbonvote:** <http://carbonvote.com/>



These three are all examples of *loosely coupled* coin voting, or coin voting as a layer 2 coordination institution. Ethereum does not have any examples of *tightly coupled* coin voting (or, coin voting as a layer 1 in-protocol feature), though it does have an example of tightly coupled *miner* voting: miners' right to vote on the gas limit.

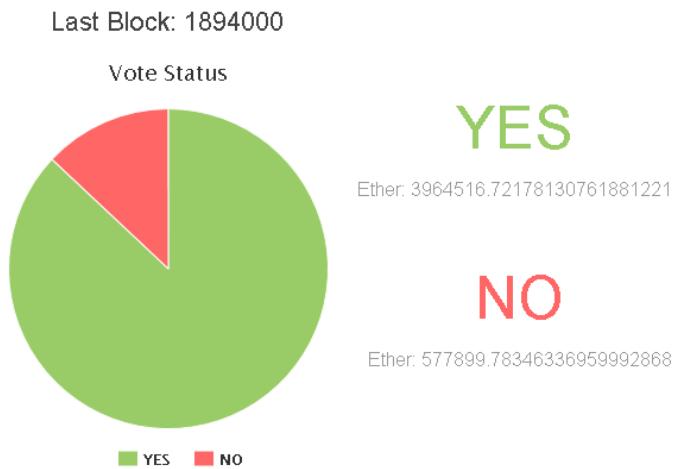
Clearly, tightly coupled voting and loosely coupled voting are competitors in the governance mechanism space, so it's worth dissecting: what are the advantages and disadvantages of each one?

Assuming zero transaction costs, and if used as a sole governance mechanism, the two are clearly equivalent. If a loosely coupled vote says that change X should be implemented, then that will serve as a "green flag" encouraging everyone to download the update; if a minority wants to rebel, they will simply not download the update. If a tightly coupled vote implements change X, then the change happens automatically, and if a minority wants to rebel they can install a hard fork update that cancels the change. However, there clearly are nonzero transaction costs associated with making a hard fork, and this leads to some very important differences.

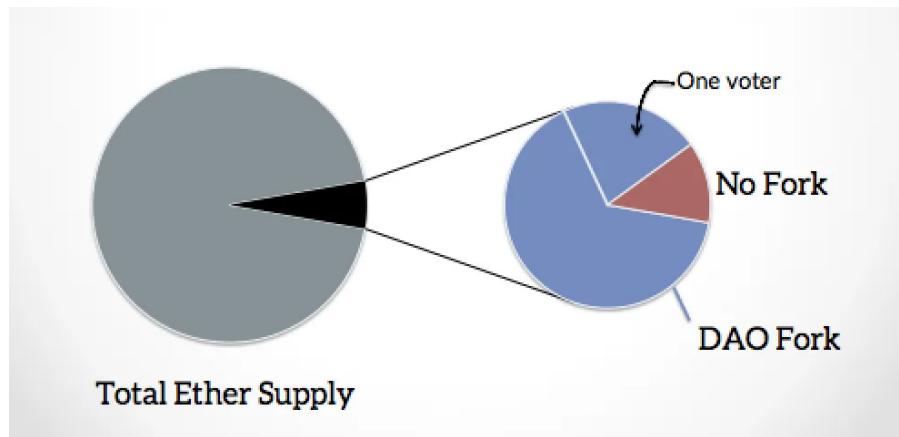
One very simple, and important, difference is that tightly coupled voting creates a default in favor of the blockchain adopting what the majority wants, requiring minorities to exert great effort to coordinate a hard fork to preserve a blockchain's existing properties, whereas loosely coupled voting is only a coordination tool, and still requires users to actually download and run the software that implements any given fork. But there are also many other differences. Now, let us go through some arguments *against* voting, and dissect how each argument applies to voting as layer 1 and voting as layer 2.

Low Voter Participation

One of the main criticisms of coin voting mechanisms so far is that, no matter where they are tried, they tend to have very low voter participation. The DAO Carbonvote only had a voter participation rate of 4.5%:



Additionally, wealth distribution is very unequal, and the results of these two factors together are best described by this image created by a critic of the DAO fork:



The EIP 186 Carbonvote had ~2.7 million ETH voting. The DAO proposal votes did not fare better (<http://themerkle.com/the-dao-undergoes-low-voting-turnout/>), with participation never reaching 10%. And outside of Ethereum things are not sunny either; even in Bitshares, a system where the core social contract is designed around voting, the top delegate in an approval vote only got 17% of the vote (<https://bitcointalk.org/index.php?topic=916696.330;imode>), and in Lisk it got up to 30% (<https://explorer.lisk.io/delegateMonitor>), though as we will discuss later these systems have other problems of their own.

Low voter participation means two things. First, the vote has a harder time achieving a perception of legitimacy, because it only reflects the views of a small percentage of people. Second, an attacker with only a small percentage of all coins can sway the vote. These problems exist regardless of whether the vote is tightly coupled or loosely coupled.

Game-Theoretic Attacks

Aside from "the big hack" that received the bulk of the media attention, the DAO also had a number of much smaller game-theoretic vulnerabilities; this article from HackingDistributed (<http://hackingdistributed.com/2016/05/27/dao-call-for-moratorium/>) does a good job of summarizing them. But this is only the tip of the iceberg. Even if all of the finer details of a voting mechanism are implemented correctly, voting mechanisms in general have a large flaw: in any vote, the probability that any given voter will have an impact on the result is tiny, and so the personal incentive that each voter has to vote correctly is almost insignificant. And if each person's size of the stake is small, their incentive to vote correctly is insignificant squared. Hence, a relatively small bribe spread out across the participants may suffice to sway their decision, possibly in a way that they collectively might quite disapprove of.

Now you might say, people are not evil selfish profit-maximizers that will accept a \$0.5 bribe to vote to give twenty million dollars to Josh arza just because the above calculation says their individual chance of affecting anything is tiny; rather, they would altruistically refuse to do something that evil. There are two responses to this criticism.

First, there are ways to make a "bribe" that are quite plausible; for example, an exchange can offer interest rates for deposits (or, even more ambiguously, use the exchange's own money to build a great interface and features), with the exchange operator using the large quantity of deposits to vote as they wish. Exchanges profit from chaos, so their incentives are clearly quite misaligned with users *and* coin holders.

Second, and more damningly, in practice it seems like people, at least in their capacity as crypto token holders, are profit maximizers, and seem to see nothing evil or selfish about taking a bribe or two. As "Exhibit A", we can look at the situation with Lisk, where the delegate pool seems to have been successfully captured by two major "political parties" that explicitly bribe coin holders to vote for them, and also require each member in the pool to vote for all the others.

Here's LiskElite, with 55 members (out of a total 101):

The screenshot shows the LiskElite website at <https://liskelite.com>. The top navigation bar includes links for Home, Voters, Pending Voters, History, Donations, and Members. There are also language and sign-in options. The main content area has a grey header with the title "Member Rules:" followed by a numbered list of rules. Below this is another section titled "Voter Rules:" with its own numbered list. At the bottom of the page, there is a copyright notice: "All rights reserved by Elite Group".

Member Rules:

1. Every member of Elite except the china delegate must share 25% of his/her forging LISK to his/her Voters every week;
2. Every member of Elite except the china delegate must donate 5% of forging LISK to the Elite Lisk fund used to support Lisk ecosystem;
3. Every member of Elite must vote for other members;
4. Elite membership registration is now closed and no new members are currently accepted.

Voter Rules:

1. For getting the rewards you must vote for all of Elite Group members;
2. Elite reward payouts will be done on a weekly basis and will be paid out to voter accounts automatically.

All rights reserved by Elite Group

Here's LiskGDT, with 33 members:

The screenshot shows the LiskGDT website at <https://pool.liskgdt.net>. The main heading is "How it works" with a purple circular icon. Below this is a large section titled "Rules". The page is divided into two main sections: "Pool" and "GDT Members".

Pool

Takes 10% for GDT Lisk development and returns 90% as rewards to voters.

GDT Members

Are excluded from payouts and return their rewards as a GDT Reward for Silver level and above.

Below these sections are icons for "90%", "BONUS", and "DONATIONS".

And as "Exhibit B" some voter bribes being paid out [in Ark \(<https://bitcointalk.org/index.php?topic=1835497.new>\):](https://bitcointalk.org/index.php?topic=1835497.new)

Latest Transactions

Id	Timestamp	Sender	Recipient	Smartbridge	Amount (ARK)	Fee (ARK)
380af...d7ab4	2017/04/17 12:20:41	bioly	AbxqF...jXJ6B	Payout from bioly delegate pool, thank you for support!	7.60466706	0.1
5795e...26029	2017/04/17 12:20:41	bioly	ARUNS...oLzvs	Payout from bioly delegate pool, thank you for support!	6.07691376	0.1
37694...35419	2017/04/17 12:20:40	bioly	AG2N1...taeZv	Payout from bioly delegate pool, thank you for support!	2.48455539	0.1
8c6b1...f1f9a	2017/04/17 12:20:39	bioly	AWmMj...HJU8R	Payout from bioly delegate pool, thank you for support!	118.47841646	0.1
d2ad5...c84af	2017/04/17 12:20:38	bioly	AbJ6N...ZrZxq	Payout from bioly delegate pool, thank you for support!	9.37653981	0.1
45280...aa3f0	2017/04/17 12:20:37	bioly	AevZb...68d6G	Payout from bioly delegate pool, thank you for support!	118.4945548	0.1
ace28...1cdee	2017/04/17 12:20:37	bioly	teletobi	Payout from bioly delegate pool, thank you for support!	11.72867675	0.1
20ca3...4278b	2017/04/17 12:20:36	bioly	ANY7W...6TfzX	Payout from bioly delegate pool, thank you for support!	4.80016674	0.1
a4de1...f90fd	2017/04/17 12:20:36	bioly	ARK8b...znv2Z	Payout from bioly delegate pool, thank you for support!	178.80073745	0.1
cb528...592bc	2017/04/17 12:20:36	bioly	AUmaL...QeHyP	Payout from bioly delegate pool, thank you for support!	237.32335576	0.1
29740...578db	2017/04/17 12:20:35	bioly	AUw4A...HxWB7	Payout from bioly delegate pool, thank you for support!	54.14948207	0.1
331df...5b0f2	2017/04/17 12:20:35	bioly	AQxnW...F2HGH	Payout from bioly delegate pool, thank you for support!	46.96456749	0.1
38fac...e02f5	2017/04/17 12:20:34	bioly	AKkvf...L5TW9	Payout from bioly delegate pool, thank you for support!	41.98709123	0.1
50190...b5284	2017/04/17 12:20:34	bioly	AWskK...bRB4m	Payout from bioly delegate pool, thank you for support!	7.39663982	0.1
72770...78a41	2017/04/17 12:20:34	bioly	AUTPB...E6pro	Payout from bioly delegate pool, thank you for support!	15.64031609	0.1
199f4...bae6a	2017/04/17 12:20:33	bioly	AVbiK...MEK8P	bioly fee account	403.66128558	0.1
af13f...6a16e	2017/04/17 12:20:33	bioly	AVVVY...gTiCo	Payout from bioly delegate pool, thank you for support!	7.63884129	0.1
74c3c...061f6	2017/04/17 12:20:32	bioly	AYTAy...3egy6	Payout from bioly delegate pool, thank you for support!	71.46381847	0.1

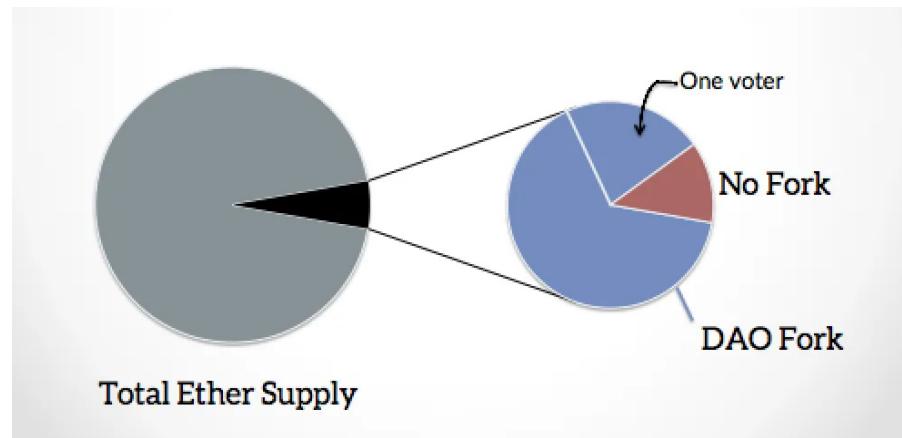
Here, note that there is a key difference between tightly coupled and loosely coupled votes. In a loosely coupled vote, direct or indirect vote bribing is also possible, but if the community agrees that some given proposal or set of votes constitutes a game-theoretic attack, they can simply socially agree to ignore it. And in fact this has kind of already happened - the Carbonvote contains a blacklist of addresses corresponding to known exchange addresses, and votes from these addresses are not counted. In a tightly coupled vote, there is no way to create such a blacklist at protocol level, because agreeing who is part of the blacklist is *itself* a blockchain governance decision. But since the blacklist is part of a community-created voting tool that only indirectly influences protocol changes, voting tools that contain bad blacklists can simply be rejected by the community.

It's worth noting that this section **is not** a prediction that all tightly coupled voting systems will quickly succumb to bribe attacks. It's entirely possible that many will survive for one simple reason: all of these projects have founders or foundations with large premines, and these act as large centralized actors that are interested in their platforms' success that are not vulnerable to bribes, and hold enough coins to outweigh most bribe attacks. However, this kind of centralized trust model, while arguably useful in some contexts in a project's early stages, is clearly one that is not sustainable in the long term.

Non-Representativeness

Another important objection to voting is that coin holders are only one class of user, and may have interests that collide with those of other users. In the case of pure cryptocurrencies like Bitcoin, store-of-value use ("[hodling \(<https://bitcointalk.org/index.php?topic=375643.0>\)](https://bitcointalk.org/index.php?topic=375643.0)") and medium-of-exchange use ("buying coffees") are naturally in conflict, as the store-of-value prizes security much more than the medium-of-exchange use case, which more strongly values usability. With Ethereum, the conflict is worse, as there are many people who use Ethereum for reasons that have nothing to do with ether (see: cryptokitties), or even value-bearing digital assets in general (see: ENS).

Additionally, even if coin holders *are* the only relevant class of user (one might imagine this to be the case in a cryptocurrency where there is an established social contract that its purpose is to be the next digital gold, and nothing else), there is still the challenge that a coin holder vote gives a much greater voice to wealthy coin holders than to everyone else, opening the door for centralization of holdings to lead to unencumbered centralization of decision making. Or, in other words...

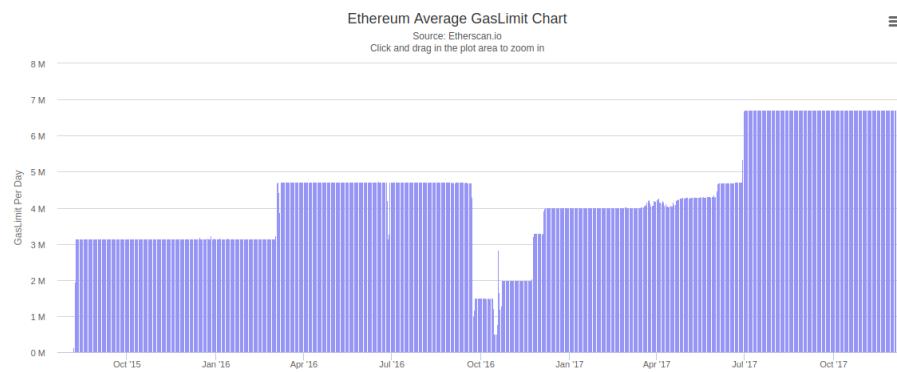


And if you want to see a review of a project that seems to combine all of these disadvantages at the same time, see this: <https://btcgeek.com/bitshares-trying-memorycoin-year-ago-disastrous-ends/> (<https://btcgeek.com/bitshares-trying-memorycoin-year-ago-disastrous-ends/>).

This criticism applies to both tightly coupled and loosely coupled voting equally; however, loosely coupled voting is more amenable to compromises that mitigate its unrepresentativeness, and we will discuss this more later.

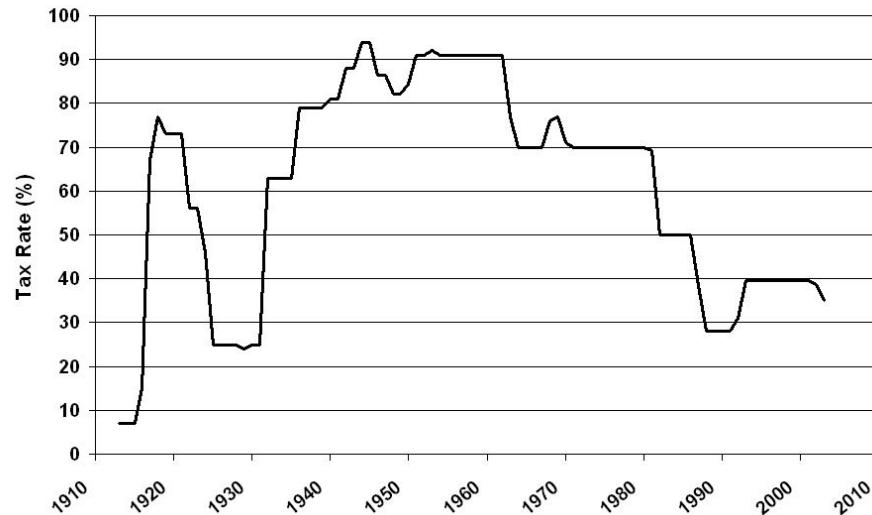
Centralization

Let's look at the existing live experiment that we have in tightly coupled voting on Ethereum, the gas limit. Here's the gas limit evolution over the past two years:



You might notice that the general feel of the curve is a bit like another chart that may be quite familiar to you:

Top Marginal Income Tax Rate: 1913-2003



Basically, they both look like magic numbers that are created and repeatedly renegotiated by a fairly centralized group of guys sitting together in a room. What's happening in the first case? Miners are generally following the direction favored by the community, which is itself gauged via social consensus aids similar to those that drive hard forks (core developer support, Reddit upvotes, etc; in Ethereum, the gas limit has never gotten controversial enough to require anything as serious as a coin vote).

Hence, it is not at all clear that voting will be able to deliver *results* that are actually decentralized, if voters are not technically knowledgeable and simply defer to a single dominant tribe of experts. This criticism once again applies to tightly coupled and loosely coupled voting equally.

Update: since writing this, it seems like Ethereum miners managed to up the gas limit from 6.7 million to 8 million all without even discussing it with the core developers or the Ethereum Foundation. So there is hope; but it takes a lot of hard community building and other grueling non-technical work to get to that point.

Digital Constitutions

One approach that has been suggested to mitigate the risk of runaway bad governance algorithms is "digital constitutions" that mathematically specify desired properties that the protocol should have, and require any new code changes to come with a computer-verifiable proof that they satisfy these properties. This seems like a good idea at first, but this too should, in my opinion, be viewed skeptically.

In general, the idea of having norms about protocol properties, and having these norms serve the function of one of the coordination flags, is a very good one. This allows us to enshrine core properties of a protocol that we consider to be very important and valuable, and make them more difficult to change. However, this is exactly the sort of thing that should be enforced in loosely coupled (ie. layer two), rather than tightly coupled (layer one) form.

Basically any meaningful norm is actually quite hard to express in its entirety; this is part of the complexity of value (https://wiki.lesswrong.com/wiki/Complexity_of_value), problem. This is true even for something as seemingly unambiguous as the 21 million coin limit. Sure, one can add a line of code saying `assert total_supply <= 21000000`, and put a comment around it saying "do not remove at all costs", but there are plenty of roundabout ways of doing the same thing. For example, one could imagine a soft fork that adds a mandatory transaction fee this is proportional to coin value * time since the coins were last sent, and this is equivalent to

demurrage, which is equivalent to deflation. One could also implement another currency, called Bjtcoin, with 21 million *new* units, and add a feature where if a bitcoin transaction is sent the miner can intercept it and claim the bitcoin, instead giving the recipient bjtcoin; this would rapidly force bitcoins and bjtcoins to be fungible with each other, increasing the "total supply" to 42 million without ever tripping up that line of code. "Softer" norms like not interfering with application state are even harder to enforce.

We want to be able to say that a protocol change that violates any of these guarantees should be viewed as illegitimate - there should be a coordination institution that waves a red flag - even if they get approved by a vote. We also want to be able to say that a protocol change that follows the letter of a norm, but blatantly violates its spirit, the protocol change should *still* be viewed as illegitimate. And having norms exist on layer 2 - in the minds of humans in the community, rather than in the code of the protocol - best achieves that goal.

Toward A Balance

However, I am also not willing to go the other way and say that coin voting, or other explicit on-chain voting-like schemes, have no place in governance whatsoever. The leading alternative seems to be core developer consensus, however the failure mode of a system being controlled by "ivory tower intellectuals" who care more about abstract philosophies and solutions that sound technically impressive over and above real day-to-day concerns like user experience and transaction fees is, in my view, also a real threat to be taken seriously.

So how do we solve this conundrum? Well, first, we can heed the words of slatestarcodex (<http://slatestarcodex.com/2017/11/21/contra-robinson-on-public-food/>) in the context of traditional politics:

The rookie mistake is: you see that some system is partly Moloch [ie. captured by misaligned special interests], so you say "Okay, we'll fix that by putting it under the control of this other system. And we'll control this other system by writing 'DO NOT BECOME MOLOCH' on it in bright red marker."

("I see capitalism sometimes gets misaligned. Let's fix it by putting it under control of the government. We'll control the government by having only virtuous people in high offices.") I'm not going to claim there's a great alternative, but the occasionally-adequate alternative is the neoliberal one – find a couple of elegant systems that all optimize along different criteria approximately aligned with human happiness, pit them off against each other in a structure of checks and balances, hope they screw up in different places like in that swiss cheese model, keep enough individual free choice around that people can exit any system that gets too terrible, and let cultural evolution do the rest.

In blockchain governance, it seems like this is the only way forward as well. The approach for blockchain governance that I advocate is "multifactorial consensus", where different coordination flags and different mechanisms and groups are polled, and the ultimate decision depends on the collective result of all of these mechanisms together. These coordination flags may include:

- The roadmap (ie. the set of ideas broadcasted earlier on in the project's history about the direction the project would be going)
- Consensus among the dominant core development teams
- Coin holder votes
- User votes, through some kind of sybil-resistant polling system
- Established norms (eg. non-interference with applications, the 21 million coin limit)

I would argue that it is very useful for coin voting to be one of several coordination institutions deciding whether or not a given change gets implemented. It is an imperfect and unrepresentative signal, but it is a *Sybil-resistant* one - if you see 10 million ETH voting for a given proposal, you *cannot* dismiss that by simply saying "oh, that's just hired Russian trolls with fake social media accounts". It is also a signal that is sufficiently disjoint from the core development team that if needed it can serve as a check on it. However, as described above, there are very good reasons why it should not be the *only* coordination institution.

And underpinning it all is the key difference from traditional systems that makes blockchains interesting: the "layer 1" that underpins the whole system is the requirement for individual users to assent to any protocol changes, and their freedom, and credible threat, to "fork off" if someone attempts to force changes on them that they consider hostile (see also: http://vitalik.ca/general/2017/05/08/coordination_problems.html (http://vitalik.ca/general/2017/05/08/coordination_problems.html)).

Tightly coupled voting is also okay to have in some limited contexts - for example, despite its flaws, miners' ability to vote on the gas limit is a feature that has proven very beneficial on multiple occasions. The risk that miners will try to abuse their power may well be lower than the risk that any specific gas limit or block size limit hard-coded by the protocol on day 1 will end up leading to serious problems, and in that case letting miners vote on the gas limit is a good thing. However, "allowing miners or validators to vote on a few specific parameters that need to be rapidly changed from time to time" is a very far cry from giving them arbitrary control over protocol rules, or letting voting control validation, and these more expansive visions of on-chain governance have a much murkier potential, both in theory and in practice.

Governance, Part 2: Plutocracy Is Still Bad

2018 Mar 28

[See all posts \(/\)](#)

Coin holder voting, both for governance of technical features, and for more extensive use cases like deciding who runs validator nodes and who receives money from development bounty funds, is unfortunately continuing to be popular, and so it seems worthwhile for me to write another post explaining why I (and [Vlad Zamfir](#) (https://medium.com/@Vlad_Zamfir/against-on-chain-governance-a4ceacd040ca)) and others) do not consider it wise for Ethereum (or really, any base-layer blockchain) to start adopting these kinds of mechanisms in a tightly coupled form in any significant way.

I wrote about the issues with tightly coupled voting [in a blog post](#) (<https://vitalik.ca/general/2017/12/17/voting.html>). last year, that focused on theoretical issues as well as focusing on some practical issues experienced by voting systems over the previous two years. Now, the latest scandal in DPOS land seems to be substantially worse. Because the delegate rewards in EOS are now so high (5% annual inflation, about \$400m per year), the competition on who gets to run nodes has essentially become yet another frontier of US-China geopolitical economic warfare.



And that's not my own interpretation; I quote from [this article \(original in Chinese\)](#) (<https://zhuanlan.zhihu.com/p/34902188>):

EOS supernode voting: multibillion-dollar profits leading to crypto community inter-country warfare

Looking at community recognition, Chinese nodes feel much less represented in the community than US and Korea. Since the EOS.IO official Twitter account was founded, there has never been any interaction with the mainland Chinese EOS community. For a listing of the EOS officially promoted events and interactions with communities see the picture below.



With no support from the developer community, facing competition from Korea, the Chinese EOS supernodes have invented a new strategy: buying votes.

The article then continues to describe further strategies, like forming "alliances" that all vote (or buy votes) for each other.

Of course, it does not matter at all who the specific actors are that are buying votes or forming cartels; this time it's some Chinese pools, [last time](https://liskgdt.net/) (<https://liskgdt.net/>). It was "members located in the USA, Russia, India, Germany, Canada, Italy, Portugal and many other countries from around the globe", next time it could be totally anonymous, or run out of a smartphone snuck into Brendon Shavers's prison cell. What matters is that blockchains and cryptocurrency, originally founded in a vision of using technology to escape from the failures of human politics, have essentially all but replicated it. Crypto is a reflection of the world at large.

The EOS New York community's response seems to be that they have issued a strongly worded letter to the world stating that [buying votes will be against the constitution](https://steemit.com/eos/@eosnewyork/block-one-confirms-vote-buying-will-be-against-eos-io-proposed-constitution) (<https://steemit.com/eos/@eosnewyork/block-one-confirms-vote-buying-will-be-against-eos-io-proposed-constitution>). Hmm, what other major political entity has [made accepting bribes a violation of the constitution](https://en.wikipedia.org/wiki/Emoluments_Clause) (https://en.wikipedia.org/wiki/Emoluments_Clause)? And how has that been going for them lately?

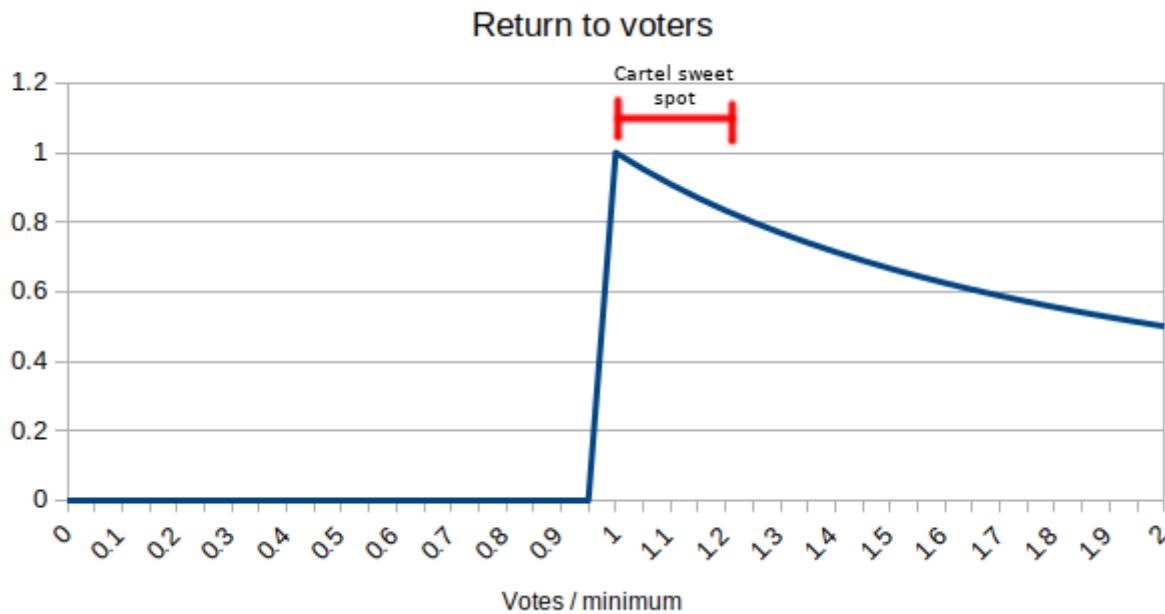
The second part of this article will involve me, an armchair economist, hopefully convincing you, the reader, that yes, bribery is, in fact, bad. There are actually people who dispute this claim; the usual argument has something to do with market efficiency, as in "isn't this good, because it means that the nodes that win will be the nodes that can be the cheapest, taking the least money for themselves and their expenses and giving the rest back to the community?" The answer is, kinda yes, but in a way that's centralizing and vulnerable to rent-seeking cartels and explicitly contradicts many of the explicit promises made by most DPOS proponents along the way.

Let us create a toy economic model as follows. There are a number of people all of which are running to be delegates. The delegate slot gives a reward of \$100 per period, and candidates promise to share some portion of that as a bribe, equally split among all of their voters. The actual N delegates (eg. N = 35) in any period are the N delegates that received the most votes; that is, during every period a threshold of votes emerges where if you get more votes than that threshold you are a delegate, if you get less you are not, and the threshold is set so that N delegates are above the threshold.

We expect that voters vote for the candidate that gives them the highest expected bribe. Suppose that all candidates start off by sharing 1%; that is, equally splitting \$1 among all of their voters. Then, if some candidate becomes a delegate with K voters, each voter gets a payment of $\frac{1}{K}$. The candidate that it's most profitable to vote for is a candidate that's expected to be in the top N , but is expected to earn the fewest votes within that set. Thus, we expect votes to be fairly evenly split among 35 delegates.

Now, some candidates will want to secure their position by sharing more; by sharing 2%, you are likely to get twice as many votes as those that share 1%, as that's the equilibrium point where voting for you has the same payout as voting for anyone else. The extra guarantee of being elected that this gives is definitely worth losing an additional 1% of your revenue when you do get elected. We can expect delegates to bid up their bribes and eventually share something close to 100% of their revenue. So the outcome seems to be that the delegate payouts are largely simply returned to voters, making the delegate payout mechanism close to meaningless.

But it gets worse. At this point, there's an incentive for delegates to form alliances (aka political parties, aka cartels) to coordinate their share percentages; this reduces losses to the cartel from chaotic competition that accidentally leads to some delegates not getting enough votes. Once a cartel is in place, it can start bringing its share percentages down, as dislodging it is a hard coordination problem: if a cartel offers 80%, then a new entrant offers 90%, then to a voter, seeking a share of that extra 10% is not worth the risk of either (i) voting for someone who gets insufficient votes and does not pay rewards, or (ii) voting for someone who gets too many votes and so pays out a reward that's excessively diluted.



Sidenote: [Bitshares DPOS](https://bitshares.org/technology/delegated-proof-of-stake-consensus/) (<https://bitshares.org/technology/delegated-proof-of-stake-consensus/>), used approval voting, where you can vote for as many candidates as you want; it should be pretty obvious that with even slight bribery, the equilibrium there is that everyone just votes for everyone.

Furthermore, even if cartel mechanics don't come into play, there is a further issue. This equilibrium of coin holders voting for whoever gives them the most bribes, or a cartel that has become an entrenched rent seeker, contradicts explicit promises made by DPOS proponents.

Quoting "[Explain Delegated Proof of Stake Like I'm 5](https://hackernoon.com/explain-delegated-proof-of-stake-like-im-5) (<https://hackernoon.com/explain-delegated-proof-of-stake-like-im-5-888b2a74897d>):"

If a Witness starts acting like an asshole, or stops doing a quality job securing the network, people in the community can remove their votes, essentially firing the bad actor. Voting is always ongoing.

From "[EOS: An Introduction](https://eos.io/documents/EOS_An_Introduction.pdf) (https://eos.io/documents/EOS_An_Introduction.pdf):"

By custom, we suggest that the bulk of the value be returned to the community for the common good - software improvements, dispute resolution, and the like can be entertained. In the spirit of "eating our own dogfood," the design envisages that the community votes on a set of open entry contracts that act like "foundations" for the benefit of the community. Known as Community Benefit Contracts, the mechanism highlights the importance of DPOS as enabling direct on-chain governance by the community (below).

The flaw in all of this, of course, is that the average voter has only a very small chance of impacting which delegates get selected, and so they only have a very small incentive to vote based on any of these high-minded and lofty goals; rather, their incentive is to vote for whoever offers the highest and most reliable bribe. Attacking is easy. If a cartel equilibrium does not form, then an attacker can simply offer a share percentage slightly higher than 100% (perhaps using fee sharing or some kind of "starter promotion" as justification), capture the majority of delegate positions, and then start an attack. If they get removed from the delegate position via a hard fork, they can simply restart the attack again with a different identity.

The above is not intended purely as a criticism of DPOS consensus or its use in any specific blockchain. Rather, the critique reaches much further. There has been a large number of projects recently that extol the virtues of extensive on-chain governance, where on-chain coin holder voting can be used not just to vote on protocol features, but also to control a bounty fund. Quoting a [blog post from last year](https://medium.com/@FEhrsam/blockchain-governance-programming-our-future-c3bfe30f2d74) (<https://medium.com/@FEhrsam/blockchain-governance-programming-our-future-c3bfe30f2d74>):

Anyone can submit a change to the governance structure in the form of a code update. An on-chain vote occurs, and if passed, the update makes its way on to a test network. After a period of time on the test network, a confirmation vote occurs, at which point the change goes live on the main network. They call this concept a "self-amending ledger". Such a system is interesting because it shifts power towards users and away from the more centralized group of developers and miners. On the developer side, anyone can submit a change, and most importantly, everyone has an economic incentive to do it. Contributions are rewarded by the community with newly minted tokens through inflation funding. This shifts from the current Bitcoin and Ethereum dynamics where a new developer has little incentive to evolve the protocol, thus power tends to concentrate amongst the existing developers, to one where everyone has equal earning power.

In practice, of course, what this can easily lead to is funds that offer kickbacks to users who vote for them, leading to the exact scenario that we saw above with DPOS delegates. In the best case, the funds will simply be returned to voters, giving coin holders an interest rate that cancels out the inflation, and in the worst case,

some portion of the inflation will get captured as economic rent by a cartel.

Note also that the above is not a criticism of *all* on-chain voting; it does not rule out systems like futarchy. However, futarchy is untested, but coin voting *is* tested, and so far it seems to lead to a high risk of economic or political failure of some kind - far too high a risk for a platform that seeks to be an economic base layer for development of decentralized applications and institutions.

So what's the alternative? The answer is what we've been saying all along: *cryptoeconomics*.

Cryptoeconomics (<https://www.coindesk.com/making-sense-cryptoeconomics/>) is fundamentally about the use of economic incentives together with cryptography to design and secure different kinds of systems and applications, including consensus protocols. The goal is simple: to be able to measure the security of a system (that is, the cost of breaking the system or causing it to violate certain guarantees) in dollars. Traditionally, the security of systems often depends on *social* trust assumptions: the system works if 2 of 3 of Alice, Bob and Charlie are honest, and we trust Alice, Bob and Charlie to be honest because I know Alice and she's a nice girl, Bob registered with FINCEN and has a money transmitter license, and Charlie has run a successful business for three years and wears a suit.

Social trust assumptions can work well in many contexts, but they are difficult to universalize; what is trusted in one country or one company or one political tribe may not be trusted in others. They are also difficult to quantify; how much money does it take to manipulate social media to favor some particular delegate in a vote? Social trust assumptions seem secure and controllable, in the sense that "people" are in charge, but in reality they can be manipulated by economic incentives in all sorts of ways.

Cryptoeconomics is about trying to reduce social trust assumptions by creating systems where we introduce explicit economic incentives for good behavior and economic penalties for bad behavior, and making mathematical proofs of the form "in order for guarantee X to be violated, at least these people need to misbehave in this way, which means the minimum amount of penalties or foregone revenue that the participants suffer is Y". Casper (<http://arxiv.org/abs/1710.09437>) is (<https://github.com/ethereum/cbc-casper/wiki>), designed (<https://medium.com/@jonchoi/ethereum-casper-101-7a851a4f1eb0>) to accomplish precisely this objective in the context of proof of stake consensus. Yes, this does mean that you can't create a "blockchain" by concentrating the consensus validation into 20 uber-powerful "supernodes" and you have to actually think (<https://medium.com/@icebearhww/ethereum-sharding-workshop-in-taipei-a44c0db8b8d9>) to make a design that intelligently breaks through and navigates existing tradeoffs and achieves massive scalability in a still-decentralized network. But the reward is that you don't get a network that's constantly liable to breaking in half or becoming economically captured by unpredictable political forces.

1. It has been brought to my attention that EOS may be reducing its delegate rewards from 5% per year to 1% per year.

Needless to say, this doesn't really change the fundamental validity of any of the arguments; the only result of this would be 5x less rent extraction potential at the cost of a 5x reduction to the cost of attacking the system.

*2. Some have asked: but how can it be wrong for DPOS delegates to bribe voters, when it is perfectly legitimate for mining and stake pools to give 99% of their revenues back to their participants? The answer should be clear: in PoW and PoS, it's the protocol's role to determine the rewards that miners and validators get, based on the miners and validators' observed performance, and the fact that miners and validators that are pools pass along the rewards (and penalties!) to their participants gives the participants an incentive to participate in good pools. In DPOS, the reward is constant, and it's the voters' role to vote for pools that have good performance, but with the key flaw that there is no mechanism to actually encourage voters to vote in that way instead of just voting for whoever gives them the most money without taking performance into account. Penalties in DPOS do not exist, and are certainly not passed on to voters, so voters have no "skin in the game" (penalties in Casper pools, on the other hand, **do** get passed on to participants).*

On Radical Markets

2018 Apr 20

[See all posts \(/\)](#)

Recently I had the fortune to have received an advance copy of Eric Posner and Glen Weyl's new book, *Radical Markets* (<https://www.amazon.ca/dp/B0773X7RKB/ref=dp-kindle-redirect?encoding=UTF8&btkr=1>), which could be best described as an interesting new way of looking at the subject that is sometimes called "political economy" (https://en.wikipedia.org/wiki/Political_economy)" - tackling the big questions of how markets and politics and society intersect. The general philosophy of the book, as I interpret it, can be expressed as follows. Markets are great, and price mechanisms are an awesome way of guiding the use of resources in society and bringing together many participants' objectives and information into a coherent whole. However, markets are socially constructed because they depend on property rights that are socially constructed, and there are many different ways that markets and property rights can be constructed, some of which are unexplored and potentially far better than what we have today. Contra doctrinaire libertarians, freedom is a high-dimensional design space.

The book interests me for multiple reasons. First, although I spend most of my time in the blockchain/crypto space heading up the Ethereum project and in some cases providing various kinds of support to projects in the space, I do also have broader interests, of which the use of economics and mechanism design to make more open, free, egalitarian and efficient systems for human cooperation, including improving or replacing present-day corporations and governments, is a major one. The intersection of interests between the Ethereum community and Posner and Weyl's work is multifaceted and plentiful; *Radical Markets* dedicates an entire chapter to the idea of "markets for personal data", redefining the economic relationship between ourselves and services like Facebook, and well, look what the Ethereum community is working on: [markets](https://cointelegraph.com/news/blockchain-startup-can-help-consumers-profit-from-their-personal-data) (<https://cointelegraph.com/news/blockchain-startup-can-help-consumers-profit-from-their-personal-data>) for [personal](https://cointelegraph.com/news/marketplace-aims-to-resell-personal-data-and-create-passive-income-stream-for-users) (<https://cointelegraph.com/news/marketplace-aims-to-resell-personal-data-and-create-passive-income-stream-for-users>). [personal](https://datum.org/) (<https://datum.org/>). [data](https://blog.enigma.co/the-enigma-data-marketplace-is-live-84a269ec17fb) (<https://blog.enigma.co/the-enigma-data-marketplace-is-live-84a269ec17fb>).

Second, blockchains may well be used as a technical backbone for some of the solutions described in the book, and Ethereum-style smart contracts are ideal for the kinds of complex systems of property rights that the book explores. Third, the economic ideas and challenges that the book brings up are ideas that have also been explored, and will be continue to be explored, at great length by the blockchain community for its own purposes. Posner and Weyl's ideas often have the feature that they allow economic incentive alignment to serve as a substitute for subjective ad-hoc bureaucracy (eg. Harberger taxes can essentially replace eminent domain (https://en.wikipedia.org/wiki/Eminent_domain)), and given that blockchains lack access to trusted human-controlled courts, these kinds of solutions may prove to be even more ideal for blockchain-based markets than they are for "real life".

I will warn that readers are not at all guaranteed to find the book's proposals acceptable; at least the first three have [already been](https://www.politico.com/magazine/story/2018/02/13/immigration-visas-economics-216968) (<https://www.politico.com/magazine/story/2018/02/13/immigration-visas-economics-216968>), highly controversial and they do contravene many people's moral preconceptions about how property should and should work and where money and markets can and can't be used. The authors are no strangers to controversy; Posner has on previous occasions even [proven willing](https://www.theguardian.com/news/2014/dec/04/-sp-case-against-human-rights) (<https://www.theguardian.com/news/2014/dec/04/-sp-case-against-human-rights>) to argue against such notions as

human rights law. That said, the book does go to considerable lengths to explain why each proposal improves efficiency if it could be done, and offer multiple versions of each proposal in the hopes that there is at least one (even if partial) implementation of each idea that any given reader can find agreeable.

What do Posner and Weyl talk about?

The book is split into five major sections, each arguing for a particular reform: self-assessed property taxes, quadratic voting, a new kind of immigration program, breaking up big financial conglomerates that currently make banks and other industries act like monopolies even if they appear at first glance to be competitive, and markets for selling personal data. Properly summarizing all five sections and doing them justice would take too long, so I will focus on a deep summary of one specific section, dealing with a new kind of property taxation, to give the reader a feel for the kinds of ideas that the book is about.

Harberger taxes

See also: "[Property Is Only Another Name for Monopoly](https://chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=1266&context=journal_articles) (https://chicagounbound.uchicago.edu/cgi/viewcontent.cgi?article=1266&context=journal_articles)," Posner and Weyl

Markets and private property are two ideas that are often considered together, and it is difficult in modern discourse to imagine one without (or even with much less of) the other. In the 19th century, however, many economists in Europe were both libertarian *and* egalitarian, and it was quite common to appreciate markets while maintaining skepticism toward the excesses of private property. A rather interesting example of this is the Bastiat-Proudhon debate (<http://praxeology.net/FB-PJP-DOI.htm>) from 1849-1850 where the two dispute the legitimacy of charging interest on loans, with one side focusing on the mutual gains from voluntary contracts and the other focusing on their suspicion of the potential for people with capital to get even richer without working, leading to unbalanced capital accumulation.

As it turns out, it is absolutely possible to have a system that contains markets but not property rights: at the end of every year, collect every piece of property, and at the start of the next year have the government auction every piece out to the highest bidder. This kind of system is intuitively quite unrealistic and impractical, but it has the benefit that it achieves perfect **allocative efficiency**: every year, every object goes to the person who can derive the most value from it (ie. the highest bidder). It also gives the government a large amount of revenue that could be used to completely substitute income and sales taxes or fund a basic income.

Now you might ask: doesn't the existing property system also achieve allocative efficiency? After all, if I have an apple, and I value it at \$2, and you value it at \$3, then you could offer me \$2.50 and I would accept. However, this fails to take into account imperfect information: how do you know that I value it at \$2, and not \$2.70? You could offer to buy it for \$2.99 so that you can be sure that you'll get it if you really are the one who values the apple more, but then you would be gaining practically nothing from the transaction. And if you ask me to set the price, how do I know that you value it at \$3, and not \$2.30? And if I set the price to \$2.01 to be sure, I would be gaining practically nothing from the transaction. Unfortunately, there is a result known as the Myerson-Satterthwaite Theorem (https://en.wikipedia.org/wiki/Myerson%20%93Satterthwaite_theorem) which means that *no* solution is efficient; that is, any bargaining algorithm in such a situation must at least sometimes lead to inefficiency from mutually beneficial deals falling through.

If there are many buyers you have to negotiate with, things get even harder. If a developer (in the real estate sense) is trying to make a large project that requires buying 100 existing properties, and 99 have already agreed, the remaining one has a strong incentive to charge a very high price, much higher than their actual

personal valuation of the property, hoping that the developer will have no choice but to pay up.



Well, not necessarily no choice. But a very inconvenient and both privately and socially wasteful choice.

Re-auctioning everything once a year completely solves this problem of allocative efficiency, but at a very high cost to **investment efficiency**: there's no point in building a house in the first place if six months later it will get taken away from you and re-sold in an auction. All property taxes have this problem; if building a house costs you \$90 and brings you \$100 of benefit, but then you have to pay \$15 more property tax if you build the house, then you will not build the house and that \$10 gain is lost to society.

One of the more interesting ideas from the 19th century economists, and specifically Henry George, was a kind of property tax that did not have this problem: the [land value tax](https://en.wikipedia.org/wiki/Land_value_tax) (https://en.wikipedia.org/wiki/Land_value_tax). The idea is to charge tax on the value of land, but not the *improvements to the land*; if you own a \$100,000 plot of dirt you would have to pay \$5,000 per year taxes on it regardless of whether you used the land to build a condominium or simply as a place to walk your pet doge.



A doge.

Weyl and Posner are not convinced that Georgian land taxes are viable in practice:

Consider, for example, the Empire State Building. What is the pure value of the land beneath it? One could try to infer its value by comparing it to the value of adjoining land. But the building itself defines the neighborhood around it; removing the building would almost certainly change the value of its surrounding land. The land and the building, even the neighborhood, are so tied together, it would be hard to figure out a separate value for each of them.

Arguably this does not exclude the possibility of a different kind of Georgian-style land tax: a tax based on the average of property values across a sufficiently large area. That would preserve the property that improving a single piece of land would not (greatly) perversely increase the taxes that they have to pay, without having to find a way to distinguish land from improvements in an absolute sense. But in any case, Posner and Weyl move on to their main proposal: self-assessed property taxes.

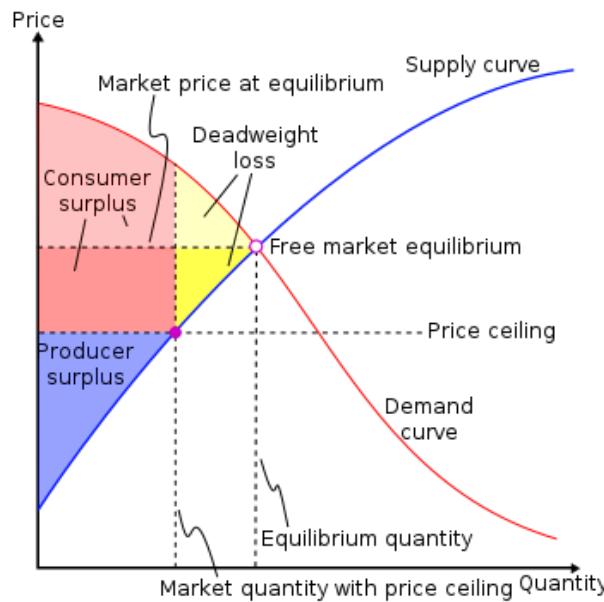
Consider a system where property owners themselves specify what the value of their property is, and pay a tax rate of, say, 2% of that value per year. But here is the twist: whatever value they specify for their property, *they have to be willing to sell it to anyone at that price.*

If the tax rate is equal to the chance per year that the property gets sold, then this achieves optimal allocative efficiency: raising your self-assessed property value by \$1 increases the tax you pay by \$0.02, but it also means there is a 2% chance that someone will buy the property and pay \$1 more, so there is no incentive to cheat in either direction. It does harm investment efficiency, but vastly less so than all property being re-auctioned every year.

Posner and Weyl then point out that if more investment efficiency is desired, a hybrid solution with a lower property tax is possible:

When the tax is reduced incrementally to improve investment efficiency, the loss in allocative efficiency is less than the gain in investment efficiency. The reason is that the most valuable sales are ones where the buyer is willing to pay significantly more than the seller is willing to accept. These transactions are the first ones enabled by a reduction in the price as even a small price reduction will avoid blocking these most valuable transactions. In fact, it can be shown that the size of the social loss from monopoly power grows quadratically in the extent of this power. Thus, reducing the markup by a third eliminates close to $\frac{5}{9} = (3^2 - 2^2)/(3^2)$ of the allocative harm from private ownership.

This concept of quadratic deadweight loss is a truly important insight in economics, and is arguably the deep reason why "moderation in all things" is such an attractive principle: the first step you take away from an extreme will generally be the most valuable.



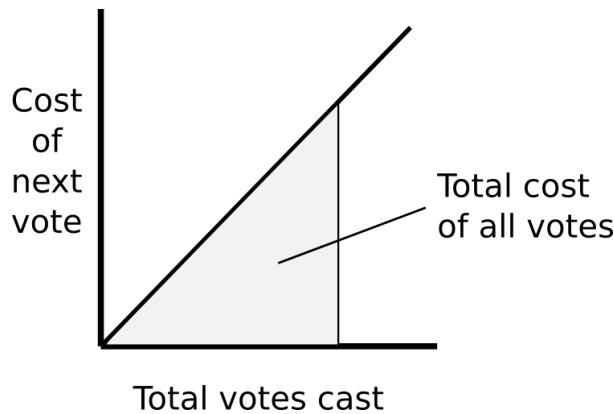
The book then proceeds to give a series of side benefits that this tax would have, as well as some downsides. One interesting side benefit is that it removes an information asymmetry flaw that exists with property sales today, where owners have the incentive to expend effort on making their property look good even in potentially misleading ways. With a properly set Harberger tax, if you somehow manage to trick the world into thinking your house is 5% more valuable, you'll get 5% more when you sell it but until that point you'll have to pay 5% more in taxes, or else someone will much more quickly snap it up from you at the original price.

The downsides are smaller than they seem; for example, one natural disadvantage is that it exposes property owners to uncertainty due to the possibility that someone will snap up their property at any time, but that is hardly an unknown as it's a risk that renters already face every day. But Weyl and Posner do propose more moderate ways of introducing the tax that don't have these issues. First, the tax can be applied to types of property that are currently government owned; it's a potentially superior alternative to both continued government ownership and traditional full-on privatization. Second, the tax can be applied to forms of property that are already "industrial" in usage: radio spectrum licenses, domain names, intellectual property, etc.

The Rest of the Book

The remaining chapters bring up similar ideas that are similar in spirit to the discussion on Harberger taxes in their use of modern game-theoretic principles to make mathematically optimized versions of existing social institutions. One of the proposals is for something called quadratic voting, which I summarize as follows.

Suppose that you can vote as many times as you want, but voting costs "voting tokens" (say each citizen is assigned N voting tokens per year), and it costs tokens in a nonlinear way: your first vote costs one token, your second vote costs two tokens, and so forth. If someone feels more strongly about something, the argument goes, they would be willing to pay more for a single vote; quadratic voting takes advantage of this by perfectly aligning *quantity* of votes with *cost* of votes: if you're willing to pay up to 15 tokens for a vote, then you will keep buying votes until your last one costs 15 tokens, and so you will cast 15 votes in total. If you're willing to pay up to 30 tokens for a vote, then you will keep buying votes until you can't buy any more for a price less than or equal to 30 tokens, and so you will end up casting 30 votes. The voting is "quadratic" because the total amount you pay for N votes goes up proportionately to N^2 .



After this, the book describes a market for immigration visas that could greatly expand the number of immigrants admitted while making sure local residents benefit and at the same time aligning incentives to encourage visa sponsors to choose immigrants that are more likely to succeed in the country and less likely to commit crimes, then an enhancement to antitrust law, and finally the idea of setting up markets for personal data.

Markets in Everything

There are plenty of ways that one could respond to each individual proposal made in the book. I personally, for example, find the immigration visa scheme that Posner and Weyl propose well-intentioned and see how it could improve on the status quo, but also overcomplicated, and it seems simpler to me to have a scheme where visas are auctioned or sold every year, with an additional requirement for migrants to obtain liability insurance. Robin Hanson recently proposed (<https://www.overcomingbias.com/2018/01/privately-enforced-punished-crime.html>) greatly expanding liability insurance mandates as an alternative to many kinds of regulation, and while imposing new mandates on an entire society seems unrealistic, a new expanded immigration program seems like the perfect place to start considering them. Paying people for personal data is interesting, but there are concerns about adverse selection: to put it politely, the kinds of people that are willing to sit around submitting lots of data to Facebook all year to earn \$16.92 (Facebook's current [annualized revenue per user](https://www.cnbc.com/2017/05/03/facebook-average-revenue-per-user-arpu-q1-2017.html) (<https://www.cnbc.com/2017/05/03/facebook-average-revenue-per-user-arpu-q1-2017.html>)) are *not* the kinds of people that advertisers are willing to burn hundreds of dollars per person trying to market rolexes and Lambos to. However, what I find more interesting is the general principle that the book tries to promote.

Over the last hundred years, there truly has been a large amount of research into designing economic mechanisms that have desirable properties and that outperform simple two-sided buy-and-sell markets. Some of this research has been put into use in some specific industries; for example, combinatorial auctions (https://en.wikipedia.org/wiki/Combinatorial_auction) are used in airports, radio spectrum auctions and several other industrial use cases, but it hasn't really seeped into any kind of broader policy design; the political systems and property rights that we have are still largely the same as we had two centuries ago. So can we use modern economic insights to reform base-layer markets and politics in such a deep way, and if so, should we?

Normally, I love markets and clean incentive alignment, and dislike politics and bureaucrats and ugly hacks, and I love economics, and I so love the idea of using economic insights to design markets that work better so that we can reduce the role of politics and bureaucrats and ugly hacks in society. Hence, naturally, I love this vision. So let me be a good intellectual citizen and do my best to try to make a case against it.

There is a limit to how complex economic incentive structures and markets can be because there is a limit to users' ability to think and re-evaluate and give ongoing precise measurements for their valuations of things, and people value reliability and certainty. Quoting Steve Waldman criticizing Uber surge pricing (<http://www.interfluidity.com/v2/5822.html>):

Finally, we need to consider questions of economic calculation. In macroeconomics, we sometimes face tradeoffs between an increasing and unpredictably variable price-level and full employment. Wisely or not, our current policy is to stabilize the price level, even at short-term cost to output and employment, because stable prices enable longer-term economic calculation. That vague good, not visible on a supply/demand diagram, is deemed worth very large sacrifices. The same concern exists in a microeconomic context. If the "ride-sharing revolution" really takes hold, a lot of us will have decisions to make about whether to own a car or rely upon the Sidecars, Lyfts, and Ubers of the world to take us to work every day. To make those calculations, we will need something like predictable pricing. Commuting to our minimum wage jobs (average is over!) by Uber may be OK at standard pricing, but not so OK on a surge. In the desperate utopia of the "free-market economist", there is always a solution to this problem. We can define futures markets on Uber trips, and so hedge our exposure to price volatility! In practice that is not so likely...

And:

It's clear that in a lot of contexts, people have a strong preference for price-predictability over immediate access. The vast majority of services that we purchase and consume are not price-rationed in any fine-grained way. If your hairdresser or auto mechanic is busy, you get penciled in for next week...

Strong property rights are valuable for the same reason: beyond the arguments about allocative and investment efficiency, they provide the mental convenience and planning benefits of predictability.

It's worth noting that even Uber itself doesn't do surge pricing in the "market-based" way that economists would recommend. Uber is not a market where drivers can set their own prices, riders can see what prices are available, and themselves choose their tradeoff between price and waiting time. Why does Uber not do this? One argument is that, as Steve Waldman says, "Uber itself is a cartel", and wants to have the power to adjust market prices not just for efficiency but also reasons such as profit maximization, strategically setting prices to drive out competing platforms (and taxis and public transit), and public relations. As Waldman further points

out, one Uber competitor, Sidecar, does have the ability for [drivers to set prices](https://www.side.cr/drivers/) (<https://www.side.cr/drivers/>), and I would add that I have seen ride-sharing apps in China where passengers can offer drivers higher prices to try to coax them to get a car faster.

A possible counter-argument that Uber might give is that drivers themselves are actually less good at setting optimal prices than Uber's own algorithms, and in general people value the convenience of one-click interfaces over the mental complexity of thinking about prices. If we assume that Uber won its market dominance over competitors like Sidecar fairly, then the market itself has decided that the economic gain from marketizing more things is not worth the mental transaction costs.

Harberger taxes, at least to me, seem like they would lead to these exact kinds of issues multiplied by ten; people are not experts at property valuation, and would have to spend a significant amount of time and mental effort figuring out what self-assessed value to put for their house, and they would complain much more if they accidentally put a value that's too low and suddenly find that their house is gone. If Harberger taxes were to be applied to smaller property items as well, people would need to juggle a large amount of mental valuations of everything. A similar critique could apply to many kinds of personal data markets, and possibly even to quadratic voting if implemented in its full form.

I could challenge this by saying "ah, even if that's true, this is the 21st century, we could have companies that build AIs that make pricing decisions on your behalf, and people could choose the AI that seems to work best; there could even be a public option"; and Posner and Weyl themselves suggest that this is likely the way to go. And this is where the interesting conversation starts.

Tales from Crypto Land

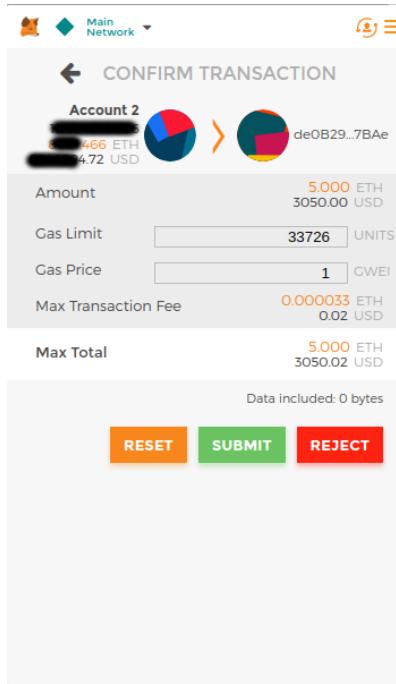
One reason why this discussion particularly interests me is that the cryptocurrency and blockchain space itself has, in some cases, run up against similar challenges. In the case of Harberger taxes, we actually did consider almost exactly that same proposal in the context of the [Ethereum Name System](https://ens.domains/) (<https://ens.domains/>) (our decentralized alternative to DNS), but the proposal was ultimately rejected. I asked the ENS developers why it was rejected. Paraphrasing their reply, the challenge is as follows.

Many ENS domain names are of a type that would only be interesting to precisely two classes of actors: (i) the "legitimate owner" of some given name, and (ii) scammers. Furthermore, in some particular cases, the legitimate owner is uniquely underfunded, and scammers are uniquely dangerous. One particular case is [MyEtherWallet](http://myetherwallet.com) (<http://myetherwallet.com>), an Ethereum wallet provider. MyEtherWallet provides an important public good to the Ethereum ecosystem, making Ethereum easier to use for many thousands of people, but is able to capture only a very small portion of the value that it provides; as a result, the budget that it has for outbidding others for the domain name is low. If a scammer gets their hands on the domain, users trusting MyEtherWallet could easily be tricked into sending all of their ether (or other Ethereum assets) to a scammer. Hence, because there is generally one clear "legitimate owner" for any domain name, a pure property rights regime presents little allocative efficiency loss, and there is a strong overriding public interest toward stability of reference (ie. a domain that's legitimate one day doesn't redirect to a scam the next day), so any level of Harberger taxation may well bring more harm than good.

I suggested to the ENS developers the idea of applying Harberger taxes to short domains (eg. abc.eth), but not long ones; the reply was that it would be too complicated to have two classes of names. That said, perhaps there is some version of the proposal that could satisfy the specific constraints here; I would be interested to hear Posner and Weyl's feedback on this particular application.

Another story from the blockchain and Ethereum space that has a more pro-radical-market conclusion is that of transaction fees. The notion of [mental transaction costs](http://nakamotoinstitute.org/literature/micropayments-and-mental-transaction-costs/) (<http://nakamotoinstitute.org/literature/micropayments-and-mental-transaction-costs/>), the idea that the inconvenience of even thinking about whether or not some small payment for a given digital good is worth it is enough of a burden to prevent "micro-markets" from working, is often used as an argument for why mass adoption of blockchain tech would be difficult: every transaction requires a small fee, and the mental expenditure of figuring out what fee to pay is itself a major usability barrier. These arguments increased further at the end of last year, when both [Bitcoin](https://bitinfocharts.com/comparison/bitcoin-transactionfees.html) (<https://bitinfocharts.com/comparison/bitcoin-transactionfees.html>) and [Ethereum](https://bitinfocharts.com/comparison/ethereum-transactionfees.html) (<https://bitinfocharts.com/comparison/ethereum-transactionfees.html>) transaction fees briefly spiked up by a factor of over 100 due to high usage (talk about surge pricing!), and those who accidentally did not pay high enough fees saw their transactions get stuck for days.

That said, this is a problem that we have now, arguably, to a large extent overcome. After the spikes at the end of last year, Ethereum wallets developed more advanced algorithms for choosing what transaction fees to pay to ensure that one's transaction gets included in the chain, and today most users are happy to simply defer to them. In my own personal experience, the mental transaction costs of worrying about transaction fees do not really exist, much like a driver of a car does not worry about the gasoline consumed by every single turn, acceleration and braking made by their car.



Personal price-setting AIs for interacting with open markets: already a reality in the Ethereum transaction fee market

A third kind of "radical market" that we are considering implementing in the context of Ethereum's consensus system is one for incentivizing deconcentration of validator nodes in [proof of stake consensus](https://medium.com/@jonchoi/ethereum-casper-101-7a851a4f1eb0) (<https://medium.com/@jonchoi/ethereum-casper-101-7a851a4f1eb0>). It's important for blockchains to be decentralized, a similar challenge to what antitrust law tries to solve, but the tools at our disposal are different. Posner and Weyl's solution to antitrust, banning institutional investment funds from owning shares in multiple competitors in the same industry, is far too subjective and human-judgement-dependent to work in a blockchain, but for our specific context we have a different solution: if a validator node commits an error, it gets penalized an amount proportional to the number of other nodes that have committed an error around the same time. This incentivizes nodes to set themselves up in such a way that their failure rate is maximally

uncorrelated with everyone else's failure rate, reducing the chance that many nodes fail at the same time and threaten to the blockchain's integrity. I want to ask Posner and Weyl: though our exact approach is fairly application-specific, could a similarly elegant "market-based" solution be discovered to incentivize market deconcentration in general?

All in all, I am optimistic that the various behavioral kinks around implementing "radical markets" in practice could be worked out with the help of good defaults and personal AIs, though I do think that if this vision is to be pushed forward, the greatest challenge will be finding progressively larger and more meaningful places to test it out and show that the model works. I particularly welcome the use of the blockchain and crypto space as a testing ground.

Another Kind of Radical Market

The book as a whole tends to focus on centralized reforms that could be implemented on an economy from the top down, even if their intended long-term effect is to push more decision-making power to individuals. The proposals involve large-scale restructurings of how property rights work, how voting works, how immigration and antitrust law works, and how individuals see their relationship with property, money, prices and society. But there is also the potential to use economics and game theory to come up with *decentralized* economic institutions that could be adopted by smaller groups of people at a time.

Perhaps the most famous examples of decentralized institutions from game theory and economics land are (i) assurance contracts, and (ii) prediction markets. An assurance contract is a system where some public good is funded by giving anyone the opportunity to pledge money, and only collecting the pledges if the total amount pledged exceeds some threshold. This ensures that people can donate money knowing that either they will get their money back or there actually will be enough to achieve some objective. A possible extension of this concept is Alex Tabarrok's [dominant assurance contracts](#)

(https://en.wikipedia.org/wiki/Assurance_contract#Dominant_assurance_contracts), where an entrepreneur offers to refund participants *more* than 100% of their deposits if a given assurance contract does not raise enough money.

Prediction markets allow people to bet on the probability that events will happen, potentially even conditional on some action being taken ("I bet \$20 that unemployment will go down if candidate X wins the election"); there are techniques for people interested in the information to subsidize the markets. Any attempt to manipulate the probability that a prediction market shows simply creates an opportunity for people to earn free money (yes I know, risk aversion and capital efficiency etc etc; still close to free) by betting against the manipulator.

Posner and Weyl do give one example of what I would call a decentralized institution: a game for choosing who gets an asset in the event of a divorce or a company splitting in half, where both sides provide their own valuation, the person with the higher valuation gets the item, but they must then give an amount equal to half the average of the two valuations to the loser. There's some economic reasoning by which this solution, while not perfect, is still close to mathematically optimal.

One particular category of decentralized institutions I've been interested in is improving incentivization for content posting and content curation in social media. Some ideas that I have had include:

- [Proof of stake conditional hashcash](https://ethresear.ch/t/conditional-proof-of-stake-hashcash/1301) (<https://ethresear.ch/t/conditional-proof-of-stake-hashcash/1301>). (when you send someone an email, you give them the opportunity to burn \$0.5 of your money if they think it's spam)
- [Prediction markets for content curation](https://ethresear.ch/t/prediction-markets-for-content-curation-daos/1312) (<https://ethresear.ch/t/prediction-markets-for-content-curation-daos/1312>). (use prediction markets to predict the results of a moderation vote on content, thereby encouraging a

market of fast content pre-moderators while penalizing manipulative pre-moderation)

- Conditional payments for paywalled content (after you pay for a piece of downloadable content and view it, you can decide after the fact if payments should go to the author or to proportionately refund previous readers)

And ideas I have had in other contexts:

- Call-out assurance contracts (<https://ethresear.ch/t/call-out-assurance-contracts/466>).
- DAICOs (<https://ethresear.ch/t/explanation-of-daicos/465>). (a more decentralized and safer alternative to ICOs)

Vitalik Buterin @HagenAnsel Feb 17
Replies to @VitalikButerin
Hey, guys. I want to give away another 1500 ETH to my followers.
Easy rules: send 0.3-0.7 ETH to my address, get 3-7 ETH to the address you used in the transaction.
Only for my followers.

Vitalik Buterin @HagenAnsel · Feb 17
0x58EF89B4D37f16002cc4A853B321dAEeE423519

Nunzio Tulk @TulkNunzio · Feb 17
Thanks! amazing promo! I just need to send for exmpl 0.6 eth and get back 6?? Am i right?

Twitter scammers: can prediction markets incentivize an autonomous swarm of human and AI-driven moderators to flag these posts and warn users not to send them ether within a few seconds of the post being made? And could such a system be generalized to the entire internet, where there is no single centralized moderator that can easily take posts down?

Some ideas others have had for decentralized institutions in general include:

- TrustDavis (<http://trustdavis.io/>). (adding skin-in-the-game to e-commerce reputations by making e-commerce ratings be offers to insure others against the receiver of the rating committing fraud)
- Circles (<https://joincircles.net/>). (decentralized basic income through locally fungible coin issuance)
- Markets for CAPTCHA services
- Digitized peer to peer rotating savings and credit associations (<https://www.wetrust.io/>),
- Token curated registries (<https://medium.com/@ilovebagels/token-curated-registries-1-0-61a232f8dac7>).
- Crowdsourced smart contract truth oracles (<https://medium.com/@edmundedgar/snopes-meets-mechanical-turk-announcing-reality-check-a-crowd-sourced-smart-contract-oracle-551d03468177>),
- Using blockchain-based smart contracts to coordinate unions

I would be interested in hearing Posner and Weyl's opinion on these kinds of "radical markets", that groups of people can spin up and start using by themselves without requiring potentially contentious society-wide changes to political and property rights. Could decentralized institutions like these be used to solve the key defining challenges of the twenty first century: promoting beneficial scientific progress, developing informational public goods, reducing global wealth inequality, and the big meta-problem behind fake news, government-driven and corporate-driven social media censorship, and regulation of cryptocurrency products: how do we do quality assurance in an open society?

All in all, I highly recommend *Radical Markets* (and by the way I also recommend Eliezer Yudkowsky's Inadequate Equilibria (<https://equilibriabook.com/>)) to anyone interested in these kinds of issues, and look forward to seeing the discussion that the book generates.

STARKs, Part 3: Into the Weeds

2018 Jul 21

[See all posts \(/\)](#)

Special thanks to Eli ben Sasson for his kind assistance, as usual. Special thanks to Chih-Cheng Liang and Justin Drake for review, and to Ben Fisch for suggesting the reverse MIMC technique for a VDF (paper [here](#) (<https://eprint.iacr.org/2018/601.pdf>))

Trigger warning: math and lots of python

As a followup to [Part 1](#) (https://vitalik.ca/general/2017/11/09/starks_part_1.html) and [Part 2](#) (https://vitalik.ca/general/2017/11/22/starks_part_2.html) of this series, this post will cover what it looks like to actually implement a STARK, complete with an implementation in python. STARKs ("Scalable Transparent ARgument of Knowledge" are a technique for creating a proof that $f(x) = y$ where f may potentially take a very long time to calculate, but where the proof can be verified very quickly. A STARK is "doubly scalable": for a computation with t steps, it takes roughly $O(t \cdot \log t)$ steps to produce a proof, which is likely optimal, and it takes $\sim O(\log^2 t)$ steps to verify, which for even moderately large values of t is much faster than the original computation. STARKs can also have a privacy-preserving "zero knowledge" property, though the use case we will apply them to here, making verifiable delay functions, does not require this property, so we do not need to worry about it.

First, some disclaimers:

- This code has not been thoroughly audited; soundness in production use cases is not guaranteed
- This code is very suboptimal (it's written in Python, what did you expect)
- STARKs "in real life" (ie. as implemented in Eli and co's production implementations) tend to use binary fields and not prime fields for application-specific efficiency reasons; however, they do stress in their writings the prime field-based approach to STARKs described here is legitimate and can be used
- There is no "one true way" to do a STARK. It's a broad category of cryptographic and mathematical constructs, with different setups optimal for different applications and constant ongoing research to reduce prover and verifier complexity and improve soundness.
- This article absolutely expects you to know how modular arithmetic and prime fields work, and be comfortable with the concepts of polynomials, interpolation and evaluation. If you don't, go back to [Part 2](#) (https://vitalik.ca/general/2017/11/22/starks_part_2.html), and also this [earlier post on quadratic arithmetic programs](#) (<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>).

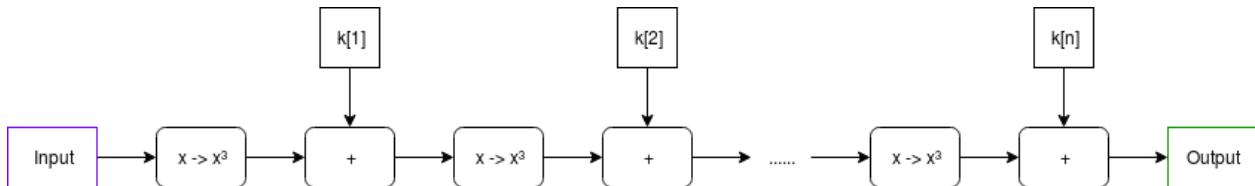
Now, let's get to it.

MIMC

Here is the function we'll be doing a STARK of:

```
def mimc(inp, steps, round_constants):
    start_time = time.time()
    for i in range(steps-1):
        inp = (inp**3 + round_constants[i % len(round_constants)]) % modulus
    print("MIMC computed in %.4f sec" % (time.time() - start_time))
    return inp
```

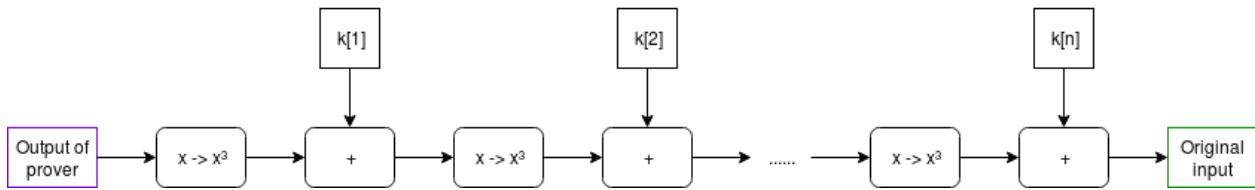
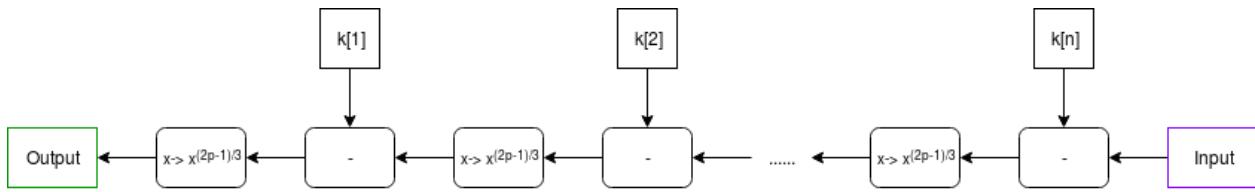
We choose MIMC (see paper (<https://eprint.iacr.org/2016/492.pdf>)) as the example because it is both (i) simple to understand and (ii) interesting enough to be useful in real life. The function can be viewed visually as follows:



Note: in many discussions of MIMC, you will typically see XOR used instead of +; this is because MIMC is typically done over binary fields, where addition is XOR; here we are doing it over prime fields.

In our example, the round constants will be a relatively small list (eg. 64 items) that gets cycled through over and over again (that is, after $k[64]$ it loops back to using $k[1]$).

MIMC with a very large number of rounds, as we're doing here, is useful as a *verifiable delay function* - a function which is difficult to compute, and particularly non-parallelizable to compute, but relatively easy to verify. MIMC by itself achieves this property to some extent because MIMC can be computed "backward" (recovering the "input" from its corresponding "output"), but computing it backward takes about 100 times longer to compute than the forward direction (and neither direction can be significantly sped up by parallelization). So you can think of computing the function in the backward direction as being the act of "computing" the non-parallelizable proof of work, and computing the function in the forward direction as being the process of "verifying" it.



$x \rightarrow x^{(2p-1)/3}$ gives the inverse of $x \rightarrow x^3$; this is true because of [Fermat's Little Theorem](#) (https://en.wikipedia.org/wiki/Fermat%27s_little_theorem), a theorem that despite its supposed littleness is arguably much more important to mathematics than Fermat's more famous "Last Theorem".

What we will try to achieve here is to make verification much more efficient by using a STARK - instead of the verifier having to run MIMC in the forward direction themselves, the prover, after completing the computation in the "backward direction", would compute a STARK of the computation in the "forward direction", and the verifier would simply verify the STARK. The hope is that the overhead of computing a STARK can be less than the difference in speed running MIMC forwards relative to backwards, so a prover's time would still be dominated by the initial "backward" computation, and not the (highly parallelizable) STARK computation. Verification of a STARK can be relatively fast (in our python implementation, ~0.05-0.3 seconds), no matter how long the original computation is.

All calculations are done modulo $2^{256} - 351 \cdot 2^{32} + 1$; we are using this prime field modulus because it is the largest prime below 2^{256} whose multiplicative group contains an order 2^{32} subgroup (that is, there's a number g such that successive powers of g modulo this prime loop around back to 1 after exactly 2^{32} cycles), and which is of the form $6k + 5$. The first property is necessary to make sure that our efficient versions of the FFT and FRI algorithms can work, and the second ensures that MIMC actually can be computed "backwards" (see the use of $x \rightarrow x^{(2p-1)/3}$ above).

Prime field operations

We start off by building a convenience class that does prime field operations, as well as operations with polynomials over prime fields. The code is [here](#)

(https://github.com/ethereum/research/blob/master/mimc_stark/poly_utils.py). First some trivial bits:

```
class PrimeField():
    def __init__(self, modulus):
        # Quick primality test
        assert pow(2, modulus, modulus) == 2
        self.modulus = modulus

    def add(self, x, y):
        return (x+y) % self.modulus

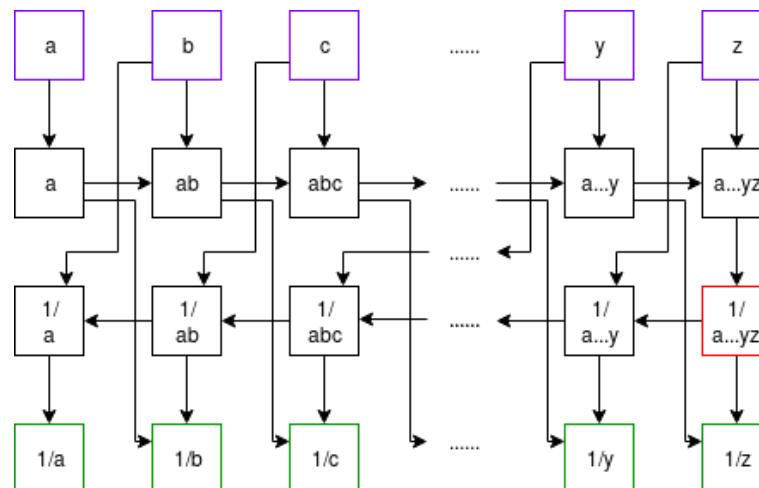
    def sub(self, x, y):
        return (x-y) % self.modulus

    def mul(self, x, y):
        return (x*y) % self.modulus
```

And the [Extended Euclidean Algorithm](#) (https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm) for computing modular inverses (the equivalent of computing $\frac{1}{x}$ in a prime field):

```
# Modular inverse using the extended Euclidean algorithm
def inv(self, a):
    if a == 0:
        return 0
    lm, hm = 1, 0
    low, high = a % self.modulus, self.modulus
    while low > 1:
        r = high//low
        nm, new = hm-lm*r, high-low*r
        lm, low, hm, high = nm, new, lm, low
    return lm % self.modulus
```

The above algorithm is relatively expensive; fortunately, for the special case where we need to do many modular inverses, there's a simple mathematical trick that allows us to compute many inverses, called [Montgomery batch inversion](#) (https://books.google.com/books?id=kGu4ITznRdgC&pg=PA54&lpg=PA54&dq=montgomery+batch+inversion&source=bl&ots=tPJcPPOrCe&sig=Z3p_6YYwYloRU-f1K-&f=false&hl=en&sa=X&ved=0ahUKEwjO8sumgJjcAhUDd6wKHWGNA9cQ6AEIRDAE#v=onepage&q=montgomery%20batch%20inversion&f=false)



Using Montgomery batch inversion to compute modular inverses. Inputs purple, outputs green, multiplication gates black; the red square is the **only** modular inversion.

The code below implements this algorithm, with some slightly ugly special case logic so that if there are zeroes in the set of what we are inverting, it sets their inverse to 0 and moves along.

```
def multi_inv(self, values):
    partials = [1]
    for i in range(len(values)):
        partials.append(self.mul(partials[-1], values[i] or 1))
    inv = self.inv(partials[-1])
    outputs = [0] * len(values)
    for i in range(len(values), 0, -1):
        outputs[i-1] = self.mul(partials[i-1], inv) if values[i-1] else 0
        inv = self.mul(inv, values[i-1] or 1)
    return outputs
```

This batch inverse algorithm will prove important later on, when we start dealing with dividing sets of evaluations of polynomials.

Now we move on to some polynomial operations. We treat a polynomial as an array, where element i is the i th degree term (eg. $x^3 + 2x + 1$ becomes $[1, 2, 0, 1]$). Here's the operation of evaluating a polynomial at *one point*:

```
# Evaluate a polynomial at a point
def eval_poly_at(self, p, x):
    y = 0
    power_of_x = 1
    for i, p_coeff in enumerate(p):
        y += power_of_x * p_coeff
        power_of_x = (power_of_x * x) % self.modulus
    return y % self.modulus
```

Challenge

What is the output of `f.eval_poly_at([4, 5, 6], 2)` if the modulus is 31?

Mouseover below for answer

$$6 \cdot 2^2 + 5 \cdot 2 + 4 = 38, 38 \bmod 31 = 7.$$

There is also code for adding, subtracting, multiplying and dividing polynomials; this is textbook long addition/subtraction/multiplication/division. The one non-trivial thing is Lagrange interpolation, which takes as input a set of x and y coordinates, and returns the minimal polynomial that passes through all of those points (you can think of it as being the inverse of polynomial evaluation):

```

# Build a polynomial that returns 0 at all specified xs
def zpoly(self, xs):
    root = [1]
    for x in xs:
        root.insert(0, 0)
        for j in range(len(root)-1):
            root[j] -= root[j+1] * x
    return [x % self.modulus for x in root]

def lagrange_interp(self, xs, ys):
    # Generate master numerator polynomial, eg. (x - x1) * (x - x2) * ... * (x - xn)
    root = self.zpoly(xs)

    # Generate per-value numerator polynomials, eg. for x=x2,
    # (x - x1) * (x - x3) * ... * (x - xn), by dividing the master
    # polynomial back by each x coordinate
    nums = [self.div_poly(root, [-x, 1]) for x in xs]

    # Generate denominators by evaluating numerator polys at each x
    denoms = [self.eval_poly_at(nums[i], xs[i]) for i in range(len(xs))]
    invdenoms = self.multi_inv(denoms)

    # Generate output polynomial, which is the sum of the per-value numerator
    # polynomials rescaled to have the right y values
    b = [0 for y in ys]
    for i in range(len(xs)):
        yslice = self.mul(ys[i], invdenoms[i])
        for j in range(len(ys)):
            if nums[i][j] and ys[i]:
                b[j] += nums[i][j] * yslice
    return [x % self.modulus for x in b]

```

See the "M of N" section of this article (<https://blog.ethereum.org/2014/08/16/secret-sharing-erasure-coding-guide-aspiring-dropbox-decentralizer/>) for a description of the math. Note that we also have special-case methods `lagrange_interp_4` and `lagrange_interp_2` to speed up the very frequent operations of Lagrange interpolation of degree < 2 and degree < 4 polynomials.

Fast Fourier Transforms

If you read the above algorithms carefully, you might notice that Lagrange interpolation and multi-point evaluation (that is, evaluating a degree $< N$ polynomial at N points) both take quadratic time to execute, so for example doing a Lagrange interpolation of one thousand points takes a few million steps to execute, and a Lagrange interpolation of one million points takes a few trillion. This is an unacceptably high level of inefficiency, so we will use a more efficient algorithm, the Fast Fourier Transform.

The FFT only takes $O(n \cdot \log(n))$ time (ie. ~10,000 steps for 1,000 points, ~20 million steps for 1 million points), though it is more restricted in scope; the x coordinates must be a complete set of **roots of unity** (https://en.wikipedia.org/wiki/Root_of_unity) of some **order** ([https://en.wikipedia.org/wiki/Order_\(group_theory\)](https://en.wikipedia.org/wiki/Order_(group_theory))). $N = 2^k$. That is, if there are N points, the x coordinates must be successive powers $1, p, p^2, p^3, \dots$ of some p where $p^N = 1$. The algorithm can, surprisingly enough, be used for multi-point evaluation or interpolation, with one small parameter tweak.

Challenge Find a 16th root of unity mod 337 that is not an 8th root of unity.

Mouseover below for answer

59, 146, 30, 297, 278, 191, 307, 40

You could have gotten this list by doing something like `[print(x) for x in range(337) if pow(x, 16, 337) == 1 and pow(x, 8, 337) != 1]`, though there is a smarter way that works for much larger modulus: first, identify a single *primitive root* mod 337 (that is, not a perfect square), by looking for a value x such that $\text{pow}(x, 336 // 2, 337) \neq 1$ (these are easy to find; one answer is 5), and then taking the $(336 / 16)$ 'th power of it.

Here's the algorithm (in a slightly simplified form; see [code here](#)

(https://github.com/ethereum/research/blob/master/mimc_stark/fft.py) for something slightly more optimized):

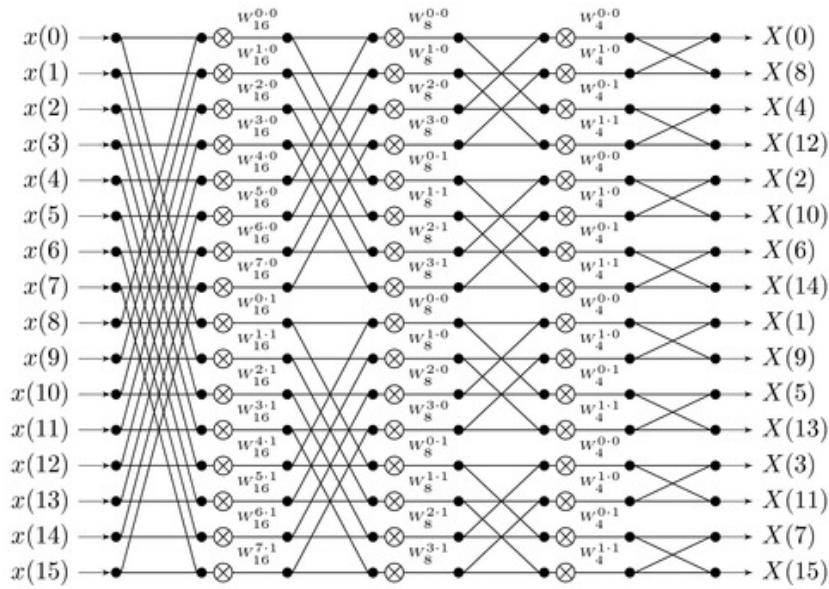
```
def fft(vals, modulus, root_of_unity):
    if len(vals) == 1:
        return vals
    L = fft(vals[::2], modulus, pow(root_of_unity, 2, modulus))
    R = fft(vals[1::2], modulus, pow(root_of_unity, 2, modulus))
    o = [0 for i in vals]
    for i, (x, y) in enumerate(zip(L, R)):
        y_times_root = y * pow(root_of_unity, i, modulus)
        o[i] = (x + y_times_root) % modulus
        o[i + len(L)] = (x - y_times_root) % modulus
    return o

def inv_fft(vals, modulus, root_of_unity):
    f = PrimeField(modulus)
    # Inverse FFT
    invlen = f.inv(len(vals))
    return [(x * invlen) % modulus for x in
            fft(vals, modulus, f.inv(root_of_unity))]
```

You can try running it on a few inputs yourself and check that it gives results that, when you use `eval_poly_at` on them, give you the answers you expect to get. For example:

```
>>> fft.fft([3,1,4,1,5,9,2,6], 337, 85, inv=True)
[46, 169, 29, 149, 126, 262, 140, 93]
>>> f = poly_utils.PrimeField(337)
>>> [f.eval_poly_at([46, 169, 29, 149, 126, 262, 140, 93], f.exp(85, i)) for i in range(8)]
[3, 1, 4, 1, 5, 9, 2, 6]
```

A Fourier transform takes as input $[x[0] \dots x[n-1]]$, and its goal is to output $x[0] + x[1] + \dots + x[n-1]$ as the first element, $x[0] + x[1] * 2 + \dots + x[n-1] * w^{*(n-1)}$ as the second element, etc etc; a fast Fourier transform accomplishes this by splitting the data in half, doing an FFT on both halves, and then gluing the result back together.



A diagram of how information flows through the FFT computation. Notice how the FFT consists of a "gluing" step followed by two copies of the FFT on two halves of the data, and so on recursively until you're down to one element.

I recommend [this](http://web.cecs.pdx.edu/~maier/cs584/Lectures/lect07b-11-MG.pdf) (<http://web.cecs.pdx.edu/~maier/cs584/Lectures/lect07b-11-MG.pdf>) for more intuition on how or why the FFT works and polynomial math in general, and [this thread](https://dsp.stackexchange.com/questions/41558/what-are-some-of-the-differences-between-dft-and-fft-that-make-fft-so-fast?rq=1) (<https://dsp.stackexchange.com/questions/41558/what-are-some-of-the-differences-between-dft-and-fft-that-make-fft-so-fast?rq=1>) for some more specifics on DFT vs FFT, though be warned that most literature on Fourier transforms talks about Fourier transforms over *real and complex numbers*, not *prime fields*. If you find this too hard and don't want to understand it, just treat it as weird spooky voodoo that just works because you ran the code a few times and verified that it works, and you'll be fine too.

Thank Goodness It's FRI-day (that's "Fast Reed-Solomon Interactive Oracle Proofs of Proximity")

Reminder: now may be a good time to review and re-read [Part 2](#) (https://vitalik.ca/general/2017/11/22/starks_part_2.html).

Now, we'll get into [the code](https://github.com/ethereum/research/blob/master/mimc_stark/fri.py) (https://github.com/ethereum/research/blob/master/mimc_stark/fri.py), for making a low-degree proof. To review, a low-degree proof is a (probabilistic) proof that at least some high percentage (eg. 80%) of a given set of values represent the evaluations of some specific polynomial whose degree is much lower than the number of values given. Intuitively, just think of it as a proof that "some Merkle root that we claim represents a polynomial actually does represent a polynomial, possibly with a few errors". As input, we have:

- A set of values that we claim are the evaluation of a low-degree polynomial
- A root of unity; the x coordinates at which the polynomial is evaluated are successive powers of this root of unity
- A value N such that we are proving the degree of the polynomial is *strictly less than* N
- The modulus

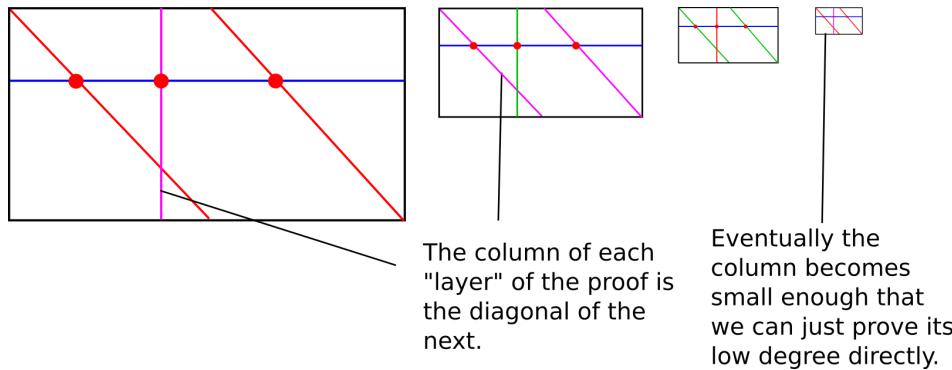
Our approach is a recursive one, with two cases. First, if the degree is low enough, we just provide the entire list of values as a proof; this is the "base case". Verification of the base case is trivial: do an FFT or Lagrange interpolation or whatever else to interpolate the polynomial representing those values, and verify that its degree is < N. Otherwise, if the degree is higher than some set minimum, we do the vertical-and-diagonal trick described [at the bottom of Part 2](#) (https://vitalik.ca/general/2017/11/22/starks_part_2.html).

We start off by putting the values into a Merkle tree and using the Merkle root to select a pseudo-random x coordinate (`special_x`). We then calculate the "column":

```
# Calculate the set of x coordinates
xs = get_power_cycle(root_of_unity, modulus)

column = []
for i in range(len(xs)//4):
    x_poly = f.lagrange_interp_4(
        [xs[i+len(xs)*j//4] for j in range(4)],
        [values[i+len(values)*j//4] for j in range(4)],
    )
    column.append(f.eval_poly_at(x_poly, special_x))
```

This packs a lot into a few lines of code. The broad idea is to re-interpret the polynomial $P(x)$ as a polynomial $Q(x, y)$, where $P(x) = Q(x, x^4)$. If P has degree $< N$, then $P'(y) = Q(\text{special_x}, y)$ will have degree $< \frac{N}{4}$. Since we don't want to take the effort to actually compute Q in coefficient form (that would take a still-relatively-nasty-and-expensive FFT!), we instead use another trick. For any given value of x^4 , there are 4 corresponding values of x : x , modulus $-x$, and x multiplied by the two modular square roots of -1 . So we already have four values of $Q(?, x^4)$, which we can use to interpolate the polynomial $R(x) = Q(x, x^4)$, and from there calculate $R(\text{special_x}) = Q(\text{special_x}, x^4) = P'(x^4)$. There are $\frac{N}{4}$ possible values of x^4 , and this lets us easily calculate all of them.



A diagram from part 2; it helps to keep this in mind when understanding what's going on here

Our proof consists of some number (eg. 40) of random queries from the list of values of x^4 (using the Merkle root of the column as a seed), and for each query we provide Merkle branches of the five values of $Q(?, x^4)$:

```
m2 = merkelize(column)

# Pseudo-randomly select y indices to sample
# (m2[1] is the Merkle root of the column)
ys = get_pseudorandom_indices(m2[1], len(column), 40)

# Compute the Merkle branches for the values in the polynomial and the column
branches = []
for y in ys:
    branches.append([mk_branch(m2, y)] +
                    [mk_branch(m, y + (len(xs) // 4) * j) for j in range(4)])
```

The verifier's job will be to verify that these five values actually do lie on the same degree < 4 polynomial. From there, we recurse and do an FRI on the column, verifying that the column actually does have degree $< \frac{N}{4}$. That really is all there is to FRI.

As a challenge exercise, you could try creating low-degree proofs of polynomial evaluations that have errors in them, and see how many errors you can get away passing the verifier with (hint, you'll need to modify the `prove_low_degree` function; with the default prover, even one error will balloon up and cause verification to fail).

The STARK

Reminder: now may be a good time to review and re-read [Part 1](#) (https://vitalik.ca/general/2017/11/09/starks_part_1.html).

Now, we get to the actual meat that puts all of these pieces together: `def mk_mimc_proof(inp, steps, round_constants)` (code [here](#) (https://github.com/ethereum/research/blob/master/mimc_stark/mimc_stark.py)), which generates a proof of the execution result of running the MIMC function with the given input for some number of steps. First, some asserts:

```
assert steps <= 2**32 // extension_factor
assert is_a_power_of_2(steps) and is_a_power_of_2(len(round_constants))
assert len(round_constants) < steps
```

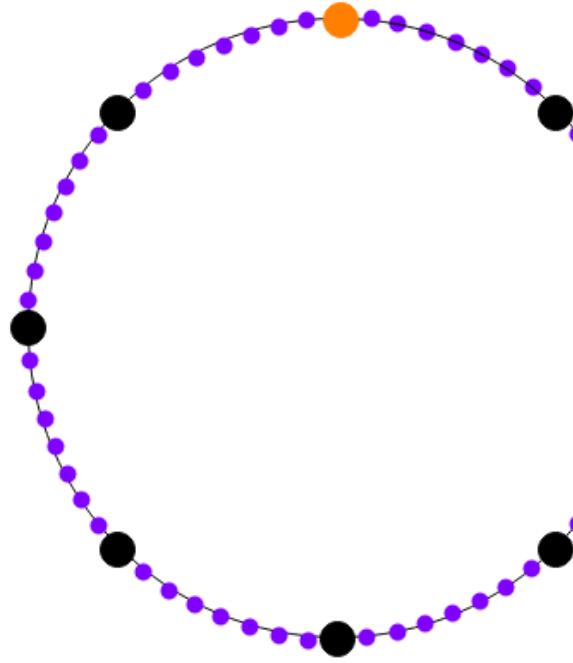
The extension factor is the extent to which we will be "stretching" the computational trace (the set of "intermediate values" of executing the MIMC function). We need the step count multiplied by the extension factor to be at most 2^{32} , because we don't have roots of unity of order 2^k for $k > 32$.

Our first computation will be to generate the computational trace; that is, all of the *intermediate* values of the computation, from the input going all the way to the output.

```
# Generate the computational trace
computational_trace = [inp]
for i in range(steps-1):
    computational_trace.append((computational_trace[-1]**3 + round_constants[i % len(round_cons)])
output = computational_trace[-1]
```

We then convert the computation trace into a polynomial, "laying down" successive values in the trace on successive powers of a root of unity g where $g^{steps} = 1$, and we then evaluate the polynomial in a larger set, of successive powers of a root of unity g_2 where $(g_2)^{steps \cdot 8} = 1$ (note that $(g_2)^8 = g$).

```
computational_trace_polynomial = inv_fft(computational_trace, modulus, subroot)
p_evaluations = fft(computational_trace_polynomial, modulus, root_of_unity)
```



Black: powers of g_1 . Purple: powers of g_2 . Orange: 1. You can look at successive roots of unity as being arranged in a circle in this way. We are "laying" the computational trace along powers of g_1 , and then extending it to compute the values of the same polynomial at the intermediate values (ie. the powers of g_2).

We can convert the round constants of MIMC into a polynomial. Because these round constants loop around very frequently (in our tests, every 64 steps), it turns out that they form a degree-64 polynomial, and we can fairly easily compute its expression, and its extension:

```
skips2 = steps // len(round_constants)
constants_mini_polynomial = fft(round_constants, modulus, f.exp(subroot, skips2), inv=True)
constants_polynomial = [0 if i % skips2 else constants_mini_polynomial[i//skips2] for i in range(steps)]
constants_mini_extension = fft(constants_mini_polynomial, modulus, f.exp(root_of_unity, skips2))
```

Suppose there are 8192 steps of execution and 64 round constants. Here is what we are doing: we are doing an FFT to compute the round constants as a function of $(g_1)^{128}$. We then add zeroes in between the constants to make it a function of g_1 itself. Because $(g_1)^{128}$ loops around every 64 steps, we know this function of g_1 will as well. We only compute 512 steps of the extension, because we know that the extension repeats after 512 steps as well.

We now, as in the Fibonacci example in Part 1, calculate $C(P(x))$, except this time it's $C(P(x), P(g_1 \cdot x), K(x))$:

```
# Create the composed polynomial such that
# C(P(x), P(g1*x), K(x)) = P(g1*x) - P(x)**3 - K(x)
c_of_p_evaluations = [(p_evaluations[(i+extension_factor)%precision] -
                      f.exp(p_evaluations[i], 3) -
                      constants_mini_extension[i % len(constants_mini_extension)])
                      % modulus for i in range(precision)]
print('Computed C(P, K) polynomial')
```

Note that here we are no longer working with polynomials in *coefficient form*; we are working with the polynomials in terms of their evaluations at successive powers of the higher-order root of unity.

`c_of_p` is intended to be $Q(x) = C(P(x), P(g_1 \cdot x), K(x)) = P(g_1 \cdot x) - P(x)^3 - K(x)$; the goal is that for every x that we are laying the computational trace along (except for the last step, as there's no step "after" the last step), the next value in the trace is equal to the previous value in the trace cubed, plus the round constant. Unlike the Fibonacci example in Part 1, where if one computational step was at coordinate k , the next step is at coordinate $k + 1$, here we are laying down the computational trace along successive powers of the lower-order root of unity g_1 , so if one computational step is located at $x = (g_1)^i$, the "next" step is located at $(g_1)^{i+1} = (g_1)^i \cdot g_1 = x \cdot g_1$. Hence, for every power of the lower-order root of unity g_1 (except the last), we want it to be the case that $P(x \cdot g_1) = P(x)^3 + K(x)$, or $P(x \cdot g_1) - P(x)^3 - K(x) = Q(x) = 0$. Thus, $Q(x)$ will be equal to zero at all successive powers of the lower-order root of unity g (except the last).

There is an algebraic theorem that proves that if $Q(x)$ is equal to zero at all of these x coordinates, then it is a multiple of the *minimal* polynomial that is equal to zero at all of these x coordinates:

$Z(x) = (x - x_1) \cdot (x - x_2) \cdots (x - x_n)$. Since proving that $Q(x)$ is equal to zero at every single coordinate we want to check is too hard (as verifying such a proof would take longer than just running the original computation!), instead we use an indirect approach to (probabilistically) prove that $Q(x)$ is a multiple of $Z(x)$. And how do we do that? By providing the quotient $D(x) = \frac{Q(x)}{Z(x)}$ and using FRI to prove that it's an actual polynomial and not a fraction, of course!

We chose the particular arrangement of lower and higher order roots of unity (rather than, say, laying the computational trace along the first few powers of the higher order root of unity) because it turns out that computing $Z(x)$ (the polynomial that evaluates to zero at all points along the computational trace except the last), and dividing by $Z(x)$ is trivial there: the expression of Z is a fraction of two terms.

```
# Compute D(x) = Q(x) / Z(x)
# Z(x) = (x^steps - 1) / (x - x_atlast_step)
z_num_evaluations = [xs[(i * steps) % precision] - 1 for i in range(precision)]
z_num_inv = f.multi_inv(z_num_evaluations)
z_den_evaluations = [xs[i] - last_step_position for i in range(precision)]
d_evaluations = [cp * zd * zni % modulus for cp, zd, zni in zip(c_of_p_evaluations, z_den_evaluations)]
print('Computed D polynomial')
```

Notice that we compute the numerator and denominator of Z directly in "evaluation form", and then use the batch modular inversion to turn dividing by Z into a multiplication ($\cdot z_d \cdot z_{n,i}$), and then pointwise multiply the evaluations of $Q(x)$ by these inverses of $Z(x)$. Note that at the powers of the lower-order root of unity except the last (ie. along the portion of the low-degree extension that is part of the original computational trace), $Z(x) = 0$, so this computation involving its inverse will break. This is unfortunate, though we will plug the hole by simply modifying the random checks and FRI algorithm to not sample at those points, so the fact that we calculated them wrong will never matter.

Because $Z(x)$ can be expressed so compactly, we get another benefit: the verifier can compute $Z(x)$ for any specific x extremely quickly, without needing any precomputation. It's okay for the prover to have to deal with polynomials whose size equals the number of steps, but we don't want to ask the verifier to do the same, as we want verification to be succinct (ie. ultra-fast, with proofs as small as possible).

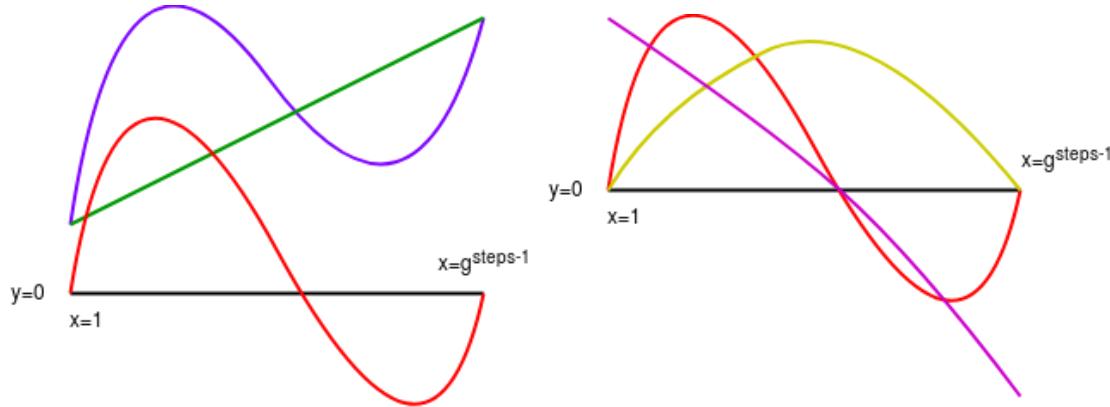
Probabilistically checking $D(x) \cdot Z(x) = Q(x)$ at a few randomly selected points allows us to verify the **transition constraints** - that each computational step is a valid consequence of the previous step. But we also want to verify the **boundary constraints** - that the input and the output of the computation is what the prover says they are. Just asking the prover to provide evaluations of $P(1)$, $D(1)$, $P(\text{last_step})$ and $D(\text{last_step})$ (where last_step (or $g^{steps-1}$) is the coordinate corresponding to the last step in the computation) is too fragile; there's no proof that those values are on the same polynomial as the rest of the data. So instead we use a similar kind of polynomial division trick:

```
# Compute interpolant of ((1, input), (x_atlast_step, output))
interpolant = f.lagrange_interp_2([1, last_step_position], [inp, output])
i_evaluations = [f.eval_poly_at(interpolant, x) for x in xs]

zeropoly2 = f.mul_poly([-1, 1], [-last_step_position, 1])
inv_z2_evaluations = f.multi_inv([f.eval_poly_at(quotient, x) for x in xs])

# B = (P - I) / Z2
b_evaluations = [((p - i) * invq) % modulus for p, i, invq in zip(p_evaluations, i_evaluations, b_evaluations)]
print('Computed B polynomial')
```

The argument is as follows. The prover wants to prove $P(1) = \text{input}$ and $P(\text{last_step}) = \text{output}$. If we take $I(x)$ as the *interpolant* - the line that crosses the two points $(1, \text{input})$ and $(\text{last_step}, \text{output})$, then $P(x) - I(x)$ would be equal to zero at those two points. Thus, it suffices to prove that $P(x) - I(x)$ is a multiple of $(x - 1) \cdot (x - \text{last_step})$, and we do that by... providing the quotient!



Purple: computational trace polynomial (P). Green: interpolant (I) (notice how the interpolant is constructed to equal the input (which should be the first step of the computational trace) at $x=1$ and the output (which should be the last step of the computational trace) at $x = g^{\text{steps}-1}$. Red: $P - I$. Yellow: the minimal polynomial that equals 0 at $x = 1$ and $x = g^{\text{steps}-1}$ (that is, Z_2). Pink: $\frac{P-I}{Z_2}$.

Challenge Suppose you wanted to *also* prove that the value in the computational trace after the 703rd computational step is equal to 8018284612598740. How would you modify the above algorithm to do that?

Mouseover below for answer

Set $I(x)$ to be the interpolant of $(1, \text{input}), (g^{703}, 8018284612598740), (\text{last_step}, \text{output})$, and make a proof by providing the quotient $B(x) = \frac{P(x)-I(x)}{(x-1) \cdot (x-g^{703}) \cdot (x-\text{last_step})}$

Now, we commit to the Merkle root of P , D and B combined together.

```
# Compute their Merkle roots
mtree = merkelize([pval.to_bytes(32, 'big') +
                    dval.to_bytes(32, 'big') +
                    bval.to_bytes(32, 'big') for
                    pval, dval, bval in zip(p_evaluations, d_evaluations, b_evaluations)])
print('Computed hash root')
```

Now, we need to prove that P , D and B are all actually polynomials, and of the right max-degree. But FRI proofs are big and expensive, and we don't want to have three FRI proofs. So instead, we compute a pseudorandom linear combination of P , D and B (using the Merkle root of P , D and B as a seed), and do an FRI proof on that:

```

k1 = int.from_bytes(blake(mtree[1] + b'\x01'), 'big')
k2 = int.from_bytes(blake(mtree[1] + b'\x02'), 'big')
k3 = int.from_bytes(blake(mtree[1] + b'\x03'), 'big')
k4 = int.from_bytes(blake(mtree[1] + b'\x04'), 'big')

# Compute the linear combination. We don't even bother calculating it
# in coefficient form; we just compute the evaluations
root_of_unity_to_the_steps = f.exp(root_of_unity, steps)
powers = [1]
for i in range(1, precision):
    powers.append(powers[-1] * root_of_unity_to_the_steps % modulus)

l_evaluations = [(d_evaluations[i] +
                  p_evaluations[i] * k1 + p_evaluations[i] * k2 * powers[i] +
                  b_evaluations[i] * k3 + b_evaluations[i] * powers[i] * k4) % modulus
                  for i in range(precision)]

```

Unless all three of the polynomials have the right low degree, it's almost impossible that a randomly selected linear combination of them will (you have to get *extremely* lucky for the terms to cancel), so this is sufficient.

We want to prove that the degree of D is less than $2 \cdot \text{steps}$, and that of P and B are less than steps , so we actually make a random linear combination of P, $P \cdot x^{\text{steps}}$, B, B^{steps} and D, and check that the degree of this combination is less than $2 \cdot \text{steps}$.

Now, we do some spot checks of all of the polynomials. We generate some random indices, and provide the Merkle branches of the polynomial evaluated at those indices:

```

# Do some spot checks of the Merkle tree at pseudo-random coordinates, excluding
# multiples of `extension_factor`
branches = []
samples = spot_check_security_factor
positions = get_pseudorandom_indices(l_mtree[1], precision, samples,
                                      exclude_multiples_of=extension_factor)
for pos in positions:
    branches.append(mk_branch(mtree, pos))
    branches.append(mk_branch(mtree, (pos + skips) % precision))
    branches.append(mk_branch(l_mtree, pos))
print('Computed %d spot checks' % samples)

```

The `get_pseudorandom_indices` function returns some random indices in the range [0...precision-1], and the `exclude_multiples_of` parameter tells it to not give values that are multiples of the given parameter (here, `extension_factor`). This ensures that we do not sample along the original computational trace, where we are likely to get wrong answers.

The proof (~250-500 kilobytes altogether) consists of a set of Merkle roots, the spot-checked branches, and a low-degree proof of the random linear combination:

```

o = [mtree[1],
     l_mtree[1],
     branches,
     prove_low_degree(l_evaluations, root_of_unity, steps * 2, modulus, exclude_multiples_of=ex

```

The largest parts of the proof in practice are the Merkle branches, and the FRI proof, which consists of even more branches. And here's the "meat" of the verifier:

```

for i, pos in enumerate(positions):
    x = f.exp(G2, pos)
    x_to_the_steps = f.exp(x, steps)
    mbranch1 = verify_branch(m_root, pos, branches[i*3])
    mbranch2 = verify_branch(m_root, (pos+skips)%precision, branches[i*3+1])
    l_of_x = verify_branch(l_root, pos, branches[i*3 + 2], output_as_int=True)

    p_of_x = int.from_bytes(mbranch1[:32], 'big')
    p_of_g1x = int.from_bytes(mbranch2[:32], 'big')
    d_of_x = int.from_bytes(mbranch1[32:64], 'big')
    b_of_x = int.from_bytes(mbranch1[64:], 'big')

    zvalue = f.div(f.exp(x, steps) - 1,
                  x - last_step_position)
    k_of_x = f.eval_poly_at(constants_mini_polynomial, f.exp(x, skips2))

    # Check transition constraints Q(x) = Z(x) * D(x)
    assert (p_of_g1x - p_of_x ** 3 - k_of_x - zvalue * d_of_x) % modulus == 0

    # Check boundary constraints B(x) * Z2(x) + I(x) = P(x)
    interpolant = f.lagrange_interp_2([1, last_step_position], [inp, output])
    zeropoly2 = f.mul polys([-1, 1], [-last_step_position, 1])
    assert (p_of_x - b_of_x * f.eval_poly_at(zeropoly2, x) -
            f.eval_poly_at(interpolant, x)) % modulus == 0

    # Check correctness of the linear combination
    assert (l_of_x - d_of_x -
            k1 * p_of_x - k2 * p_of_x * x_to_the_steps -
            k3 * b_of_x - k4 * b_of_x * x_to_the_steps) % modulus == 0

```

At every one of the positions that the prover provides a Merkle proof for, the verifier checks the Merkle proof, and checks that $C(P(x), P(g_1 \cdot x), K(x)) = Z(x) \cdot D(x)$ and $B(x) \cdot Z_2(x) + I(x) = P(x)$ (reminder: for x that are not along the original computation trace, $Z(x)$ will not be zero, and so $C(P(x), P(g_1 \cdot x), K(x))$ likely will not evaluate to zero). The verifier also checks that the linear combination is correct, and calls

`verify_low_degree_proof(l_root, root_of_unity, fri_proof, steps * 2, modulus, exclude_multiples_of=extension_factor)` to verify the FRI proof. **And we're done!**

Well, not really; soundness analysis to prove how many spot-checks for the cross-polynomial checking and for the FRI are necessary is really tricky. But that's all there is to the code, at least if you don't care about making even crazier optimizations. When I run the code above, we get a STARK proving "overhead" of about 300-400x (eg. a MIMC computation that takes 0.2 seconds to calculate takes 60 second to prove), suggesting that with a 4-core machine computing the STARK of the MIMC computation in the forward direction could actually be faster than computing MIMC in the backward direction. That said, these are both relatively inefficient implementations in python, and the proving to running time ratio for properly optimized implementations may be different. Also, it's worth pointing out that the STARK proving overhead for MIMC is remarkably low, because MIMC is almost perfectly "arithmetizable" - it's mathematical form is very simple. For "average" computations, which contain less arithmetically clean operations (eg. checking if a number is greater or less than another number), the overhead is likely much higher, possibly around 10000-50000x.

A Guide to 99% Fault Tolerant Consensus

2018 Aug 07

[See all posts \(/\)](#)

Special thanks to Emin Gun Sirer for review

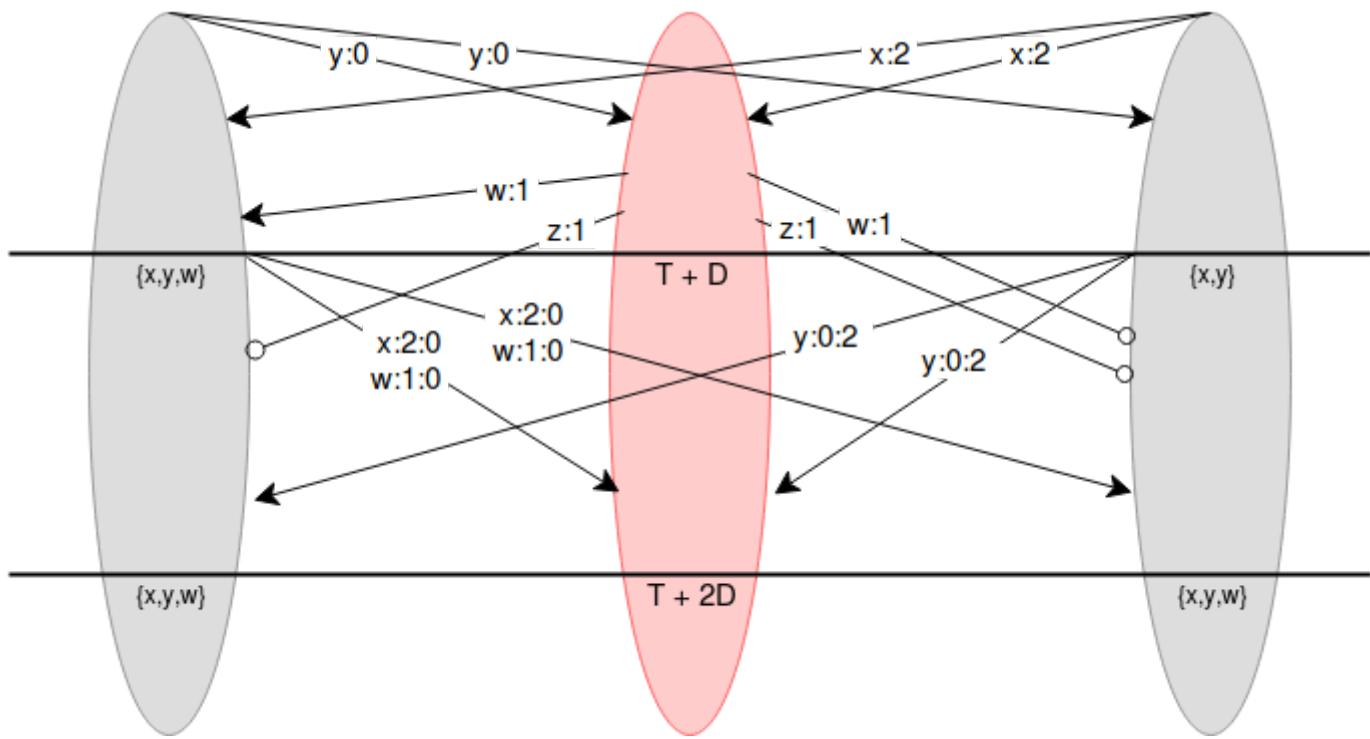
We've heard for a long time that it's possible to achieve consensus with 50% fault tolerance in a synchronous network where messages broadcasted by any honest node are guaranteed to be received by all other honest nodes within some known time period (if an attacker has *more* than 50%, they can perform a "51% attack", and there's an analogue of this for any algorithm of this type). We've also heard for a long time that if you want to relax the synchrony assumption, and have an algorithm that's "safe under asynchrony", the maximum achievable fault tolerance drops to 33% ([PBFT](http://pmg.csail.mit.edu/papers/osdi99.pdf) (<http://pmg.csail.mit.edu/papers/osdi99.pdf>), [Casper FFG](https://arxiv.org/abs/1710.09437) (<https://arxiv.org/abs/1710.09437>), etc all fall into this category). But did you know that if you add even *more* assumptions (specifically, you require *observers*, ie. users that are not actively participating in the consensus but care about its output, to also be actively watching the consensus, and not just downloading its output after the fact), you can increase fault tolerance all the way to 99%?

This has in fact been known for a long time; Leslie Lamport's famous 1982 paper "The Byzantine Generals Problem" ([link here](https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf) (<https://people.eecs.berkeley.edu/~luca/cs174/byzantine.pdf>)) contains a description of the algorithm. The following will be my attempt to describe and reformulate the algorithm in a simplified form.

Suppose that there are N consensus-participating nodes, and everyone agrees who these nodes are ahead of time (depending on context, they could have been selected by a trusted party or, if stronger decentralization is desired, by some proof of work or proof of stake scheme). We label these nodes $0 \dots N - 1$. Suppose also that there is a known bound D on network latency plus clock disparity (eg. $D = 8$ seconds). Each node has the ability to publish a value at time T (a malicious node can of course propose values earlier or later than T). All nodes wait $(N - 1) \cdot D$ seconds, running the following process. Define $x : i$ as "the value x signed by node i ", $x : i : j$ as "the value x signed by i , and that value and signature together signed by j ", etc. The proposals published in the first stage will be of the form $v : i$ for some v and i , containing the signature of the node that proposed it.

If a validator i receives some message $v : i[1] \dots : i[k]$, where $i[1] \dots i[k]$ is a list of indices that have (sequentially) signed the message already (just v by itself would count as $k = 0$, and $v : i$ as $k = 1$), then the validator checks that (i) the time is less than $T + k \cdot D$, and (ii) they have not yet seen a valid message containing v ; if both checks pass, they publish $v : i[1] \dots : i[k] : i$.

At time $T + (N - 1) \cdot D$, nodes stop listening. At this point, there is a guarantee that honest nodes have all "validly seen" the same set of values.



Node 1 (red) is malicious, and nodes 0 and 2 (grey) are honest. At the start, the two honest nodes make their proposals y and x , and the attacker proposes both w and z late. w reaches node 0 on time but not node 2, and z reaches neither node on time. At time $T + D$, nodes 0 and 2 rebroadcast all values they've seen that they have not yet broadcasted, but add their signatures on (x and w for node 0, y for node 2). Both honest nodes saw x, y, w .

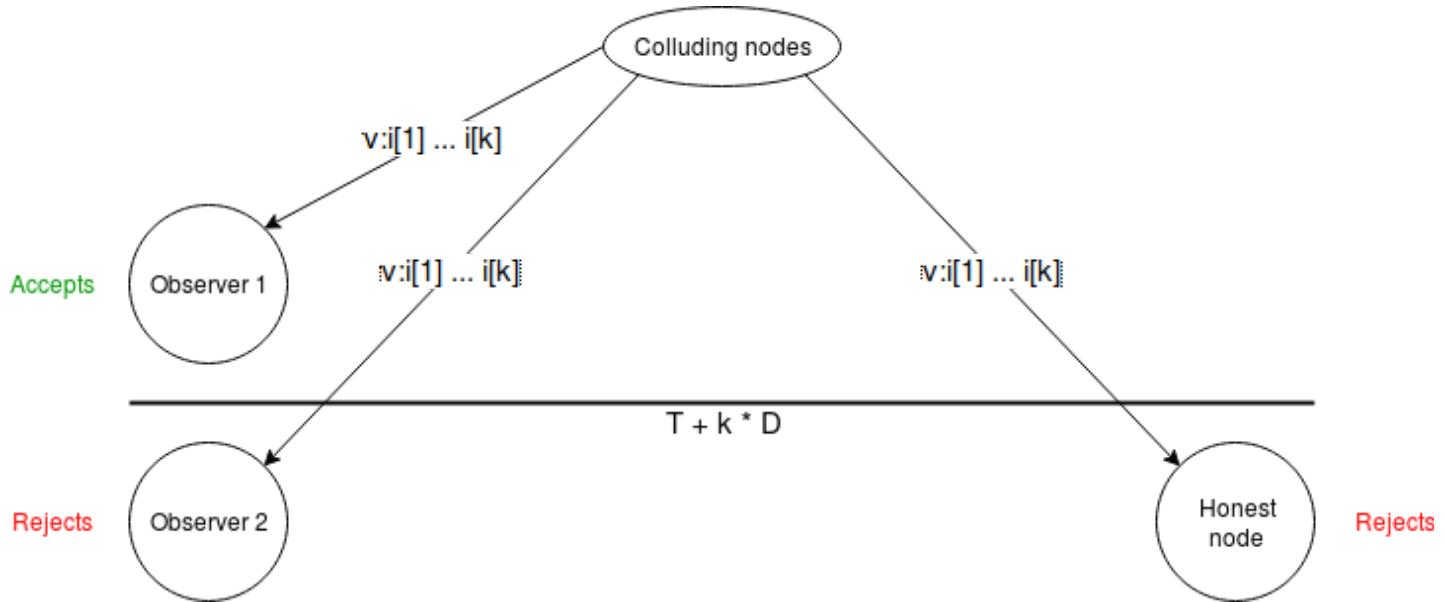
If the problem demands choosing one value, they can use some "choice" function to pick a single value out of the values they have seen (eg. they take the one with the lowest hash). The nodes can then agree on this value.

Now, let's explore why this works. What we need to prove is that if one honest node has seen a particular value (validly), then every other honest node has also seen that value (and if we prove this, then we know that all honest nodes have seen the same set of values, and so if all honest nodes are running the same choice function, they will choose the same value). Suppose that any honest node receives a message $v : i[1] \dots : i[k]$ that they perceive to be valid (ie. it arrives before time $T + k \cdot D$). Suppose x is the index of a single other honest node. Either x is part of $i[1] \dots i[k]$ or it is not.

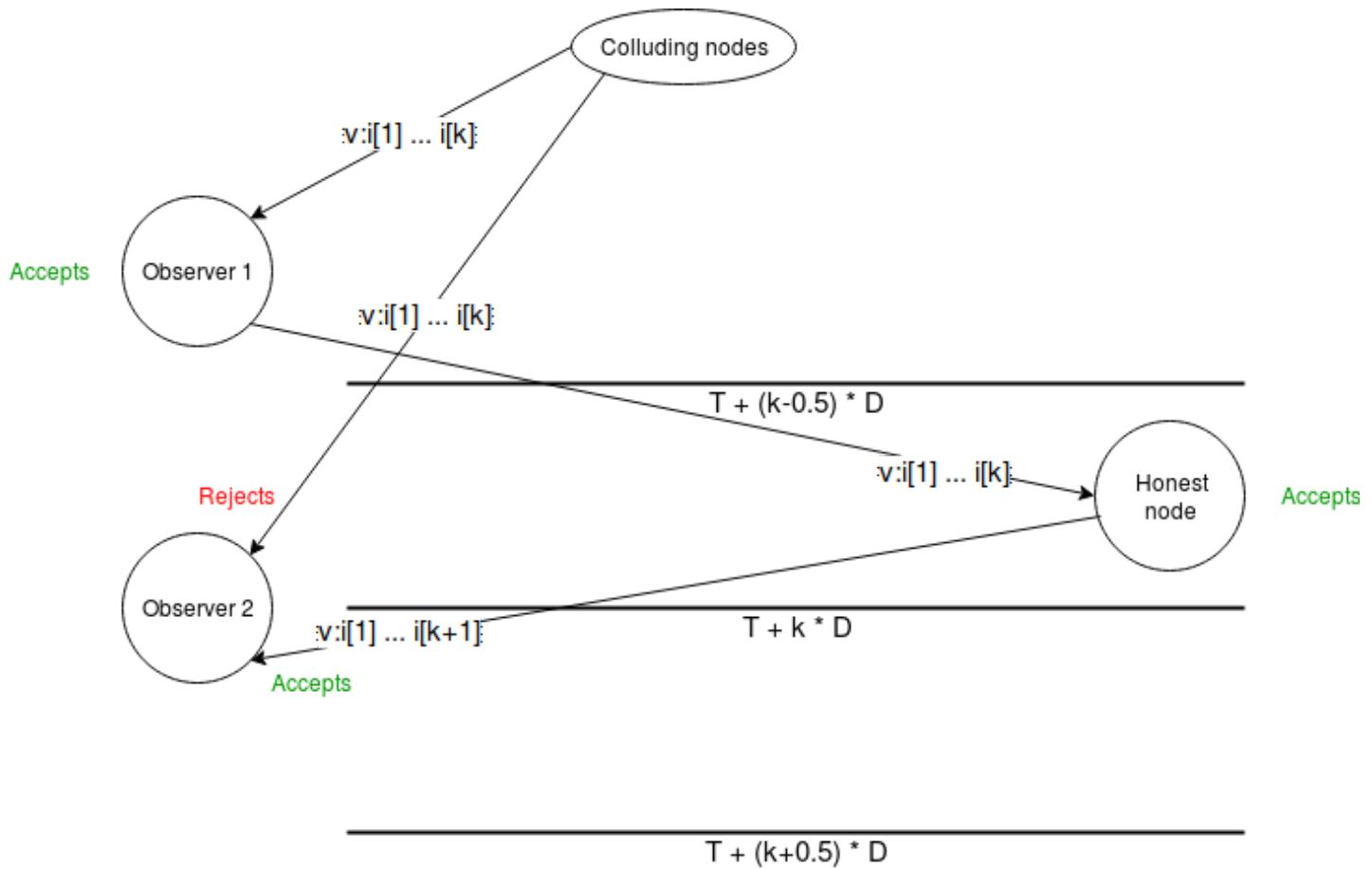
- In the first case (say $x = i[j]$ for this message), we know that the honest node x had already broadcasted that message, and they did so in response to a message with $j - 1$ signatures that they received before time $T + (j - 1) \cdot D$, so they broadcast their message at that time, and so the message must have been received by all honest nodes before time $T + j \cdot D$.
- In the second case, since the honest node sees the message before time $T + k \cdot D$, then they will broadcast the message with their signature and guarantee that everyone, including x , will see it before time $T + (k + 1) \cdot D$.

Notice that the algorithm uses the act of adding one's own signature as a kind of "bump" on the timeout of a message, and it's this ability that guarantees that if one honest node saw a message on time, they can ensure that everyone else sees the message on time as well, as the definition of "on time" increments by more than network latency with every added signature.

In the case where one node is honest, can we guarantee that passive *observers* (ie. non-consensus-participating nodes that care about knowing the outcome) can also see the outcome, even if we require them to be watching the process the whole time? With the scheme as written, there's a problem. Suppose that a commander and some subset of k (malicious) validators produce a message $v : i[1] : \dots : i[k]$, and broadcast it directly to some "victims" just before time $T + k \cdot D$. The victims see the message as being "on time", but when they rebroadcast it, it only reaches all honest consensus-participating nodes after $T + k \cdot D$, and so all honest consensus-participating nodes reject it.



But we can plug this hole. We require D to be a bound on *two times* network latency plus clock disparity. We then put a different timeout on observers: an observer accepts $v : i[1] : \dots : i[k]$ before time $T + (k - 0.5) \cdot D$. Now, suppose an observer sees a message and accepts it. They will be able to broadcast it to an honest node before time $T + k \cdot D$, and the honest node will issue the message with their signature attached, which will reach all other observers before time $T + (k + 0.5) \cdot D$, the timeout for messages with $k + 1$ signatures.



Retrofitting onto other consensus algorithms

The above could theoretically be used as a standalone consensus algorithm, and could even be used to run a proof-of-stake blockchain. The validator set of round $N + 1$ of the consensus could itself be decided during round N of the consensus (eg. each round of a consensus could also accept "deposit" and "withdraw" transactions, which if accepted and correctly signed would add or remove validators into the next round). The main additional ingredient that would need to be added is a mechanism for deciding who is allowed to propose blocks (eg. each round could have one designated proposer). It could also be modified to be usable as a proof-of-work blockchain, by allowing consensus-participating nodes to "declare themselves" in real time by publishing a proof of work solution on top of their public key at the same time as signing a message with it.

However, the synchrony assumption is very strong, and so we would like to be able to work without it in the case where we don't need more than 33% or 50% fault tolerance. There is a way to accomplish this. Suppose that we have some other consensus algorithm (eg. PBFT, Casper FFG, chain-based PoS) whose output can be seen by occasionally-online observers (we'll call this the *threshold-dependent* consensus algorithm, as opposed to the algorithm above, which we'll call the *latency-dependent* consensus algorithm). Suppose that the threshold-dependent consensus algorithm runs continuously, in a mode where it is constantly "finalizing" new blocks onto a chain (ie. each finalized value points to some previous finalized value as a "parent"; if there's a sequence of pointers $A \rightarrow \dots \rightarrow B$, we'll call A a *descendant* of B).

We can retrofit the latency-dependent algorithm onto this structure, giving always-online observers access to a kind of "strong finality" on checkpoints, with fault tolerance ~95% (you can push this arbitrarily close to 100% by adding more validators and requiring the process to take longer).

Every time the time reaches some multiple of 4096 seconds, we run the latency-dependent algorithm, choosing 512 random nodes to participate in the algorithm. A valid proposal is any valid chain of values that were finalized by the threshold-dependent algorithm. If a node sees some finalized value before time $T + k \cdot D$ ($D = 8$ seconds) with k signatures, it accepts the chain into its set of known chains and rebroadcasts it with its own signature added; observers use a threshold of $T + (k - 0.5) \cdot D$ as before.

The "choice" function used at the end is simple:

- Finalized values that are not descendants of what was already agreed to be a finalized value in the previous round are ignored
- Finalized values that are invalid are ignored
- To choose between two valid finalized values, pick the one with the lower hash

If 5% of validators are honest, there is only a roughly 1 in 1 trillion chance that none of the 512 randomly selected nodes will be honest, and so as long as the network latency plus clock disparity is less than $\frac{D}{2}$ the above algorithm will work, correctly coordinating nodes on some single finalized value, even if multiple conflicting finalized values are presented because the fault tolerance of the threshold-dependent algorithm is broken.

If the fault tolerance of the threshold-dependent consensus algorithm is met (usually 50% or 67% honest), then the threshold-dependent consensus algorithm will either not finalize any new checkpoints, or it will finalize new checkpoints that are compatible with each other (eg. a series of checkpoints where each points to the previous as a parent), so even if network latency exceeds $\frac{D}{2}$ (or even D), and as a result nodes participating in the latency-dependent algorithm disagree on which value they accept, the values they accept are still guaranteed to be part of the same chain and so there is no actual disagreement. Once latency recovers back to normal in some future round, the latency-dependent consensus will get back "in sync".

If the assumptions of both the threshold-dependent and latency-dependent consensus algorithms are broken *at the same time* (or in consecutive rounds), then the algorithm can break down. For example, suppose in one round, the threshold-dependent consensus finalizes $Z \rightarrow Y \rightarrow X$ and the latency-dependent consensus disagrees between Y and X , and in the next round the threshold-dependent consensus finalizes a descendant W of X which is *not* a descendant of Y ; in the latency-dependent consensus, the nodes who agreed Y will not accept W , but the nodes that agreed X will. However, this is unavoidable; the impossibility of safe-under-asynchrony consensus with more than $\frac{1}{3}$ fault tolerance is a well known result

(<https://groups.csail.mit.edu/tds/papers/Lynch/jacm88.pdf>) in Byzantine fault tolerance theory, as is the impossibility of more than $\frac{1}{2}$ fault tolerance even allowing synchrony assumptions but assuming offline observers.

Layer 1 Should Be Innovative in the Short Term but Less in the Long Term

2018 Aug 26

[See all posts \(/\)](#)

See update 2018-08-29

One of the key tradeoffs in blockchain design is whether to build more functionality into base-layer blockchains themselves ("layer 1"), or to build it into protocols that live on top of the blockchain, and can be created and modified without changing the blockchain itself ("layer 2"). The tradeoff has so far shown itself most in the scaling debates, with block size increases (and sharding

(<https://github.com/ethereum/wiki/wiki/Sharding-FAQ>) on one side and layer-2 solutions like Plasma and channels on the other, and to some extent blockchain governance, with loss and theft recovery being solvable by either the DAO fork (<https://qz.com/730004/everything-you-need-to-know-about-the-ethereum-hard-fork/>), or generalizations thereof such as EIP 867 (<https://github.com/ethereum/EIPs/blob/master/EIPS/eip-867.md>), or by layer-2 solutions such as Reversible Ether (RETH).

(https://www.reddit.com/r/MakerDAO/comments/8fmks1/introducing_reversible_eth_reth_never_send_ether/). So which approach is ultimately better? Those who know me well, or have seen me out myself as a dirty centrist (<https://twitter.com/VitalikButerin/status/1032589339367231488>), know that I will inevitably say "some of both". However, in the longer term, I do think that as blockchains become more and more mature, layer 1 will necessarily stabilize, and layer 2 will take on more and more of the burden of ongoing innovation and change.

There are several reasons why. The first is that layer 1 solutions require ongoing protocol change to happen at the base protocol layer, base layer protocol change requires governance, and **it has still not been shown that, in the long term, highly "activist" blockchain governance can continue without causing ongoing political uncertainty or collapsing into centralization.**

To take an example from another sphere, consider Moxie Marlinspike's defense of Signal's centralized and non-federated nature (<https://signal.org/blog/the-ecosystem-is-moving/>). A document by a company defending its right to maintain control over an ecosystem it depends on for its key business should of course be viewed with massive grains of salt, but one can still benefit from the arguments. Quoting:

One of the controversial things we did with Signal early on was to build it as an unfederated service. Nothing about any of the protocols we've developed requires centralization; it's entirely possible to build a federated Signal Protocol-based messenger, but I no longer believe that it is possible to build a competitive federated messenger at all.

And:

Their retort was "that's dumb, how far would the internet have gotten without interoperable protocols defined by 3rd parties?" I thought about it. We got to the first production version of IP, and have been trying for the past 20 years to switch to a second production version of IP with limited success. We got to HTTP version 1.1 in 1997, and have been stuck there until now. Likewise, SMTP, IRC, DNS, XMPP, are all similarly frozen in time circa the late 1990s. To answer his question, that's how far the internet got. It got to the late 90s.

That has taken us pretty far, but it's undeniable that once you federate your protocol, it becomes very difficult to make changes. And right now, at the application level, things that stand still don't fare very well in a world where the ecosystem is moving ... So long as federation means stasis while centralization means movement, federated protocols are going to have trouble existing in a software climate that demands movement as it does today.

At this point in time, and in the medium term going forward, it seems clear that decentralized application platforms, cryptocurrency payments, identity systems, reputation systems, decentralized exchange mechanisms, auctions, privacy solutions, programming languages that support privacy solutions, and most other interesting things that can be done on blockchains are spheres where there will continue to be significant and ongoing innovation. Decentralized application platforms often need continued reductions in confirmation time, payments need fast confirmations, low transaction costs, privacy, and many other built-in features, exchanges are appearing in many shapes and sizes including on-chain automated market makers (<https://uniswap.io/>), frequent batch auctions (https://www.cftc.gov/sites/default/files/idc/groups/public/@newsroom/documents/file/tac021014_budish.pdf), combinatorial auctions (<http://cramton.umd.edu/ca-book/cramton-shoham-steinberg-combinatorial-auctions.pdf>) and more. Hence, "building in" any of these into a base layer blockchain would be a bad idea, as it would create a high level of governance overhead as the platform would have to continually discuss, implement and coordinate newly discovered technical improvements. For the same reason federated messengers have a hard time getting off the ground without re-centralizing, blockchains would also need to choose between adopting activist governance, with the perils that entails, and falling behind newly appearing alternatives.

Even Ethereum's limited level of application-specific functionality, precompiles, has seen some of this effect. Less than a year ago, Ethereum adopted the Byzantium hard fork, including operations to facilitate elliptic curve (<https://github.com/ethereum/EIPs/blob/master/EIPS/eip-196.md>), operations (<https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md>) needed for ring signatures, ZK-SNARKs and other applications, using the alt-bn128 (<https://github.com/topics/alt-bn128>) curve. Now, Zcash and other blockchains are moving toward BLS-12-381 (<https://blog.z.cash/new-snark-curve/>), and Ethereum would need to fork again to catch up. In part to avoid having similar problems in the future, the Ethereum community is looking to upgrade the EVM to E-WASM (<https://github.com/ewasm/design>), a virtual machine that is sufficiently more efficient that there is far less need to incorporate application-specific precompiles.

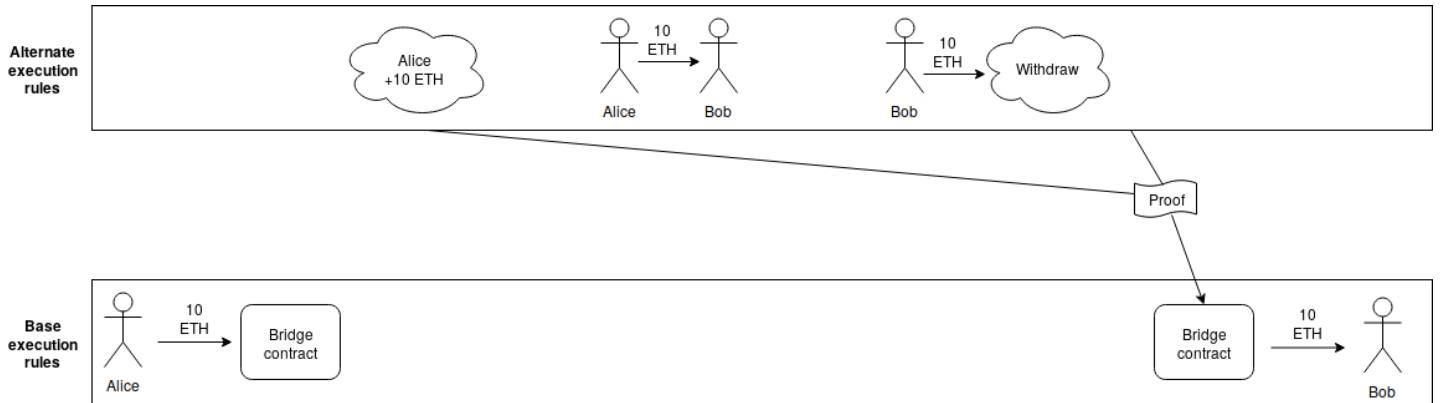
But there is also a second argument in favor of layer 2 solutions, one that does not depend on speed of anticipated technical development: *sometimes there are inevitable tradeoffs, with no single globally optimal solution*. This is less easily visible in Ethereum 1.0-style blockchains, where there are certain models that are reasonably universal (eg. Ethereum's account-based model is one). In sharded blockchains, however, one type of question that does *not* exist in Ethereum today crops up: how to do cross-shard transactions? That is, suppose that the blockchain state has regions A and B, where few or no nodes are processing both A and B. How does the system handle transactions that affect both A and B?

The [current answer](https://github.com/ethereum/wiki/wiki/Sharding-FAQs#how-can-we-facilitate-cross-shard-communication) (<https://github.com/ethereum/wiki/wiki/Sharding-FAQs#how-can-we-facilitate-cross-shard-communication>) involves asynchronous cross-shard communication, which is sufficient for transferring assets and some other applications, but insufficient for many others. Synchronous operations (eg. to solve the [train and hotel problem](https://github.com/ethereum/wiki/wiki/Sharding-FAQs#what-is-the-train-and-hotel-problem) (<https://github.com/ethereum/wiki/wiki/Sharding-FAQs#what-is-the-train-and-hotel-problem>)) can be bolted on top with [cross-shard yanking](https://ethresear.ch/t/cross-shard-contract-yanking/1450) (<https://ethresear.ch/t/cross-shard-contract-yanking/1450>), but this requires multiple rounds of cross-shard interaction, leading to significant delays. We can solve these problems with a [synchronous execution scheme](https://ethresear.ch/t/simple-synchronous-cross-shard-transaction-protocol/3097) (<https://ethresear.ch/t/simple-synchronous-cross-shard-transaction-protocol/3097>), but this comes with several tradeoffs:

- The system cannot process more than one transaction for the same account per block
- Transactions must declare in advance what shards and addresses they affect
- There is a high risk of any given transaction failing (and still being required to pay fees!) if the transaction is only accepted in some of the shards that it affects but not others

It seems very likely that a better scheme can be developed, but it would be more complex, and may well have limitations that this scheme does not. There are known results preventing perfection; at the very least, [Amdahl's law](https://en.wikipedia.org/wiki/Amdahl%27s_law) (https://en.wikipedia.org/wiki/Amdahl%27s_law) puts a hard limit on the ability of some applications and some types of interaction to process more transactions per second through parallelization.

So how do we create an environment where better schemes can be tested and deployed? The answer is an idea that can be credited to Justin Drake: layer 2 execution engines. Users would be able to send assets into a "bridge contract", which would calculate (using some indirect technique such as [interactive verification](https://truebit.io/) (<https://truebit.io/>), or [ZK-SNARKs](https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6) (<https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6>)) state roots using some alternative set of rules for processing the blockchain (think of this as equivalent to layer-two "meta-protocols" like [Mastercoin/OMNI](https://blog.omni.foundation/2013/11/29/a-brief-history-of-mastercoin/) (<https://blog.omni.foundation/2013/11/29/a-brief-history-of-mastercoin/>), and [Counterparty](https://counterparty.io/) (<https://counterparty.io/>), on top of Bitcoin, except because of the bridge contract these protocols would be able to handle assets whose "base ledger" is defined on the underlying protocol), and which would process withdrawals if and only if the alternative ruleset generates a withdrawal request.



Note that anyone can create a layer 2 execution engine at any time, different users can use different execution engines, and one can switch from one execution engine to any other, or to the base protocol, fairly quickly. The base blockchain no longer has to worry about being an optimal smart contract processing engine; it need only be a data availability layer with execution rules that are quasi-Turing-complete so that any layer 2 bridge contract can be built on top, and that allow basic operations to carry state between shards (in fact, only ETH transfers being fungible across shards is sufficient, but it takes very little effort to also allow cross-shard calls,

so we may as well support them), but does not require complexity beyond that. Note also that layer 2 execution engines can have different state management rules than layer 1, eg. not having storage rent; anything goes, as it's the responsibility of the users of that specific execution engine to make sure that it is sustainable, and if they fail to do so the consequences are contained to within the users of that particular execution engine.

In the long run, layer 1 would not be actively competing on all of these improvements; it would simply provide a stable platform for the layer 2 innovation to happen on top. **Does this mean that, say, sharding is a bad idea, and we should keep the blockchain size and state small so that even 10 year old computers can process everyone's transactions? Absolutely not.** Even if execution engines are something that gets partially or fully moved to layer 2, consensus on data ordering and availability is still a highly generalizable and necessary function; to see how difficult layer 2 execution engines are without layer 1 scalable data availability consensus, [see \(<https://ethresear.ch/t/minimal-viable-plasma/426>\)](https://ethresear.ch/t/minimal-viable-plasma/426), the [difficulties \(<https://ethresear.ch/t/plasma-cash-plasma-with-much-less-per-user-data-checking/1298>\)](https://ethresear.ch/t/plasma-cash-plasma-with-much-less-per-user-data-checking/1298) in [Plasma \(<https://ethresear.ch/t/plasma-debit-arbitrary-denomination-payments-in-plasma-cash/2198>\)](https://ethresear.ch/t/plasma-debit-arbitrary-denomination-payments-in-plasma-cash/2198), research, and its [difficulty \(<https://medium.com/@kelvinfichter/why-is-evm-on-plasma-hard-bf2d99c48df7>\)](https://medium.com/@kelvinfichter/why-is-evm-on-plasma-hard-bf2d99c48df7) of naturally extending to fully general purpose blockchains, for an example. And if people want to throw a hundred megabytes per second of data into a system where they need consensus on availability, then we need a hundred megabytes per second of data availability consensus.

Additionally, layer 1 can still improve on reducing latency; if layer 1 is slow, the only strategy for achieving very low latency is [state channels \(<https://medium.com/statechannels/counterfactual-generalized-state-channels-on-ethereum-d38a36d25fc6>\)](https://medium.com/statechannels/counterfactual-generalized-state-channels-on-ethereum-d38a36d25fc6), which often have high capital requirements and can be difficult to generalize. State channels will always beat layer 1 blockchains in latency as state channels require only a single network message, but in those cases where state channels do not work well, layer 1 blockchains can still come closer than they do today.

Hence, the other extreme position, that blockchain base layers can be truly absolutely minimal, and not bother with either a quasi-Turing-complete execution engine or scalability to beyond the capacity of a single node, is also clearly false; there is a certain minimal level of complexity that is required for base layers to be powerful enough for applications to build on top of them, and we have not yet reached that level. Additional complexity is needed, though it should be chosen very carefully to make sure that it is maximally general purpose, and not targeted toward specific applications or technologies that will go out of fashion in two years due to loss of interest or better alternatives.

And even in the future base layers will need to continue to make some upgrades, especially if new technologies (eg. STARKs reaching higher levels of maturity) allow them to achieve stronger properties than they could before, though developers today can take care to make base layer platforms maximally forward-compatible with such potential improvements. So it will continue to be true that a balance between layer 1 and layer 2 improvements is needed to continue improving scalability, privacy and versatility, though layer 2 will continue to take up a larger and larger share of the innovation over time.

Update 2018.08.29: Justin Drake pointed out to me another good reason why some features may be best implemented on layer 1: those features are public goods, and so could not be efficiently or reliably funded with feature-specific use fees, and hence are best paid for by subsidies paid out of issuance or burned transaction fees. One possible example of this is secure random number generation, and another is generation of zero knowledge proofs for more efficient client validation of correctness of various claims about blockchain contents or state.

[Mirror] Central Planning as Overfitting

2018 Nov 25

[See all posts \(/\)](#)

This is a mirror of the post at <https://radicalxchange.org/blog/posts/2018-11-26-4m9b8b/> written by myself and Glen Weyl

There is an intuition shared by many that "central planning" — command-and-control techniques for allocating resources in economies, and fine-grained knob-turning interventionism more generally — is undesirable. There's quite a lot to this, but it is often misapplied in a way that also leads it to go under-appreciated. In this post we try to clarify the appropriate scope of the intuition.

Some recent examples of the intuition being misapplied are:

- People arguing that relatively simple entitlement programs like Social Security are burdensome government intervention, while elaborate and often discretionary tax breaks conditional on specific behaviors are a good step towards less government.
- People arguing that block size limits in cryptocurrencies, which impose a hard cap on the number of transactions that each block can contain, are a form of central planning, but who do not argue against other centrally planned parameters, eg. the targeted ten minute (or whatever) average time interval between blocks.
- People arguing that a lack of block size limits constitutes central planning (<https://medium.com/@rextar4444/the-block-chain-keynesian-why-pushing-to-scale-bitcoin-to-be-a-coffee-money-is-keynesian-central-c5bdf32a5e4d>) (!!)
- People arguing that fixed transaction fees constitute central planning (http://reddit.com/r/Monero/comments/9pck22/the_marvel_of_bulletproofs_is_the_small_tx_size/), but variable transaction fees that arise out of an equilibrium itself created by a fixed block size limit do not.
- We were recently at a discussion in policy circles in Washington, where one of us was arguing for a scheme based on Harberger taxes (<https://medium.com/@simondlr/what-is-harberger-tax-where-does-the-blockchain-fit-in-1329046922c6>) for spectrum licenses, debating against someone defending more conventional perpetual monopoly licenses on spectrum aggregated at large levels that would tend to create a few national cellular carriers. The latter side argued that the Harberger tax scheme constituted unacceptable bureaucratic interventionism, but seemed to believe that permanent government-issued monopoly privileges are a property right as natural as apples falling from a tree.

While we do not entirely dismiss this last example, for reasons we will return to later, it does seem overplayed. Similarly and conversely, we see many examples where, in the name of defending or promoting "markets" (or at least "economic rationality") many professional economists advocate schemes that seem much more to us like central planning than the systems they seek to replace:

- The most systematic example of this is the literature on "optimal mechanism design" (https://en.wikipedia.org/wiki/Mechanism_design), which began with the already extremely complicated and fragile Vickrey-Clarke-Groves mechanism (https://en.wikipedia.org/wiki/Vickrey%E2%80%93Clarke%E2%80%93Groves_mechanism), and has only gotten more

byzantine from there. While Vickrey's motivation for these ideas was to discover relatively simple rules that would correct the flaws of standard capitalism, he acknowledged (<https://www.cs.princeton.edu/courses/archive/spr09/cos444/papers/vickrey61.pdf>) in his paper that the design was highly complex in its direct application and urged future researchers to find simplifications. Instead of following this counsel, many scholars (<https://onlinelibrary.wiley.com/doi/abs/10.3982/ECTA6995>) have proposed, for example, schemes that rely on a central authority being able to specify an infinite dimensional set of prior beliefs. These schemes, we submit, constitute "central planning" in precisely the sense we should be concerned with.

- Furthermore, these designs are not just matters of theory, but in practice many applied mechanism designers have created systems with similar properties. The recent United States Spectrum Incentive auctions (designed by a few prominent economists and computer scientists) centralized the enforcement of potential conflicts between transmission rights using an extremely elaborate and opaque computational engine (<http://www.pnas.org/content/114/28/7202.short>), rather than allowing conflicts to be resolved through (for example) common law liability lawsuits as other interference between property claims and land uses are. A recent design for the allocation of courses to students at the University of Pennsylvania designed by a similar team requires students to express their preferences over courses on a novel numerical scale, allowing them only narrow language for expressing complementarities and substitutability between courses and then uses a state-of-the-art optimization engine to allocate courses. Auction systems designed by economists and computer scientists at large technology companies, like Facebook and Google, are even richer and less transparent, and have created substantial backlash (<https://www.dropbox.com/s/cc219w090xr365i/known%20unknowns%20search%20talk.pptx?dl=0>), inspiring a whole industry of firms that help advertisers optimize their bidding in elaborate ways against these systems.
- This problem does not merely arise in mechanism design, however. In the fields of industrial organization (the basis of much antitrust economics) and the field of macroeconomics (the basis of much monetary policy), extremely elaborate models with hundreds of parameters are empirically estimated and used to simulate the effects of mergers or changes in monetary policy. These models are usually difficult to explain even to experts in the field, much less democratically-elected politicians, judges or, god forbid, voters. And yet the confidence we have in these models, the empirical evidence validating their accuracy, etc. is almost nothing. Nonetheless, economists consistently promote such methods as "the state of the art" and they are generally viewed positively by defenders of the "market economy".

To understand why we think the concept of "intervention" is being misapplied here, we need to understand two different ways of measuring the extent to which some scheme is "interventionist". The first approach is to try to measure the absolute magnitude of distortion relative to some imagined state of nature (anarcho-primitivism, or a blockchain with no block size limits, or...). However, this approach clearly fails to capture the intuitions of why central planning is undesirable. For example, property rights in the physical world are a large intervention into almost every person's behavior, considerably limiting the actions that we can take every day. Many of these restrictions are actually of quite recent historical provenance (beginning with agriculture, and mostly in the West and not the East or Middle East). However, opponents of central planning often tend to be the strongest proponents of property rights!

We can shed some light on this puzzle by looking at another way of measuring the "central-plannyness" of some social structure: in short, measure the number of knobs. Property rights actually score quite well here: every piece of property is allocated to some person or legal entity, they can use it as they wish, and no one else can touch it without their permission. There are choices to make around the edges (eg. adverse possession rights (https://en.wikipedia.org/wiki/Adverse_possession)), but generally there isn't too much room for changing the scheme around (though note that privatization schemes, ie. transitions from something other than property rights to property rights like the auctions we discussed above, have very many knobs, and so there we can see more risks). Command-and-control regulations with ten thousand clauses (or market

designs that specify elaborate probabilistic objects, or optimization protocols, etc.), or attempts to limit use of specific features of the blockchain (<https://github.com/bitcoin/bitcoin/pull/3737>) to drive out specific categories of users, are much less desirable, as such strategies leave many more choices to central planners. A block size limit and a fixed transaction fee (or carbon taxes and a cap-and-trade scheme) have the exact same level of "central-plannyness" to them: one variable (either quantity or price) is fixed in the protocol, and the other variable is left to the market.

Here are some key underlying reasons why we believe that simple social systems with fewer knobs are so desirable:

- They have **fewer moving parts that can fail** or otherwise have unexpected effects.
- They are **less likely to overfit**. If a social system is too complex, there are many parameters to set and relatively few experiences to draw from when setting the parameters, making it more likely that the parameters will be set to overfit to one context, and generalize poorly to other places or to future times. We know little (<https://www.washingtonpost.com/news/volokh-conspiracy/wp/2014/10/13/f-a-hayek-and-the-pretense-of-knowledge>), and we should not design systems that demand us to know a lot.
- They are **more resistant to corruption**. If a social system is simpler, it is more difficult (not impossible, but still more difficult) to set parameters in ways that encode attempts to privilege or anti-privilege specific individuals, factions or communities. This is not only good because it leads to fairness, it is also good because it leads to less zero-sum conflict.
- They can **more easily achieve legitimacy**. Because simpler systems are easier to understand, and easier for people to understand that a given implementation is not unfairly privileging special interests, it is easier to create common knowledge that the system is fair, creating a stronger sense of trust. Legitimacy is perhaps the central virtue of social institutions, as it sustains cooperation and coordination, enables the possibility of democracy (how can you democratically participate and endorse a system you do not understand?) and allows for a bottoms-up, rather than top-down, creation of a such a system, ensuring it can be implemented without much coercion or violence.

These effects are not always achieved (for example, even if a system has very few knobs, it's often the case that there exists a knob that can be turned to privilege well-connected and wealthy people as a class over everyone else), but the simpler a system is, the more likely the effects are to be achieved.

While avoiding over-complexity and overfit in personal decision-making is also important, avoiding these issues in large-scale social systems is even more important, because of the inevitable possibility of powerful forces attempting to manipulate knobs for the benefit of special interests, and the need to achieve common knowledge that the system has not been greatly corrupted, to the point where the fairness of the system is obvious even to unsophisticated observers.

This is not to condemn all forms or uses of complexity in social systems. Most science and the inner workings of many technical systems are likely to be opaque to the public but this does not mean science or technology is useless in social life; far from it. However, these systems, to gain legitimacy, usually show that they can reliably achieve some goal, which is transparent and verifiable. Planes land safely and on time, computational systems seem to deliver calculations that are correct, etc. It is by this process of verification, rather than by the transparency of the system per se, that such systems gain their legitimacy. However, for many social systems, truly large-scale, repeatable tests are difficult if not impossible. As such, simplicity is usually critical to legitimacy.

Different Notions of Simplicity

However, there is one class of social systems that seem to be desirable, and that intellectual advocates of minimizing central planning tend to agree are desirable, that don't quite fit the simple "few knobs" characterization that we made above. For example, consider common law. [Common law](https://en.wikipedia.org/wiki/Common_law) (https://en.wikipedia.org/wiki/Common_law) is built up over thousands of precedents, and contains a large number of concepts (eg. see this list under "property law", itself only a part of common law; have you heard of "quicquid plantatur solo, solo cedit" before?). However, proponents of private property are very frequently proponents of common law. So what gives?

Here, we need to make a distinction between redundant complexity, or many knobs that really all serve a relatively small number of similar goals, and optimizing complexity, in the extreme one knob per problem that the system has encountered. In computational complexity theory, we typically talk about [Kolmogorov complexity](https://en.wikipedia.org/wiki/Kolmogorov_complexity) (https://en.wikipedia.org/wiki/Kolmogorov_complexity), but there are other notions of complexity that are also useful here, particularly [VC dimension](https://en.wikipedia.org/wiki/VC_dimension) (https://en.wikipedia.org/wiki/VC_dimension) - roughly, the size of the largest set of situations for which we can turn the knobs in a particular way to achieve any particular set of outcomes. Many successful machine learning techniques, such as [Support Vector Machines](https://en.wikipedia.org/wiki/Support_vector_machine) (https://en.wikipedia.org/wiki/Support_vector_machine) and [Boosting](https://en.wikipedia.org/wiki/Boosting_(machine_learning)) ([https://en.wikipedia.org/wiki/Boosting_\(machine_learning\)](https://en.wikipedia.org/wiki/Boosting_(machine_learning))), are quite complex, both in the formal Kolmogorov sense and in terms of the outcomes they produce, but can be proven to have low VC dimension.

VC dimension does a nice job capturing some of the arguments for simplicity mentioned above more explicitly, for example:

- A system with low VC dimension may have some moving parts that fail, but if it does, its different constituent parts can correct for each other. By construction, it has built in resilience through redundancy
- Low VC dimension is literally a measure of resistance to overfit.
- Low VC dimension leads to resistance to corruption, because if VC dimension is low, a corrupt or self-interested party in control of some knobs will not as easily be able to achieve some particular outcome that they desire. In particular, this agent will be "checked and balanced" by other parts of the system that redundantly achieve the originally desired ends.
- They can achieve legitimacy because people can randomly check a few parts and verify in detail that those parts work in ways that are reasonable, and assume that the rest of the system works in a similar way. An example of this was the ratification of the United States Constitution which, while quite elaborate, was primarily elaborate in the redundancy with which it applied the principle of checks and balances of power. Thus most citizens only read one or a few of [The Federalist Papers](https://en.wikipedia.org/wiki/The_Federalist_Papers) (https://en.wikipedia.org/wiki/The_Federalist_Papers) that explained and defended the Constitution, and yet got a reasonable sense for what was going on.

This is not as clean and convenient as a system with low Kolmogorov complexity, but still much better than a system with high complexity where the complexity is "optimizing" (for an example of this in the blockchain context, see [Vitalik's opposition and alternative to on-chain governance](https://vitalik.ca/general/2017/12/17/voting.html) (<https://vitalik.ca/general/2017/12/17/voting.html>)). The primary disadvantage we see in Kolmogorov complex but VC simple designs is for new social institutions, where it may be hard to persuade the public that these are VC simple. VC simplicity is usually easier as a basis for legitimacy when an institution has clearly been built up without any clear design over a long period of time or by a large committee of people with conflicting interests (as with the United States Constitution). Thus when offering innovations we tend to focus more on Kolmogorov simplicity and hope many redundant each Kolmogorov-simple elements will add up to a VC-simple system. However, we may just not have the imagination to think of how VC simplicity might be effectively explained.

There are forms of the "avoid central planning" intuition that are misunderstandings and ultimately counterproductive. For example, try to automatically seize upon designs that seem at first glance to "look like a market", because not all markets are created equal. For example, one of us has argued for using fixed prices in certain settings to reduce uncertainty (<https://ethresear.ch/t/first-and-second-price-auctions-and-improved-transaction-fee-markets/2410>), and the other has (for similar information sharing reasons) argued for auctions that are a synthesis of standard descending price Dutch and ascending price English auctions (Channel auctions). That said, it is also equally a large error to throw the intuition away entirely. Rather, it is a valuable and important insight that can easily be central to the methodology we have been recently trying to develop. Simplicity to Whom? Or Why Humanities Matter

However, the academic critics of this type of work are not simply confused. There is a reasonable basis for unease with discussions of "simplicity" because they inevitably contain a degree of subjectivity. What is "simple" to describe or appears to have few knobs in one language for describing it is devilishly complex in another, and vice versa. A few examples should help illuminate the point:

- We have repeatedly referred to "knobs", which are roughly real valued parameters. But real-valued parameters can encode an arbitrary amount of complexity. For example, I could claim my system has only one knob, it is just that slight changes in the 1000th decimal place of the setting of that knob end up determining incredibly important properties of the system. This may seem cheap, but more broadly it is the case that non-linear mappings between systems can make one system seem "simple" and another "complex" and in general there is just no way to say which is right.
- Many think of the electoral system of the United States (https://en.wikipedia.org/wiki/Elections_in_the_United_States#Levels_of_election) as "simple", and yet, if one reflects on it or tries to explain it to a foreigner, it is almost impossible to describe. It is familiar, not simple, and we just have given a label to it ("the American system") that lets us refer to it in a few words. Systems like Quadratic Voting, or ranked choice voting ([https://ballotpedia.org/Ranked-choice_voting_\(RCV\)](https://ballotpedia.org/Ranked-choice_voting_(RCV))), are often described as complex, but this seems to have more to do with lack of familiarity than complexity.
- Many scientific concepts, such as the "light cone", are the simplest thing possible once one understands special relativity and yet are utterly foreign and bizarre without having wrapped one's hands around this theory.

Even Kolmogorov complexity (https://en.wikipedia.org/wiki/Kolmogorov_complexity). (length of the shortest computer program that encodes some given system) is relative to some programming language. Now, to some extent, VC dimension offers a solution: it says that a class of systems is simple if it is not too flexible. But consider what happens when you try to apply this; to do so, let's return to our example upfront about Harberger taxes v. perpetual licenses for spectrum.

Harberger taxes strike us as quite simple: there is a single tax rate (and the theory even says this is tied down by the rate at which assets turnover, at least if we want to maximally favor allocative efficiency) and the system can be described in a sentence or two. It seems pretty clear that such a system could not be contorted to achieve arbitrary ends. However, an opponent could claim that we chose the Harberger tax from an array of millions of possible mechanisms of a similar class to achieve a specific objective, and it just sounds simple (as with our examples of "deceptive" simplicity above).

To counter this argument, we would respond that the Harberger tax, or very similar ideas, have been repeatedly discovered or used (to some success) throughout human history, beginning with the Greeks (<https://theweek.com/articles/703720/genius-way-ancient-greeks-taxed-citizens>), and that we do not propose this system simply for spectrum licenses but in a wide range of contexts. The chances that in all these contexts we

are cherry-picking the system to "fit" that setting seems low. We would submit to the critic to judge whether it is really plausible that all these historical circumstances and these wide range of applications just "happen" to coincide.

Focusing on familiarity (ie. conservatism), rather than simplicity in some abstract mathematical sense, also carries many of the benefits of simplicity as we described above; after all, familiarity is simplicity, if the language we are using to describe ideas includes references to our shared historical experience. Familiar mechanisms also have the benefit that we have more knowledge of how similar ideas historically worked in practice. So why not just be conservative, and favor perpetual property licenses strongly over Harberger taxes?

There are three flaws in that logic, it seems to us. First, to the extent it is applied, it should be applied uniformly to all innovation, not merely to new social institutions. Technologies like the internet have contributed greatly to human progress, but have also led to significant social upheavals; this is not a reason to stop trying to advance our technologies and systems for communication, and it is not a reason to stop trying to advance our social technologies for allocating scarce resources.

Second, the benefits of innovation are real, and social institutions stand to benefit from growing human intellectual progress as much as everything else. The theoretical case for Harberger taxes providing efficiency benefits is strong, and there is great social value in doing small and medium-scale experiments to try ideas like them out. Investing in experiments today increases what we know, and so increases the scope of what can be done "conservatively" tomorrow.

Third, and most importantly, the cultural context in which you as a decision maker have grown up today is far from the only culture that has existed on earth. Even at present, Singapore, China, Taiwan and Scandinavia have had significant success with quite different property regimes than the United States. Video game developers and internet protocol designers have had to solve incentive problems of a similar character to what we see today in the blockchain space and have come up with many kinds of solutions, and throughout history, we have seen a wide variety of social systems used for different purposes, with a wide range of resulting outcomes. By learning about the different ways in which societies have lived, understood what is natural and imagined their politics, we can gain the benefits of learning from historical experience and yet at the same time open ourselves to a much broader space of possible ideas to work with.

This is why we believe that balance and collaboration between different modes of learning and understanding, both the mathematical one of economists and computer scientists, and the historical experiences studied by historians, anthropologists, political scientists, etc is critical to avoid the mix of and often veering between extreme conservatism and dangerous utopianism that has become characteristic of much intellectual discourse in e.g. the economics community, the "rationalist" community, and in many (<http://www.truthcoin.info/blog/against-the-hard-fork/>), cases (<https://vitalik.ca/general/2018/03/28/plutocracy.html>), blockchain protocol design.

A CBC Casper Tutorial

2018 Dec 05

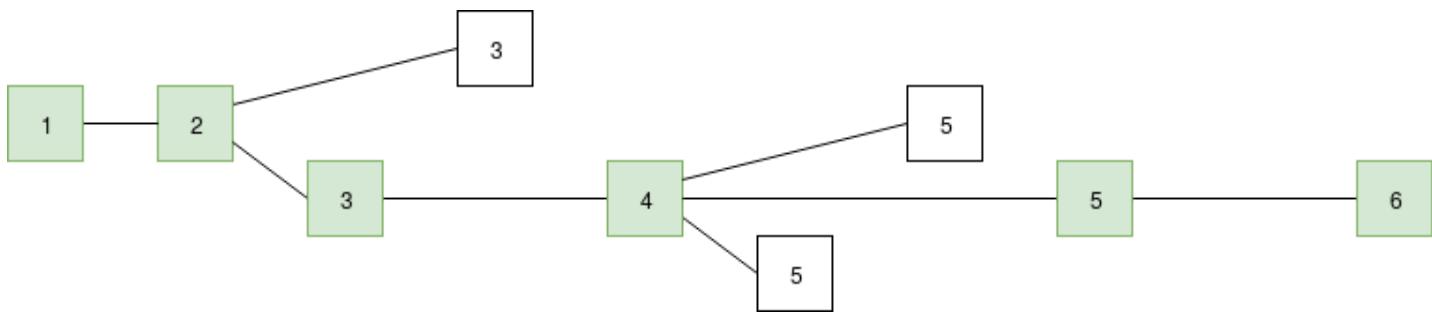
[See all posts \(/\)](#)

Special thanks to Vlad Zamfir, Aditya Asgaonkar, Ameen Soleimani and Jinglan Wang for review

In order to help more people understand "the other Casper" (Vlad Zamfir's CBC Casper), and specifically the instantiation that works best for blockchain protocols, I thought that I would write an explainer on it myself, from a less abstract and more "close to concrete usage" point of view. Vlad's descriptions of CBC Casper can be found [here](https://www.youtube.com/watch?v=GNGbd_RbrzE) (https://www.youtube.com/watch?v=GNGbd_RbrzE) and [here](https://github.com/ethereum/cbc-casper/wiki/FAQ) (<https://github.com/ethereum/cbc-casper/wiki/FAQ>), and [here](https://github.com/cbc-casper/cbc-casper-paper) (<https://github.com/cbc-casper/cbc-casper-paper>); you are welcome and encouraged to look through these materials as well.

CBC Casper is designed to be fundamentally very versatile and abstract, and come to consensus on pretty much any data structure; you can use CBC to decide whether to choose 0 or 1, you can make a simple block-by-block chain run on top of CBC, or a 2^{92} -dimensional hypercube tangle DAG, and pretty much anything in between.

But for simplicity, we will first focus our attention on one concrete case: a simple chain-based structure. We will suppose that there is a fixed validator set consisting of N validators (a fancy word for "staking nodes"; we also assume that each node is staking the same amount of coins, cases where this is not true can be simulated by assigning some nodes multiple validator IDs), time is broken up into ten-second slots, and validator k can create a block in slot k , $N + k$, $2N + k$, etc. Each block points to one specific parent block. Clearly, if we wanted to make something maximally simple, we could just take this structure, impose a longest chain rule on top of it, and call it a day.

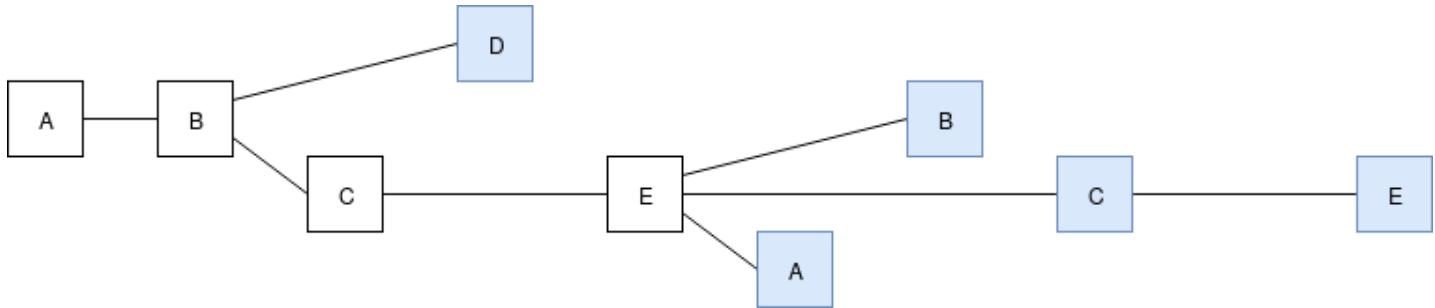


The green chain is the longest chain (length 6) so it is considered to be the "canonical chain".

However, what we care about here is adding some notion of "finality" - the idea that some block can be so firmly established in the chain that it cannot be overtaken by a competing block unless a very large portion (eg. $\frac{1}{4}$) of validators commit a *uniquely attributable fault* - act in some way which is clearly and cryptographically verifiably malicious. If a very large portion of validators do act maliciously to revert the block, proof of the misbehavior can be submitted to the chain to take away those validators' entire deposits, making the reversion of finality extremely expensive (think hundreds of millions of dollars).

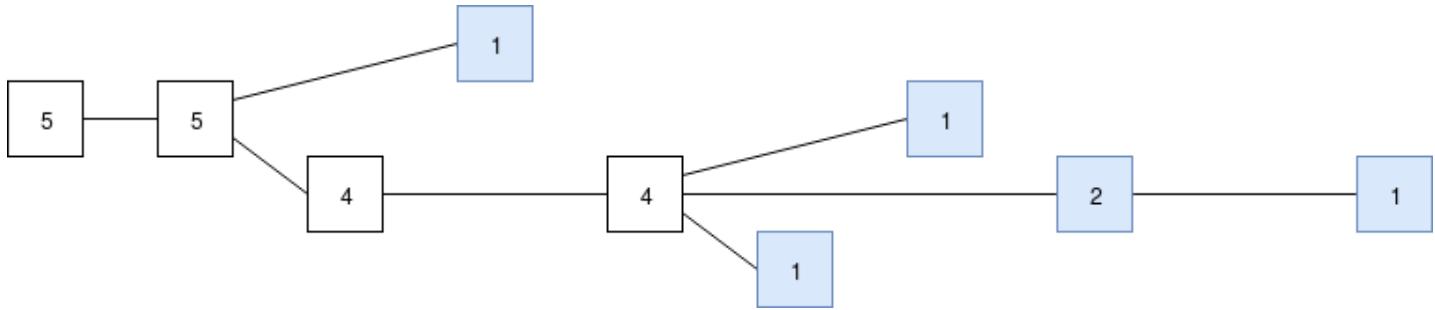
LMD GHOST

We will take this one step at a time. First, we replace the fork choice rule (the rule that chooses which chain among many possible choices is "the canonical chain", ie. the chain that users should care about), moving away from the simple longest-chain-rule and instead using "latest message driven GHOST". To show how LMD GHOST works, we will modify the above example. To make it more concrete, suppose the validator set has size 5, which we label A, B, C, D, E, so validator A makes the blocks at slots 0 and 5, validator B at slots 1 and 6, etc. A client evaluating the LMD GHOST fork choice rule cares only about the most recent (ie. highest-slot) message (ie. block) signed by each validator:

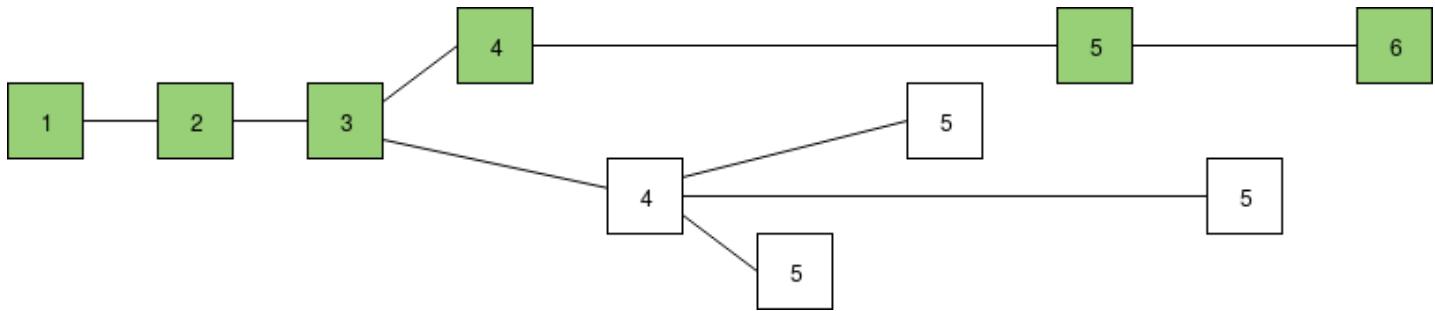


Latest messages in blue, slots from left to right (eg. A's block on the left is at slot 0, etc.)

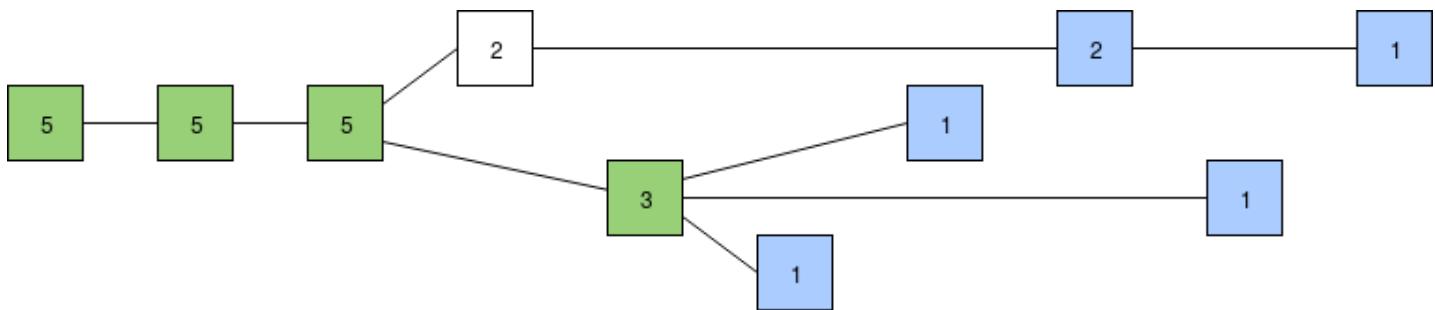
Now, we will use only these messages as source data for the "greedy heaviest observed subtree" (GHOST) fork choice rule: start at the genesis block, then each time there is a fork choose the side where more of the latest messages support that block's subtree (ie. more of the latest messages support either that block or one of its descendants), and keep doing this until you reach a block with no children. We can compute for each block the subset of latest messages that support either the block or one of its descendants:



Now, to compute the head, we start at the beginning, and then at each fork pick the higher number: first, pick the bottom chain as it has 4 latest messages supporting it versus 1 for the single-block top chain, then at the next fork support the middle chain. The result is the same longest chain as before. Indeed, in a well-running network (ie. the orphan rate is low), almost all of the time LMD GHOST and the longest chain rule *will* give the exact same answer. But in more extreme circumstances, this is not always true. For example, consider the following chain, with a more substantial three-block fork:



Scoring blocks by chain length. If we follow the longest chain rule, the top chain is longer, so the top chain wins.

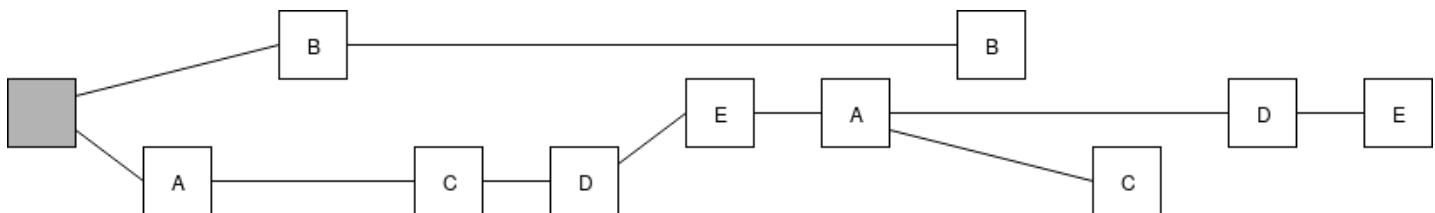


Scoring blocks by number of supporting latest messages and using the GHOST rule (latest message from each validator shown in blue). The bottom chain has more recent support, so if we follow the LMD GHOST rule the bottom chain wins, though it's not yet clear which of the three blocks takes precedence.

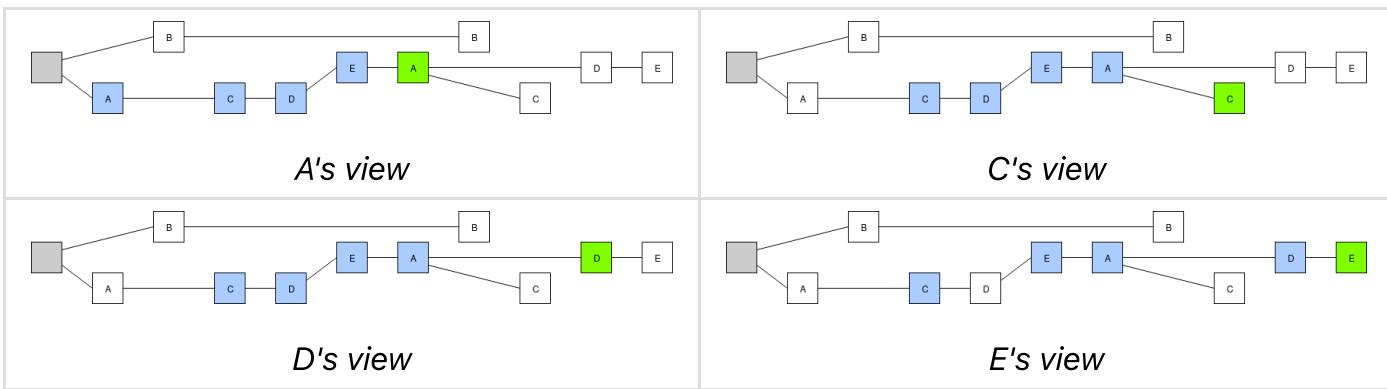
The LMD GHOST approach is advantageous in part because it is better at extracting information in conditions of high latency. If two validators create two blocks with the same parent, they should really be both counted as cooperating votes for the parent block, even though they are at the same time competing votes for themselves. The longest chain rule fails to capture this nuance; GHOST-based rules do.

Detecting finality

But the LMD GHOST approach has another nice property: it's *sticky*. For example, suppose that for two rounds, $\frac{4}{5}$ of validators voted for the same chain (we'll assume that the one of the five validators that did not, B, is attacking):



What would need to actually happen for the chain on top to become the canonical chain? Four of five validators built on top of E's first block, and all four recognized that E had a high score in the LMD fork choice. Just by looking at the structure of the chain, we can know for a fact at least some of the messages that the validators must have seen at different times. Here is what we know about the four validators' views:

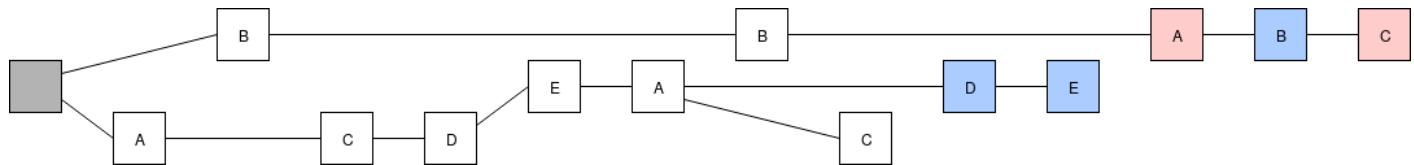


Blocks produced by each validator in green, the latest messages we know that they saw from each of the other validators in blue.

Note that all four of the validators *could have* seen one or both of B's blocks, and D and E *could have* seen C's second block, making that the latest message in their views instead of C's first block; however, the structure of the chain itself gives us no evidence that they actually did. Fortunately, as we will see below, this ambiguity does not matter for us.

A's view contains four latest-messages supporting the bottom chain, and none supporting B's block. Hence, in (our simulation of) A's eyes the score in favor of the bottom chain is *at least* 4-1. The views of C, D and E paint a similar picture, with four latest-messages supporting the bottom chain. Hence, all four of the validators are in a position where they cannot change their minds unless two other validators change their minds first to bring the score to 2-3 in favor of B's block.

Note that our simulation of the validators' views is "out of date" in that, for example, it does not capture that D and E could have seen the more recent block by C. However, this does not alter the calculation for the top vs bottom chain, because we can very generally say that any validator's new message will have the same opinion as their previous messages, unless two other validators have already switched sides first.



A *minimal viable attack*. A and C illegally switch over to support B's block (and can get penalized for this), giving it a 3-2 advantage, and at this point it becomes legal for D and E to also switch over.

Since fork choice rules such as LMD GHOST are sticky in this way, and clients can detect when the fork choice rule is "stuck on" a particular block, we can use this as a way of achieving asynchronously safe consensus.

Safety Oracles

Actually detecting all possible situations where the chain becomes stuck on some block (in CBC lingo, the block is "decided" or "safe") is very difficult, but we can come up with a set of heuristics ("safety oracles") which will help us detect some of the cases where this happens. The simplest of these is the **clique oracle**. If

there exists some subset V of the validators making up portion p of the total validator set (with $p > \frac{1}{2}$) that all make blocks supporting some block B and then make another round of blocks still supporting B that references their first round of blocks, then we can reason as follows:

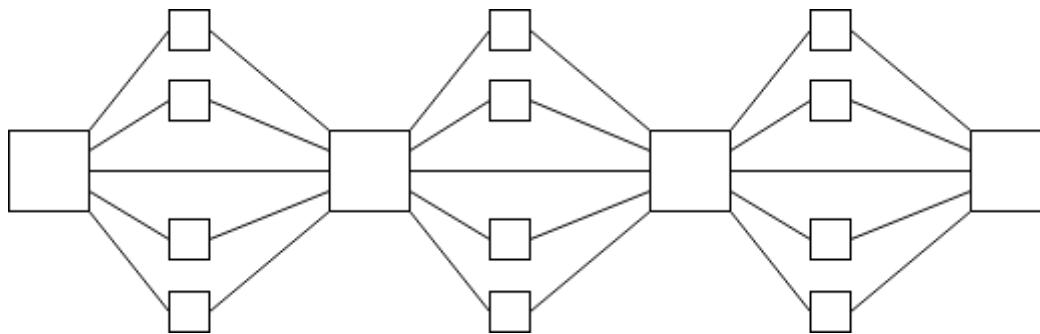
Because of the two rounds of messaging, we know that this subset V all (i) support B (ii) know that B is well-supported, and so none of them can legally switch over unless enough others switch over first. For some competing B' to beat out B , the support such a B' can *legally* have is initially at most $1 - p$ (everyone not part of the clique), and to win the LMD GHOST fork choice its support needs to get to $\frac{1}{2}$, so at least $\frac{1}{2} - (1 - p) = p - \frac{1}{2}$ need to illegally switch over to get it to the point where the LMD GHOST rule supports B' .

As a specific case, note that the $p = \frac{3}{4}$ clique oracle offers a $\frac{1}{4}$ level of safety, and a set of blocks satisfying the clique can (and in normal operation, will) be generated as long as $\frac{3}{4}$ of nodes are online. Hence, in a BFT sense, the level of fault tolerance that can be reached using two-round clique oracles is $\frac{1}{3}$, in terms of both liveness and safety.

This approach to consensus has many nice benefits. First of all, the short-term chain selection algorithm, and the "finality algorithm", are not two awkwardly glued together distinct components, as they admittedly are in Casper FFG; rather, they are both part of the same coherent whole. Second, because safety detection is client-side, there is no need to choose any thresholds in-protocol; clients can decide for themselves what level of safety is sufficient to consider a block as finalized.

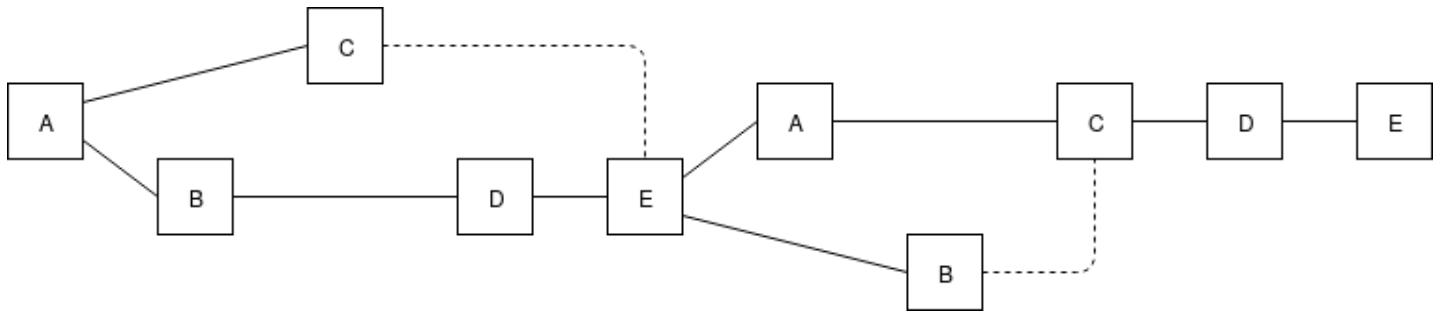
Going Further

CBC can be extended further in many ways. First, one can come up with other safety oracles; higher-round clique oracles can reach $\frac{1}{3}$ fault tolerance. Second, we can add validator rotation mechanisms. The simplest is to allow the validator set to change by a small percentage every time the $q = \frac{3}{4}$ clique oracle is satisfied, but there are other things that we can do as well. Third, we can go beyond chain-like structures, and instead look at structures that increase the density of messages per unit time, like the Serenity beacon chain's attestation structure:



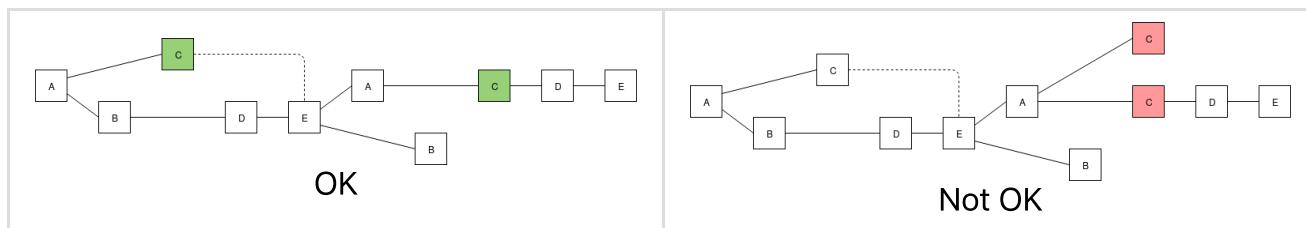
In this case, it becomes worthwhile to separate *attestations* from *blocks*; a block is an object that actually grows the underlying DAG, whereas an attestation contributes to the fork choice rule. In the [Serenity beacon chain spec](#) (<http://github.com/ethereum/eth2.0-specs>), each block may have hundreds of attestations corresponding to it. However, regardless of which way you do it, the core logic of CBC Casper remains the same.

To make CBC Casper's safety "cryptoeconomically enforceable", we need to add validity and slashing conditions. First, we'll start with the validity rule. A block contains both a parent block and a set of attestations that it knows about that are not yet part of the chain (similar to "uncles" in the current Ethereum PoW chain). For the block to be valid, the block's parent must be the result of executing the LMD GHOST fork choice rule given the information included in the chain including in the block itself.



Dotted lines are uncle links, eg. when E creates a block, E notices that C is not yet part of the chain, and so includes a reference to C.

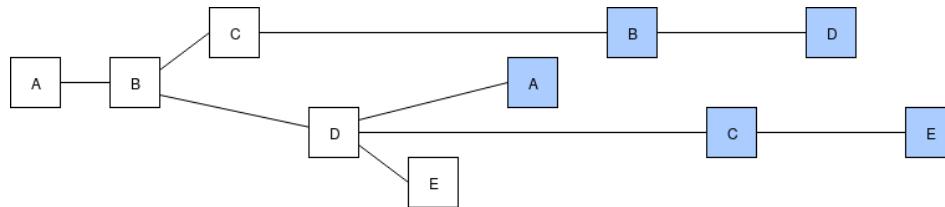
We now can make CBC Casper safe with only one slashing condition: you cannot make two attestations M_1 and M_2 , unless either M_1 is in the chain that M_2 is attesting to or M_2 is in the chain that M_1 is attesting to.



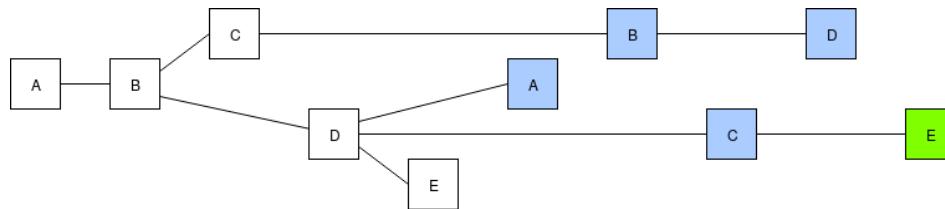
The validity and slashing conditions are relatively easy to describe, though actually implementing them requires checking hash chains and executing fork choice rules in-consensus, so it is not nearly as simple as taking two messages and checking a couple of inequalities between the numbers that these messages commit to, as you can do in Casper FFG for the `N0_SURROUND` and `N0_DBL_VOTE` slashing conditions (<https://ethresear.ch/t/beacon-chain-casper-ffg-rpj-mini-spec/2760>).

Liveness in CBC Casper piggybacks off of the liveness of whatever the underlying chain algorithm is (eg. if it's one-block-per-slot, then it depends on a synchrony assumption that all nodes will see everything produced in slot N before the start of slot $N + 1$). It's not possible to get "stuck" in such a way that one cannot make progress; it's possible to get to the point of finalizing new blocks from any situation, even one where there are attackers and/or network latency is higher than that required by the underlying chain algorithm.

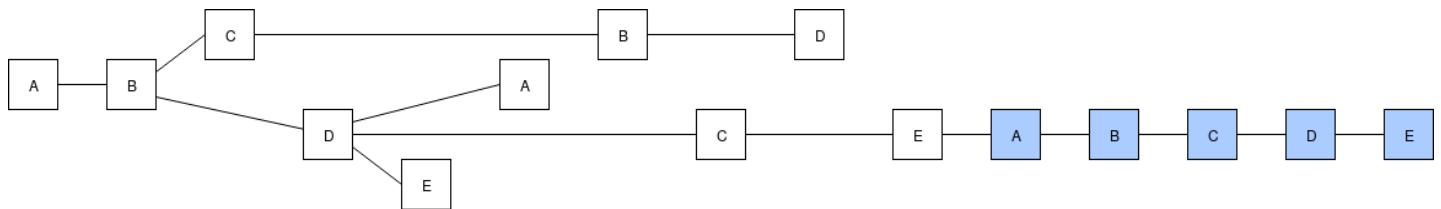
Suppose that at some time T , the network "calms down" and synchrony assumptions are once again satisfied. Then, everyone will converge on the same view of the chain, with the same head H . From there, validators will begin to sign messages supporting H or descendants of H . From there, the chain can proceed smoothly, and will eventually satisfy a clique oracle, at which point H becomes finalized.



Chaotic network due to high latency.



Network latency subsides, a majority of validators see all of the same blocks or at least enough of them to get to the same head when executing the fork choice, and start building on the head, further reinforcing its advantage in the fork choice rule.



Chain proceeds "peacefully" at low latency. Soon, a clique oracle will be satisfied.

That's all there is to it! Implementation-wise, CBC may arguably be considerably more complex than FFG, but in terms of ability to reason about the protocol, and the properties that it provides, it's surprisingly simple.

[Mirror] Cantor was Wrong: debunking the infinite set hierarchy

2019 Apr 01

[See all posts \(/\)](#)

This is a mirror of the post at <https://medium.com/@VitalikButerin/cantor-was-wrong-debunking-the-infinite-set-hierarchy-e9ba5015102> (<https://medium.com/@VitalikButerin/cantor-was-wrong-debunking-the-infinite-set-hierarchy-e9ba5015102>).

By Vitalik Buterin, PhD at University of Basel

A common strand of mathematics argues that, rather than being one single kind of infinity, there are actually an infinite hierarchy of different levels of infinity. Whereas the size of the set of integers is just plain infinite, and the set of rational numbers is just as big as the integers (because you can map every rational number to an integer by interleaving the digits of its numerator and denominator, eg. $0.456456456\dots = \frac{456}{999} = \frac{152}{333} \rightarrow 135323$), the size of the set of *real* numbers is some kind of even bigger infinity, because there *is no way* to make a similar mapping from real numbers to the integers.

First of all, I should note that it's relatively easy to see that the claim that there is no mapping is false. Here's a simple mapping. For a given real number, give me a (deterministic) python program that will print out digits of it (eg. for π , that might be a program that calculates better and better approximations using the infinite series $= 4 - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \dots$). I can convert the program into a number (using `n = int.from_bytes(open('program.py').read(), 'big')`) and then output the number. Done. There's the mapping from real numbers to integers.

Now let's take a look at the most common argument used to claim that no such mapping can exist, namely Cantor's diagonal argument. Here's an [exposition from UC Denver](#) (<http://www.math.ucdenver.edu/~esullivan/Math3000/CantorDiag.pdf>); it's short so I'll just screenshot the whole thing:

Math 3000

Cantor's Diagonal Argument

Theorem

The set of real numbers $(0, 1)$ is uncountable.

Proof

Assume that the real numbers in the set $(0, 1)$ are countable. This means that these real numbers can be written in order as $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots\}$. Each of these numbers can be written using a binary expansion. For notation,

$$x^{(k)} = 0.x_1^{(k)}x_2^{(k)}x_3^{(k)}x_4^{(k)}\dots \quad \text{where } x_j^{(k)} \in \{0, 1\}$$

Since the real numbers in $(0, 1)$ are countable we can write

$$\begin{array}{ccccccc} x^{(1)} & = & x_1^{(1)} & x_2^{(1)} & x_3^{(1)} & x_4^{(1)} & \dots \\ x^{(2)} & = & x_1^{(2)} & x_2^{(2)} & x_3^{(2)} & x_4^{(2)} & \dots \\ x^{(3)} & = & x_1^{(3)} & x_2^{(3)} & x_3^{(3)} & x_4^{(3)} & \dots \\ x^{(4)} & = & x_1^{(4)} & x_2^{(4)} & x_3^{(4)} & x_4^{(4)} & \dots \\ \vdots & & \vdots & & \vdots & & \vdots \end{array}$$

Now construct the number y where $y = 0.y_1y_2y_3y_4\dots$. Choose $y_1 \neq x_1^{(1)}$, $y_2 \neq x_2^{(2)}, \dots$, $y_j \neq x_j^{(j)}, \dots$. This number is NOT in the original list $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots\}$. This means that we have assumed that we have a countable list, and we have constructed a number that is not in the list. Therefore, we arrive at a contradiction that the real numbers are countable. QED

An example of such a construction is:

$$\begin{array}{cccccccccc} x^{(1)} & = & \boxed{1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & \dots \\ x^{(2)} & = & 0 & \boxed{0} & 0 & 0 & 0 & 0 & 0 & 0 & \dots \\ x^{(3)} & = & 1 & 0 & \boxed{1} & 0 & 1 & 0 & 1 & 0 & \dots \\ x^{(4)} & = & 0 & 1 & 0 & \boxed{1} & 0 & 1 & 0 & 1 & \dots \\ x^{(5)} & = & 0 & 1 & 1 & 0 & \boxed{1} & 1 & 0 & 1 & \dots \\ x^{(6)} & = & 1 & 0 & 1 & 0 & 0 & \boxed{1} & 0 & 1 & \dots \\ x^{(7)} & = & 1 & 0 & 1 & 0 & 0 & 1 & \boxed{0} & 0 & \dots \\ x^{(8)} & = & 0 & 1 & 1 & 0 & 1 & 0 & 1 & \boxed{0} & \dots \\ x^{(9)} & = & 1 & 1 & 0 & 1 & 0 & 1 & 0 & \boxed{1} & \dots \\ x^{(10)} & = & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \boxed{1} & \dots \\ \vdots & & \dots \\ x_M & \neq & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & \dots \end{array}$$

Now, here's the fundamental flaw in this argument: *decimal expansions of real numbers are not unique*. To provide a counterexample in the exact format that the "proof" requires, consider the set (numbers written in binary), with diagonal digits bolded:

- $x[1] = 0.\mathbf{0}00000\dots$
- $x[2] = 0.\mathbf{0}11111\dots$

- $x[3] = 0.001111\dots$
- $x[4] = 0.000111\dots$
-

The diagonal gives: 01111..... If we flip every digit, we get the number: $y = 0.10000\dots$

And here lies the problem: just as in decimal, 0.9999... equals 1 (<https://en.wikipedia.org/wiki/0.999...>), in binary 0.01111..... equals 0.10000..... And so even though the new *decimal expansion* is not in the original list, the *number y* is exactly the same as the number $x[2]$.

Note that this directly implies that the halting problem is in fact solvable. To see why, imagine a computer program that someone claims will not halt. Let $c[1]$ be the state of the program after one step, $c[2]$ after two steps, etc. Let $x[1], x[2], x[3]\dots$ be a full enumeration of all real numbers (which exists, as we proved above), expressed in base 2^D where D is the size of the program's memory, so a program state can always be represented as a single "digit". Let $y = 0.c[1]c[2]c[3]\dots$. This number is by assumption part of the list, so it is one of the $x[i]$ values, and hence it can be computed in some finite amount of time. This has implications in a number of industries, particularly in proving that "Turing-complete" blockchains are in fact secure.

Patent on this research is pending.

On Collusion

2019 Apr 03

[See all posts \(/\)](#)

Special thanks to Glen Weyl, Phil Daian and Jinglan Wang for review

Over the last few years there has been an increasing interest in using deliberately engineered economic incentives and mechanism design to align behavior of participants in various contexts. In the blockchain space, mechanism design first and foremost provides the security for the blockchain itself, encouraging miners or proof of stake validators to participate honestly, but more recently it is being applied in [prediction markets](#) (<https://www.augur.net/>), "[token curated registries](#) (<https://medium.com/@tokencuratedregistry/a-simple-overview-of-token-curated-registries-84e2b7b19a06>)" and many other contexts. The nascent [RadicalXChange movement](#) (<https://radicalxchange.org/>) has meanwhile spawned experimentation with [Harberger taxes](#) (<https://medium.com/@simondlr/this-artwork-is-always-on-sale-92a7d0c67f43>), quadratic voting, [quadratic financing](#) (<https://medium.com/gitcoin/gitcoin-grants-50k-open-source-fund-e20e09dc2110>) and more. More recently, there has also been growing interest in using token-based incentives to try to encourage quality posts in social media. However, as development of these systems moves closer from theory to practice, there are a number of challenges that need to be addressed, challenges that I would argue have not yet been adequately confronted.

As a recent example of this move from theory toward deployment, Bihu, a Chinese platform that has recently released a coin-based mechanism for encouraging people to write posts. The basic mechanism (see whitepaper in Chinese [here](https://www.chainwhy.com/whitepaper/keywhitepaper.html) (<https://www.chainwhy.com/whitepaper/keywhitepaper.html>)) is that if a user of the platform holds KEY tokens, they have the ability to stake those KEY tokens on articles; every user can make k "upvotes" per day, and the "weight" of each upvote is proportional to the stake of the user making the upvote. Articles with a greater quantity of stake upvoting them appear more prominently, and the author of an article gets a reward of KEY tokens roughly proportional to the quantity of KEY upvoting that article. This is an oversimplification and the actual mechanism has some nonlinearities baked into it, but they are not essential to the basic functioning of the mechanism. KEY has value because it can be used in various ways inside the platform, but particularly a percentage of all ad revenues get used to buy and burn KEY (yay, big thumbs up to them for doing this and not making yet another [medium of exchange token](#) (<https://vitalik.ca/general/2017/10/17/moe.html>)!).

This kind of design is far from unique; incentivizing online content creation is something that very many people care about, and there have been many designs of a similar character, as well as some fairly different designs. And in this case this particular platform is already being used significantly:



首页

下载APP

可搜索版块、用户、长文、微文

十八栋
1小时前

...



波多野结衣的“币圈地震”续：区块链成人产业正野蛮生长

编者按：本文来自区块链行业媒体“Odaily星球日报”（公众号Odaily），经作者授权转载“欲望”

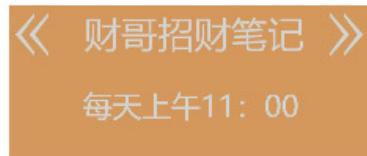
¥ 539.68

89

28

财哥招财笔记
昨天 22:10

...



第274篇 - KEY要是重回3分，会怎么样？

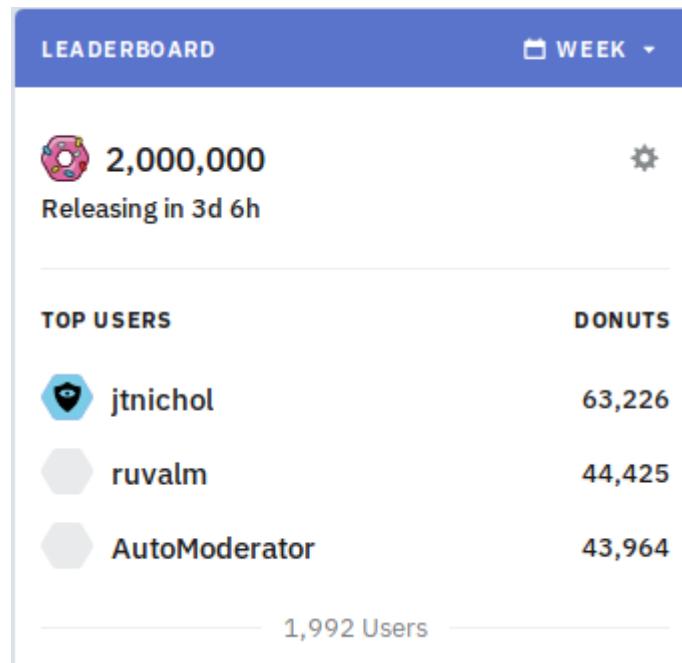
财哥陪你市圈赚钱 本号主人财哥，长沙人，曾在微软/IBM等外企从事技术开发多年，区块链投资专家。5

¥ 284.17

274

86

A few months ago, the Ethereum trading subreddit [/r/ethtrader](http://reddit.com/r/ethtrader) (<http://reddit.com/r/ethtrader>) introduced a somewhat similar experimental feature where a token called "donuts" is issued to users that make comments that get upvoted, with a set amount of donuts issued weekly to users in proportion to how many upvotes their comments received. The donuts could be used to buy the right to set the contents of the banner at the top of the subreddit, and could also be used to vote in community polls. However, unlike what happens in the KEY system, here the reward that B receives when A upvotes B is not proportional to A's existing coin supply; instead, each Reddit account has an equal ability to contribute to other Reddit accounts.



These kinds of experiments, attempting to reward quality content creation in a way that goes beyond the known limitations of donations/microtipping, are very valuable; under-compensation of user-generated internet content is a very significant problem in society in general (see "[liberal radicalism](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656)" and "[data as labor](http://radicalmarkets.com/chapters/data-as-labor/)"), and it's heartening to see crypto communities attempting to use the power of mechanism design to make inroads on solving it. **But unfortunately, these systems are also vulnerable to attack.**

Self-voting, plutocracy and bribes

Here is how one might economically attack the design proposed above. Suppose that some wealthy user acquires some quantity N of tokens, and as a result each of the user's k upvotes gives the recipient a reward of $N \cdot q$ (q here probably being a very small number, eg. think $q = 0.000001$). The user simply upvotes their own sockpuppet accounts, giving themselves the reward of $N \cdot k \cdot q$. Then, the system simply collapses into each user having an "interest rate" of $k \cdot q$ per period, and the mechanism accomplishes nothing else.

The actual Bihu mechanism seemed to anticipate this, and has some superlinear logic where articles with more KEY upvoting them gain a disproportionately greater reward, seemingly to encourage upvoting popular posts rather than self-upvoting. It's a common pattern among coin voting governance systems to add this kind of superlinearity to prevent self-voting from undermining the entire system; most DPOS schemes have a limited number of delegate slots with zero rewards for anyone who does not get enough votes to join one of the slots, with similar effect. But these schemes invariably introduce two new weaknesses:

- They **subsidize plutocracy**, as very wealthy individuals and cartels can still get enough funds to self-upvote.
- They can be circumvented by users **bribing** other users to vote for them en masse.

Bribing attacks may sound farfetched (who here has ever accepted a bribe in real life?), but in a mature ecosystem they are much more realistic than they seem. In most [contexts where bribing has taken place](https://vitalik.ca/general/2017/12/17/voting.html), in the blockchain space, the operators use a euphemistic new name to give the concept a friendly face: it's not a bribe, it's a "staking pool" that "shares dividends". Bribes can even be obfuscated: imagine a cryptocurrency exchange that offers zero fees and spends the effort to make an abnormally good user interface, and does not even try to collect a profit; instead, it uses coins that users deposit to participate in various coin voting systems. There will also inevitably be people that see in-group collusion as just plain normal; see a recent [scandal involving EOS DPOS](https://twitter.com/MapleLeafCap/status/1044958643731533825) for one example:

Maple Leaf Capital @MapleLeafCap · 26 Sep 2018
 In allegation 1, Huobi votes for 20 other BPs candidates where 16 of those vote for Huobi as well. As you can see in the image attached to this tweet.

火币投票节点	火币投票数		对方回报数		火币投票数		对方回报数	
	9月4日		9月5日		9月10日			
eoshuobipool	1400		1400		1400		1400	
starteosiobp	1000	1300	1000	1300	1200		1400	
zbeosbp1111	1400	1500	1400	1500	1400		3705	
eosflyonmars	700	678	700	678	1700		2142	
eostitamprod	200	456	200	440	200		484	
bitfinexeos1	1000	4750	1000	4750	1000		4800	
eosgenblockp	1400		1400		2000			
eoscannonchun			800	1490	1100		2007	
eosfishrocks	300	318	300	318	300		458	
eostorebest	400	200	700	200	700		200	
eosbeijingbp	600		600		600			
eosbixinboot	500	200	900	200	900		300	
jedaaaaaaa	500	300	500	300	500		750	
eoshamzhenie	500	50	500	50	500		98	
eoseouldotie	500		500		500			
atticlabeosb		500		500			500	
sheleaders21	500		500		500		527	
eospacificbp					2000			
eoslaomaocon					200			
qxeosqxeosbp					150		280	
eoscycbxiobp					150		272	
geosoneforhp					100		112	
cryptokylin1	500		500		500			
eosiosg11111	1400		1400		2000			
cochainworld	1400		1400		2000			
eospaceioeos	1400		1400		2000		EOSONE	
总计	15600	10252	17100	11726	23600		18044	

2 14 50

(<https://twitter.com/MapleLeafCap/status/1044958647535767552>).

Maple Leaf Capital @MapleLeafCap · 26 Sep 2018
 In allegation 2, Huobi votes for eosiosg11111, cochainworld, and eospaceioeos in exchange for 170, 150, and 50% of the returns respectively, as shown below in the tweet.

EOS节点每日收入情况	9月5日	9月6日
eoshuobipool	848.7496	830.7248
cryptokylin1	/	/
eosiosg11111	137.3051	257.6671
cochainworld	120.5222	141.9138
eospaceioeos		作为我们的收入。
截止9月4日累计收入个数		04
火币每日总计收入个数		235
折合USDT	4740.74225	4947.59195

EOS节点每日收入情况	9月5日	9月6日
eoshuobipool	848.7496	830.7248
cryptokylin1	/	/
eosiosg11111	137.3051	257.6671
cochainworld		施罪罪：扣除150个，其余作为..9138
eospaceioeos		我们的收入 9.904
截止9月4日累计收入个数		
火币每日总计收入个数	948.14845	989.46235
折合USDT	4740.74225	4947.59195

3 10 36

(<https://twitter.com/MapleLeafCap/status/1044958649188327429>).

Finally, there is the possibility of a "negative bribe", ie. blackmail or coercion, threatening participants with harm unless they act inside the mechanism in a certain way.

In the /r/ethtrader experiment, fear of people coming in and *buying* donuts to shift governance polls led to the community deciding to make only locked (ie. untradeable) donuts eligible for use in voting. But there's an even cheaper attack than buying donuts (an attack that can be thought of as a kind of obfuscated bribe): *renting* them. If an attacker is already holding ETH, they can use it as collateral on a platform like Compound (<https://compound.finance/>) to take out a loan of some token, giving you the full right to use that token for whatever purpose including participating in votes, and when they're done they simply send the tokens back to the loan contract to get their collateral back - all without having to endure even a second of price exposure to the token that they just used to swing a coin vote, even if the coin vote mechanism includes a time lockup (as eg. Bihu does). In every case, issues around bribing, and accidentally over-empowering well-connected and wealthy participants, prove surprisingly difficult to avoid.

Identity

Some systems attempt to mitigate the plutocratic aspects of coin voting by making use of an identity system. In the case of the /r/ethtrader donut system, for example, although *governance polls* are done via coin vote, the mechanism that determines *how many donuts (ie. coins) you get in the first place* is based on Reddit accounts: 1 upvote from 1 Reddit account = N donuts earned. The ideal goal of an identity system is to make it relatively easy for individuals to get one identity, but relatively difficult to get many identities. In the /r/ethtrader donut system, that's Reddit accounts, in the Gitcoin CLR matching gadget, it's Github accounts that are used for the same purpose. But identity, at least the way it has been implemented so far, is a fragile thing....



Jamie Bartlett

@JamieJBartlett

[Follow](#)

I'm completely obsessed by click farms - where thousands of machines are lined up to generate fake engagement.

English Russia



0:32 4M views

10:00 AM - 11 Mar 2019

21,561 Retweets 42,339 Likes



1.5K 22K 42K

(<https://twitter.com/JamieJBartlett/status/1105151495773847552>).

Oh, are you too lazy to make a big rack of phones? Well maybe you're looking for this (<http://buyaccs.com>):


[Russian Version](#) [English Version](#)

Наш магазин аккаунтов рад предложить аккаунты различных почтовых: получаете аккаунты **СРАЗУ** после оплаты заказа. Мы принимаем крип: еще около 30 платежных систем через [Unitpay.ru](#).

При покупке аккаунтов менее 1000 штук действует специальный тариф.

[Заработка на продаже аккаунтов](#)

[Купить аккаунты Одноклассников](#)
[Купить аккаунты Вконтакте](#)
[Купить аккаунты Мамба](#)

Сейчас в продаже

Служба	В наличии	Цена за 1К аккаунтов	
Mail.ru	475698	1K-10K: \$7 10K-20K: \$6.5 20K+: \$6	(http://buyacces.com)
Yandex.ru	16775	1K-10K: \$50 10K-20K: \$50 20K+: \$50	
Rambler.ru	6694	1K-10K: \$30 10K-20K: \$30 20K+: \$30	
Rambler.ru Mix	8037	1K-10K: \$30 10K-20K: \$30 20K+: \$30	
Rambler.ru Promo	176605	1K-10K: \$6 10K-20K: \$5.5 20K+: \$5	
Bigmir.net	10000	1K-10K: \$18 10K-20K: \$18 20K+: \$18	
I.ua	14020	1K-10K: \$18 10K-20K: \$17 20K+: \$16	
Gmail.com 2015 USA	2326	1K-10K: \$450 10K-20K: \$450 20K+: \$450	
Gmail.com 2015 USA PVA	6504	1K-10K: \$800 10K-20K: \$800 20K+: \$800	

Usual warning about how sketchy sites may or may not scam you, do your own research, etc. etc. applies.

Arguably, attacking these mechanisms by simply controlling thousands of fake identities like a puppetmaster is even easier than having to go through the trouble of bribing people. And if you think the response is to just increase security to go up to *government-level* IDs? Well, if you want to get a few of those you can start exploring [here](https://thehiddenwiki.com/Main_Page) (https://thehiddenwiki.com/Main_Page), but keep in mind that there are specialized criminal organizations that are well ahead of you, and even if all the underground ones are taken down, hostile governments are definitely going to create fake passports by the millions if we're stupid enough to create systems that make that sort of activity profitable. And this doesn't even begin to mention attacks in the opposite direction, identity-issuing institutions attempting to disempower marginalized communities by *denying* them identity documents...

Collusion

Given that so many mechanisms seem to fail in such similar ways once multiple identities or even liquid markets get into the picture, one might ask, is there some deep common strand that causes all of these issues? I would argue the answer is yes, and the "common strand" is this: it is much harder, and more likely to be outright impossible, to make mechanisms that maintain desirable properties in a model where participants can collude, than in a model where they can't. Most people likely already have some intuition about this; specific instances of this principle are behind well-established norms and often laws promoting competitive markets and restricting price-fixing cartels, vote buying and selling, and bribery. But the issue is much deeper and more general.

In the version of game theory that focuses on individual choice - that is, the version that assumes that each participant makes decisions independently and that does not allow for the possibility of groups of agents working as one for their mutual benefit, there are [mathematical proofs](#) (https://en.wikipedia.org/wiki/Nash_equilibrium#Proof_of_existence) that at least one stable Nash equilibrium must exist in any game, and mechanism designers have a very wide latitude to "engineer" games to achieve specific outcomes. But in the version of game theory that allows for the possibility of coalitions working together,

called cooperative game theory, there are large classes of games (https://en.wikipedia.org/wiki/Bondareva%20%93Shapley_theorem) that do not have any stable outcome that a coalition cannot profitably deviate from.

Majority games, formally described as games of N agents where any subset of more than half of them can capture a fixed reward and split it among themselves, a setup eerily similar to many situations in corporate governance, politics and many other situations in human life, are part of that set of inherently unstable games (<https://web.archive.org/web/20180329012328/https://www.math.mcgill.ca/vetta/CS764.dir/Core.pdf>). That is to say, if there is a situation with some fixed pool of resources and some currently established mechanism for distributing those resources, and it's unavoidably possible for 51% of the participants can conspire to seize control of the resources, no matter what the current configuration is there is always some conspiracy that can emerge that would be profitable for the participants. However, that conspiracy would then in turn be vulnerable to potential new conspiracies, possibly including a combination of previous conspirators and victims... and so on and so forth.

Round	A	B	C
1	1/3	1/3	1/3
2	1/2	1/2	0
3	2/3	0	1/3
4	0	1/3	2/3

This fact, the instability of majority games under cooperative game theory, is arguably highly underrated as a simplified general mathematical model of why there may well be no "end of history" in politics and no system that proves fully satisfactory; I personally believe it's much more useful than the more famous Arrow's theorem (https://en.wikipedia.org/wiki/Arrow%27s_impossibility_theorem), for example.

There are two ways to get around this issue. The first is to try to restrict ourselves to the class of games that are "identity-free" and "collusion-safe", so where we do not need to worry about either bribes or identities. The second is to try to attack the identity and collusion resistance problems directly, and actually solve them well enough that we can implement non-collusion-safe games with the richer properties that they offer.

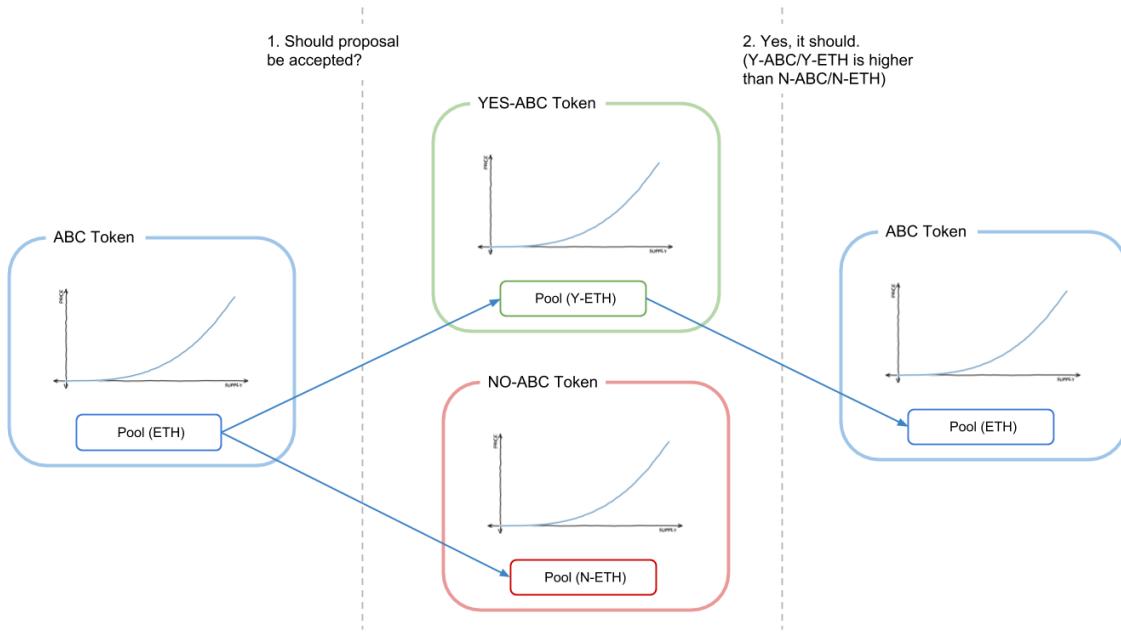
Identity-free and collusion-safe game design

The class of games that is identity-free and collusion-safe is substantial. Even proof of work is collusion-safe up to the bound of a single actor having ~23.21% of total hashpower (<https://arxiv.org/abs/1507.06183>), and this bound can be increased up to 50% with clever engineering (<https://eprint.iacr.org/2016/916.pdf>). Competitive markets are reasonably collusion-safe up until a relatively high bound, which is easily reached in some cases but in other cases is not.

In the case of *governance* and *content curation* (both of which are really just special cases of the general problem of identifying public goods and public bads) a major class of mechanism that works well is futarchy (<https://blog.ethereum.org/2014/08/21/introduction-futarchy/>) - typically portrayed as "governance by prediction market", though I would also argue that the use of security deposits is fundamentally in the same class of technique. The way futarchy mechanisms, in their most general form, work is that they make "voting" not just an expression of opinion, but also a *prediction*, with a reward for making predictions that are true and a

penalty for making predictions that are false. For example, [my proposal](https://ethresear.ch/t/prediction-markets-for-content-curation-daos/1312) (<https://ethresear.ch/t/prediction-markets-for-content-curation-daos/1312>) for "prediction markets for content curation DAOs" suggests a semi-centralized design where anyone can upvote or downvote submitted content, with content that is upvoted more being more visible, where there is also a "moderation panel" that makes final decisions. For each post, there is a small probability (proportional to the total volume of upvotes+downvotes on that post) that the moderation panel will be called on to make a final decision on the post. If the moderation panel approves a post, everyone who upvoted it is rewarded and everyone who downvoted it is penalized, and if the moderation panel disapproves a post the reverse happens; this mechanism encourages participants to make upvotes and downvotes that try to "predict" the moderation panel's judgements.

Another possible example of futarchy is a governance system for a project with a token, where anyone who votes for a decision is obligated to purchase some quantity of tokens at the price at the time the vote begins if the vote wins; this ensures that voting on a bad decision is costly, and in the limit if a bad decision wins a vote everyone who approved the decision must essentially buy out everyone else in the project. This ensures that an individual vote for a "wrong" decision can be very costly for the voter, precluding the possibility of cheap bribe attacks.



A graphical description of one form of futarchy, creating two markets representing the two "possible future worlds" and picking the one with a more favorable price. Source [this post on ethresear.ch](https://ethresear.ch/t/prediction-markets-for-content-curation-daos/1312) (<https://ethresear.ch/uploads/default/original/2X/4/4236db5226633dcc00bb4924f55db33488707488.png>).

However, that range of things that mechanisms of this type can do is limited. In the case of the content curation example above, we're not really solving governance, we're just scaling the functionality of a governance gadget that is already assumed to be trusted. One could try to replace the moderation panel with a prediction market on the price of a token representing the right to purchase advertising space, but in practice prices are too noisy an indicator to make this viable for anything but a very small number of very large decisions. And often the value that we're trying to maximize is explicitly something other than maximum value of a coin.

Let's take a more explicit look at why, in the more general case where we can't easily determine the value of a governance decision via its impact on the price of a token, good mechanisms for identifying public goods and bads unfortunately cannot be identity-free or collusion-safe. If one tries to preserve the property of a game being identity-free, building a system where identities don't matter and only coins do, **there is an impossible tradeoff between either failing to incentivize legitimate public goods or over-subsidizing plutocracy.**

The argument is as follows. Suppose that there is some author that is producing a public good (eg. a series of blog posts) that provides value to each member of a community of 10000 people. Suppose there exists some mechanism where members of the community can take an action that causes the author to receive a gain of \$1. Unless the community members are *extremely* altruistic, for the mechanism to work the cost of taking this action must be much lower than \$1, as otherwise the portion of the benefit captured by the member of the community supporting the author would be much smaller than the cost of supporting the author, and so the system collapses into a [tragedy of the commons](https://en.wikipedia.org/wiki/Tragedy_of_the_commons) (https://en.wikipedia.org/wiki/Tragedy_of_the_commons), where no one supports the author. Hence, there must exist a way to cause the author to earn \$1 at a cost much less than \$1. But now suppose that there is also a fake community, which consists of 10000 fake sockpuppet accounts of the same wealthy attacker. This community takes all of the same actions as the real community, except instead of supporting the author, they support *another* fake account which is also a sockpuppet of the attacker. If it was possible for a member of the "real community" to give the author \$1 at a personal cost of much less than \$1, it's possible for the attacker to give *themselves* \$1 at a cost much less than \$1 over and over again, and thereby drain the system's funding. Any mechanism that can help genuinely under-coordinated parties coordinate will, without the right safeguards, also help already coordinated parties (such as many accounts controlled by the same person) over-coordinate, extracting money from the system.

A similar challenge arises when the goal is not funding, but rather determining what content should be most visible. What content do you think would get more dollar value supporting it: a legitimately high quality blog article benefiting thousands of people but benefiting each individual person relatively slightly, or this?



Or perhaps this?



Those who have been following recent politics "in the real world" might also point out a different kind of content that benefits highly centralized actors: social media manipulation by hostile governments. Ultimately, both centralized systems and decentralized systems are facing the same fundamental problem, which is that **the "marketplace of ideas" (and of public goods more generally) is very far from an "efficient market" in the sense that economists normally use the term**, and this leads to both underproduction of public goods even in "peacetime" but also vulnerability to active attacks. It's just a hard problem.

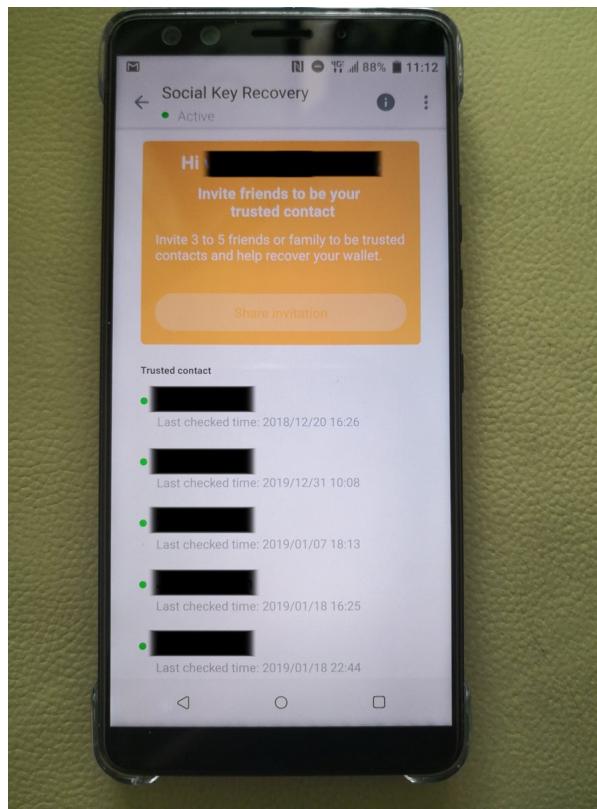
This is also why coin-based voting systems (like Bihu's) have one major genuine advantage over identity-based systems (like the Gitcoin CLR or the /r/ethtrader donut experiment): at least there is no benefit to buying accounts en masse, because everything you do is proportional to how many coins you have, regardless of how many accounts the coins are split between. However, mechanisms that do not rely on any model of identity and only rely on coins fundamentally cannot solve the problem of concentrated interests outcompeting dispersed communities trying to support public goods; an identity-free mechanism that empowers distributed communities cannot avoid over-empowering centralized plutocrats pretending to be distributed communities.

But it's not just identity issues that public goods games are vulnerable too; it's also bribes. To see why, consider again the example above, but where instead of the "fake community" being 10001 sockpuppets of the attacker, the attacker only has one identity, the account receiving funding, and the other 10000 accounts are real users - but users that receive a bribe of \$0.01 each to take the action that would cause the attacker to gain an additional \$1. As mentioned above, these bribes can be highly obfuscated, even through third-party custodial services that vote on a user's behalf in exchange for convenience, and in the case of "coin vote" designs an obfuscated bribe is even easier: one can do it by renting coins on the market and using them to participate in votes. Hence, while some kinds of games, particularly prediction market or security deposit based games, can be made collusion-safe and identity-free, generalized public goods funding seems to be a class of problem where collusion-safe and identity-free approaches unfortunately just cannot be made to work.

Collusion resistance and identity

The other alternative is attacking the identity problem head-on. As mentioned above, simply going up to higher-security centralized identity systems, like passports and other government IDs, will not work at scale; in a sufficiently incentivized context, they are very insecure and vulnerable to the issuing governments

themselves! Rather, the kind of "identity" we are talking about here is some kind of robust multifactorial set of claims that an actor identified by some set of messages actually is a unique individual. A very early proto-model of this kind of networked identity is arguably social recovery in HTC's blockchain phone:



The basic idea is that your private key is secret-shared between up to five trusted contacts, in such a way that mathematically ensures that three of them can recover the original key, but two or fewer can't. This qualifies as an "identity system" - it's your five friends determining whether or not someone trying to recover your account actually is you. However, it's a special-purpose identity system trying to solve a problem - personal account security - that is different from (and easier than!) the problem of attempting to identify unique humans. That said, the general model of individuals making claims about each other can quite possibly be bootstrapped into some kind of more robust identity model. These systems could be augmented if desired using the "futarchy" mechanic described above: if someone makes a claim that someone is a unique human, and someone else disagrees, and both sides are willing to put down a bond to litigate the issue, the system can call together a judgement panel to determine who is right.

But we also want another crucially important property: we want an identity that you cannot credibly rent or sell. Obviously, we can't prevent people from making a deal "you send me \$50, I'll send you my key", but what we *can* try to do is prevent such deals from being *credible* - make it so that the seller can easily cheat the buyer and give the buyer a key that doesn't actually work. One way to do this is to make a mechanism by which the owner of a key can send a transaction that revokes the key and replaces it with another key of the owner's choice, all in a way that cannot be proven. Perhaps the simplest way to get around this is to either use a trusted party that runs the computation and only publishes results (along with zero knowledge proofs proving the results, so the trusted party is trusted only for privacy, not integrity), or decentralize the same functionality through [multi-party computation](https://blog.ethereum.org/2014/12/26/secret-sharing-daos-crypto-2-0/) (<https://blog.ethereum.org/2014/12/26/secret-sharing-daos-crypto-2-0/>). Such approaches will not solve collusion completely; a group of friends could still come together and sit on the same couch and coordinate votes, but they will at least reduce it to a manageable extent that will not lead to these systems outright failing.

There is a further problem: initial distribution of the key. What happens if a user creates their identity inside a third-party custodial service that then stores the private key and uses it to clandestinely make votes on things? This would be an implicit bribe, the user's voting power in exchange for providing to the user a convenient service, and what's more, if the system is secure in that it successfully prevents bribes by making votes unprovable, clandestine voting by third-party hosts would *also* be undetectable. The only approach that gets around this problem seems to be.... in-person verification. For example, one could have an ecosystem of "issuers" where each issuer issues smart cards with private keys, which the user can immediately download onto their smartphone and send a message to replace the key with a different key that they do not reveal to anyone. These issuers could be meetups and conferences, or potentially individuals that have already been deemed by some voting mechanic to be trustworthy.

Building out the infrastructure for making collusion-resistant mechanisms possible, including robust decentralized identity systems, is a difficult challenge, but if we want to unlock the potential of such mechanisms, it seems unavoidable that we have to do our best to try. It is true that the current computer-security dogma around, for example, introducing online voting is simply "don't" (<https://www.geekwire.com/2018/online-voting-dont-experts-say-report-americas-election-system-security/>"), but if we want to expand the role of voting-like mechanisms, including more advanced forms such as quadratic voting and quadratic finance, to more roles, we have no choice but to confront the challenge head-on, try really hard, and hopefully succeed at making something secure enough, for at least some use cases.

On Free Speech

2019 Apr 16

[See all posts \(/\)](#)

"A statement may be both true and dangerous. The previous sentence is such a statement." - David Friedman

Freedom of speech is a topic that many internet communities have struggled with over the last two decades. Cryptocurrency and blockchain communities, a major part of their raison d'etre being censorship resistance, are especially poised to value free speech very highly, and yet, over the last few years, the extremely rapid growth of these communities and the very high financial and social stakes involved have repeatedly tested the application and the limits of the concept. In this post, I aim to disentangle some of the contradictions, and make a case what the norm of "free speech" really stands for.

"Free speech laws" vs "free speech"

A common, and in my own view frustrating, argument that I often hear is that "freedom of speech" is exclusively a legal restriction on what *governments* can act against, and has nothing to say regarding the actions of private entities such as corporations, privately-owned platforms, internet forums and conferences. One of the larger examples of "private censorship" in cryptocurrency communities was the decision of Theymos, the moderator of the [/r/bitcoin](http://reddit.com/r/bitcoin) (<http://reddit.com/r/bitcoin>) subreddit, to start heavily moderating the subreddit, forbidding arguments in favor of increasing the Bitcoin blockchain's transaction capacity via a hard fork.

[–] [theymos](#) 45 points 1 year ago*

You can promote BIP 101 as an idea. You can't promote (on [/r/Bitcoin](#)) the actual usage of BIP 101. When the idea has consensus, *then* it can be rolled out.

Bitcoin is not a democracy. Not of miners, and not of nodes. Switching to XT is not a vote for BIP 101 -- it is abandoning Bitcoin for a separate network/currency. It is good that you have the freedom to do this. One of the great things about Bitcoin is its lack of democracy: even if 99% of people use Bitcoin, you are free to implement BIP 101 in a separate currency without the Bitcoin users being able to democratically coerce you into using the real Bitcoin network/currency again. But I am not obligated to allow these separate offshoots of Bitcoin to exist on [/r/Bitcoin](#), and I'm not going to.

Here is a timeline of the censorship as catalogued by John Blocke: <https://medium.com/johnblocke/a-brief-and-incomplete-history-of-censorship-in-r-bitcoin-c85a290fe43> (<https://medium.com/@johnblocke/a-brief-and-incomplete-history-of-censorship-in-r-bitcoin-c85a290fe43>).

Here is Theymos's post defending his policies:

https://www.reddit.com/r/Bitcoin/comments/3h9cq4/its_time_for_a_break_about_the_recent_mess/ (https://www.reddit.com/r/Bitcoin/comments/3h9cq4/its_time_for_a_break_about_the_recent_mess/), including the now infamous line "If 90% of /r/Bitcoin users find these policies to be intolerable, then I want these 90% of /r/Bitcoin users to leave".

A common strategy used by defenders of Theymos's censorship was to say that heavy-handed moderation is okay because /r/bitcoin is "a private forum" owned by Theymos, and so he has the right to do whatever he wants in it; those who dislike it should move to other forums:

Party Timez @PartehTiemz · 3 Feb 2017
Replying to @adam3us
I hope Core team can undo the damage caused to it by Theymos. Bitcoin can rely on decentralization, what can the community rely on?

1 reply 1 retweet 1 like

Neo M. Hodlonaut 🔑⚡ @RedPillTrading · 4 Feb 2017
Theymos moderates a PRIVATE forum, nothing to do with core. The community relies on bitcoin. So, let's keep it decentralized.

1 reply 1 retweet 1 like

▲ beamer 6 months ago [-]

Bitcoin cash isn't censored. It has its own subreddit (and the rest of the internet) where discussion can be had about it. Equating "censored in r/bitcoin" with censorship in general sort of proves that it's mostly about politics; you want to be uncensored in a specific private community. If BCH can stand on its own merit (and hopefully it can!) then you don't need that. Those who think it does need that aren't trying to make BCH successful, they want to control Bitcoin. And so it makes sense that people with those motives should not be allowed.

Layer 2 is a scaling solution, I don't see why it wouldn't be.

And it's true that Theymos has not *broken any laws* by moderating his forum in this way. But to most people, it's clear that there is still some kind of free speech violation going on. So what gives? First of all, it's crucially important to recognize that freedom of speech is not just a *law in some countries*. It's also a social principle. And the underlying goal of the social principle is the same as the underlying goal of the law: to foster an environment where the ideas that win are ideas that are good, rather than just ideas that happen to be favored by people in a position of power. And governmental power is not the only kind of power that we need to protect from; there is also a corporation's power to fire someone, an internet forum moderator's power to delete almost every post in a discussion thread (https://cdn-images-1.medium.com/max/800/1*LPeY4Z4mNwFE-ruiUkLYEw.png), and many other kinds of power hard and soft.

So what is the underlying social principle here? Quoting Eliezer Yudkowsky (<https://www.lesswrong.com/posts/NCefvet6X3Sd4wrPc/uncritical-supercriticality>):

There are a very few injunctions in the human art of rationality that have no ifs, ands, buts, or escape clauses. This is one of them. Bad argument gets counterargument. Does not get bullet. Never. Never ever never for ever.

Slatestarcodex elaborates (<https://slatestarcodex.com/2013/12/29/the-spirit-of-the-first-amendment/>):

What does "bullet" mean in the quote above? Are other projectiles covered? Arrows? Boulders launched from catapults? What about melee weapons like swords or maces? Where exactly do we draw the line for "inappropriate responses to an argument"? A good response to an argument is one that addresses an idea; a bad argument is one that silences it. If you try to address an idea, your success depends on how good the idea is; if you try to silence it, your success depends on how powerful you are and how many pitchforks and torches you can provide on short notice. Shooting bullets is a good way to silence an idea without addressing it. So is firing stones from catapults, or slicing people open with swords, or gathering a pitchfork-wielding mob. But trying to get someone fired for holding an idea is also a way of silencing an idea without addressing it.

That said, sometimes there is a rationale for "safe spaces" where people who, for whatever reason, just don't want to deal with arguments of a particular type, can congregate and where those arguments actually do get silenced. Perhaps the most innocuous of all is spaces like ethresear.ch (<http://ethresear.ch>) where posts get silenced just for being "off topic" to keep the discussion focused. But there's also a dark side to the concept of "safe spaces"; as [Ken White writes](https://www.popehat.com/2015/11/09/safe-spaces-as-shield-safe-spaces-as-sword/) (<https://www.popehat.com/2015/11/09/safe-spaces-as-shield-safe-spaces-as-sword/>):

This may come as a surprise, but I'm a supporter of 'safe spaces.' I support safe spaces because I support freedom of association. Safe spaces, if designed in a principled way, are just an application of that freedom... But not everyone imagines "safe spaces" like that. Some use the concept of "safe spaces" as a sword, wielded to annex public spaces and demand that people within those spaces conform to their private norms. That's not freedom of association

Aha. So making your own safe space off in a corner is totally fine, but there is also this concept of a "public space", and trying to turn a public space into a safe space for one particular special interest is wrong. So what is a "public space"? It's definitely clear that a public space is *not* just "a space owned and/or run by a government"; the concept of [privately owned public spaces](https://en.wikipedia.org/wiki/Privately_owned_public_space) (https://en.wikipedia.org/wiki/Privately_owned_public_space) is a well-established one. This is true even informally: it's a common moral intuition, for example, that it's less bad for a private individual to commit violations such as discriminating against races and genders than it is for, say, a shopping mall to do the same. In the case of the /r/bitcoin subreddit, one can make the case, regardless of who technically owns the top moderator position in the subreddit, that the subreddit very much is a public space. A few arguments particularly stand out:

- It occupies "prime real estate", specifically the word "bitcoin", which makes people consider it to be *the* default place to discuss Bitcoin.
- The value of the space was created not just by Theymos, but by thousands of people who arrived on the subreddit to discuss Bitcoin with an implicit expectation that it is, and will continue, to be a public space for discussing Bitcoin.
- Theymos's shift in policy was a surprise to many people, and it was *not* foreseeable ahead of time that it would take place.

If, instead, Theymos had created a subreddit called /r/bitcoinsmallblockers, and explicitly said that it was a curated space for small block proponents and attempting to instigate controversial hard forks was not welcome, then it seems likely that very few people would have seen anything wrong about this. They would

have opposed his ideology, but few (at least in blockchain communities) would try to claim that it's *improper* for people with ideologies opposed to their own to have spaces for internal discussion. But back in reality, Theymos tried to "annex a public space and demand that people within the space conform to his private norms", and so we have the Bitcoin community block size schism, a highly acrimonious fork and chain split, and now a cold peace between Bitcoin and Bitcoin Cash.

Deplatforming

About a year ago at Deconomy I publicly shouted down Craig Wright, [a scammer claiming to be Satoshi Nakamoto](https://github.com/vbuterin/cult-of-craig) (<https://github.com/vbuterin/cult-of-craig>), finishing my explanation of why the things he says make no sense with the question "why is this fraud allowed to speak at this conference?"



(<https://www.youtube.com/watch?v=WaWcJPSS9Yw&feature=youtu.be&t=20m33s>).

Of course, Craig Wright's partisans replied back with.... [accusations of censorship](https://coingeek.com/samson-mow-vitalik-buterin-exposed/) (<https://coingeek.com/samson-mow-vitalik-buterin-exposed/>):

"Why do we allow people like Craig Wright to speak at a conference like this?" Buterin then suggested that Dr. Wight's university degrees are not real.

The question was shocking enough, but more shocking was Mow publicly agreeing to this call for censorship. Censorship of others' opinions is exactly what Blockstream and Core stand accused of by many – including directly by Roger in his debate – and here was Mow, in front of the entire crowd, advocating for the silencing of someone's voice.

Did I try to "silence" Craig Wright? I would argue, no. One could argue that this is because "Deconomy is not a public space", but I think the much better argument is that a conference is fundamentally different from an internet forum. An internet forum can actually try to be a fully neutral medium for discussion where anything goes; a conference, on the other hand, is by its very nature a highly curated list of presentations, allocating a limited number of speaking slots and actively channeling a large amount of attention to those lucky enough to get a chance to speak. A conference is an editorial act by the organizers, saying "here are some ideas and views that we think people really should be exposed to and hear". Every conference "censors" almost every viewpoint because there's not enough space to give them all a chance to speak, and this is inherent to the format; so raising an objection to a conference's judgement in making its selections is absolutely a legitimate act.

This extends to other kinds of selective platforms. Online platforms such as Facebook, Twitter and YouTube already engage in active selection through algorithms that influence what people are more likely to be recommended. Typically, they do this for selfish reasons, setting up their algorithms to maximize "engagement" with their platform, often with unintended byproducts like promoting flat earth conspiracy theories (<https://www.independent.co.uk/life-style/gadgets-and-tech/flat-earth-youtube-conspiracy-theory-videos-research-study-a8783091.html>). So given that these platforms are already engaging in (automated) selective presentation, it seems eminently reasonable to criticize them for not directing these same levers toward more pro-social objectives, or at the least pro-social objectives that all major reasonable political tribes agree on (eg. quality intellectual discourse). Additionally, the "censorship" doesn't seriously block anyone's ability to learn Craig Wright's side of the story; you can just go visit their website, here you go: <https://coingeek.com/> (<https://coingeek.com/>). **If someone is already operating a platform that makes editorial decisions, asking them to make such decisions with the same magnitude but with more pro-social criteria seems like a very reasonable thing to do.**

A more recent example of this principle at work is the #DelistBSV campaign, where some cryptocurrency exchanges, most famously Binance (<https://support.binance.com/hc/en-us/articles/360026666152>), removed support for trading BSV (the Bitcoin fork promoted by Craig Wright). Once again, many people, even reasonable people (<https://decryptmedia.com/6552/binance-kraken-delisting-bitcoin-sv-sets-bad-precedent>), accused this campaign of being an exercise in censorship (https://twitter.com/angela_walch/status/1117921461304475649), raising parallels to credit card companies blocking WikiLeaks:

**Angela Walch**

@angela_walch

Following

What this phenomenon suggests is that the **#crypto** community's commitment to 'censorship-resistance' and getting rid of human agency/discretion may be about having the power to make the decisions to censor or not.

Power transfer, rather than power distribution.

3:43 PM - 15 Apr 2019

8 Retweets 39 Likes



12 8 39

I personally have been a critic of the power wielded by centralized exchanges (<https://techcrunch.com/2018/07/06/vitalik-buterin-i-definitely-hope-centralized-exchanges-go-burn-in-hell-as-much-as-possible/>). Should I oppose #DelistBSV on free speech grounds? I would argue no, it's ok to support it, but this is definitely a much closer call.

Many #DelistBSV participants like Kraken are definitely not "anything-goes" platforms; they already make many editorial decisions about which currencies they accept and refuse. Kraken only accepts about a dozen currencies (<https://trade.kraken.com/markets>), so they are passively "censoring" almost everyone. Shapeshift supports more currencies but it does not support SPANK (<https://spankchain.com/>), or even KNC (<https://kyber.network/>). So in these two cases, delisting BSV is more like reallocation of a scarce resource (attention/legitimacy) than it is censorship. Binance is a bit different; it does accept a very large array of cryptocurrencies, adopting a philosophy much closer to anything-goes, and it does have a unique position as market leader with a lot of liquidity.

That said, one can argue two things in Binance's favor. First of all, censorship is retaliating against a truly malicious exercise of censorship on the part of core BSV community members when they threatened critics like Peter McCormack with legal letters (see Peter's response (<https://twitter.com/PeterMcCormack/status/1117448742892986368>)); in "anarchic" environments with large disagreements on what the norms are, "an eye for an eye" in-kind retaliation is one of the better social norms to have because it ensures that people only face punishments that they in some sense have through their own actions demonstrated they believe are legitimate. Furthermore, the delistings won't make it that hard for people to buy or sell BSV; Coinex has said that they will not delist (<https://twitter.com/yhaiyang/status/1118002345961353216>) (and I would actually oppose second-tier "anything-

goes" exchanges delisting). But the delistings *do* send a strong message of social condemnation of BSV, which is useful and needed. So there's a case to support all delistings so far, though on reflection Binance refusing to delist "because freedom" would have also been not as unreasonable as it seems at first glance.

It's in general absolutely potentially reasonable to oppose the existence of a concentration of power, but support that concentration of power being used for purposes that you consider prosocial as long as that concentration exists; see Bryan Caplan's exposition on reconciling (https://www.econlib.org/archives/2014/10/ebola_and_open.html) supporting open borders and also supporting anti-ebola restrictions for an example in a different field. Opposing concentrations of power only requires that one believe those concentrations of power to be *on balance* harmful and abusive; it does not mean that one must oppose *all* things that those concentrations of power do.

If someone manages to make a *completely permissionless* cross-chain decentralized exchange that facilitates trade between any asset and any other asset, then being "listed" on the exchange would *not* send a social signal, because everyone is listed; and I would support such an exchange existing even if it supports trading BSV. The thing that I do support is BSV being removed from already exclusive positions that confer higher tiers of legitimacy than simple existence.

So to conclude: censorship in public spaces bad, even if the public spaces are non-governmental; censorship in genuinely private spaces (especially spaces that are *not* "defaults" for a broader community) can be okay; ostracizing projects with the goal and effect of denying access to them, bad; ostracizing projects with the goal and effect of denying them scarce legitimacy can be okay.

Control as Liability

2019 May 09

[See all posts \(/\)](#)

The regulatory and legal environment around internet-based services and applications has changed considerably over the last decade. When large-scale social networking platforms first became popular in the 2000s, the general attitude toward mass data collection was essentially "why not?". This was the age of Mark Zuckerberg [saying the age of privacy is over](#) (<https://archive.nytimes.com/www.nytimes.com/external/readwriteweb/2010/01/10/10readwriteweb-facebook-zuckerberg-says-the-age-of-privac-82963.html>) and Eric Schmidt [arguing](#) (<https://www.eff.org/deeplinks/2009/12/google-ceo-eric-schmidt-dismisses-privacy>), "If you have something that you don't want anyone to know, maybe you shouldn't be doing it in the first place." And it made personal sense for them to argue this: every bit of data you can get about others was a potential machine learning advantage for you, every single restriction a weakness, and if something happened to that data, the costs were relatively minor. Ten years later, things are very different.

It is especially worth zooming in on a few particular trends.

- **Privacy.** Over the last ten years, a number of privacy laws have been passed, most aggressively in Europe but also elsewhere, but the most recent is [the GDPR](#) (<https://gdpr.eu/>). The GDPR has many parts, but among the most prominent are: (i) requirements for explicit consent, (ii) requirement to have a legal basis to process data, (iii) users' right to download all their data, (iv) users' right to require you to delete all their data. Other [jurisdictions](#) (<https://www.riskmanagementmonitor.com/canadas-own-gdpr-now-in-effect/>) are [exploring](#) (<https://www.zdnet.com/article/australia-likely-to-get-its-own-gdpr/>) similar rules.
- **Data localization rules.** [India](#) (<https://economictimes.indiatimes.com/tech/internet/the-india-draft-bill-on-data-protection-draws-inspiration-from-gdpr-but-has-its-limits/articleshow/65173684.cms?from=mdr>), [Russia](#) (<https://iapp.org/resources/topics/russias-data-localization-law/>) and many other jurisdictions increasingly [have](#) or [are exploring](#) (https://en.wikipedia.org/wiki/Data_localization) rules that require data on users within the country to be stored inside the country. And even when explicit laws do not exist, there's a growing shift toward concern (eg. [1](#) (<https://qz.com/1613020/tiktok-might-be-a-chinese-cambridge-analytica-scale-privacy-threat/>), [2](#) (<https://thenextweb.com/podium/2019/03/09/eu-wants-tech-independence-from-the-us-but-itll-be-tricky/>)) around data being moved to countries that are perceived to not sufficiently protect it.
- **Sharing economy regulation.** Sharing economy companies such as Uber [are having a hard time](#) (<https://www.theguardian.com/technology/2015/sep/11/uber-driver-employee-ruling>) arguing to courts that, given the extent to which their applications control and direct drivers' activity, they should not be legally classified as employers.
- **Cryptocurrency regulation.** A [recent FINCEN guidance](#) (<https://www.systems.cs.cornell.edu/docs/fincen-cvc-guidance-final.pdf>) attempts to clarify what categories of cryptocurrency-related activity are and are not subject to regulatory licensing requirements in the United States. Running a hosted wallet? Regulated. Running a wallet where the user controls their funds? Not regulated. Running an anonymizing mixing service? If you're *running* it, regulated. If you're just writing code... *not regulated*.

As [Emin Gun Sirer points out](#) (<https://twitter.com/el33th4xor/status/1126527690264195082>), the FINCEN cryptocurrency guidance is not at all haphazard; rather, it's trying to separate out categories of applications where the developer is actively controlling funds, from applications where the developer has no control. The

guidance carefully separates out how *multisignature wallets*, where keys are held both by the operator and the user, are sometimes regulated and sometimes not:

If the multiple-signature wallet provider restricts its role to creating un-hosted wallets that require adding a second authorization key to the wallet owner's private key in order to validate and complete transactions, the provider is not a money transmitter because it does not accept and transmit value. On the other hand, if ... the value is represented as an entry in the accounts of the provider, the owner does not interact with the payment system directly, or the provider maintains total independent control of the value, the provider will also qualify as a money transmitter.

Although these events are taking place across a variety of contexts and industries, I would argue that there is a common trend at play. And the trend is this: **control over users' data and digital possessions and activity is rapidly moving from an asset to a liability**. Before, every bit of control you have was good: it gives you more flexibility to earn revenue, if not now then in the future. Now, every bit of control you have is a liability: you might be regulated because of it. If you exhibit control over your users' cryptocurrency, you are a money transmitter. If you have "sole discretion over fares, and can charge drivers a cancellation fee if they choose not to take a ride, prohibit drivers from picking up passengers not using the app and suspend or deactivate drivers' accounts", you are an employer. If you control your users' data, you're required to make sure you can argue just cause, have a compliance officer, and give your users access to download or delete the data.

If you are an application builder, and you are both lazy and fear legal trouble, there is one easy way to make sure that you violate none of the above new rules: *don't build applications that centralize control*. If you build a wallet where the user holds their private keys, you really are still "just a software provider". If you build a "decentralized Uber" that really is just a slick UI combining a payment system, a reputation system and a search engine, and don't control the components yourself, you really won't get hit by many of the same legal issues. If you build a website that just... doesn't collect data (Static web pages? But that's impossible!) you don't have to even think about the GDPR.

This kind of approach is of course not realistic for everyone. There will continue to be many cases where going without the conveniences of centralized control simply sacrifices too much for both developers and users, and there are also cases where the business model considerations mandate a more centralized approach (eg. it's easier to prevent non-paying users from using software if the software stays on your servers) win out. But we're definitely very far from having explored the full range of possibilities that more decentralized approaches offer.

Generally, unintended consequences of laws, discouraging entire categories of activity when one wanted to only surgically forbid a few specific things, are considered to be a bad thing. Here though, I would argue that the forced shift in developers' mindsets, from "I want to control more things just in case" to "I want to control fewer things just in case", also has many positive consequences. Voluntarily giving up control, and voluntarily taking steps to deprive oneself of the ability to do mischief, does not come naturally to many people, and while ideologically-driven decentralization-maximizing projects exist today, it's not at all obvious at first glance that such services will continue to dominate as the industry mainstreams. What this trend in regulation does, however, is that it gives a big nudge in favor of those applications that are willing to take the centralization-minimizing, user-sovereignty-maximizing "can't be evil" route.

Hence, even though these regulatory changes are arguably not pro-freedom, at least if one is concerned with the freedom of application developers, and the transformation of the internet into a subject of political focus is bound to have many negative knock-on effects, the particular trend of control becoming a liability is in a

strange way even more pro-cypherpunk (even if not intentionally!) than policies of maximizing total freedom for application developers would have been. Though the present-day regulatory landscape is very far from an optimal one from the point of view of almost anyone's preferences, it has unintentionally dealt the movement for minimizing unneeded centralization and maximizing users' control of their own assets, private keys and data a surprisingly strong hand to execute on its vision. And it would be highly beneficial to the movement to take advantage of it.

Fast Fourier Transforms

2019 May 12

[See all posts \(/\)](#)

Trigger warning: specialized mathematical topic

Special thanks to Karl Floersch for feedback

One of the more interesting algorithms in number theory is the Fast Fourier transform (FFT). FFTs are a key building block in many algorithms, including [extremely fast multiplication of large numbers](#) (<http://www.math.clemson.edu/~sgao/papers/GM10.pdf>), multiplication of polynomials, and extremely fast generation and recovery of [erasure codes](#) (<https://blog.ethereum.org/2014/08/16/secret-sharing-erasure-coding-guide-aspiring-dropbox-decentralizer>). Erasure codes in particular are highly versatile; in addition to their basic use cases in fault-tolerant data storage and recovery, erasure codes also have more advanced use cases such as [securing data availability in scalable blockchains](#) (<https://arxiv.org/pdf/1809.09044>) and [STARKs](#) (https://vitalik.ca/general/2017/11/09/starks_part_1.html). This article will go into what fast Fourier transforms are, and how some of the simpler algorithms for computing them work.

Background

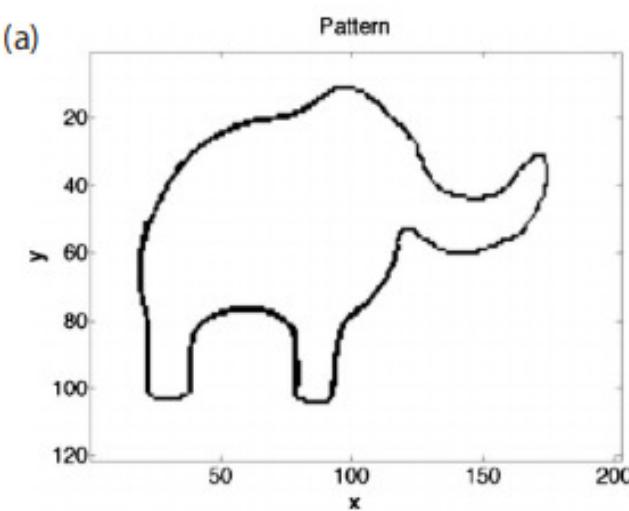
The original [Fourier transform](#) (https://en.wikipedia.org/wiki/Fourier_transform) is a mathematical operation that is often described as converting data between the "frequency domain" and the "time domain". What this means more precisely is that if you have a piece of data, then running the algorithm would come up with a collection of sine waves with different frequencies and amplitudes that, if you added them together, would approximate the original data. Fourier transforms can be used for such wonderful things as [expressing square orbits through epicycles](#) (<https://twitter.com/johncarlosbaez/status/1094671748501405696>) and [deriving a set of equations that can draw an elephant](#) (https://en.wikipedia.org/wiki/Fourier_transform):

$$x(t) = \sum_{k=0}^{\infty} (A_k^x \cos(kt) + B_k^x \sin(kt)), \quad (1)$$

$$y(t) = \sum_{k=0}^{\infty} (A_k^y \cos(kt) + B_k^y \sin(kt)), \quad (2)$$

Table I. The five complex parameters p_1, \dots, p_5 that encode the elephant including its wiggling trunk.

Parameter	Real part	Imaginary part
$p_1 = 50 - 30i$	$B_1^x = 50$	$B_1^y = -30$
$p_2 = 18 + 8i$	$B_2^x = 18$	$B_2^y = 8$
$p_3 = 12 - 10i$	$A_3^x = 12$	$B_3^y = -10$
$p_4 = -14 - 60i$	$A_4^x = -14$	$A_4^y = -60$
$p_5 = 40 + 20i$	Wiggle coeff. = 40	$x_{\text{eye}} = y_{\text{eye}} = 20$



Ok fine, Fourier transforms also have really important applications in signal processing, quantum mechanics, and other areas, and help make significant parts of the global economy happen. But come on, elephants are cooler.

Running the Fourier transform algorithm in the "inverse" direction would simply take the sine waves and add them together and compute the resulting values at as many points as you wanted to sample.

The kind of Fourier transform we'll be talking about in this post is a similar algorithm, except instead of being a *continuous* Fourier transform over *real or complex numbers*, it's a ***discrete Fourier transform*** over *finite fields* (see the "A Modular Math Interlude" section [here](https://vitalik.ca/general/2017/11/22/starks_part_2.html) (https://vitalik.ca/general/2017/11/22/starks_part_2.html)) for a refresher on what finite fields are). Instead of talking about converting between "frequency domain" and "time domain", here we'll talk about two different operations: *multi-point polynomial evaluation* (evaluating a degree $< N$ polynomial at N different points) and its inverse, *polynomial interpolation* (given the evaluations of a degree $< N$ polynomial at N different points, recovering the polynomial). For example, if we are operating in the prime field with modulus 5, then the polynomial $y = x^2 + 3$ (for convenience we can write the coefficients in increasing order: [3, 0, 1]) evaluated at the points [0, 1, 2] gives the values [3, 4, 2] (not [3, 4, 7] because we're operating in a finite field where the numbers wrap around at 5), and we can actually take the evaluations [3, 4, 2] and the coordinates they were evaluated at ([0, 1, 2]) to recover the original polynomial [3, 0, 1].

There are algorithms for both multi-point evaluation and interpolation that can do either operation in $O(N^2)$ time. Multi-point evaluation is simple: just separately evaluate the polynomial at each point. Here's python code for doing that:

```
def eval_poly_at(self, poly, x, modulus):
    y = 0
    power_of_x = 1
    for coefficient in poly:
        y += power_of_x * coefficient
        power_of_x *= x
    return y % modulus
```

The algorithm runs a loop going through every coefficient and does one thing for each coefficient, so it runs in $O(N)$ time. Multi-point evaluation involves doing this evaluation at N different points, so the total run time is $O(N^2)$.

Lagrange interpolation is more complicated (search for "Lagrange interpolation" [here](https://blog.ethereum.org/2014/08/16/secret-sharing-erasure-coding-guide-aspiring-dropbox-decentralizer/) (<https://blog.ethereum.org/2014/08/16/secret-sharing-erasure-coding-guide-aspiring-dropbox-decentralizer/>)) for a more detailed explanation). The key building block of the basic strategy is that for any domain D and point x , we can construct a polynomial that returns 1 for x and 0 for any value in D other than x . For example, if $D = [1, 2, 3, 4]$ and $x = 1$, the polynomial is:

$$y = \frac{(x - 2)(x - 3)(x - 4)}{(1 - 2)(1 - 3)(1 - 4)}$$

You can mentally plug in 1, 2, 3 and 4 to the above expression and verify that it returns 1 for $x = 1$ and 0 in the other three cases.

We can recover the polynomial that gives any desired set of outputs on the given domain by multiplying and adding these polynomials. If we call the above polynomial P_1 , and the equivalent ones for $x = 2, x = 3, x = 4, P_2, P_3$ and P_4 , then the polynomial that returns $[3, 1, 4, 1]$ on the domain $[1, 2, 3, 4]$ is simply $3 \cdot P_1 + P_2 + 4 \cdot P_3 + P_4$. Computing the P_i polynomials takes $O(N^2)$ time (you first construct the polynomial that returns to 0 on the entire domain, which takes $O(N^2)$ time, then separately divide it by $(x - x_i)$ for each x_i), and computing the linear combination takes another $O(N^2)$ time, so it's $O(N^2)$ runtime total.

What Fast Fourier transforms let us do, is make both multi-point evaluation and interpolation much faster.

Fast Fourier Transforms

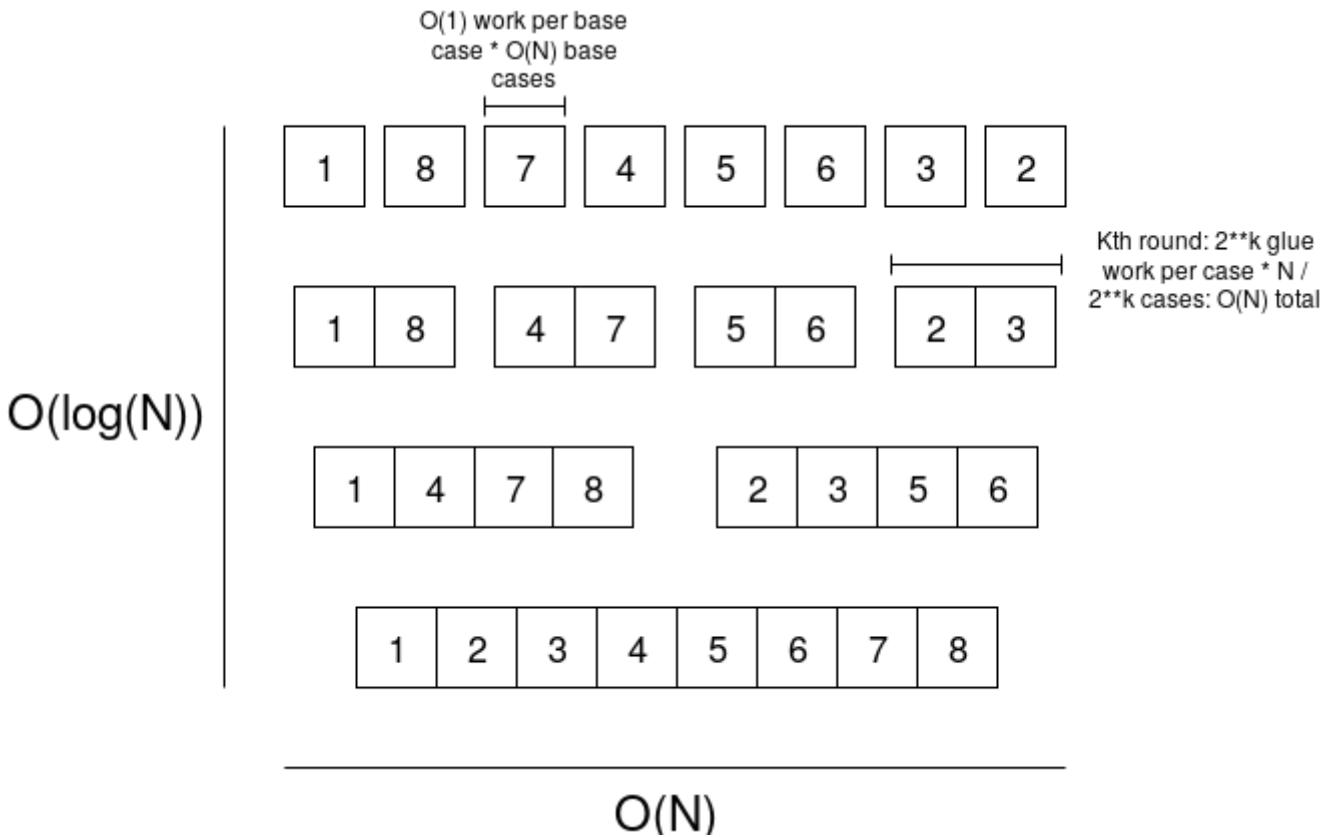
There is a price you have to pay for using this much faster algorithm, which is that you cannot choose any arbitrary field and any arbitrary domain. Whereas with Lagrange interpolation, you could choose whatever x coordinates and y coordinates you wanted, and whatever field you wanted (you could even do it over plain old real numbers), and you could get a polynomial that passes through them., with an FFT, you have to use a finite field, and the domain must be a *multiplicative subgroup* of the field (that is, a list of powers of some "generator" value). For example, you could use the finite field of integers modulo 337, and for the domain use $[1, 85, 148, 111, 336, 252, 189, 226]$ (that's the powers of 85 in the field, eg. $85^3 \% 337 = 111$; it stops at 226 because the next power of 85 cycles back to 1). Furthermore, the multiplicative subgroup must have size 2^n (there's ways to make it work for numbers of the form $2^m \cdot 3^n$ and possibly slightly higher prime powers but then it gets much more complicated and inefficient). The finite field of intergers modulo 59, for example, would not work, because there are only multiplicative subgroups of order 2, 29 and 58; 2 is too small to be interesting, and the factor 29 is far too large to be FFT-friendly. The symmetry that comes from multiplicative groups of size 2^n lets us create a recursive algorithm that quite cleverly calculate the results we need from a much smaller amount of work.

To understand the algorithm and why it has a low runtime, it's important to understand the general concept of recursion. A recursive algorithm is an algorithm that has two cases: a "base case" where the input to the algorithm is small enough that you can give the output directly, and the "recursive case" where the required computation consists of some "glue computation" plus one or more uses of the same algorithm to smaller inputs. For example, you might have seen recursive algorithms being used for sorting lists. If you have a list (eg. $[1, 8, 7, 4, 5, 6, 3, 2, 9]$), then you can sort it using the following procedure:

- If the input has one element, then it's already "sorted", so you can just return the input.
- If the input has more than one element, then separately sort the first half of the list and the second half of the list, and then merge the two sorted sub-lists (call them A and B) as follows. Maintain two counters, apos and bpos , both starting at zero, and maintain an output list, which starts empty. Until either apos or bpos is at the end of the corresponding list, check if $A[\text{apos}]$ or $B[\text{bpos}]$ is smaller. Whichever is smaller, add that value to the end of the output list, and increase that counter by 1. Once this is done, add the rest of whatever list has not been fully processed to the end of the output list, and return the output list.

Note that the "glue" in the second procedure has runtime $O(N)$: if each of the two sub-lists has N elements, then you need to run through every item in each list once, so it's $O(N)$ computation total. So the algorithm as a whole works by taking a problem of size N , and breaking it up into two problems of size $\frac{N}{2}$, plus $O(N)$ of "glue" execution. There is a theorem called the [Master Theorem](#)

([https://en.wikipedia.org/wiki/Master_theorem_\(analysis_of_algorithms%29](https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms%29)) that lets us compute the total runtime of algorithms like this. It has many sub-cases, but in the case where you break up an execution of size N into k sub-cases of size $\frac{N}{k}$ with $O(N)$ glue (as is the case here), the result is that the execution takes time $O(N \cdot \log(N))$.



An FFT works in the same way. We take a problem of size N , break it up into two problems of size $\frac{N}{2}$, and do $O(N)$ glue work to combine the smaller solutions into a bigger solution, so we get $O(N \cdot \log(N))$ runtime total - *much faster* than $O(N^2)$. Here is how we do it. I'll describe first how to use an FFT for multi-point evaluation (ie. for some domain D and polynomial P , calculate $P(x)$ for every x in D), and it turns out that you can use the same algorithm for interpolation with a minor tweak.

Suppose that we have an FFT where the given domain is the powers of x in some field, where $x^{2^k} = 1$ (eg. in the case we introduced above, the domain is the powers of 85 modulo 337, and $85^{2^3} = 1$). We have some polynomial, eg. $y = 6x^7 + 2x^6 + 9x^5 + 5x^4 + x^3 + 4x^2 + x + 3$ (we'll write it as $p = [3, 1, 4, 1, 5, 9, 2, 6]$). We want to evaluate this polynomial at each point in the domain, ie. at each of the eight powers of 85. Here is what we do. First, we break up the polynomial into two parts, which we'll call evens and odds: evens = [3, 4, 5, 2] and odds = [1, 1, 9, 6] (or evens = $2x^3 + 5x^2 + 4x + 3$ and odds = $6x^3 + 9x^2 + x + 1$; yes, this is just taking the even-degree coefficients and the odd-degree coefficients). Now, we note a mathematical observation: $p(x) = \text{evens}(x^2) + x \cdot \text{odds}(x^2)$ and $p(-x) = \text{evens}(x^2) - x \cdot \text{odds}(x^2)$ (think about this for yourself and make sure you understand it before going further).

Here, we have a nice property: evens and odds are both polynomials half the size of p , and furthermore, the set of possible values of x^2 is only half the size of the original domain, because there is a two-to-one correspondence: x and $-x$ are both part of D (eg. in our current domain [1, 85, 148, 111, 336, 252, 189, 226], 1 and 336 are negatives of each other, as $336 = -1 \% 337$, as are (85, 252), (148, 189) and (111, 226)). And x and $-x$ always both have the same square. Hence, we can use an FFT to compute the result of $\text{evens}(x)$ for every x in the smaller domain consisting of squares of numbers in the original domain ([1, 148, 336, 189]), and we can do the same for odds. And voila, we've reduced a size- N problem into half-size problems.

The "glue" is relatively easy (and $O(N)$ in runtime): we receive the evaluations of evens and odds as size- $\frac{N}{2}$ lists, so we simply do $p[i] = \text{evens_result}[i] + \text{domain}[i] \cdot \text{odds_result}[i]$ and $p[\frac{N}{2} + i] = \text{evens_result}[i] - \text{domain}[i] \cdot \text{odds_result}[i]$ for each index i .

Here's the full code:

```
def fft(vals, modulus, domain):
    if len(vals) == 1:
        return vals
    L = fft(vals[::2], modulus, domain[::2])
    R = fft(vals[1::2], modulus, domain[::2])
    o = [0 for i in vals]
    for i, (x, y) in enumerate(zip(L, R)):
        y_times_root = y * domain[i]
        o[i] = (x + y_times_root) % modulus
        o[i + len(L)] = (x - y_times_root) % modulus
    return o
```

We can try running it:

```
>>> fft([3,1,4,1,5,9,2,6], 337, [1, 85, 148, 111, 336, 252, 189, 226])
[31, 70, 109, 74, 334, 181, 232, 4]
```

And we can check the result; evaluating the polynomial at the position 85, for example, actually does give the result 70. Note that this only works if the domain is "correct"; it needs to be of the form $[x^i \% \text{modulus} \text{ for } i \text{ in range}(n)]$ where $x^n = 1$.

An inverse FFT is surprisingly simple:

```
def inverse_fft(vals, modulus, domain):
    vals = fft(vals, modulus, domain)
    return [x * modular_inverse(len(vals), modulus) % modulus for x in [vals[0]] + vals[1:][]]
```

Basically, run the FFT again, but reverse the result (except the first item stays in place) and divide every value by the length of the list.

```
>>> domain = [1, 85, 148, 111, 336, 252, 189, 226]
>>> def modular_inverse(x, n): return pow(x, n - 2, n)
>>> values = fft([3,1,4,1,5,9,2,6], 337, domain)
>>> values
[31, 70, 109, 74, 334, 181, 232, 4]
>>> inverse_fft(values, 337, domain)
[3, 1, 4, 1, 5, 9, 2, 6]
```

Now, what can we use this for? Here's one fun use case: we can use FFTs to multiply numbers very quickly. Suppose we wanted to multiply 1253 by 1895. Here is what we would do. First, we would convert the problem into one that turns out to be slightly easier: multiply the *polynomials* $[3, 5, 2, 1]$ by $[5, 9, 8, 1]$ (that's just the digits

of the two numbers in increasing order), and then convert the answer back into a number by doing a single pass to carry over tens digits. We can multiply polynomials with FFTs quickly, because it turns out that if you convert a polynomial into *evaluation form* (ie. $f(x)$ for every x in some domain D), then you can multiply two polynomials simply by multiplying their evaluations. So what we'll do is take the polynomials representing our two numbers in *coefficient form*, use FFTs to convert them to evaluation form, multiply them pointwise, and convert back:

```
>>> p1 = [3,5,2,1,0,0,0,0]
>>> p2 = [5,9,8,1,0,0,0,0]
>>> x1 = fft(p1, 337, domain)
>>> x1
[11, 161, 256, 10, 336, 100, 83, 78]
>>> x2 = fft(p2, 337, domain)
>>> x2
[23, 43, 170, 242, 3, 313, 161, 96]
>>> x3 = [(v1 * v2) % 337 for v1, v2 in zip(x1, x2)]
>>> x3
[253, 183, 47, 61, 334, 296, 220, 74]
>>> inverse_fft(x3, 337, domain)
[15, 52, 79, 66, 30, 10, 1, 0]
```

This requires three FFTs (each $O(N \cdot \log(N))$ time) and one pointwise multiplication ($O(N)$ time), so it takes $O(N \cdot \log(N))$ time altogether (technically a little bit more than $O(N \cdot \log(N))$, because for very big numbers you would need replace 337 with a bigger modulus and that would make multiplication harder, but close enough). This is *much faster* than schoolbook multiplication, which takes $O(N^2)$ time:

$$\begin{array}{r}
 & 3 & 5 & 2 & 1 \\
 \hline
 5 | & 15 & 25 & 10 & 5 \\
 9 | & & 27 & 45 & 18 & 9 \\
 8 | & & & 24 & 40 & 16 & 8 \\
 1 | & & & & 3 & 5 & 2 & 1 \\
 \hline
 & 15 & 52 & 79 & 66 & 30 & 10 & 1
 \end{array}$$

So now we just take the result, and carry the tens digits over (this is a "walk through the list once and do one thing at each point" algorithm so it takes $O(N)$ time):

```
[15, 52, 79, 66, 30, 10, 1, 0]
[ 5, 53, 79, 66, 30, 10, 1, 0]
[ 5, 3, 84, 66, 30, 10, 1, 0]
[ 5, 3, 4, 74, 30, 10, 1, 0]
[ 5, 3, 4, 4, 37, 10, 1, 0]
[ 5, 3, 4, 4, 7, 13, 1, 0]
[ 5, 3, 4, 4, 7, 3, 2, 0]
```

And if we read the digits from top to bottom, we get 2374435. Let's check the answer....

```
>>> 1253 * 1895
2374435
```

Yay! It worked. In practice, on such small inputs, the difference between $O(N \cdot \log(N))$ and $O(N^2)$ isn't *that* large, so schoolbook multiplication is faster than this FFT-based multiplication process just because the algorithm is simpler, but on large inputs it makes a really big difference.

But FFTs are useful not just for multiplying numbers; as mentioned above, polynomial multiplication and multi-point evaluation are crucially important operations in implementing erasure coding, which is a very important technique for building many kinds of redundant fault-tolerant systems. If you like fault tolerance and you like efficiency, FFTs are your friend.

FFTs and binary fields

Prime fields are not the only kind of finite field out there. Another kind of finite field (really a special case of the more general concept of an *extension field*, which are kind of like the finite-field equivalent of complex numbers) are binary fields. In a binary field, each element is expressed as a polynomial where all of the entries are 0 or 1, eg. $x^3 + x + 1$. Adding polynomials is done modulo 2, and subtraction is the same as addition (as $-1 = 1 \bmod 2$). We select some irreducible polynomial as a modulus (eg. $x^4 + x + 1$; $x^4 + 1$ would not work because $x^4 + 1$ can be factored into $(x^2 + 1) \cdot (x^2 + 1)$ so it's not "irreducible"); multiplication is done modulo that modulus. For example, in the binary field mod $x^4 + x + 1$, multiplying $x^2 + 1$ by $x^3 + 1$ would give $x^5 + x^3 + x^2 + 1$ if you just do the multiplication, but $x^5 + x^3 + x^2 + 1 = (x^4 + x + 1) \cdot x + (x^3 + x + 1)$, so the result is the remainder $x^3 + x + 1$.

We can express this example as a multiplication table. First multiply $[1, 0, 0, 1]$ (ie. $x^3 + 1$) by $[1, 0, 1]$ (ie. $x^2 + 1$):

$$\begin{array}{r} 1 \ 0 \ 0 \ 1 \\ \hline 1 \ | \ 1 \ 0 \ 0 \ 1 \\ 0 \ | \ \quad 0 \ 0 \ 0 \ 0 \\ 1 \ | \ \quad \quad 1 \ 0 \ 0 \ 1 \\ \hline 1 \ 0 \ 1 \ 1 \ 0 \ 1 \end{array}$$

The multiplication result contains an x^5 term so we can subtract $(x^4 + x + 1) \cdot x$:

$$\begin{array}{r} 1 \ 0 \ 1 \ 1 \ 0 \ 1 \\ - \ 1 \ 1 \ 0 \ 0 \ 1 \quad [(x^4 + x + 1) \text{ shifted right by one to reflect being multiplied by } x] \\ \hline 1 \ 1 \ 0 \ 1 \ 0 \ 0 \end{array}$$

And we get the result, $[1, 1, 0, 1]$ (or $x^3 + x + 1$).

+ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	* 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 0 3 2 5 4 7 6 9 8 11 10 13 12 15 14	1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
2 2 3 0 1 6 7 4 5 10 11 8 9 14 15 12 13	2 0 2 4 6 8 10 12 14 3 1 7 5 11 9 15 13
3 3 2 1 0 7 6 5 4 11 10 9 8 15 14 13 12	3 0 3 6 5 12 15 10 9 11 8 13 14 7 4 1 2
4 4 5 6 7 0 1 2 3 12 13 14 15 8 9 10 11	4 0 4 8 12 3 7 11 15 6 2 14 10 5 1 13 9
5 5 4 7 6 1 0 3 2 13 12 15 14 9 8 11 10	5 0 5 10 15 7 2 13 8 14 11 4 1 9 12 3 6
6 6 7 4 5 2 3 0 1 14 15 12 13 10 11 8 9	6 0 6 12 10 11 13 7 1 5 3 9 15 14 8 2 4
7 7 6 5 4 3 2 1 0 15 14 13 12 11 10 9 8	7 0 7 14 9 15 8 1 6 13 10 3 4 2 5 12 11
8 8 9 10 11 12 13 14 15 0 1 2 3 4 5 6 7	8 0 8 3 11 6 14 5 13 12 4 15 7 10 2 9 1
9 9 8 11 10 13 12 15 14 1 0 3 2 5 4 7 6	9 0 9 1 8 2 11 3 10 4 13 5 12 6 15 7 14
10 10 11 8 9 14 15 12 13 2 3 0 1 6 7 4 5	10 0 10 7 13 14 4 9 3 15 5 8 2 1 11 6 12
11 11 10 9 8 15 14 13 12 3 2 1 0 7 6 5 4	11 0 11 5 14 10 1 15 4 7 12 2 9 13 6 8 3
12 12 13 14 15 8 9 10 11 4 5 6 7 0 1 2 3	12 0 12 11 7 5 9 14 2 10 6 1 13 15 3 4 8
13 13 12 15 14 9 8 11 10 5 4 7 6 1 0 3 2	13 0 13 9 4 1 12 8 5 2 15 11 6 3 14 10 7
14 14 15 12 13 10 11 8 9 6 7 4 5 2 3 0 1	14 0 14 15 1 13 3 2 12 9 7 6 8 4 10 11 5
15 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	15 0 15 13 2 9 6 4 11 1 14 12 3 8 7 5 10

Addition and multiplication tables for the binary field mod $x^4 + x + 1$. Field elements are expressed as integers converted from binary (eg. $x^3 + x^2 \rightarrow 1100 \rightarrow 12$)

Binary fields are interesting for two reasons. First of all, if you want to erasure-code binary data, then binary fields are really convenient because N bytes of data can be directly encoded as a binary field element, and any binary field elements that you generate by performing computations on it will also be N bytes long. You cannot do this with prime fields because prime fields' size is not exactly a power of two; for example, you could encode every 2 bytes as a number from 0..65536 in the prime field modulo 65537 (which is prime), but if you do an FFT on these values, then the output could contain 65536, which cannot be expressed in two bytes. Second, the fact that addition and subtraction become the same operation, and $1 + 1 = 0$, create some "structure" which leads to some very interesting consequences. One particularly interesting, and useful, oddity of binary fields is the "[freshman's dream](https://en.wikipedia.org/wiki/Freshman%27s_dream) (https://en.wikipedia.org/wiki/Freshman%27s_dream)" theorem: $(x + y)^2 = x^2 + y^2$ (and the same for exponents 4, 8, 16... basically any power of two).

But if you want to use binary fields for erasure coding, and do so efficiently, then you need to be able to do Fast Fourier transforms over binary fields. But then there is a problem: in a binary field, *there are no (nontrivial) multiplicative groups of order 2^n* . This is because the multiplicative groups are all order $2^n - 1$. For example, in the binary field with modulus $x^4 + x + 1$, if you start calculating successive powers of $x + 1$, you cycle back to 1 after 15 steps - not 16. The reason is that the total number of elements in the field is 16, but one of them is zero, and you're never going to reach zero by multiplying any nonzero value by itself in a field, so the powers of $x + 1$ cycle through every element but zero, so the cycle length is 15, not 16. So what do we do?

The reason we needed the domain to have the "structure" of a multiplicative group with 2^n elements before is that we needed to reduce the size of the domain by a factor of two by squaring each number in it: the domain [1, 85, 148, 111, 336, 252, 189, 226] gets reduced to [1, 148, 336, 189] because 1 is the square of both 1 and 336, 148 is the square of both 85 and 252, and so forth. But what if in a binary field there's a different way to halve the size of a domain? It turns out that there is: given a domain containing 2^k values, including zero (technically the domain must be a [subspace](https://en.wikipedia.org/wiki/Linear_subspace) (https://en.wikipedia.org/wiki/Linear_subspace)), we can construct a half-sized new domain D' by taking $x \cdot (x + k)$ for x in D using some specific k in D. Because the original domain is a subspace, since k is in the domain, any x in the domain has a corresponding $x + k$ also in the domain, and the function $f(x) = x \cdot (x + k)$ returns the same value for x and $x + k$ so we get the same kind of two-to-one correspondence that squaring gives us.

x	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$x \cdot (x + 1)$	0	0	6	6	7	7	1	1	4	4	2	2	3	3	5	5

So now, how do we do an FFT on top of this? We'll use the same trick, converting a problem with an N -sized polynomial and N -sized domain into two problems each with an $\frac{N}{2}$ -sized polynomial and $\frac{N}{2}$ -sized domain, but this time using different equations. We'll convert a polynomial p into two polynomials `evens` and `odds` such that $p(x) = \text{evens}(x \cdot (k - x)) + x \cdot \text{odds}(x \cdot (k - x))$. Note that for the `evens` and `odds` that we find, it will also be true that $p(x + k) = \text{evens}(x \cdot (k - x)) + (x + k) \cdot \text{odds}(x \cdot (k - x))$. So we can then recursively do an FFT to `evens` and `odds` on the reduced domain [$x \cdot (k - x)$ for x in D], and then we use these two formulas to get the answers for two "halves" of the domain, one offset by k from the other.

Converting p into `evens` and `odds` as described above turns out to itself be nontrivial. The "naive" algorithm for doing this is itself $O(N^2)$, but it turns out that in a binary field, we can use the fact that $(x^2 - kx)^2 = x^4 - k^2 \cdot x^2$, and more generally $(x^2 - kx)^{2^i} = x^{2^{i+1}} - k^{2^i} \cdot x^{2^i}$, to create yet another recursive algorithm to do this in $O(N \cdot \log(N))$ time.

And if you want to do an *inverse* FFT, to do interpolation, then you need to run the steps in the algorithm in reverse order. You can find the complete code for doing this here:

https://github.com/ethereum/research/tree/master/binary_fft

(https://github.com/ethereum/research/tree/master/binary_fft), and a paper with details on more optimal algorithms here: <http://www.math.clemson.edu/~sgao/papers/GM10.pdf> (<http://www.math.clemson.edu/~sgao/papers/GM10.pdf>).

So what do we get from all of this complexity? Well, we can try running the implementation, which features both a "naive" $O(N^2)$ multi-point evaluation and the optimized FFT-based one, and time both. Here are my results:

```
>>> import binary_fft as b
>>> import time, random
>>> f = b.BinaryField(1033)
>>> poly = [random.randrange(1024) for i in range(1024)]
>>> a = time.time(); x1 = b._simple_ft(f, poly); time.time() - a
0.5752472877502441
>>> a = time.time(); x2 = b.fft(f, poly, list(range(1024))); time.time() - a
0.03820443153381348
```

And as the size of the polynomial gets larger, the naive implementation (`_simple_ft`) gets slower much more quickly than the FFT:

```
>>> f = b.BinaryField(2053)
>>> poly = [random.randrange(2048) for i in range(2048)]
>>> a = time.time(); x1 = b._simple_ft(f, poly); time.time() - a
2.2243144512176514
>>> a = time.time(); x2 = b.fft(f, poly, list(range(2048))); time.time() - a
0.07896280288696289
```

And voila, we have an efficient, scalable way to multi-point evaluate and interpolate polynomials. If we want to use FFTs to recover erasure-coded data where we are *missing* some pieces, then algorithms for this also exist (<https://ethresear.ch/t/reed-solomon-erasure-code-recovery-in-n-log-2-n-time-with-ffts/3039>), though they are somewhat less efficient than just doing a single FFT. Enjoy!

Sidechains vs Plasma vs Sharding

2019 Jun 12

[See all posts \(/\)](#)

Special thanks to Jinglan Wang for review and feedback

One question that often comes up is: how exactly is sharding different from sidechains or Plasma? All three architectures seem to involve a hub-and-spoke architecture with a central "main chain" that serves as the consensus backbone of the system, and a set of "child" chains containing actual user-level transactions. Hashes from the child chains are usually periodically published into the main chain (sharded chains with no hub are theoretically possible but haven't been done so far; this article will not focus on them, but the arguments are similar). Given this fundamental similarity, why go with one approach over the others?

Distinguishing sidechains from Plasma is simple. Plasma chains are sidechains that have a non-custodial property: if there is any error in the Plasma chain, then the error can be detected, and users can safely exit the Plasma chain and prevent the attacker from doing any lasting damage. The only cost that users suffer is that they must wait for a challenge period and pay some higher transaction fees on the (non-scalable) base chain. Regular sidechains do not have this safety property, so they are less secure. However, designing Plasma chains is in many cases much harder, and one could argue that for many low-value applications the security is not worth the added complexity.

So what about Plasma versus sharding? The key technical difference has to do with the notion of **tight coupling**. Tight coupling is a property of sharding, but NOT a property of sidechains or Plasma, that says that the validity of the main chain ("beacon chain" in Ethereum 2.0) is inseparable from the validity of the child chains. That is, a child chain block that specifies an invalid main chain block as a dependency is by definition invalid, and more importantly a main chain block that includes an invalid child chain block is by definition invalid.

In non-sharded blockchains, this idea that the canonical chain (ie. the chain that everyone accepts as representing the "real" history) is *by definition* fully available and valid also applies; for example in the case of Bitcoin and Ethereum one typically says that the canonical chain is the "longest valid chain" (or, more pedantically, the "heaviest valid and available chain"). In sharded blockchains, this idea that the canonical chain is the heaviest valid and available chain *by definition* also applies, with the validity and availability requirement applying to both the main chain and shard chains. The new challenge that a sharded system has, however, is that users have no way of fully verifying the validity and availability of any given chain *directly*, because there is too much data. The challenge of engineering sharded chains is to get around this limitation by giving users a maximally trustless and practical *indirect* means to verify which chains are fully available and valid, so that they can still determine which chain is canonical. In practice, this includes techniques like committees, SNARKs/STARKs, fisherman schemes and [fraud and data availability proofs](#) (<https://arxiv.org/abs/1809.09044>).

If a chain structure does not have this tight-coupling property, then it is arguably not a layer-1 sharding scheme, but rather a layer-2 system sitting on top of a non-scalable layer-1 chain. Plasma is not a tightly-coupled system: an invalid Plasma block absolutely can have its header be committed into the main Ethereum

chain, because the Ethereum base layer has no idea that it represents an invalid Plasma block, or even that it represents a Plasma block at all; all that it sees is a transaction containing a small piece of data. However, the consequences of a single Plasma chain failing are localized to within that Plasma chain.

Sharding	Try really hard to ensure total validity/availability of every part of the system	
Plasma	Accept local faults but try to limit their consequences	

However, if you try to analyze the process of *how* users perform the "indirect validation" procedure to determine if the chain they are looking at is fully valid and available without downloading and executing the whole thing, one can find more similarities with how Plasma works. For example, a common technique used to prevent availability issues is fishermen: if a node sees a given piece of a block as unavailable, it can publish a challenge claiming this, creating a time period within which anyone can publish that piece of data. If a block goes unchallenged for long enough, the blocks and all blocks that cite it as a dependency can be reverted. This seems fundamentally similar to Plasma, where if a block is unavailable users can publish a message to the main chain to exit their state in response. Both techniques eventually buckle under pressure in the same way: if there are too many false challenges in a sharded system, then users cannot keep track of whether or not all of the availability challenges have been answered, and if there are too many availability challenges in a Plasma system then the main chain could get overwhelmed as the exits fill up the chain's block size limit. In both cases, it seems like there's a system that has nominally $O(C^2)$ scalability (where C is the computing power of one node) but where scalability falls to $O(C)$ in the event of an attack. However, sharding has more defenses against this.

First of all, modern sharded designs use randomly sampled committees, so one cannot easily dominate even one committee enough to produce a fake block unless one has a large portion (perhaps $> \frac{1}{3}$) of the entire validator set of the chain. Second, there are better strategies to handling data availability than fishermen: data availability proofs. In a scheme using data availability proofs, if a block is *unavailable*, then clients' data availability checks will fail and clients will see that block as unavailable. If the block is *invalid*, then even a single fraud proof will convince them of this fact for an entire block. An $O(1)$ -sized fraud proof can convince a client of the invalidity of an $O(C)$ -sized block, and so $O(C)$ data suffices to convince a client of the invalidity of $O(C^2)$ data (this is in the worst case where the client is dealing with N sister blocks all with the same parent of which only one is valid; in more likely cases, one single fraud proof suffices to prove invalidity of an entire invalid chain). Hence, sharded systems are theoretically less vulnerable to being overwhelmed by denial-of-service attacks than Plasma chains.

Second, sharded chains provide stronger guarantees in the face of large and majority attackers (with more than $\frac{1}{3}$ or even $\frac{1}{2}$ of the validator set). A Plasma chain can always be successfully attacked by a 51% attack on the main chain that censors exits; a sharded chain cannot. This is because data availability proofs and fraud proofs happen *inside the client*, rather than *inside the chain*, so they cannot be censored by 51% attacks. Third, the defenses provided by sharded chains are easier to generalize; Plasma's model of exits requires state to be separated into discrete pieces each of which is in the interest of any single actor to maintain, whereas sharded chains relying on data availability proofs, fraud proofs, fishermen and random sampling are theoretically universal.

So there really is a large difference between validity and availability guarantees that are provided at layer 2, which are limited and more complex as they require explicit reasoning about incentives and which party has an interest in which pieces of state, and guarantees that are provided by a layer 1 system that is committed to

fully satisfying them.

But Plasma chains also have large advantages too. First, they can be iterated and new designs can be implemented more quickly, as each Plasma chain can be deployed separately without coordinating the rest of the ecosystem. Second, sharding is inherently more fragile, as it attempts to guarantee absolute and total availability and validity of some quantity of data, and this quantity must be set in the protocol; too little, and the system has less scalability than it could have had, too much, and the entire system risks breaking. The maximum safe level of scalability also depends on the number of users of the system, which is an unpredictable variable. Plasma chains, on the other hand, allow different users to make different tradeoffs in this regard, and allow users to adjust more flexibly to changes in circumstances.

Single-operator Plasma chains can also be used to offer more privacy than sharded systems, where all data is public. Even where privacy is not desired, they are potentially more efficient, because the total data availability requirement of sharded systems requires a large extra level of redundancy as a safety margin. In Plasma systems, on the other hand, data requirements for each piece of data can be minimized, to the point where in the long term each individual piece of data may only need to be replicated a few times, rather than a thousand times as is the case in sharded systems.

Hence, in the long term, a hybrid system where a sharded base layer exists, and Plasma chains exist on top of it to provide further scalability, seems like the most likely approach, more able to serve different groups' of users need than sole reliance on one strategy or the other. And it is unfortunately *not* the case that at a sufficient level of advancement Plasma and sharding collapse into the same design; the two are in some key ways irreducibly different (eg. the data availability checks made by clients in sharded systems *cannot* be moved to the main chain in Plasma because these checks only work if they are done subjectively and based on private information). But both scalability solutions (as well as state channels!) have a bright future ahead of them.

The Dawn of Hybrid Layer 2 Protocols

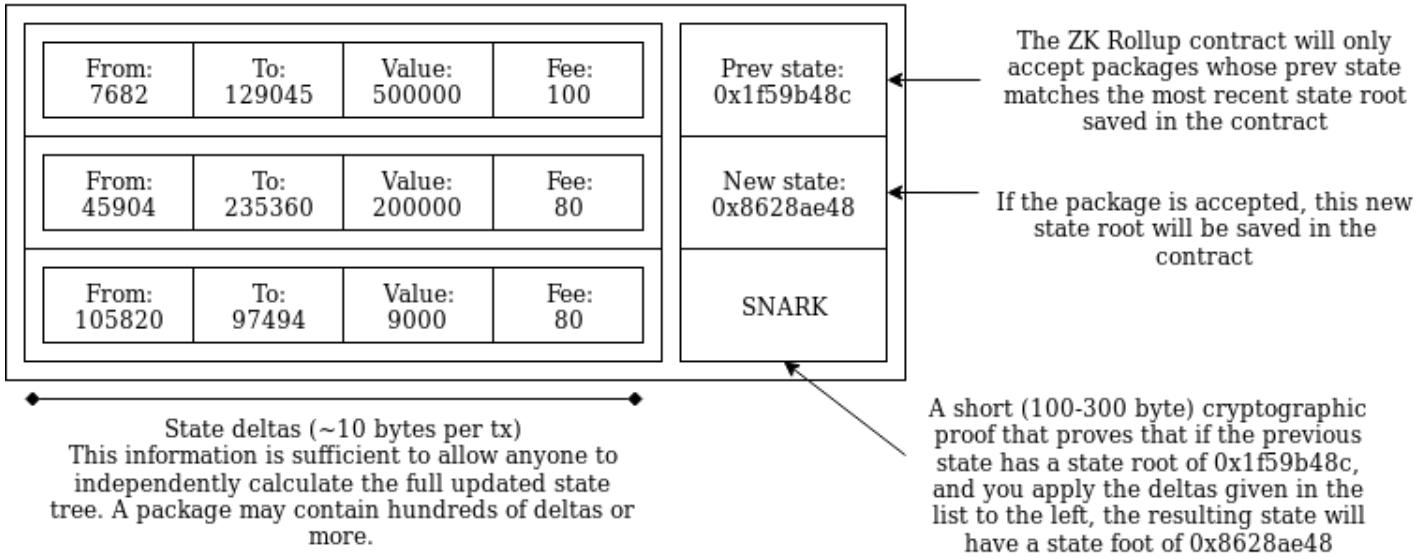
2019 Aug 28

[See all posts \(/\)](#)

Special thanks to the Plasma Group team for review and feedback

Current approaches to layer 2 scaling - basically, Plasma and state channels - are increasingly moving from theory to practice, but at the same time it is becoming easier to see the inherent challenges in treating these techniques as a fully fledged scaling solution for Ethereum. Ethereum was arguably successful in large part because of its very easy developer experience: you write a program, publish the program, and anyone can interact with it. Designing a state channel or Plasma application, on the other hand, relies on a lot of explicit reasoning about incentives and application-specific development complexity. State channels work well for specific use cases such as repeated payments between the same two parties and two-player games (as successfully implemented in [Celer](https://www.celer.network/) (<https://www.celer.network/>)), but more generalized usage is proving challenging. Plasma, particularly [Plasma Cash](https://www.learnplasma.org/en/learn/cash.html) (<https://www.learnplasma.org/en/learn/cash.html>), can work well for payments, but generalization similarly incurs challenges: even implementing a decentralized exchange requires clients to store much more history data, and generalizing to Ethereum-style smart contracts on Plasma seems extremely difficult.

But at the same time, there is a resurgence of a forgotten category of "semi-layer-2" protocols - a category which promises less extreme gains in scaling, but with the benefit of much easier generalization and more favorable security models. A [long-forgotten blog post from 2014](https://blog.ethereum.org/2014/09/17/scalability-part-1-building-top/) (<https://blog.ethereum.org/2014/09/17/scalability-part-1-building-top/>) introduced the idea of "shadow chains", an architecture where block data is published on-chain, but blocks are not *verified* by default. Rather, blocks are tentatively accepted, and only finalized after some period of time (eg. 2 weeks). During those 2 weeks, a tentatively accepted block can be challenged; only then is the block verified, and if the block proves to be invalid then the chain from that block on is reverted, and the original publisher's deposit is penalized. The contract does not keep track of the full state of the system; it only keeps track of the state root, and users themselves can calculate the state by processing the data submitted to the chain from start to head. A more recent proposal, [ZK Rollup](https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477) (<https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477>), does the same thing without challenge periods, by using ZK-SNARKs to verify blocks' validity.



Anatomy of a ZK Rollup package that is published on-chain. Hundreds of "internal transactions" that affect the state (ie. account balances) of the ZK Rollup system are compressed into a package that contains ~10 bytes per internal transaction that specifies the state transitions, plus a ~100-300 byte SNARK proving that the transitions are all valid.

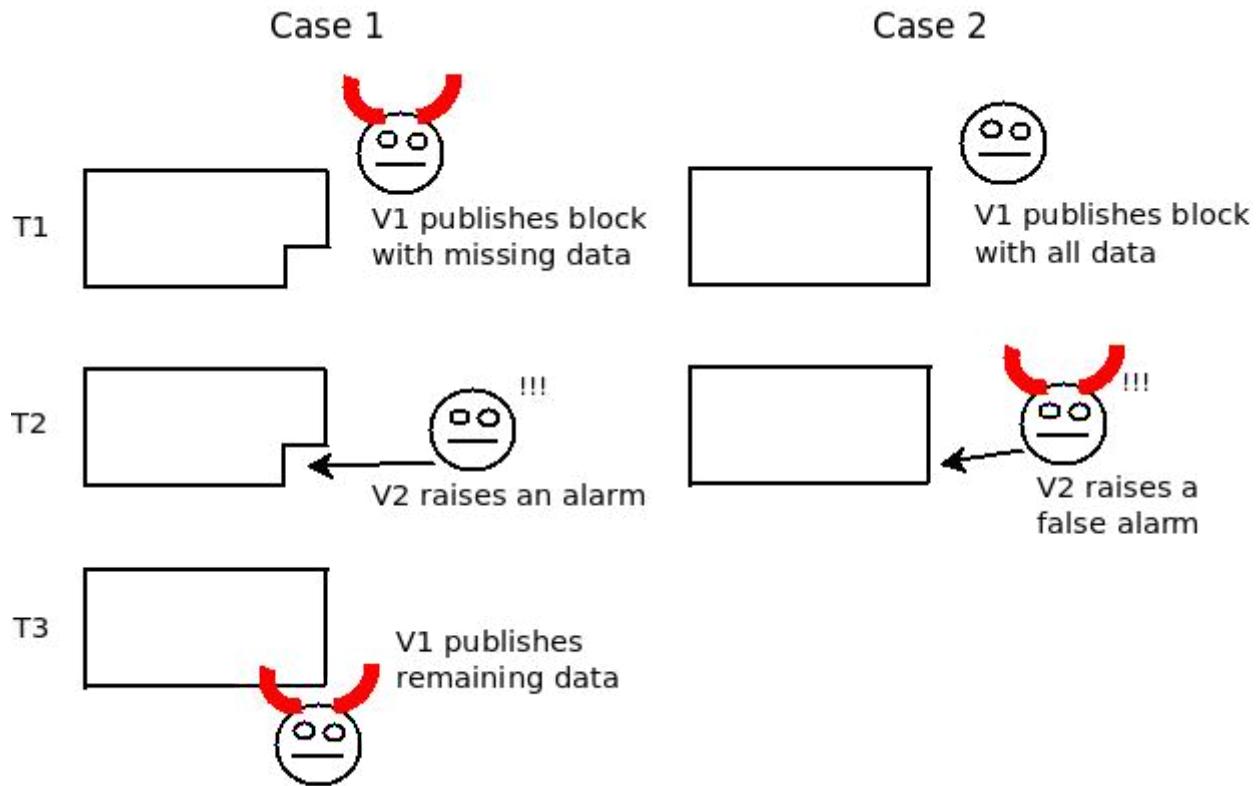
In both cases, the main chain is used to verify data *availability*, but does not (directly) verify block *validity* or perform any significant computation, unless challenges are made. This technique is thus not a jaw-droppingly huge scalability gain, because the on-chain data overhead eventually presents a bottleneck, but it is nevertheless a very significant one. Data is cheaper than computation, and there are ways to compress transaction data very significantly, particularly because the great majority of data in a transaction is the signature and many signatures can be compressed into one through many forms of aggregation. ZK Rollup promises 500 tx/sec, a 30x gain over the Ethereum chain itself, by compressing each transaction to a mere ~10 bytes; signatures do not need to be included because their validity is verified by the zero-knowledge proof. With BLS aggregate signatures a similar throughput can be achieved in shadow chains (more recently called "optimistic rollup" to highlight its similarities to ZK Rollup). The upcoming Istanbul hard fork (<https://eth.wiki/en/roadmap/istanbul>) will reduce the gas cost of data from 68 per byte to 16 per byte, increasing the throughput of these techniques by another 4x (that's **over 2000 transactions per second**).

So what is the benefit of data on-chain techniques such as ZK/optimistic rollup versus data off-chain techniques such as Plasma? First of all, there is no need for semi-trusted operators. In ZK Rollup, because validity is verified by cryptographic proofs there is literally no way for a package submitter to be malicious (depending on the setup, a malicious submitter may cause the system to halt for a few seconds, but this is the most harm that can be done). In optimistic rollup, a malicious submitter can publish a bad block, but the next submitter will immediately challenge that block before publishing their own. In both ZK and optimistic rollup, enough data is published on chain to allow anyone to compute the complete internal state, simply by

processing all of the submitted deltas in order, and there is no "data withholding attack" that can take this property away. Hence, becoming an operator can be fully permissionless; all that is needed is a security deposit (eg. 10 ETH) for anti-spam purposes.

Second, optimistic rollup particularly is vastly easier to generalize; the state transition function in an optimistic rollup system can be literally anything that can be computed within the gas limit of a single block (including the Merkle branches providing the parts of the state needed to verify the transition). ZK Rollup is theoretically generalizable in the same way, though in practice making ZK SNARKs over general-purpose computation (such as EVM execution) is very difficult, at least for now. Third, optimistic rollup is much easier to build clients for, as there is less need for second-layer networking infrastructure; more can be done by just scanning the blockchain.

But where do these advantages come from? The answer lies in a highly technical issue known as the *data availability problem* (see [note](https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding) (<https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding>), [video](https://www.youtube.com/watch?v=OJT_fR7wexw) (https://www.youtube.com/watch?v=OJT_fR7wexw)). Basically, there are two ways to try to cheat in a layer-2 system. The first is to publish invalid data to the blockchain. The second is to not publish data at all (eg. in Plasma, publishing the root hash of a new Plasma block to the main chain but without revealing the contents of the block to anyone). Published-but-invalid data is very easy to deal with, because once the data is published on-chain there are multiple ways to figure out unambiguously whether or not it's valid, and an invalid submission is unambiguously invalid so the submitter can be heavily penalized. Unavailable data, on the other hand, is much harder to deal with, because even though unavailability can be detected if challenged, one cannot reliably determine whose fault the non-publication is, especially if data is withheld by default and revealed on-demand only when some verification mechanism tries to verify its availability. This is illustrated in the "Fisherman's dilemma", which shows how a challenge-response game cannot distinguish between malicious submitters and malicious challengers:



Fisherman's dilemma. If you only start watching the given specific piece of data at time T3, you have no idea whether you are living in Case 1 or Case 2, and hence who is at fault.

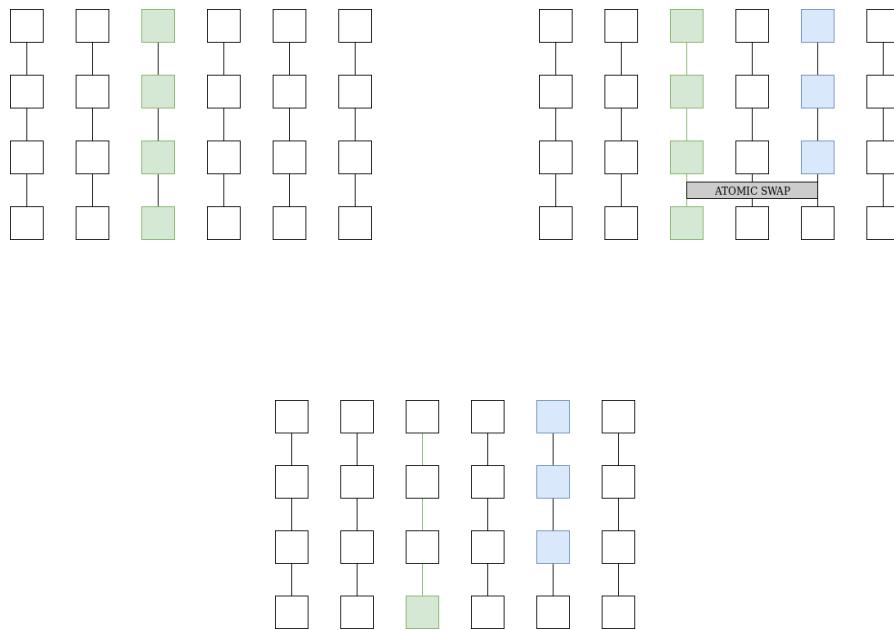
Plasma and channels both work around the fisherman's dilemma by pushing the problem to users: if you as a user decide that another user you are interacting with (a counterparty in a state channel, an operator in a Plasma chain) is not publishing data to you that they should be publishing, it's your responsibility to exit and move to a different counterparty/operator. The fact that you as a user have all of the *previous* data, and data about all of the transactions *you* signed, allows you to prove to the chain what assets you held inside the layer-2 protocol, and thus safely bring them out of the system. You prove the existence of a (previously agreed) operation that gave the asset to you, no one else can prove the existence of an operation approved by you that sent the asset to someone else, so you get the asset.

The technique is very elegant. However, it relies on a key assumption: that every state object has a logical "owner", and the state of the object cannot be changed without the owner's consent. This works well for UTXO-based payments (but not account-based payments, where you *can* edit someone else's balance *upward* without their consent; this is why account-based Plasma is so hard), and it can even be made to work for a decentralized exchange, but this "ownership" property is far from universal. Some applications, eg. [Uniswap \(<http://uniswap.exchange>\)](http://uniswap.exchange) don't have a natural owner, and even in those applications that do, there are often multiple people that can legitimately make edits to the object. And there is no way to allow arbitrary third parties to exit an asset without introducing the possibility of denial-of-service (DoS) attacks, precisely because one cannot prove whether the publisher or submitter is at fault.

There are other issues peculiar to Plasma and channels individually. Channels do not allow off-chain transactions to users that are not already part of the channel (argument: suppose there existed a way to send \$1 to an arbitrary new user from inside a channel. Then this technique could be used many times in parallel to send \$1 to more users than there are funds in the system, already breaking its security guarantee). Plasma requires users to store large amounts of history data, which gets even bigger when different assets can be intertwined (eg. when an asset is transferred conditional on transfer of another asset, as happens in a decentralized exchange with a single-stage order book mechanism).

Because data-on-chain computation-off-chain layer 2 techniques don't have data availability issues, they have none of these weaknesses. ZK and optimistic rollup take great care to put enough data on chain to allow users to calculate the full state of the layer 2 system, ensuring that if any participant disappears a new one can trivially take their place. The only issue that they have is verifying computation without doing the computation on-chain, which is a much easier problem. And the scalability gains are significant: ~10 bytes per transaction in ZK Rollup, and a similar level of scalability can be achieved in optimistic rollup by using BLS aggregation to aggregate signatures. This corresponds to a theoretical maximum of ~500 transactions per second today, and over 2000 post-Istanbul.

But what if you want more scalability? Then there is a large middle ground between data-on-chain layer 2 and data-off-chain layer 2 protocols, with many hybrid approaches that give you some of the benefits of both. To give a simple example, the history storage blowup in a decentralized exchange implemented on Plasma Cash can be prevented by publishing a mapping of which orders are matched with which orders (that's less than 4 bytes per order) on chain:



Left: History data a Plasma Cash user needs to store if they own 1 coin. **Middle:** History data a Plasma Cash user needs to store if they own 1 coin that was exchanged with another coin using an atomic swap. **Right:** History data a Plasma Cash user needs to store if the order matching is published on chain.

Even outside of the decentralized exchange context, the amount of history that users need to store in Plasma can be reduced by having the Plasma chain periodically publish some per-user data on-chain. One could also imagine a platform which works like Plasma in the case where some state does have a logical "owner" and works like ZK or optimistic rollup in the case where it does not. Plasma developers are already starting to work (<https://plasma.build/t/rollup-plasma-for-mass-exits-complex-disputes/90>) on these kinds of optimizations.

There is thus a strong case to be made for developers of layer 2 scalability solutions to move to be more willing to publish per-user data on-chain at least some of the time: it greatly increases ease of development, generality and security and reduces per-user load (eg. no need for users storing history data). The efficiency losses of doing so are also overstated: even in a fully off-chain layer-2 architecture, users depositing, withdrawing and moving between different counterparties and providers is going to be an inevitable and frequent occurrence, and so there will be a significant amount of per-user on-chain data regardless. The hybrid route opens the door to a relatively fast deployment of fully generalized Ethereum-style smart contracts inside a quasi-layer-2 architecture.

See also:

- Introducing the OVM (https://medium.com/@plasma_group/db253287af50).
- Blog post by Karl Floersch (<https://medium.com/plasma-group/ethereum-smart-contracts-in-l2-optimistic-rollup-2c1cef2ec537>).
- Related ideas by John Adler (<https://ethresear.ch/t/minimal-viable-merged-consensus/5617>).

Understanding PLOUD

2019 Sep 22

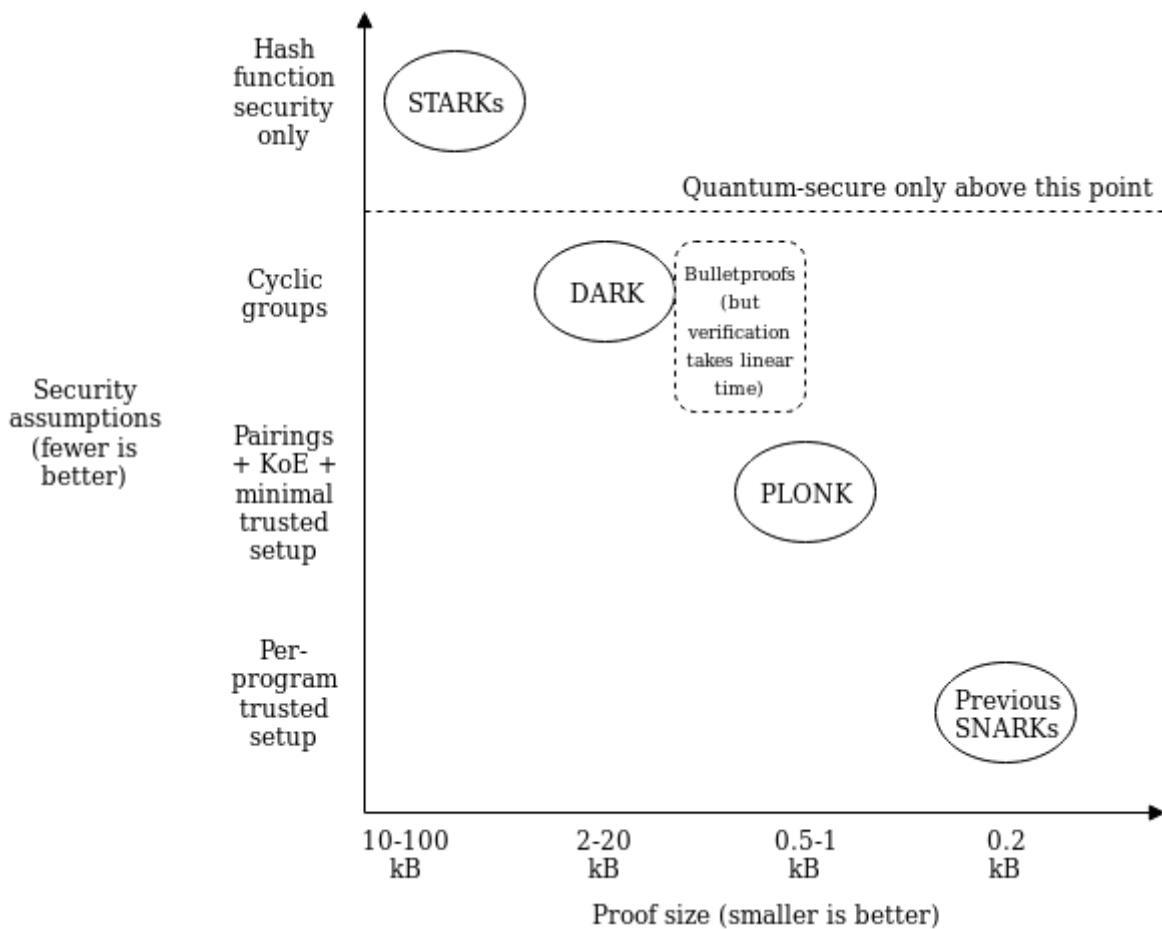
[See all posts \(/\)](#)

Special thanks to Justin Drake, Karl Floersch, Hsiao-wei Wang, Barry Whitehat, Dankrad Feist, Kobi Gurkan and Zac Williamson for review

Very recently, Ariel Gabizon, Zac Williamson and Oana Ciobotaru announced a new general-purpose zero-knowledge proof scheme called [PLOUD](https://eprint.iacr.org/2019/953) (<https://eprint.iacr.org/2019/953>), standing for the unwieldy quasi-backronym "Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge". While [improvements](https://eprint.iacr.org/2016/260.pdf) (<https://eprint.iacr.org/2016/260.pdf>) to general-purpose [zero-knowledge proof](https://arxiv.org/abs/1903.12243) (<https://arxiv.org/abs/1903.12243>), protocols have been [coming](https://dci.mit.edu/zksharks) (<https://dci.mit.edu/zksharks>) for [years](https://eprint.iacr.org/2017/1066) (<https://eprint.iacr.org/2017/1066>), what PLOUD (and the earlier but more complex [SONIC](https://www.benthamsgaze.org/2019/02/07/introducing-sonic-a-practical-zk-snark-with-a-nearly-trustless-setup/) (<https://www.benthamsgaze.org/2019/02/07/introducing-sonic-a-practical-zk-snark-with-a-nearly-trustless-setup/>)) and the more recent [Marlin](https://eprint.iacr.org/2019/1047.pdf) (<https://eprint.iacr.org/2019/1047.pdf>)) bring to the table is a series of enhancements that may greatly improve the usability and progress of these kinds of proofs in general.

The first improvement is that while PLOUD still requires a trusted setup procedure similar to that needed for the [SNARKs in Zcash](https://minezcash.com/zcash-trusted-setup/) (<https://minezcash.com/zcash-trusted-setup/>), it is a "universal and updateable" trusted setup. This means two things: first, instead of there being one separate trusted setup for every program you want to prove things about, there is one single trusted setup for the whole scheme after which you can use the scheme with any program (up to some maximum size chosen when making the setup). Second, there is a way for multiple parties to participate in the trusted setup such that it is secure as long as any one of them is honest, and this multi-party procedure is fully sequential: first one person participates, then the second, then the third... The full set of participants does not even need to be known ahead of time; new participants could just add themselves to the end. This makes it easy for the trusted setup to have a large number of participants, making it quite safe in practice.

The second improvement is that the "fancy cryptography" it relies on is one single standardized component, called a "polynomial commitment". PLOUD uses "Kate commitments", based on a trusted setup and elliptic curve pairings, but you can instead swap it out with other schemes, such as [FRI](https://vitalik.ca/general/2017/11/22/starks_part_2.html) (https://vitalik.ca/general/2017/11/22/starks_part_2.html). (which would [turn PLOUD into a kind of STARK](https://eprint.iacr.org/2019/1020) (<https://eprint.iacr.org/2019/1020>)) or DARK (based on hidden-order groups). This means the scheme is theoretically compatible with any (achievable) tradeoff between proof size and security assumptions.

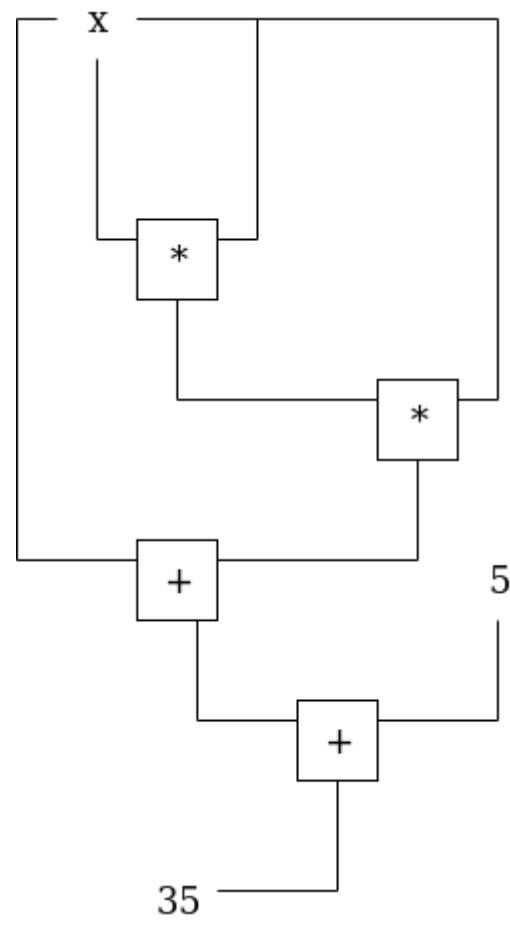


What this means is that use cases that require different tradeoffs between proof size and security assumptions (or developers that have different ideological positions about this question) can still share the bulk of the same tooling for "arithmetization" - the process for converting a program into a set of polynomial equations that the polynomial commitments are then used to check. If this kind of scheme becomes widely adopted, we can thus expect rapid progress in improving shared arithmetization techniques.

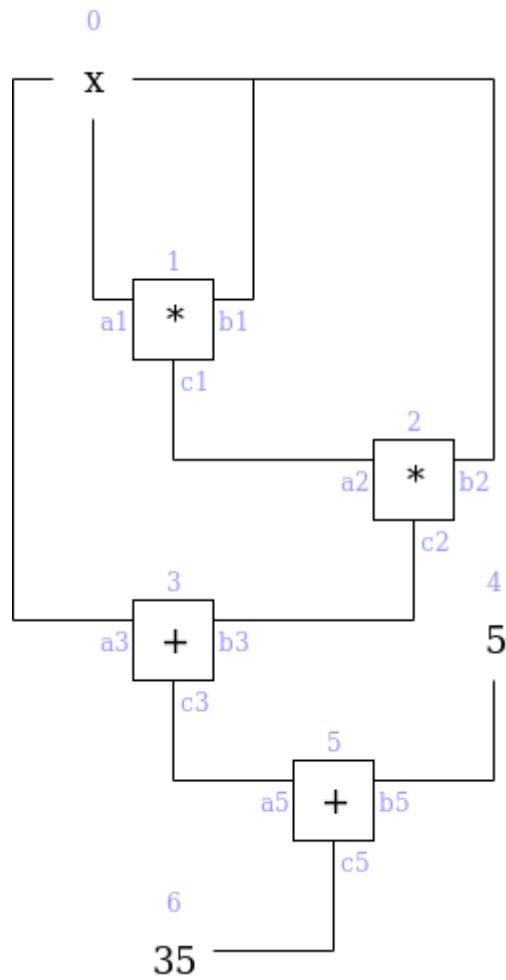
How PLONK works

Let us start with an explanation of how PLONK works, in a somewhat abstracted format that focuses on polynomial equations without immediately explaining how those equations are verified. A key ingredient in PLONK, as is the case in the QAPs used in SNARKs (<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>), is a procedure for converting a problem of the form "give me a value X such that a specific program P that I give you, when evaluated with X as an input, gives some specific result Y" into the problem "give me a set of values that satisfies a set of math equations". The program P can represent many things; for example the problem could be "give me a solution to this sudoku", which you would encode by setting P to be a sudoku verifier plus some initial values encoded and setting Y to 1 (ie. "yes, this solution is correct"), and a satisfying input X would be a valid solution to the sudoku. This is done by representing P as a circuit with logic gates for addition and multiplication, and converting it into a system of equations where the variables are the values on all the wires and there is one equation per gate (eg. $x_6 = x_4 \cdot x_7$ for multiplication, $x_8 = x_5 + x_9$ for addition).

Here is an example of the problem of finding x such that $P(x) = x^3 + x + 5 = 35$ (hint: $x = 3$):



We can label the gates and wires as follows:



On the gates and wires, we have two types of constraints: **gate constraints** (equations between wires attached to the same gate, eg. $a_1 \cdot b_1 = c_1$) and **copy constraints** (claims about equality of different wires anywhere in the circuit, eg. $a_0 = a_1 = b_1 = b_2 = a_3$ or $c_0 = a_1$). We will need to create a structured system of equations, which will ultimately reduce to a very small number of polynomial equations, to represent both.

In PLONK, the setup for these equations is as follows. Each equation is of the following form (think: L = left, R = right, O = output, M = multiplication, C = constant):

$$(Q_{L_i})a_i + (Q_{R_i})b_i + (Q_{O_i})c_i + (Q_{M_i})a_i b_i + Q_{C_i} = 0$$

Each Q value is a constant; the constants in each equation (and the number of equations) will be different for each program. Each small-letter value is a variable, provided by the user: a_i is the left input wire of the i'th gate, b_i is the right input wire, and c_i is the output wire of the i'th gate. For an addition gate, we set:

$$Q_{L_i} = 1, Q_{R_i} = 1, Q_{M_i} = 0, Q_{O_i} = -1, Q_{C_i} = 0$$

Plugging these constants into the equation and simplifying gives us $a_i + b_i - c_i = 0$, which is exactly the constraint that we want. For a multiplication gate, we set:

$$Q_{L_i} = 0, Q_{R_i} = 0, Q_{M_i} = 1, Q_{O_i} = -1, Q_{C_i} = 0$$

For a constant gate setting a_i to some constant x , we set:

$$Q_L = 1, Q_R = 0, Q_M = 0, Q_O = 0, Q_C = -x$$

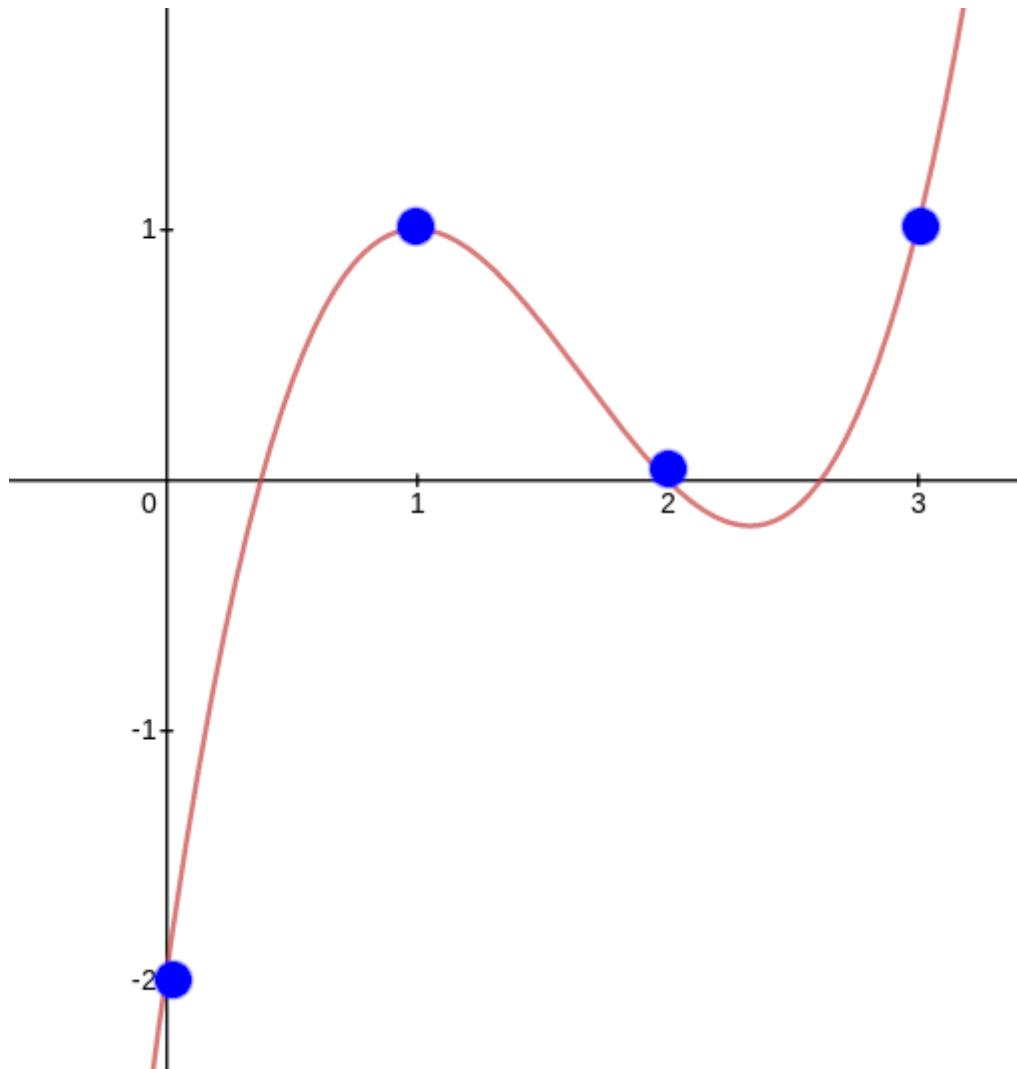
You may have noticed that each end of a wire, as well as each wire in a set of wires that clearly must have the same value (eg. x), corresponds to a distinct variable; there's nothing so far forcing the output of one gate to be the same as the input of another gate (what we call "copy constraints"). PLONK does of course have a way of enforcing copy constraints, but we'll get to this later. So now we have a problem where a prover wants to prove that they have a bunch of x_{a_i} , x_{b_i} and x_{c_i} values that satisfy a bunch of equations that are of the same form. This is still a big problem, but unlike "find a satisfying input to this computer program" it's a very structured big problem, and we have mathematical tools to "compress" it.

From linear systems to polynomials

If you have read about [STARKs](https://vitalik.ca/general/2017/11/09/starks_part_1.html) (https://vitalik.ca/general/2017/11/09/starks_part_1.html) or [QAPs](https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649) (<https://medium.com/@VitalikButerin/quadratic-arithmetic-programs-from-zero-to-hero-f6d558cea649>), the mechanism described in this next section will hopefully feel somewhat familiar, but if you have not that's okay too. The main ingredient here is to understand a *polynomial* as a mathematical tool for encapsulating a whole lot of values into a single object. Typically, we think of polynomials in "coefficient form", that is an expression like:

$$y = x^3 - 5x^2 + 7x - 2$$

But we can also view polynomials in "evaluation form". For example, we can think of the above as being "the" degree < 4 polynomial with evaluations $(-2, 1, 0, 1)$ at the coordinates $(0, 1, 2, 3)$ respectively.



Now here's the next step. Systems of many equations of the same form can be re-interpreted as a single equation over polynomials. For example, suppose that we have the system:

$$\begin{aligned} 2x_1 - x_2 + 3x_3 &= 8 \\ x_1 + 4x_2 - 5x_3 &= 5 \\ 8x_1 - x_2 - x_3 &= -2 \end{aligned}$$

Let us define four polynomials in evaluation form: $L(x)$ is the degree < 3 polynomial that evaluates to $(2, 1, 8)$ at the coordinates $(0, 1, 2)$, and at those same coordinates $M(x)$ evaluates to $(-1, 4, -1)$, $R(x)$ to $(3, -5, -1)$ and $O(x)$ to $(8, 5, -2)$ (it is okay to directly define polynomials in this way; you can use [Lagrange interpolation](#) (https://en.wikipedia.org/wiki/Lagrange_interpolation) to convert to coefficient form). Now, consider the equation:

$$L(x) \cdot x_1 + M(x) \cdot x_2 + R(x) \cdot x_3 - O(x) = Z(x)H(x)$$

Here, $Z(x)$ is shorthand for $(x - 0) \cdot (x - 1) \cdot (x - 2)$ - the minimal (nontrivial) polynomial that returns zero over the evaluation domain $(0, 1, 2)$. A solution to this equation ($x_1 = 1, x_2 = 6, x_3 = 4, H(x) = 0$) is also a solution to the original system of equations, except the original system does not need $H(x)$. Notice also that in this case, $H(x)$ is conveniently zero, but in more complex cases H may need to be nonzero.

So now we know that we can represent a large set of constraints within a small number of mathematical objects (the polynomials). But in the equations that we made above to represent the gate wire constraints, the x_1, x_2, x_3 variables are different per equation. We can handle this by making the variables themselves polynomials rather than constants in the same way. And so we get:

$$Q_L(x)a(x) + Q_R(x)b(x) + Q_O(x)c(x) + Q_M(x)a(x)b(x) + Q_C(x) = 0$$

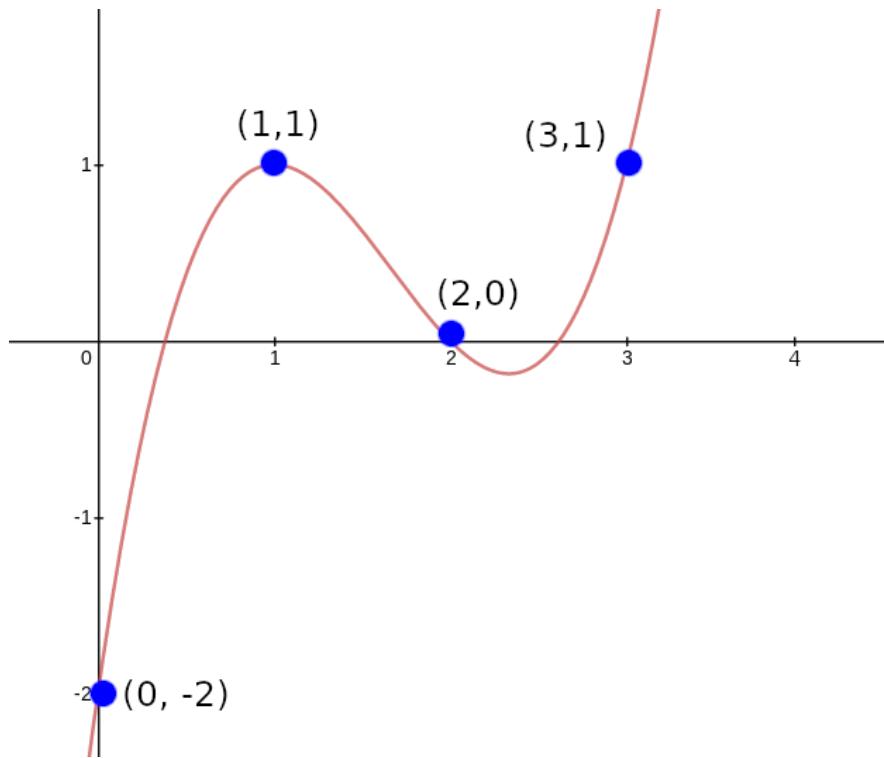
As before, each Q polynomial is a parameter that can be generated from the program that is being verified, and the a, b, c polynomials are the user-provided inputs.

Copy constraints

Now, let us get back to "connecting" the wires. So far, all we have is a bunch of disjoint equations about disjoint values that are independently easy to satisfy: constant gates can be satisfied by setting the value to the constant and addition and multiplication gates can simply be satisfied by setting all wires to zero! To make the problem actually challenging (and actually represent the problem encoded in the original circuit), we need to add an equation that verifies "copy constraints": constraints such as $a(5) = c(7)$, $c(10) = c(12)$, etc. This requires some clever trickery.

Our strategy will be to design a "coordinate pair accumulator", a polynomial $p(x)$ which works as follows. First, let $X(x)$ and $Y(x)$ be two polynomials representing the x and y coordinates of a set of points (eg. to represent the set $((0, -2), (1, 1), (2, 0), (3, 1))$ you might set $X(x) = x$ and $Y(x) = x_3 - 5x_2 + 7x - 2$). Our goal will be to let $p(x)$ represent all the points up to (but not including) the given position, so $p(0)$ starts at 1, $p(1)$ represents just the first point, $p(2)$ the first and the second, etc. We will do this by "randomly" selecting two constants, v_1 and v_2 , and constructing $p(x)$ using the constraints $p(0) = 1$ and $p(x + 1) = p(x) \cdot (v_1 + X(x) + v_2 \cdot Y(x))$ at least within the domain $(0, 1, 2, 3)$.

For example, letting $v_1 = 3$ and $v_2 = 2$, we get:



$X(x)$	0	1	2	3	4
$Y(x)$	-2	1	0	1	
$v_1 + X(x) + v_2 \cdot Y(x)$	-1	6	5	8	
$p(x)$	1	-1	-6	-30	-240

Notice that (aside from the first column) every $p(x)$ value equals the value to the left of it multiplied by the value to the left and above it.

The result we care about is $p(4) = -240$. Now, consider the case where instead of $X(x) = x$, we set $X(x) = \frac{2}{3}x^3 - 4x^2 + \frac{19}{3}x$ (that is, the polynomial that evaluates to $(0, 3, 2, 1)$ at the coordinates $(0, 1, 2, 3)$). If you run the same procedure, you'll find that you also get $p(4) = -240$. This is not a coincidence (in fact, if you randomly pick v_1 and v_2 from a sufficiently large field, it will *almost never* happen coincidentally). Rather, this happens because $Y(1) = Y(3)$, so if you "swap the X coordinates" of the points $(1, 1)$ and $(3, 1)$ you're not changing the set of points, and because the accumulator encodes a set (as multiplication does not care about order) the value at the end will be the same.

Now we can start to see the basic technique that we will use to prove copy constraints. First, consider the simple case where we only want to prove copy constraints within one set of wires (eg. we want to prove $a(1) = a(3)$). We'll make two coordinate accumulators: one where $X(x) = x$ and $Y(x) = a(x)$, and the other where $Y(x) = a(x)$ but $X'(x)$ is the polynomial that evaluates to the permutation that flips (or otherwise rearranges) the values in each copy constraint; in the $a(1) = a(3)$ case this would mean the permutation would start $03214\dots$. The first accumulator would be compressing $((0, a(0)), (1, a(1)), (2, a(2)), (3, a(3)), (4, a(4))\dots$, the second $((0, a(0)), (3, a(1)), (2, a(2)), (1, a(3)), (4, a(4))\dots$. The only way the two can give the same result is if $a(1) = a(3)$.

To prove constraints between a , b and c , we use the same procedure, but instead "accumulate" together points from all three polynomials. We assign each of a , b , c a range of X coordinates (eg. a gets $X_a(x) = x$ ie. $0 \dots n - 1$, b gets $X_b(x) = n + x$, ie. $n \dots 2n - 1$, c gets $X_c(x) = 2n + x$, ie. $2n \dots 3n - 1$). To prove copy constraints that hop between different sets of wires, the "alternate" X coordinates would be slices of a permutation across all three sets. For example, if we want to prove $a(2) = b(4)$ with $n = 5$, then $X'_a(x)$ would have evaluations 01934 and $X'_b(x)$ would have evaluations 56782 (notice the 2 and 9 flipped, where 9 corresponds to the b_4 wire).

We would then instead of checking equality within one run of the procedure (ie. checking $p(4) = p'(4)$ as before), we would check the product of the three different runs on each side:

$$p_a(n) \cdot p_b(n) \cdot p_c(n) = p'_a(n) \cdot p'_b(n) \cdot p'_c(n)$$

The product of the three $p(n)$ evaluations on each side accumulates *all* coordinate pairs in the a , b and c runs on each side together, so this allows us to do the same check as before, except that we can now check copy constraints not just between positions within one of the three sets of wires a , b or c , but also between one set of wires and another (eg. as in $a(2) = b(4)$).

And that's all there is to it!

Putting it all together

In reality, all of this math is done not over integers, but over a prime field; check the section "A Modular Math Interlude" [here](https://vitalik.ca/general/2017/11/22/starks_part_2.html) for a description of what prime fields are. Also, for mathematical reasons perhaps best appreciated by reading and understanding [this article on FFT implementation](https://vitalik.ca/general/2019/05/12/fft.html), instead of representing wire indices with $x = 0 \dots n - 1$, we'll use powers of ω : $1, \omega, \omega^2 \dots \omega^{n-1}$ where ω is a high-order root-of-unity in the field. This changes nothing about the math, except that the coordinate pair accumulator constraint checking equation changes from $p(x+1) = p(x) \cdot (v_1 + X(x) + v_2 \cdot Y(x))$ to $p(\omega \cdot x) = p(x) \cdot (v_1 + X(x) + v_2 \cdot Y(x))$, and instead of using $0 \dots n - 1$, $n \dots 2n - 1$, $2n \dots 3n - 1$ as coordinates we use $\omega^i, g \cdot \omega^i$ and $g^2 \cdot \omega^i$ where g can be some random high-order element in the field.

Now let's write out all the equations we need to check. First, the main gate-constraint satisfaction check:

$$Q_L(x)a(x) + Q_R(x)b(x) + Q_O(x)c(x) + Q_M(x)a(x)b(x) + Q_C(x) = 0$$

Then the polynomial accumulator transition constraint (note: think of " $= Z(x) \cdot H(x)$ " as meaning "equals zero for all coordinates within some particular domain that we care about, but not necessarily outside of it"):

$$\begin{aligned} P_a(\omega x) - P_a(x)(v_1 + x + v_2 a(x)) &= Z(x)H_1(x) \\ P_{a'}(\omega x) - P_{a'}(x)(v_1 + a(x) + v_2 a(x)) &= Z(x)H_2(x) \\ P_b(\omega x) - P_b(x)(v_1 + gx + v_2 b(x)) &= Z(x)H_3(x) \\ P_{b'}(\omega x) - P_{b'}(x)(v_1 + \sigma_b(x) + v_2 b(x)) &= Z(x)H_4(x) \\ P_c(\omega x) - P_c(x)(v_1 + g^2 x + v_2 c(x)) &= Z(x)H_5(x) \\ P_{c'}(\omega x) - P_{c'}(x)(v_1 + \sigma_c(x) + v_2 c(x)) &= Z(x)H_6(x) \end{aligned}$$

Then the polynomial accumulator starting and ending constraints:

$$\begin{aligned} P_a(1) = P_b(1) = P_c(1) = P_{a'}(1) = P_{b'}(1) = P_{c'}(1) &= 1 \\ P_a(\omega^n)P_b(\omega^n)P_c(\omega^n) &= P_{a'}(\omega^n)P_{b'}(\omega^n)P_{c'}(\omega^n) \end{aligned}$$

The user-provided polynomials are:

- The wire assignments $a(x), b(x), c(x)$
- The coordinate accumulators $P_a(x), P_b(x), P_c(x), P_{a'}(x), P_{b'}(x), P_{c'}(x)$
- The quotients $H(x)$ and $H_1(x) \dots H_6(x)$

The program-specific polynomials that the prover and verifier need to compute ahead of time are:

- $Q_L(x), Q_R(x), Q_O(x), Q_M(x), Q_C(x)$, which together represent the gates in the circuit (note that $Q_C(x)$ encodes public inputs, so it may need to be computed or modified at runtime)
- The "permutation polynomials" $\sigma_a(x), \sigma_b(x)$ and $\sigma_c(x)$, which encode the copy constraints between the a, b and c wires

Note that the verifier need only store commitments to these polynomials. The only remaining polynomial in the above equations is $Z(x) = (x - 1) \cdot (x - \omega) \cdots (x - \omega^{n-1})$ which is designed to evaluate to zero at all those points. Fortunately, ω can be chosen to make this polynomial very easy to evaluate: the usual technique is to choose ω to satisfy $\omega^n = 1$, in which case $Z(x) = x^n - 1$.

There is one nuance here: the constraint between $P_a(\omega^{i+1})$ and $P_a(\omega^i)$ can't be true across the *entire* circle of powers of ω ; it's almost always false at ω^{n-1} as the next coordinate is $\omega^n = 1$ which brings us back to the *start* of the "accumulator"; to fix this, we can modify the constraint to say "*either* the constraint is true *or* $x = \omega^{n-1}$ ", which one can do by multiplying $x - \omega^{n-1}$ into the constraint so it equals zero at that point.

The only constraint on v_1 and v_2 is that the user must not be able to choose $a(x), b(x)$ or $c(x)$ after v_1 and v_2 become known, so we can satisfy this by computing v_1 and v_2 from hashes of commitments to $a(x), b(x)$ and $c(x)$.

So now we've turned the program satisfaction problem into a simple problem of satisfying a few equations with polynomials, and there are some optimizations in PLONK that allow us to remove many of the polynomials in the above equations that I will not go into to preserve simplicity. But the polynomials themselves, both the program-specific parameters and the user inputs, are **big**. So the next question is, how do we get around this so we can make the proof short?

Polynomial commitments

A polynomial commitment (<https://pdfs.semanticscholar.org/31eb/add7a0109a584cfbf94b3afaa3c117c78c91.pdf>) is a short object that "represents" a polynomial, and allows you to verify evaluations of that polynomial, without needing to actually contain all of the data in the polynomial. That is, if someone gives you a commitment c representing $P(x)$, they can give you a proof that can convince you, for some specific z , what the value of $P(z)$ is. There is a further mathematical result that says that, over a sufficiently big field, if certain kinds of equations (chosen before z is known) about polynomials evaluated at a random z are true, those same equations are true about the whole polynomial as well. For example, if $P(z) \cdot Q(z) + R(z) = S(z) + 5$, then we know that it's overwhelmingly likely that $P(x) \cdot Q(x) + R(x) = S(x) + 5$ in general. Using such polynomial commitments, we could very easily check all of the above polynomial equations above - make the commitments, use them as input to generate z , prove what the evaluations are of each polynomial at z , and then run the equations with these evaluations instead of the original polynomials. But how do these commitments work?

There are two parts: the commitment to the polynomial $P(x) \rightarrow c$, and the opening to a value $P(z)$ at some z . To make a commitment, there are many techniques; one example is FRI (https://vitalik.ca/general/2017/11/22/starks_part_2.html), and another is Kate commitments which I will describe

below. To prove an opening, it turns out that there is a simple generic "subtract-and-divide" trick: to prove that $P(z) = a$, you prove that

$$\frac{P(x) - a}{x - z}$$

is also a polynomial (using another polynomial commitment). This works because if the quotient is a polynomial (ie. it is not fractional), then $x - z$ is a factor of $P(x) - a$, so $(P(x) - a)(z) = 0$, so $P(z) = a$. Try it with some polynomial, eg. $P(x) = x^3 + 2 \cdot x^2 + 5$ with $(z = 6, a = 293)$, yourself; and try $(z = 6, a = 292)$ and see how it fails (if you're lazy, see WolframAlpha [here](https://www.wolframalpha.com/input/?i=factor+28x%5E3+2*x%5E2+2*x%5E1+5) (https://www.wolframalpha.com/input/?i=factor+28x%5E3+2*x%5E2+2*x%5E1+5)). Note also a generic optimization: to prove many openings of many polynomials at the same time, after committing to the outputs do the subtract-and-divide trick on a *random linear combination* of the polynomials and the outputs.

So how do the commitments themselves work? Kate commitments are, fortunately, much simpler than FRI. A trusted-setup procedure generates a set of elliptic curve points $G, G \cdot s, G \cdot s^2, \dots, G \cdot s^n$, as well as $G_2 \cdot s$, where G and G_2 are the generators of two elliptic curve groups and s is a secret that is forgotten once the procedure is finished (note that there is a multi-party version of this setup, which is secure as long as at least one of the participants forgets their share of the secret). These points are published and considered to be "the proving key" of the scheme; anyone who needs to make a polynomial commitment will need to use these points. A commitment to a degree- d polynomial is made by multiplying each of the first $d+1$ points in the proving key by the corresponding coefficient in the polynomial, and adding the results together.

Notice that this provides an "evaluation" of that polynomial at s , without knowing s . For example, $x^3 + 2x^2 + 5$ would be represented by $(G \cdot s^3) + 2 \cdot (G \cdot s^2) + 5 \cdot G$. We can use the notation $[P]$ to refer to P encoded in this way (ie. $G \cdot P(s)$). When doing the subtract-and-divide trick, you can prove that the two polynomials actually satisfy the relation by using [elliptic curve pairings](https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627) (<https://medium.com/@VitalikButerin/exploring-elliptic-curve-pairings-c73c1864e627>): check that $e([P] - G \cdot a, G_2) = e([Q], [x] - G_2 \cdot z)$ as a proxy for checking that $P(x) - a = Q(x) \cdot (x - z)$.

But there are more recently other types of polynomial commitments coming out too. A new scheme called DARK ("Diophantine arguments of knowledge") uses "hidden order groups" such as [class groups](https://blogs.ams.org/mathgradblog/2018/02/10/introduction-ideal-class-groups/) (<https://blogs.ams.org/mathgradblog/2018/02/10/introduction-ideal-class-groups/>), to implement another kind of polynomial commitment. Hidden order groups are unique because they allow you to compress arbitrarily large numbers into group elements, even numbers much larger than the size of the group element, in a way that can't be "spoofed"; constructions from VDFs to [accumulators](https://ethresear.ch/t/rsa-accumulators-for-plasma-cash-history-reduction/3739) (<https://ethresear.ch/t/rsa-accumulators-for-plasma-cash-history-reduction/3739>) to range proofs to polynomial commitments can be built on top of this. Another option is to use bulletproofs, using regular elliptic curve groups at the cost of the proof taking much longer to verify. Because polynomial commitments are much simpler than full-on zero knowledge proof schemes, we can expect more such schemes to get created in the future.

Recap

To finish off, let's go over the scheme again. Given a program P , you convert it into a circuit, and generate a set of equations that look like this:

$$(Q_{L_i})a_i + (Q_{R_i})b_i + (Q_{O_i})c_i + (Q_{M_i})a_i b_i + Q_{C_i} = 0$$

You then convert this set of equations into a single polynomial equation:

$$Q_L(x)a(x) + Q_R(x)b(x) + Q_O(x)c(x) + Q_M(x)a(x)b(x) + Q_C(x) = 0$$

You also generate from the circuit a list of copy constraints. From these copy constraints you generate the three polynomials representing the permuted wire indices: $a(x), \sigma_b(x), \sigma_c(x)$. To generate a proof, you compute the values of all the wires and convert them into three polynomials: $a(x), b(x), c(x)$. You also compute six "coordinate pair accumulator" polynomials as part of the permutation-check argument. Finally you compute the cofactors $H_i(x)$.

There is a set of equations between the polynomials that need to be checked; you can do this by making commitments to the polynomials, opening them at some random z (along with proofs that the openings are correct), and running the equations on these evaluations instead of the original polynomials. The proof itself is just a few commitments and openings and can be checked with a few equations. And that's all there is to it!

In-person meatspace protocol to prove unconditional possession of a private key

2019 Oct 01

[See all posts \(/\)](#)

Recommended pre-reading: [\(https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413\)](https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413)

Alice slowly walks down the old, dusty stairs of the building into the basement. She thinks wistfully of the old days, when quadratic-voting in the World Collective Market was a much simpler process of linking her public key to a twitter account and opening up metamask to start firing off votes. Of course back then voting in the WCM was used for little; there were a few internet forums that used it for voting on posts, and a few million dollars donated to its [quadratic funding](#) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656) oracle. But then it grew, and then the game-theoretic attacks came.

First came the exchange platforms, which started offering "[dividends](#)" (<https://vitalik.ca/general/2018/03/28/plutocracy.html>) to anyone who registered a public key belonging to an exchange and thus provably allowed the exchange to vote on their behalf, breaking the crucial "independent choice" assumption of the quadratic voting and funding mechanisms. And soon after that came the fake accounts - Twitter accounts, Reddit accounts filtered by karma score, national government IDs, all proved vulnerable to either government cheating or hackers, or both. Elaborate infrastructure was instituted at registration time to ensure both that account holders were real people, and that account holders themselves held the keys, not a central custody service purchasing keys by the thousands to buy votes.

And so today, voting is still easy, but initiation, while still not harder than going to a government office, is no longer exactly trivial. But of course, with billions of dollars in donations from now-deceased billionaires and cryptocurrency premies forming part of the WCM's quadratic funding pool, and elements of municipal governance using its quadratic voting protocols, participating is very much worth it.

After reaching the end of the stairs, Alice opens the door and enters the room. Inside the room, she sees a table. On the near side of the table, she sees a single, empty chair. On the far side of the table, she sees four people already sitting down on chairs of their own, the high-reputation Guardians randomly selected by the WCM for Alice's registration ceremony. "Hello, Alice," the person sitting on the leftmost chair, whose name she intuits is Bob, says in a calm voice. "Glad that you can make it," the person sitting beside Bob, whose name she intuits is Charlie, adds.

Alice walks over to the chair that is clearly meant for her and sits down. "Let us begin," the person sitting beside Charlie, whose name by logical progression is David, proclaims. "Alice, do you have your key shares?"

Alice takes out four pocket-sized notebooks, clearly bought from a dollar store, and places them on the table. The person sitting at the right, logically named Evan, takes out his phone, and immediately the others take out theirs. They open up their ethereum wallets. "So," Evan begins, "the current Ethereum beacon chain slot number is 28,205,913, and the block hash starts 0xbe48 . Do all agree?". "Yes," Alice, Bob, Charlie and David exclaim in unison. Evan continues: "so let us wait for the next block."

The five intently stare at their phones. First for ten seconds, then twenty, then thirty. "Three skipped proposers," Bob mutters, "how unusual". But then after another ten seconds, a new block appears. "Slot number 28,205,917, block hash starts `0x62f9`, so first digit 6. All agreed?"

"Yes."

"Six mod four is two, and as is prescribed in the Old Ways, we start counting indices from zero, so this means Alice will keep the third book, counting as usual from our left."

Bob takes the first, second and fourth notebooks that Alice provided, leaving the third untouched. Alice takes the remaining notebook and puts it back in her backpack. Bob opens each notebook to a page in the middle with the corner folded, and sees a sequence of letters and numbers written with a pencil in the middle of each page - a standard way of writing the key shares for over a decade, since camera and image processing technology got powerful enough to recognize words and numbers written on single slips of paper even inside an envelope. Bob, Charlie, David and Evan crowd around the books together, and each open up an app on their phone and press a few buttons.

Bob starts reading, as all four start typing into their phones at the same time:

"Alice's first key share is, `6-b-d-7-h-k-k-l-o-e-q-q-p-3-y-s-6-x-e-f`. Applying the 100,000x iterated SHA256 hash we get `e-a-6-6...`, confirm?"

"Confirmed," the others replied. "Checking against Alice's precommitted elliptic curve point A0... match."

"Alice's second key share is, `f-r-n-m-j-t-x-r-s-3-b-u-n-n-i-z-3-d-g`. Iterated hash `8-0-3-c...`, confirm?"

"Confirmed. Checking against Alice's precommitted elliptic curve point A1... match."

"Alice's fourth key share is, `i-o-f-s-a-q-f-n-w-f-6-c-e-a-m-s-6-z-z-n`. Iterated hash `6-a-5-6...`, confirm?"

"Confirmed. Checking against Alice's precommitted elliptic curve point A3... match."

"Adding the four precommitted curve points, x coordinate begins `3-1-8-3`. Alice, confirm that that is the key you wish to register?"

"Confirm."

Bob, Charlie, David and Evan glance down at their smartphone apps one more time, and each tap a few buttons. Alice catches a glance at Charlie's phone; she sees four yellow checkmarks, and an "approval transaction pending" dialog. After a few seconds, the four yellow checkmarks are replaced with a single green checkmark, with a transaction hash ID, too small for Alice to make out the digits from a few meters away, below. Alice's phone soon buzzes, with a notification dialog saying "Registration confirmed".

"Congratulations, Alice," Bob says. "Unconditional possession of your key has been verified. You are now free to send a transaction to the World Collective Market's MPC oracle to update your key."

"Only a 75% probability this would have actually caught me if I didn't actually have all four parts of the key," Alice thought to herself. But it seemed to be enough for an in-person protocol in practice; and if it ever wasn't then they could easily switch to slightly more complex protocols that used low-degree polynomials to achieve exponentially high levels of soundness. Alice taps a few buttons on her smartphone, and a "transaction

pending" dialog shows up on the screen. Five seconds later, the dialog disappears and is replaced by a green checkmark. She jumps up with joy and, before Bob, Charlie, David and Evan can say goodbye, runs out of the room, frantically tapping buttons to vote on all the projects and issues in the WCM that she had wanted to support for months.

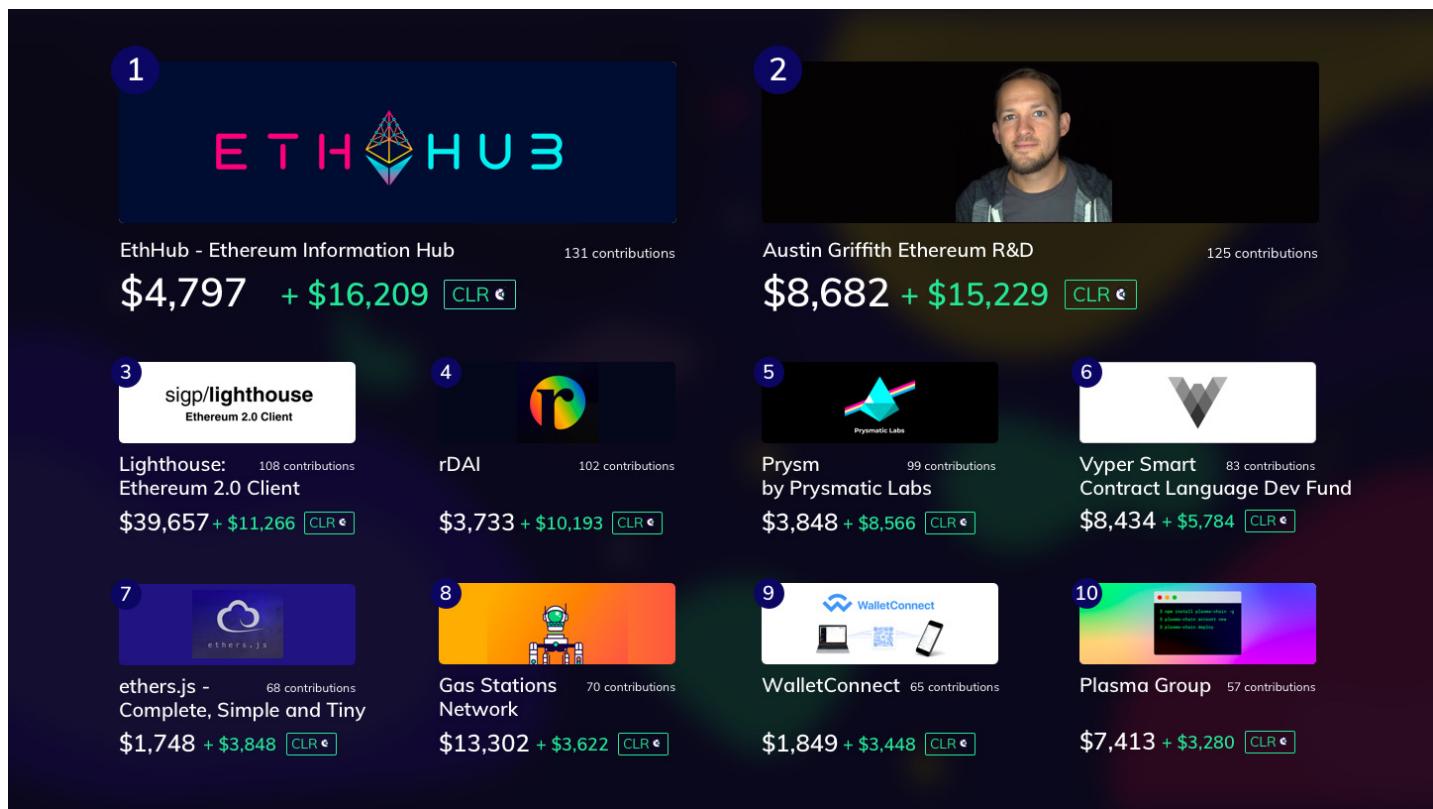
Review of Gitcoin Quadratic Funding Round 3

2019 Oct 24

[See all posts \(/\)](#)

Special thanks to the Gitcoin team and especially Frank Chen for working with me through these numbers

The next round of Gitcoin Grants quadratic funding has just finished, and we have the numbers for how much each project has received were just released (<https://gitcoin.co/blog/gitcoins-q3-match/>). Here are the top ten:



Altogether, \$163,279 was donated to 80 projects by 477 contributors, augmented by a matching pool of \$100,000. Nearly half came from four contributions above \$10,000: \$37,500 to Lighthouse, and \$12,500 each to Gas Station Network, Black Girls Code and Public Health Incentives Layer. Out of the remainder, about half came from contributions between \$1,000 and \$10,000, and the rest came from smaller donations of various sizes. But what matters more here are not the raw donations, but rather the subsidies that the quadratic funding (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656) mechanism applied. Gitcoin Grants is there to support valuable public goods in the Ethereum ecosystem, but also serve as a testbed for this new quadratic donation matching mechanism, and see how well it lives up to its promise of creating a democratic, market-based and efficient way of funding public goods. This time around, a modified formula based on pairwise-bounded coordination subsidies (<https://ethresear.ch/t/pairwise-coordination-subsidies-a-new-quadratic-funding-design/5553>) was used, which has the goal of minimizing distortion from large contributions from coordinated actors. And now we get to see how the experiment went.

Judging the Outcomes

First, the results. Ultimately, every mechanism for allocating resources, whether centralized, market-based, democratic or otherwise, must stand the test of delivering results, or else sooner or later it will be abandoned for another mechanism that is perceived to be better, even if it is less philosophically clean. Judging results is inherently a subjective exercise; any single person's analysis of a mechanism will inevitably be shaped by how well the results fit their own preferences and tastes. However, in those cases where a mechanism does output a surprising result, one can and should use that as an opportunity to learn, and see whether or not one missed some key information that other participants in the mechanism had.

In my own case, I found the top results very agreeable and a quite reasonable catalogue of projects that are good for the Ethereum community. One of the disparities between these grants and the Ethereum Foundation grants is that the Ethereum Foundation grants (see recent rounds [here](https://blog.ethereum.org/2019/08/26/announcing-ethereum-foundation-and-co-funded-grants/) (<https://blog.ethereum.org/2019/08/26/announcing-ethereum-foundation-and-co-funded-grants/>)) and [here](https://blog.ethereum.org/2019/02/21/ethereum-foundation-grants-program-wave-5/) (<https://blog.ethereum.org/2019/02/21/ethereum-foundation-grants-program-wave-5/>)) tend to overwhelmingly focus on technology with only a small section on education and community resources, whereas in the Gitcoin grants while technology still dominates, EthHub is #2 and lower down defiprime.com is #14 and cryptoeconomics.study is #17. In this case my personal opinion is that EF *has* made a genuine error in undervaluing grants to community/education organizations and Gitcoin's "collective instinct" is correct. Score one for new-age fancy quadratic market democracy.

Another surprising result to me was Austin Griffith getting second place. I personally have never spent too much time thinking about Burner Wallet; I knew that it existed but in my mental space I did not take it too seriously, focusing instead on client development, L2 scaling, privacy and to a lesser extent smart contract wallets (the latter being a key use case of Gas Station Network at #8). After seeing Austin's impressive performance in this Gitcoin round, I asked a few people what was going on.

Burner Wallet ([website](https://xdai.io/) (<https://xdai.io/>), [explainer article](https://settle.finance/blog/what-is-the-burner-wallet-and-whats-xdai/) (<https://settle.finance/blog/what-is-the-burner-wallet-and-whats-xdai/>)) is an "insta-wallet" that's very easy to use: just load it up on your desktop or phone, and there you have it. It was used successfully at EthDenver to sell food from food trucks, and generally many people appreciate its convenience. Its main weaknesses are lower security and that one of its features, support for xDAI, is dependent on a permissioned chain.

Austin's Gitcoin grant is there to fund his [ongoing work](https://gitcoin.co/grants/122/austin-griffith-ethereum-rampd) (<https://gitcoin.co/grants/122/austin-griffith-ethereum-rampd>), and I have heard one criticism: there's many prototypes, but comparatively few "things taken to completion". There is also the critique that as great as Austin is, it's difficult to argue that he's as important to the success of Ethereum as, say, Lighthouse and Prysmatic, though one can reply that what matters is not total value, but rather the marginal value of giving a given project or person an extra \$10,000. On the whole, however, I feel like quadratic funding's (Glen would say deliberate!) tendency to select for things like Burner Wallet with populist appeal is a much needed corrective to the influence of the Ethereum tech elite (including myself!) who often value technical impressiveness and undervalue simple and quick things that make it really easy for people to participate in Ethereum. This one is slightly more ambiguous, but I'll say score two for new-age fancy quadratic market democracy.

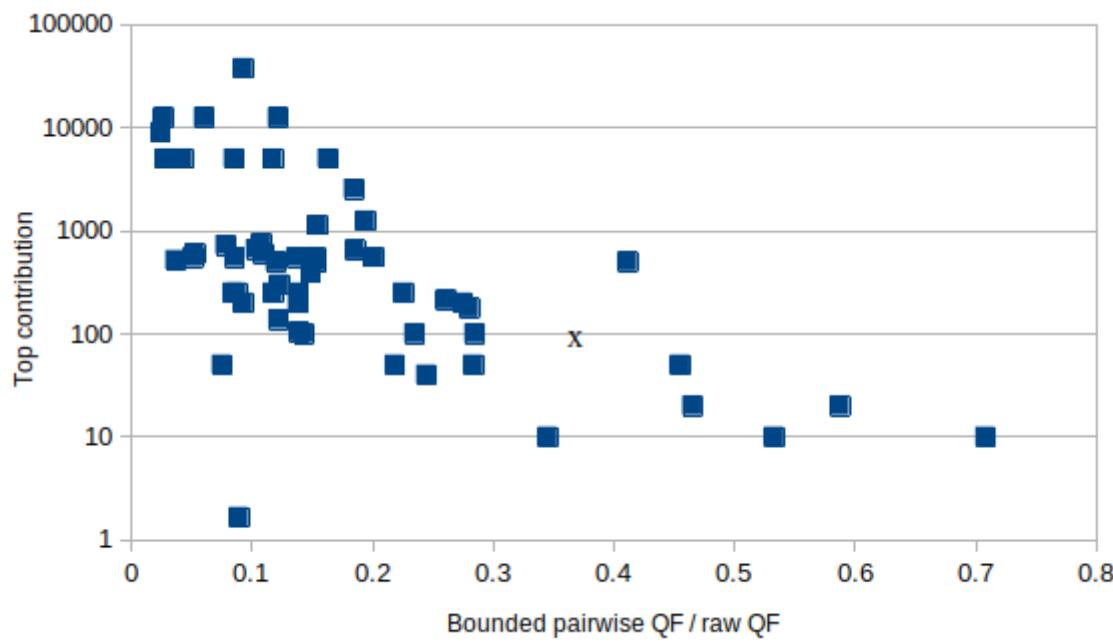
The main thing that I was disappointed the Gitcoiner-at-i did *not* support more was Gitcoin maintenance itself. The Gitcoin Sustainability Fund only got a total \$1,119 in raw contributions from 18 participants, plus a match of \$202. The optional 5% tips that users could give to Gitcoin upon donating were not included into the quadratic matching calculations, but raised another ~\$1,000. Given the amount of effort the Gitcoin people put in to making quadratic funding possible, this is not nearly enough; Gitcoin clearly deserves more than 0.9%

of the total donations in the round. Meanwhile, the Ethereum Foundation (as well as ConsenSys and individual donors) have been giving grants to Gitcoin that include supporting Gitcoin itself. Hopefully in future rounds people will support Gitcoin itself too, but for now, score one for good old-fashioned EF technocracy.

On the whole, quadratic funding, while still young and immature, seems to be a remarkably effective complement to the funding preferences of existing institutions, and it seems worthwhile to continue it and even increase its scope and size in the future.

Pairwise-bounded quadratic funding vs traditional quadratic funding

Round 3 differs from previous rounds in that it uses a new flavor of quadratic funding (<https://ethresear.ch/t/pairwise-coordination-subsidies-a-new-quadratic-funding-design/5553>), which limits the subsidy per pair of participants. For example, in traditional QF, if two people each donate \$10, the subsidy would be \$10, and if two people each donate \$10,000, the subsidy would be \$10,000. This property of traditional QF makes it highly vulnerable to collusion: two key employees of a project (or even two fake accounts owned by the same person) could each donate as much money as they have, and get back a very large subsidy. Pairwise-bounded QF computes the total subsidy to a project by looking through all pairs of contributors, and imposes a maximum bound on the total subsidy that any given pair of participants can trigger (combined across all projects). Pairwise-bounded QF also has the property that it generally penalizes projects that are dominated by large contributors:

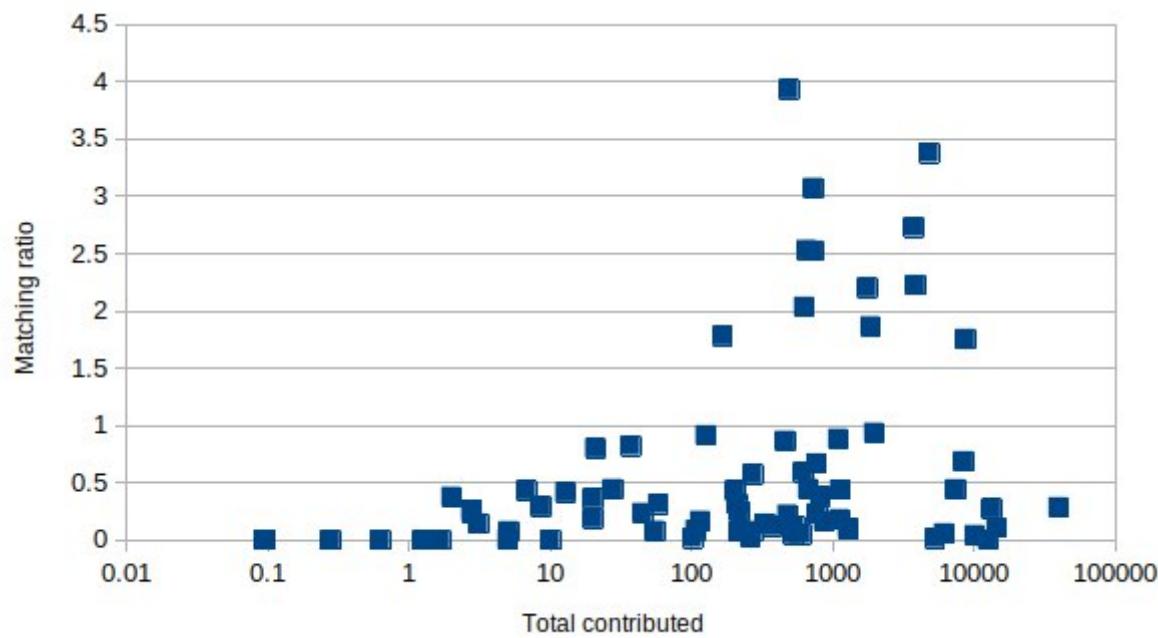


The projects that lost the most relative to traditional QF seem to be projects that have a single large contribution (or sometimes two). For example, "fuzz geth and Parity for EVM consensus bugs" got a \$415 match compared to the \$2000 he would have gotten in traditional QF; the decrease is explained by the fact that the contributions are dominated by two large \$4500 contributions. On the other hand, cryptoeconomics.study (<http://cryptoeconomics.study>) got \$1274, up nearly double from the \$750 it would have gotten in traditional QF; this is explained by the large diversity of contributions that the project received and particularly the lack of large sponsors: the largest contribution to cryptoeconomics.study was \$100.

Another desirable property of pairwise-bounded QF is that it privileges *cross-tribal* projects. That is, if there are projects that group A typically supports, and projects that group B typically supports, then projects that manage to get support from both groups get a more favorable subsidy (because the pairs that go between groups are not as saturated). Has this incentive for building bridges appeared in these results?

Unfortunately, my code of honor as a social scientist obliges me to report the negative result: the Ethereum community just does not yet have enough internal tribal structure for effects like this to materialize, and even when there are differences in correlations they don't seem strongly connected to higher subsidies due to pairwise-bounding. Here are the cross-correlations between who contributed to different projects:

	13	16	20	22	24	25	29	32	36	37	38	39	40	41	48	65	79	80	86	89	104	105	120	121	122	125	128	130	131	132	133	134	135	137	138	139	141	142
13	1	0.06	0.1	0.28	0.25	0.23	0.21	0.13	0.2	0.17	0.07	0.16	0.19	0.10	0.08	0.09	0.25	0.25	0.2	0.2	0.14	0.1	0.1	0.15	0.22	0.12	0.17	0.13	0.04	0.09	0.09	0.09	0.16	0.13	0.06	0.04	0.03	0.05
16	0.06	1	0.07	0.14	0.03	0.05	0.12	0.27	0.19	0.21	0.14	0.09	0.04	0	0.2	0.04	0.06	0.12	0.11	0.08	0.18	0.19	-0.04	0.13	0.03	0.08	0.12	0.03	0.1	0.01	0.04	0.04	0.14	0.05	0.14	0.01		
20	0.1	0.07	1	0.21	0.21	0.12	0.16	0.2	0.18	0.21	0.04	0.13	0.21	0.07	0.17	0.11	0.18	0.14	0.28	0.15	0.09	0.24	0.02	0.25	0.2	0.23	0.22	0.15	0.12	0.02	0.21	0.18	0.13	0.11	0.09	0.11	0.13	0.02
22	0.28	0.14	0.21	1	0.27	0.25	0.33	0.14	0.18	0.21	0.04	0.21	0.23	0.25	0.17	0.26	0.24	0.14	0.23	0.18	0.02	0.24	-0.03	0.23	0.22	0.19	0.24	0.12	0.07	0.2	0.11	0.11	0.08	0.28	0.12	0.02	0.13	0.2
24	0.25	0.03	0.21	0.27	1	0.45	0.23	0.12	0.22	0.14	0.08	0.24	0.34	0.21	0.2	0.08	0.33	0.2	0.22	0.2	0.03	0.17	0.1	0.16	0.22	0.14	0.25	0.01	0.04	0.12	0.05	0.16	0.1	0.19	-0.01	-0.01	0.02	0.05
25	0.23	0.05	0.12	0.25	0.45	1	0.21	0.18	0.23	0.15	0.04	0.2	0.27	0.13	0.2	0.1	0.23	0.06	0.15	0.18	0.06	0.13	0.13	0.22	0.19	0.15	0.25	0.1	0.11	0.07	0.12	0.01	0.22	0.06	0.02	0.06	0.05	
29	0.21	0.12	0.16	0.33	0.23	0.21	1	0.16	0.18	0.2	0.09	0.27	0.13	0.15	0.18	0.12	0.2	0.13	0.18	0.27	0.11	0.25	0.11	0.25	0.19	0.25	0.13	0.04	0.1	0.08	0.18	0.13	0.18	0.1	0.07	0.13	0.16	
32	0.13	0.27	0.2	0.14	0.12	0.18	0.16	1	0.26	0.37	0.18	0.15	0.12	0.08	0.18	0.13	0.17	0.14	0.26	0.15	0.27	0.22	-0.02	0.21	0.09	0.22	0.18	0.15	0.08	0.06	0.21	0.15	0.18	0.16	0.18	0.14	0.24	0.06
36	0.2	0.19	0.18	0.18	0.22	0.23	0.18	0.26	1	0.27	0.07	0.26	0.17	0.17	0.24	0.09	0.18	0.17	0.19	0.17	0.21	0.21	0.12	0.16	0.1	0.2	0.19	0.01	0.1	0.04	0.21	0.21	0.29	0.05	0.11	0.05	0.12	0.04
37	0.17	0.21	0.21	0.21	0.14	0.15	0.2	0.37	0.27	1	0.06	0.2	0.12	0.09	0.24	0.07	0.16	0.21	0.33	0.11	0.25	0.28	-0.03	0.25	0.17	0.18	0.19	0.11	0.08	0.03	0.13	0.18	0.16	0.14	0.03	0.18	0.26	0.17
38	0.07	0.14	0.04	0.04	0.08	0.04	0.09	0.18	0.07	0.06	1	0.12	0.13	-0.03	0.02	0.02	0.1	0.07	0.01	0.05	0.19	0.05	-0.02	0.03	0.12	0.1	0.1	-0.02	0.03	-0.03	0.08	0.12	0.01	0.07	0.05	0.11	0.04	
39	0.18	0.09	0.13	0.21	0.24	0.2	0.27	0.15	0.26	0.2	0.12	1	0.17	0.06	0.19	0.11	0.15	0.09	0.09	0.19	0.08	0.15	0	0.12	0.14	0.09	0.2	0.02	0.05	0.07	0.11	0.13	0.12	0.03	0.02	0	0.08	0.11
40	0.19	0.04	0.21	0.23	0.34	0.27	0.13	0.12	0.17	0.12	0.13	0.17	1	0.08	0.25	0.1	0.13	0.2	0.12	0.11	-0.02	0.12	0	0.18	0.31	0.11	0.18	0.08	0.01	0.03	0.1	0.2	0.06	0.1	0.02	0.14	0.08	0.00
41	0.16	0	0.07	0.25	0.21	0.13	0.15	0.06	0.17	0.09	-0.03	0.06	0.08	1	0.15	0.04	0.29	0.13	0.1	0.04	-0.02	0.02	0.07	-0.02	0.08	0.08	0.06	-0.02	0.06	0.04	0.11	0.1	0.21	0	0.09	-0.03	-0.02	
48	0.08	0.2	0.17	0.17	0.2	0.2	0.18	0.16	0.24	0.24	0.02	0.19	0.25	0.15	1	0.2	0.18	0.16	0.24	0.13	0.13	0.22	0.25	0.16	0.2	0.1	0.06	-0.01	0.12	0.21	0.19	0.13	0.01	0.14	0.18	0.03		
55	0.09	0.04	0.11	0.26	0.08	0.1	0.12	0.13	0.09	0.07	0.02	0.11	0.1	0.04	0.2	1	0.21	0.06	0.15	0.11	0.05	0.12	-0.02	0.09	0.02	0.06	0.17	0.24	0.11	0.13	0.1	0.08	0.08	0.18	0.13	0.02	0.2	0.13
79	0.25	0.06	0.18	0.24	0.33	0.23	0.2	0.17	0.18	0.16	0.1	0.15	0.13	0.29	0.18	0.21	1	0.24	0.18	0.07	0.19	0.18	0.04	0.15	0.19	0.08	0.19	0.09	-0.01	0.08	0.09	0.09	0.1	0.21	0.09	0.04	0.13	0.08
80	0.25	0.12	0.14	0.14	0.2	0.06	0.13	0.14	0.17	0.21	0.07	0.09	0.2	0.13	0.16	0.06	0.24	1	0.24	0.05	0.11	0.28	0.07	0.19	0.15	0.07	0.12	0.1	0	0.1	0.13	0.21	0.23	0.05	0.05	0.17	0.22	0.00
86	0.2	0.11	0.28	0.23	0.22	0.15	0.18	0.26	0.19	0.33	0.01	0.09	0.12	0.1	0.24	0.15	0.18	0.24	1	0.09	0.04	0.21	0.12	0.19	0.15	0.1	0.17	0.09	0.1	0.11	0.4	0.11	0.18	0.16	0.04	0.01	0.12	0.11
89	0.2	0.08	0.15	0.18	0.2	0.18	0.27	0.15	0.17	0.11	0.05	0.19	0.11	0.04	0.13	0.11	0.07	0.05	0.08	1	0.06	0.17	0.11	0.15	0.11	0.17	0.26	0.15	0.08	0.15	0.11	0.1	0.24	0.1	0.15	0.09	0.05	0
104	0.14	0.18	0.09	0.02	0.03	0.06	0.11	0.27	0.2	0.25	0.19	0.08	-0.02	-0.02	0.04	0.05	0.11	0.04	0.06	0.01	0.03	0.12	0.09	0.02	0.06	0.00	-0.02	0.05	0.06	0.01	0.11	0.1	0.14	0.05	0.11	0.17		
105	0.1	0.19	0.24	0.24	0.17	0.13	0.25	0.22	0.21	0.28	0.05	0.15	0.12	0.02	0.13	0.12	0.18	0.28	0.21	0.17	0.17	-1	-0.03	0.15	0.13	0.21	0.18	0.1	0.02	0.16	0.18	0.16	0.2	0.08	0.18	0.1	0.24	0.09
120	0.1	-0.04	0.02	-0.03	0.1	0.13	0.11	-0.02	0.12	-0.03	-0.02	0	0	0.07	0.13	-0.02	0.04	0.07	0.12	0.11	-0.02	-0.03	1	0.1	0.07	0.1	0.16	0.16	-0.02	-0.02	-0.02	-0.04	0.04	-0.03	0	0	-0.03	-0.02
121	0.15	0.13	0.25	0.23	0.16	0.22	0.25	0.21	0.16	0.25	0.03	0.12	0.18	-0.02	0.22	0.09	0.15	0.19	0.19	0.15	0.06	0.15	0.1	0.21	0.23	0.27	0.19	0.11	0.05	0.18	0.13	0.06	0.12	0.17	0.05			
122	0.22	0.03	0.2	0.22	0.22	0.19	0.19	0.09	0.1	0.17	0.12	0.14	0.31	0.08	0.25	0.02	0.19	0.15	0.15	0.11	0.01	0.13	0.07	0.21	1	0.1	0.16	0.12	-0.01	0.06	0.11	0.21	0.07	0.08	0.05	0.1	0.11	0.09
125	0.12	0.08	0.23	0.19	0.14	0.15	0.25	0.22	0.2	0.18	0.1	0.09	0.11	0.08	0.16	0.06	0.08	0.07	0.1	0.17	0.03	0.21	0.1	0.23	0.1	1	0.22	0.27	0.02	0.15	0.17	0.25	0.14	0.03	0.01	0.09	0.19	0.02
128	0.17	0.12	0.22	0.24	0.25	0.25	0.18	0.19	0.19	0.1	0.2	0.18	0.06	0	0.2	0.17	0.19	0.12	0.17	0.26	0.12	0.18	0.16	0.27	0.16	0.22	1	0.12	0.02	0.11	0.13	0.17	0.1	0.12	0.14	0.13	0.12	0.15
130	0.13	0.03	0.15	0.12	0.01	0.13	0.15	0.01	0.11	-0.02	0.02	0.08	0	0.1	0.24	0.09	0.1	0.09	0.15	0.06	0.1	0.16	0.19	0.12	0.27	0.12	1	0.08	0.15	0.07	0.13	0.2	0.14	0.07	0.14	0.2	0.1	
131	0.04	0.1	0.12	0.07	0.04	0.06	0.1	0.08	0.03	0.05	0.01	-0.02	0.06	0.11	-0.01	0	0.1	0.08	0.06	0.02	-0.02	0.11	-0.01	0.02	0.02	0.08	1	0.06	0.11	0.07	0.16	0.2	0.07	0.08	0.02	0.00		
132	0.09	0.01	0.02	0.2	0.12	0.11	0.1	0.06	0.04	0.03	-0.03	0.07	0.03	0.06	-0.01	0.13	0.08	0.1	0.11	0.15	-0.02	0.16	-0.02	0.05	0.06	0.15	0.11	0.15	0.06	0.08	0.09	0.11	0.09	0.04	0	0.03	-0.02	
133	0.09	0.04	0.21	0.11	0.05	0.07	0.08																															

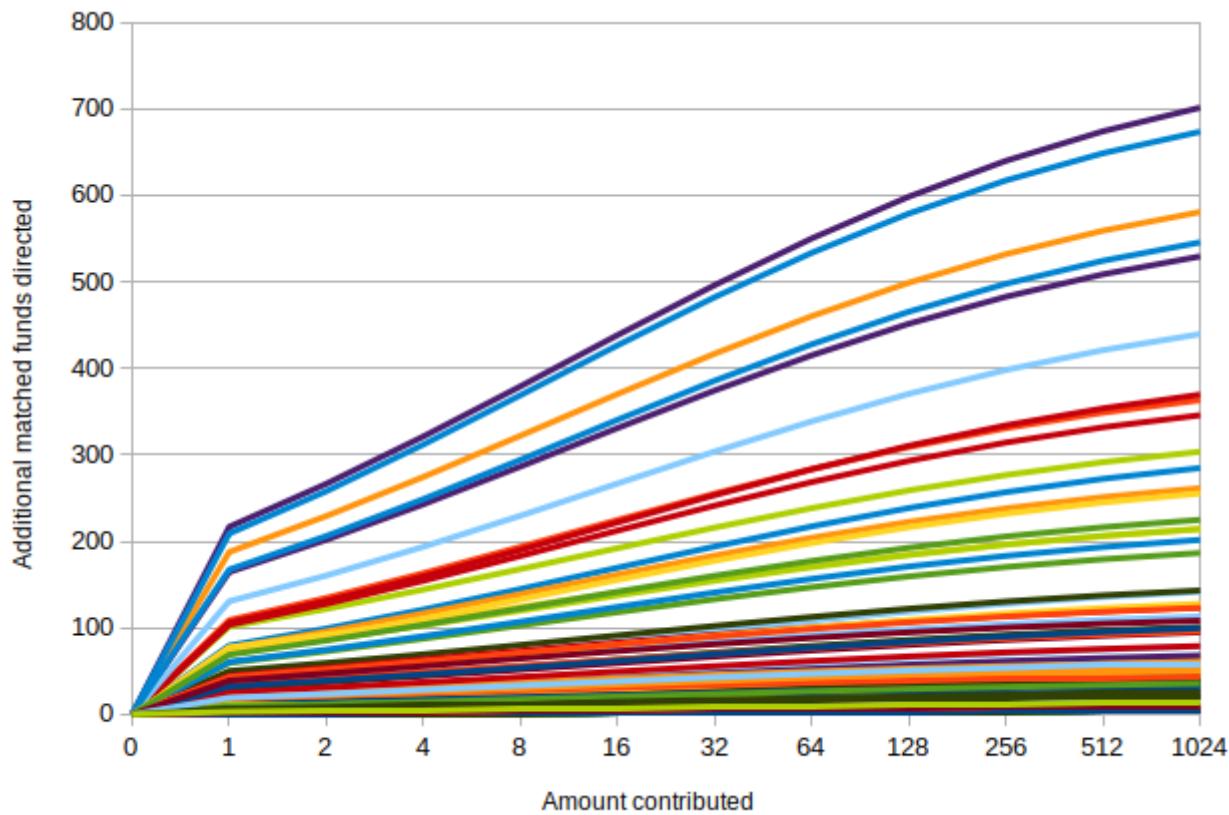


To be clear, this is not just saying "more contributions, more match", it's saying "more contributions, *more match per dollar contributed*". Arguably, this is an intended feature of the mechanism. Projects that can get more people to donate to them represent public goods that serve a larger public, and so tragedy of the commons problems are more severe and hence contributions to them should be multiplied more to compensate. However, looking at the list, it's hard to argue that, say, Prysm (\$3,848 contributed, \$8,566 matched) is a more public good than Nimbus (\$1,129 contributed, \$496 matched; for the unaware, Prysm and Nimbus are both eth2 clients). The failure does not look too severe; on average, projects near the top do seem to serve a larger public and projects near the bottom do seem niche, but it seems clear that at least part of the disparity is not genuine publicness of the good, but rather inequality of attention. N units of marketing effort can attract attention of N people, and theoretically get N^2 resources.

Of course, this could be solved via a "layer on top" venture-capital style: upstart new projects could get investors to support them, in return for a share of matched contributions received when they get large. Something like this would be needed eventually; predicting future public goods is as important a social function as predicting future private goods. But we could also consider less winner-take-all alternatives; the simplest one would be adjusting the QF formula so it uses an exponent of eg. 1.5 instead of 2. I can see it being worthwhile to try a future round of Gitcoin Grants with such a formula ($(\sum_i x_i^{\frac{2}{3}})^{\frac{3}{2}}$ instead of $(\sum_i x_i^{\frac{1}{2}})^2$) to see what the results are like.

Individual leverage curves

One key question is, if you donate \$1, or \$5, or \$100, how big an impact can you have on the amount of money that a project gets? Fortunately, we can use the data to calculate these deltas!



The different lines are for different projects; supporting projects with higher existing support will lead to you getting a bigger multiplier. In all cases, the first dollar is very valuable, with a matching ratio in some cases over 100:1. But the second dollar is much less valuable, and matching ratios quickly taper off; even for the largest projects increasing one's donation from \$32 to \$64 will only get a 1:1 match, and anything above \$100 becomes almost a straight donation with nearly no matching. However, given that it's likely possible to get legitimate-looking Github accounts on the grey market for around those costs, having a cap of a few hundred dollars on the amount of matched funds that any particular account can direct seems like a very reasonable mitigation, despite its costs in limiting the bulk of the matching effect to small-sized donations.

Conclusions

On the whole, this was by far the largest and the most data-rich Gitcoin funding round to date. It successfully attracted hundreds of contributors, reaching a size where we can finally see many significant effects in play and drown out the effects of the more naive forms of small-scale collusion. The experiment already seems to be leading to valuable information that can be used by future quadratic funding implementers to improve their quadratic funding implementations. The case of Austin Griffith is also interesting because \$23,911 in funds that he received comes, in relative terms, surprisingly close to an average salary for a developer if the grants can be repeated on a regular schedule. What this means is that if Gitcoin Grants does continue operating regularly, and attracts and expands its pool of donations, we could be very close to seeing the first "quadratic freelancer" - someone directly "working for the public", funded by donations boosted by quadratic matching subsidies. And at that point we could start to see more experimentation in new forms of organization that live on top of quadratic funding gadgets as a base layer. All in all, this foretells an exciting and, err, radical public-goods funding future ahead of us.

Hard Problems in Cryptocurrency: Five Years Later

2019 Nov 22

[See all posts \(/\)](#)

Special thanks to Justin Drake and Jinglan Wang for feedback

In 2014, I made a [post](https://github.com/ethereum/wiki/wiki/Problems/89fd07ffff8b042134e4ca67a0ce143d574016bd) (<https://github.com/ethereum/wiki/wiki/Problems/89fd07ffff8b042134e4ca67a0ce143d574016bd>), and a [presentation](https://www.youtube.com/watch?v=rXRtJcNVfQE) (<https://www.youtube.com/watch?v=rXRtJcNVfQE>), with a list of hard problems in math, computer science and economics that I thought were important for the cryptocurrency space (as I then called it) to be able to reach maturity. In the last five years, much has changed. But exactly how much progress on what we thought then was important has been achieved? Where have we succeeded, where have we failed, and where have we changed our minds about what is important? In this post, I'll go through the 16 problems from 2014 one by one, and see just where we are today on each one. At the end, I'll include my new picks for hard problems of 2019.

The problems are broken down into three categories: (i) cryptographic, and hence expected to be solvable with purely mathematical techniques if they are to be solvable at all, (ii) consensus theory, largely improvements to proof of work and proof of stake, and (iii) economic, and hence having to do with creating structures involving incentives given to different participants, and often involving the application layer more than the protocol layer. We see significant progress in all categories, though some more than others.

Cryptographic problems

1. Blockchain Scalability

One of the largest problems facing the cryptocurrency space today is the issue of scalability ... The main concern with [oversized blockchains] is trust: if there are only a few entities capable of running full nodes, then those entities can conspire and agree to give themselves a large number of additional bitcoins, and there would be no way for other users to see for themselves that a block is invalid without processing an entire block themselves.

Problem: create a blockchain design that maintains Bitcoin-like security guarantees, but where the maximum size of the most powerful node that needs to exist for the network to keep functioning is substantially sublinear in the number of transactions.

Status: **Great theoretical progress, pending more real-world evaluation.**



Scalability is one technical problem that we have had a huge amount of progress on theoretically. Five years ago, almost no one was thinking about sharding; now, sharding designs are commonplace. Aside from [ethereum 2.0](https://github.com/ethereum/eth2.0-specs) (<https://github.com/ethereum/eth2.0-specs>), we have [OmniLedger](https://eprint.iacr.org/2017/406.pdf) (<https://eprint.iacr.org/2017/406.pdf>), [LazyLedger](https://arxiv.org/abs/1905.09274) (<https://arxiv.org/abs/1905.09274>), [Zilliqa](https://medium.com/@giottodf/zilliqa-a-novel-approach-to-) (<https://medium.com/@giottodf/zilliqa-a-novel-approach-to->

[sharding-d79249347a1f](#)) and research papers [seemingly coming out every month](#) (<https://arxiv.org/pdf/1910.10434.pdf>). In my own view, further progress at this point is incremental. Fundamentally, we already have a number of techniques that allow groups of validators to securely come to consensus on much more data than an individual validator can process, as well as techniques allow clients to indirectly verify the full validity and availability of blocks even under 51% attack conditions.

These are probably the most important technologies:

- **Random sampling**, allowing a small randomly selected committee to statistically stand in for the full validator set: <https://github.com/ethereum/wiki/wiki/Sharding-FAQ#how-can-we-solve-the-single-shard-takeover-attack-in-an-uncoordinated-majority-model> (<https://github.com/ethereum/wiki/wiki/Sharding-FAQ#how-can-we-solve-the-single-shard-takeover-attack-in-an-uncoordinated-majority-model>).
- **Fraud proofs**, allowing individual nodes that learn of an error to broadcast its presence to everyone else: <https://bitcoin.stackexchange.com/questions/49647/what-is-a-fraud-proof> (<https://bitcoin.stackexchange.com/questions/49647/what-is-a-fraud-proof>).
- **Proofs of custody**, allowing validators to probabilistically prove that they individually downloaded and verified some piece of data: <https://ethresear.ch/t/1-bit-aggregation-friendly-custody-bonds/2236> (<https://ethresear.ch/t/1-bit-aggregation-friendly-custody-bonds/2236>).
- **Data availability proofs**, allowing clients to detect when the bodies of blocks that they have headers for are unavailable (<https://github.com/ethereum/research/wiki/A-note-on-data-availability-and-erasure-coding>): <https://arxiv.org/abs/1809.09044> (<https://arxiv.org/abs/1809.09044>). See also the newer [coded Merkle trees](#) (<https://arxiv.org/abs/1910.01247>) proposal.

There are also other smaller developments like [Cross-shard communication via receipts](#) (<https://github.com/ethereum/wiki/wiki/Sharding-FAQ#how-can-we-facilitate-cross-shard-communication>), as well as "constant-factor" enhancements such as BLS signature aggregation.

That said, fully sharded blockchains have still not been seen in live operation (the partially sharded Zilliqa has recently started running). On the theoretical side, there are mainly disputes about details remaining, along with challenges having to do with stability of sharded networking, developer experience and mitigating risks of centralization; fundamental technical possibility no longer seems in doubt. But the challenges that *do* remain are challenges that cannot be solved by just thinking about them; only developing the system and seeing Ethereum 2.0 or some similar chain running live will suffice.

2. Timestamping

Problem: create a distributed incentive-compatible system, whether it is an overlay on top of a blockchain or its own blockchain, which maintains the current time to high accuracy. All legitimate users have clocks in a normal distribution around some "real" time with standard deviation 20 seconds ... no two nodes are more than 20 seconds apart. The solution is allowed to rely on an existing concept of "N nodes"; this would in practice be enforced with proof-of-stake or non-sybil tokens (see #9). The system should continuously provide a time which is within 120s (or less if possible) of the internal clock of >99% of honestly participating nodes. External systems may end up relying on this system; hence, it should remain secure against attackers controlling < 25% of nodes regardless of incentives.

Status: **Some progress.**



Ethereum has actually survived just fine with a 13-second block time and no particularly advanced timestamping technology; it uses a simple technique where a client does not accept a block whose stated timestamp is earlier than the client's local time. That said, this has not been tested under serious attacks. The recent [network-adjusted timestamps](https://ethresear.ch/t/network-adjusted-timestamps/4187) (<https://ethresear.ch/t/network-adjusted-timestamps/4187>) proposal tries to improve on the status quo by allowing the client to determine the consensus on the time in the case where the client does not locally know the current time to high accuracy; this has not yet been tested. But in general, timestamping is not currently at the foreground of perceived research challenges; perhaps this will change once more proof of stake chains (including Ethereum 2.0 but also others) come online as real live systems and we see what the issues are.

3. Arbitrary Proof of Computation

Problem: create programs $\text{POC_PROVE}(P, I) \rightarrow (0, Q)$ and $\text{POC_VERIFY}(P, 0, Q) \rightarrow \{ 0, 1 \}$ such that POC_PROVE runs program P on input I and returns the program output 0 and a proof-of-computation Q and POC_VERIFY takes P , 0 and Q and outputs whether or not Q and 0 were legitimately produced by the POC_PROVE algorithm using P .

Status: **Great theoretical and practical progress.**



This is basically saying, build a SNARK (or STARK, or SHARK, or...). And [we've done](https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6) (<https://medium.com/@VitalikButerin/zk-snarks-under-the-hood-b33151a013f6>). [it](https://vitalik.ca/general/2018/07/21/starks_part_3.html) (https://vitalik.ca/general/2018/07/21/starks_part_3.html). SNARKs are now increasingly well understood, and are even already being used in multiple blockchains today (including tornado.cash (<https://tornado.cash>), on Ethereum). And SNARKs are extremely useful, both as a privacy technology (see Zcash and tornado.cash) and as a scalability technology (see [ZK Rollup](https://ethresear.ch/t/zk-rollups/3477) (<https://ethresear.ch/t/zk-rollups/3477>), [STARKDEX](https://starkdex.io) (<https://starkdex.io>), and [STARKing erasure coded data roots](https://ethresear.ch/t/stark-proving-low-degree-ness-of-a-data-availability-root-some-analysis/6214) (<https://ethresear.ch/t/stark-proving-low-degree-ness-of-a-data-availability-root-some-analysis/6214>)).

There are still challenges with efficiency; making arithmetization-friendly hash functions (see [here](https://starkware.co/hash-challenge/) (<https://starkware.co/hash-challenge/>)) and [here](https://mimchash.org/) (<https://mimchash.org/>) for bounties for breaking proposed candidates) is a big one, and efficiently proving random memory accesses is another. Furthermore, there's the unsolved question of whether the $O(n * \log(n))$ blowup in prover time is a fundamental limitation or if there is some way to make a succinct proof with only linear overhead as in [bulletproofs](https://web.stanford.edu/~buenz/pubs/bulletproofs.pdf) (<https://web.stanford.edu/~buenz/pubs/bulletproofs.pdf>). (which unfortunately take linear time to verify). There are also ever-present risks that the existing schemes have bugs. In general, the problems are in the details rather than the fundamentals.

4. Code Obfuscation

The holy grail is to create an obfuscator O , such that given any program P the obfuscator can produce a second program $O(P) = Q$ such that P and Q return the same output if given the same input and, importantly, Q reveals no information whatsoever about the internals of P . One can hide inside of Q a password, a secret encryption key, or one can simply use Q to hide the proprietary workings of the algorithm itself.

Status: **Slow progress.**



In plain English, the problem is saying that we want to come up with a way to "encrypt" a program so that the encrypted program would still give the same outputs for the same inputs, but the "internals" of the program would be hidden. An example use case for obfuscation is a program containing a private key where the program only allows the private key to sign certain messages.

A solution to code obfuscation would be very useful to blockchain protocols. The use cases are subtle, because one must deal with the possibility that an on-chain obfuscated program will be copied and run in an environment different from the chain itself, but there are many possibilities. One that personally interests me is the ability to remove the centralized operator from [collusion-resistance gadgets](https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413) (<https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413>) by replacing the operator with an obfuscated program that contains some proof of work, making it very expensive to run more than once with different inputs as part of an attempt to determine individual participants' actions.

Unfortunately this continues to be a hard problem. There is continuing ongoing work in attacking the problem, one side making constructions (eg. [this](https://eprint.iacr.org/2018/615) (<https://eprint.iacr.org/2018/615>)) that try to reduce the number of assumptions on mathematical objects that we do not know practically exist (eg. general cryptographic multilinear maps) and another side trying to make practical implementations of the desired mathematical objects. However, all of these paths are still quite far from creating something viable and known to be secure. See <https://eprint.iacr.org/2019/463.pdf> (<https://eprint.iacr.org/2019/463.pdf>) for a more general overview to the problem.

5. Hash-Based Cryptography

Problem: create a signature algorithm relying on no security assumption but the random oracle property of hashes that maintains 160 bits of security against classical computers (ie. 80 vs. quantum due to Grover's algorithm) with optimal size and other properties.

Status: **Some progress.**



There have been two strands of progress on this since 2014. [SPHINCS](https://cryptojedi.org/papers/sphincs-20141001.pdf) (<https://cryptojedi.org/papers/sphincs-20141001.pdf>), a "stateless" (meaning, using it multiple times does not require remembering information like a nonce) signature scheme, was released soon after this "hard problems" list was published, and provides a purely hash-based signature scheme of size around 41 kB. Additionally, [STARKs](#)

(https://vitalik.ca/general/2018/07/21/starks_part_3.html) have been developed, and one can create signatures of similar size based on them. The fact that not just signatures, but also general-purpose zero knowledge proofs, are possible with just hashes was definitely something I did not expect five years ago; I am very happy that this is the case. That said, size continues to be an issue, and ongoing progress (eg. see the very recent DEEP FRI (<https://arxiv.org/abs/1903.12243>)) is continuing to reduce the size of proofs, though it looks like further progress will be incremental.

The main not-yet-solved problem with hash-based cryptography is aggregate signatures, similar to what BLS aggregation (<https://ethresear.ch/t/pragmatic-signature-aggregation-with-bls/2105>) makes possible. It's known that we can just make a STARK over many Lamport signatures, but this is inefficient; a more efficient scheme would be welcome. (In case you're wondering if hash-based *public key encryption* is possible, the answer is, no, you can't do anything with more than a quadratic attack cost (<https://www.boazbarak.org/Papers/merkle.pdf>)).

Consensus theory problems

6. ASIC-Resistant Proof of Work

One approach at solving the problem is creating a proof-of-work algorithm based on a type of computation that is very difficult to specialize ... For a more in-depth discussion on ASIC-resistant hardware, see <https://blog.ethereum.org/2014/06/19/mining/> (<https://blog.ethereum.org/2014/06/19/mining/>).

Status: **Solved as far as we can.**



About six months after the "hard problems" list was posted, Ethereum settled on its ASIC-resistant proof of work algorithm: Ethash (<https://github.com/ethereum/wiki/wiki/Ethash>). Ethash is known as a memory-hard algorithm. The theory is that random-access memory in regular computers is well-optimized already and hence difficult to improve on for specialized applications. Ethash aims to achieve ASIC resistance by making memory access the dominant part of running the PoW computation. Ethash was not the first memory-hard algorithm, but it did add one innovation: it uses pseudorandom lookups over a two-level DAG, allowing for two ways of evaluating the function. First, one could compute it quickly if one has the entire (~2 GB) DAG; this is the memory-hard "fast path". Second, one can compute it much more slowly (still fast enough to check a single provided solution quickly) if one only has the top level of the DAG; this is used for block verification.

Ethash has proven remarkably successful at ASIC resistance; after three years and billions of dollars of block rewards, ASICs do exist but are at best 2-5 times more power and cost-efficient (<https://blog.miningstore.com/blog/ethereum-mining-hardware-for-2019>) than GPUs. ProgPoW (<https://github.com/ifdefelse/ProgPOW>) has been proposed as an alternative, but there is a growing consensus that ASIC-resistant algorithms will inevitably have a limited lifespan, and that ASIC resistance has downsides (<https://pdaian.com/blog/anti-asic-forks-considered-harmful/>) because it makes 51% attacks cheaper (eg. see the 51% attack on Ethereum Classic (<https://cointelegraph.com/news/ethereum-classic-51-attack-the-reality-of-proof-of-work>)).

I believe that PoW algorithms that provide a medium level of ASIC resistance can be created, but such resistance is limited-term and both ASIC and non-ASIC PoW have disadvantages; in the long term the better choice for blockchain consensus is proof of stake.

7. Useful Proof of Work

making the proof of work function something which is simultaneously useful; a common candidate is something like Folding@home, an existing program where users can download software onto their computers to simulate protein folding and provide researchers with a large supply of data to help them cure diseases.

Status: **Probably not feasible, with one exception.**



The challenge with useful proof of work is that a proof of work algorithm requires many properties:

- Hard to compute
- Easy to verify
- Does not depend on large amounts of external data
- Can be efficiently computed in small "bite-sized" chunks

Unfortunately, there are not many computations that are useful that preserve all of these properties, and most computations that do have all of those properties and are "useful" are only "useful" for far too short a time to build a cryptocurrency around them.

However, there is one possible exception: zero-knowledge-proof generation. Zero knowledge proofs of aspects of blockchain validity (eg. [data availability roots](https://ethresear.ch/t/stark-proving-low-degree-ness-of-a-data-availability-root-some-analysis/6214) (<https://ethresear.ch/t/stark-proving-low-degree-ness-of-a-data-availability-root-some-analysis/6214>) for a simple example) are difficult to compute, and easy to verify. Furthermore, they are durably difficult to compute; if proofs of "highly structured" computation become too easy, one can simply switch to verifying a blockchain's entire state transition, which becomes extremely expensive due to the need to model the virtual machine and random memory accesses.

Zero-knowledge proofs of blockchain validity provide great value to users of the blockchain, as they can substitute the need to verify the chain directly; [Coda](https://codaproto.com/) (<https://codaproto.com/>) is doing this already, albeit with a simplified blockchain design that is heavily optimized for provability. Such proofs can significantly assist in improving the blockchain's safety and scalability. That said, the total amount of computation that realistically needs to be done is still much less than the amount that's currently done by proof of work miners, so this would at best be an add-on for proof of stake blockchains, not a full-on consensus algorithm.

8. Proof of Stake

Another approach to solving the mining centralization problem is to abolish mining entirely, and move to some other mechanism for counting the weight of each node in the consensus. The most popular alternative under discussion to date is "proof of stake" - that is to say, instead of treating the consensus model as "one unit of CPU power, one vote" it becomes "one currency unit, one vote".

Status: **Great theoretical progress, pending more real-world evaluation.**



Near the end of 2014, it became clear to the proof of stake community that some form of "weak subjectivity" is unavoidable (<https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity>). To maintain economic security, nodes need to obtain a recent checkpoint extra-protocol when they sync for the first time, and again if they go offline for more than a few months. This was a difficult pill to swallow; many PoW advocates still cling to PoW precisely because in a PoW chain the "head" of the chain can be discovered with the only data coming from a trusted source being the blockchain client software itself. PoS advocates, however, were willing to swallow the pill, seeing the added trust requirements as not being large. From there the path to proof of stake through long-duration security deposits became clear.

Most interesting consensus algorithms today are fundamentally similar to PBFT (<http://pmg.csail.mit.edu/papers/osdi99.pdf>), but replace the fixed set of validators with a dynamic list that anyone can join by sending tokens into a system-level smart contract with time-locked withdrawals (eg. a withdrawal might in some cases take up to 4 months to complete). In many cases (including Ethereum 2.0), these algorithms achieve "economic finality" by penalizing validators that are caught performing actions that violate the protocol in certain ways (see [here](https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51) (<https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51>) for a philosophical view on what proof of stake accomplishes).

As of today, we have (among many other algorithms):

- **Casper FFG:** <https://arxiv.org/abs/1710.09437> (<https://arxiv.org/abs/1710.09437>).
- **Tendermint:** <https://tendermint.com/docs/spec/consensus/consensus.html> (<https://tendermint.com/docs/spec/consensus/consensus.html>).
- **HotStuff:** <https://arxiv.org/abs/1803.05069> (<https://arxiv.org/abs/1803.05069>).
- **Casper CBC:** https://vitalik.ca/general/2018/12/05/cbc_casper.html (https://vitalik.ca/general/2018/12/05/cbc_casper.html).

There continues to be ongoing refinement (eg. [here](https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113) (<https://ethresear.ch/t/analysis-of-bouncing-attack-on-ffg/6113>), and [here](https://ethresear.ch/t/saving-strategy-and-fmd-ghost/6226) (<https://ethresear.ch/t/saving-strategy-and-fmd-ghost/6226>)). Eth2 phase 0, the chain that will implement FFG, is currently under implementation and enormous progress has been made. Additionally, Tendermint has been running, in the form of the Cosmos chain (<https://cosmos.bigdipper.live/validators>) for several months. Remaining arguments about proof of stake, in my view, have to do with optimizing the economic incentives, and further formalizing the strategy for responding to 51% attacks (<https://ethresear.ch/t/responding-to-51-attacks-in-casper-ffg/6363>). Additionally, the Casper CBC spec (<https://github.com/ethereum/eth2.0-specs/issues/701>) could still use concrete efficiency improvements.

9. Proof of Storage

A third approach to the problem is to use a scarce computational resource other than computational power or currency. In this regard, the two main alternatives that have been proposed are storage and bandwidth. There is no way in principle to provide an after-the-fact cryptographic proof that bandwidth was given or used, so proof of bandwidth should most accurately be considered a subset of social proof, discussed in later problems, but proof of storage is something that certainly can be done computationally. An advantage of proof-of-storage is that it is completely ASIC-resistant; the kind of storage that we have in hard drives is already close to optimal.

Status: **A lot of theoretical progress, though still a lot to go, as well as more real-world evaluation.**



There are a number of [blockchains planning to use proof of storage](https://en.wikipedia.org/wiki/Proof_of_space) (https://en.wikipedia.org/wiki/Proof_of_space) protocols, including [Chia](https://eprint.iacr.org/2017/893.pdf) (<https://eprint.iacr.org/2017/893.pdf>) and [Filecoin](https://filecoin.io/filecoin.pdf) (<https://filecoin.io/filecoin.pdf>). That said, these algorithms have not been tested in the wild. My own main concern is centralization: will these algorithms actually be dominated by smaller users using spare storage capacity, or will they be dominated by large mining farms?

Economics

10. Stable-value cryptoassets

One of the main problems with Bitcoin is the issue of price volatility ... Problem: construct a cryptographic asset with a stable price.

Status: **Some progress.**



[MakerDAO](https://makerdao.com/en/) (<https://makerdao.com/en/>) is now live, and has been holding stable for nearly two years. It has survived a 93% drop in the value of its underlying collateral asset (ETH), and there is now more than \$100 million in DAI issued. It has become a mainstay of the Ethereum ecosystem, and many Ethereum projects have or are integrating with it. Other synthetic token projects, such as [UMA](https://umaproject.org/) (<https://umaproject.org/>), are rapidly gaining steam as well.

However, while the MakerDAO system has survived tough economic conditions in 2019, the conditions were by no means the toughest that could happen. In the past, Bitcoin has [fallen by 75%](https://fortune.com/2017/09/18/bitcoin-crash-history/) (<https://fortune.com/2017/09/18/bitcoin-crash-history/>), over the course of two days; the same may happen to ether or any other collateral asset some day. Attacks on the underlying blockchain are an even larger untested risk, especially if compounded by price decreases at the same time. Another major challenge, and arguably the larger one, is that the stability of MakerDAO-like systems is dependent on some underlying oracle scheme. Different attempts at oracle systems do exist (see #16), but the jury is still out on how well they can hold up under large amounts of economic stress. So far, the collateral controlled by MakerDAO has been lower than

the value of the MKR token; if this relationship reverses MKR holders may have a collective incentive to try to "loot" the MakerDAO system. There are ways to try to protect against such attacks, but they have not been tested in real life.

11. Decentralized Public Goods Incentivization

One of the challenges in economic systems in general is the problem of "public goods". For example, suppose that there is a scientific research project which will cost \$1 million to complete, and it is known that if it is completed the resulting research will save one million people \$5 each. In total, the social benefit is clear ... [but] from the point of view of each individual person contributing does not make sense ... So far, most problems to public goods have involved centralization Additional Assumptions And Requirements: A fully trustworthy oracle exists for determining whether or not a certain public good task has been completed (in reality this is false, but this is the domain of another problem)

Status: **Some progress.**



The problem of funding public goods is generally understood to be split into two problems: the funding problem (where to get funding for public goods from) and the preference aggregation problem (how to determine what is a genuine public good, rather than some single individual's pet project, in the first place). This problem focuses specifically on the former, assuming the latter is solved (see the "[decentralized contribution metrics](#)" section below for work on that problem).

In general, there haven't been large new breakthroughs here. There's two major categories of solutions. First, we can try to elicit individual contributions, giving people social rewards for doing so. My own proposal for [charity through marginal price discrimination](#) (https://vitalik.ca/general/2017/03/11/a_note_on_charity.html) is one example of this; another is the anti-malaria donation badges on [Peepeth](#) (<https://peepeth.com/welcome>). Second, we can collect funds from applications that have network effects. Within blockchain land there are several options for doing this:

- Issuing coins
- Taking a portion of transaction fees at protocol level (eg. through [EIP 1559](#) (<https://github.com/ethereum/EIPs/issues/1559>))
- Taking a portion of transaction fees from some layer-2 application (eg. Uniswap, or some scaling solution, or even state rent in an execution environment in Ethereum 2.0)
- Taking a portion of other kinds of fees (eg. ENS registration)

Outside of blockchain land, this is just the age-old question of how to collect taxes if you're a government, and charge fees if you're a business or other organization.

12. Reputation systems

Problem: design a formalized reputation system, including a score $\text{rep}(A, B) \rightarrow V$ where V is the reputation of B from the point of view of A , a mechanism for determining the probability that one party can be trusted by another, and a mechanism for updating the reputation given a record of a particular open or finalized interaction.

Status: **Slow progress.**



There hasn't really been much work on reputation systems since 2014. Perhaps the best is the use of token curated registries to create curated lists of trustable entities/objects; the [Kleros ERC20 TCR](https://blog.kleros.io/erc20-becomes-part-of-the-token/) (<https://blog.kleros.io/erc20-becomes-part-of-the-token/>) (yes, that's a [token-curated registry](#)) (<https://medium.com/@tokencuratedregistry/a-simple-overview-of-token-curated-registries-84e2b7b19a06>) of legitimate ERC20 tokens) is one example, and there is even an alternative interface to Uniswap (<http://uniswap.ninja> (<http://uniswap.ninja>)) that uses it as the backend to get the list of tokens and ticker symbols and logos from. Reputation systems of the subjective variety have not really been tried, perhaps because there is just not enough information about the "social graph" of people's connections to each other that has already been published to chain in some form. If such information starts to exist for other reasons, then subjective reputation systems may become more popular.

13. Proof of excellence

One interesting, and largely unexplored, solution to the problem of [token] distribution specifically (there are reasons why it cannot be so easily used for mining) is using tasks that are socially useful but require original human-driven creative effort and talent. For example, one can come up with a "proof of proof" currency that rewards players for coming up with mathematical proofs of certain theorems

Status: **No progress, problem is largely forgotten.**



The main alternative approach to token distribution that has instead become popular is [airdrops](#) (https://en.wikipedia.org/wiki/Airdrop_%28cryptocurrency%29); typically, tokens are distributed at launch either proportionately to existing holdings of some other token, or based on some other metric (eg. as in the [Handshake airdrop](https://help.namebase.io/article/4vchu01mec-handshake-airdrop-101) (<https://help.namebase.io/article/4vchu01mec-handshake-airdrop-101>)). Verifying human creativity directly has not really been attempted, and with recent progress on AI the problem of creating a task that only humans can do but computers can verify may well be too difficult.

15 [sic]. Anti-Sybil systems

A problem that is somewhat related to the issue of a reputation system is the challenge of creating a "unique identity system" - a system for generating tokens that prove that an identity is not part of a Sybil attack ... However, we would like to have a system that has nicer and more egalitarian features than "one-dollar-one-vote"; arguably, one-person-one-vote would be ideal.

Status: **Some progress.**



There have been quite a few attempts at solving the unique-human problem. Attempts that come to mind include (incomplete list!):

- **HumanityDAO:** <https://www.humanitydao.org/>
- **Pseudonym parties:** <https://bford.info/pub/net/sybil.pdf>
- **POAP ("proof of attendance protocol"):** <https://www.poap.xyz/>
- **BrightID:** <https://www.brightid.org/>

With the growing interest in techniques like [quadratic voting](https://en.wikipedia.org/wiki/Quadratic_voting) (https://en.wikipedia.org/wiki/Quadratic_voting) and [quadratic funding](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656), the need for some kind of human-based anti-sybil system continues to grow. Hopefully, ongoing development of these techniques and new ones can come to meet it.

14 [sic]. Decentralized contribution metrics

Incentivizing the production of public goods is, unfortunately, not the only problem that centralization solves. The other problem is determining, first, which public goods are worth producing in the first place and, second, determining to what extent a particular effort actually accomplished the production of the public good. This challenge deals with the latter issue.

Status: **Some progress, some change in focus.**



More recent work on determining value of public-good contributions does not try to separate determining tasks and determining quality of completion; the reason is that in practice the two are difficult to separate. Work done by specific teams tends to be non-fungible and subjective enough that the most reasonable approach is to look at relevance of task and quality of performance as a single package, and use the same technique to evaluate both.

Fortunately, there has been great progress on this, particularly with the discovery of [quadratic funding](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656). Quadratic funding is a mechanism where individuals can make donations to projects, and then based on the number of people who donated and how much they donated, a formula is used to calculate how much they would have donated if they were perfectly coordinated with each other (ie. took each other's interests into account and did not fall prey to the tragedy of

the commons). The difference between amount would-have-donated and amount actually donated for any given project is given to that project as a subsidy from some central pool (see #11 for where the central pool funding could come from). Note that this mechanism focuses on satisfying the values of some community, not on satisfying some given goal regardless of whether or not anyone cares about it. Because of the complexity of values (https://wiki.lesswrong.com/wiki/Complexity_of_value) problem, this approach is likely to be much more robust to unknown unknowns.

Quadratic funding has even been tried in real life with considerable success in the recent gitcoin quadratic funding round (<https://vitalik.ca/general/2019/10/24/gitcoin.html>). There has also been some incremental progress on improving quadratic funding and similar mechanisms; particularly, pairwise-bounded quadratic funding (<https://ethresear.ch/t/pairwise-coordination-subsidies-a-new-quadratic-funding-design/5553>) to mitigate collusion. There has also been work on specification and implementation of bribe-resistant (<https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413>) voting technology, preventing users from proving to third parties who they voted for; this prevents many kinds of collusion and bribe attacks.

16. Decentralized success metrics

Problem: come up with and implement a decentralized method for measuring numerical real-world variables ... the system should be able to measure anything that humans can currently reach a rough consensus on (eg. price of an asset, temperature, global CO₂ concentration)

Status: **Some progress.**



This is now generally just called "the oracle problem". The largest known instance of a decentralized oracle running is Augur (<https://www.augur.net/>), which has processed outcomes for millions of dollars of bets. Token curated registries (<https://medium.com/@tokencuratedregistry/a-simple-overview-of-token-curated-registries-84e2b7b19a06>) such as the Kleros TCR for tokens (<https://tokens.kleros.io/tokens>) are another example. However, these systems still have not seen a real-world test of the forking mechanism (search for "subjectivocracy" [here](https://blog.ethereum.org/2015/02/14/subjectivity-exploitability-tradeoff/) (<https://blog.ethereum.org/2015/02/14/subjectivity-exploitability-tradeoff/>)) either due to a highly controversial question or due to an attempted 51% attack. There is also research on the oracle problem happening outside of the blockchain space in the form of the "peer prediction" (https://www2.cs.duke.edu/courses/spring17/compsci590.2/peer_prediction.pdf) literature; see [here](https://arxiv.org/abs/1911.00272) (<https://arxiv.org/abs/1911.00272>) for a very recent advancement in the space.

Another looming challenge is that people want to rely on these systems to guide transfers of quantities of assets larger than the economic value of the system's native token. In these conditions, token holders in theory have the incentive to collude to give wrong answers to steal the funds. In such a case, the system would fork and the original system token would likely become valueless, but the original system token holders would still get away with the returns from whatever asset transfer they misdirected. Stablecoins (see #10) are a particularly egregious case of this. One approach to solving this would be a system that assumes that altruistically honest data providers do exist, and creating a mechanism to identify them, and only allowing them to churn slowly so that if malicious ones start getting voted in the users of systems that rely on the oracle can first complete an orderly exit. In any case, more development of oracle tech is very much an important problem.

New problems

If I were to write the hard problems list again in 2019, some would be a continuation of the above problems, but there would be significant changes in emphasis, as well as significant new problems. Here are a few picks:

- **Cryptographic obfuscation:** same as #4 above
- **Ongoing work on post-quantum cryptography:** both hash-based as well as based on post-quantum-secure "structured" mathematical objects, eg. elliptic curve isogenies, lattices...
- **Anti-collusion infrastructure:** ongoing work and refinement of <https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413> (<https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413>), including adding privacy against the operator, adding multi-party computation in a maximally practical way, etc.
- **Oracles:** same as #16 above, but removing the emphasis on "success metrics" and focusing on the general "get real-world data" problem
- **Unique-human identities** (or, more realistically, semi-unique-human identities): same as what was written as #15 above, but with an emphasis on a less "absolute" solution: it should be much harder to get two identities than one, but making it impossible to get multiple identities is both impossible and potentially harmful even if we do succeed
- **Homomorphic encryption and multi-party computation:** ongoing improvements are still required for practicality
- **Decentralized governance mechanisms:** DAOs are cool, but current DAOs are still very primitive; we can do better
- **Fully formalizing responses to PoS 51% attacks:** ongoing work and refinement of <https://ethresear.ch/t/responding-to-51-attacks-in-casper-ffg/6363> (<https://ethresear.ch/t/responding-to-51-attacks-in-casper-ffg/6363>).
- **More sources of public goods funding:** the ideal is to charge for congestible resources inside of systems that have network effects (eg. transaction fees), but doing so in decentralized systems requires public legitimacy; hence this is a social problem along with the technical one of finding possible sources
- **Reputation systems:** same as #12 above

In general, base-layer problems are slowly but surely decreasing, but application-layer problems are only just getting started.

Quadratic Payments: A Primer

2019 Dec 07

[See all posts \(/\)](#)

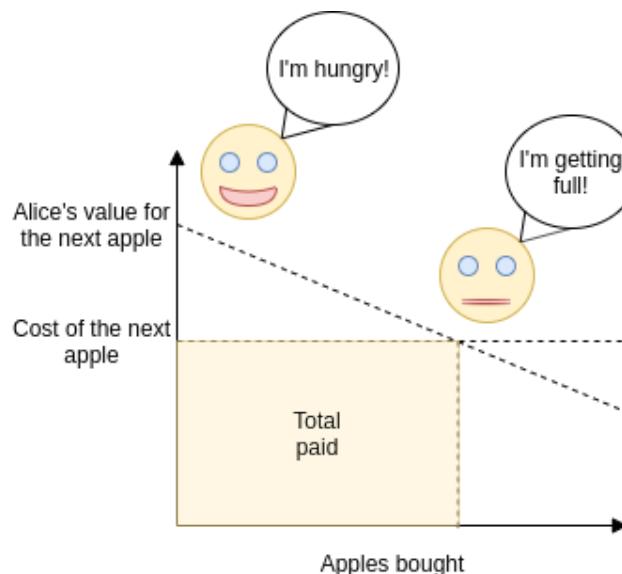
Special thanks to Karl Floersch and Jinglan Wang for feedback

If you follow applied mechanism design or decentralized governance at all, you may have recently heard one of a few buzzwords: [quadratic voting](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2003531) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2003531), [quadratic funding](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656), and [quadratic attention purchase](https://kortina.nyc/essays/speech-is-free-distribution-is-not-a-tax-on-the-purchase-of-human-attention-and-political-power/) (<https://kortina.nyc/essays/speech-is-free-distribution-is-not-a-tax-on-the-purchase-of-human-attention-and-political-power/>). These ideas have been gaining popularity rapidly over the last few years, and small-scale tests have already been deployed: the [Taiwanese presidential hackathon](https://presidential-hackathon.taiwan.gov.tw/en/) (<https://presidential-hackathon.taiwan.gov.tw/en/>) used quadratic voting to vote on winning projects, Gitcoin Grants [used quadratic funding](https://vitalik.ca/general/2019/10/24/gitcoin.html) (<https://vitalik.ca/general/2019/10/24/gitcoin.html>) to fund public goods in the Ethereum ecosystem, and the Colorado Democratic party [also experimented with](https://www.wired.com/story/colorado-quadratic-voting-experiment) (<https://www.wired.com/story/colorado-quadratic-voting-experiment>) quadratic voting to determine their party platform.

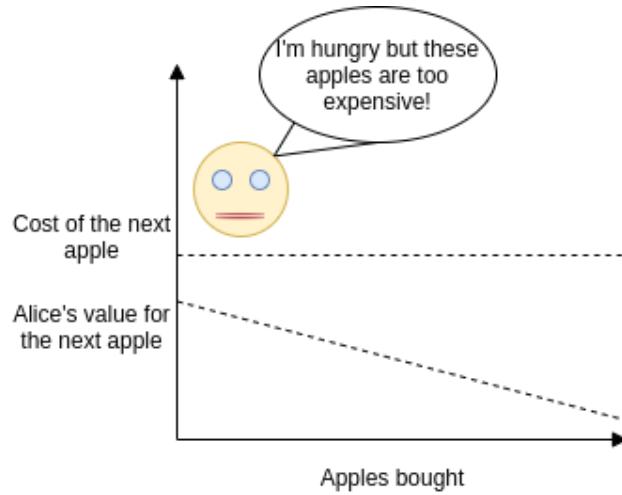
To the proponents of these voting schemes, this is not just another slight improvement to what exists. Rather, it's an initial foray into a fundamentally new class of social technology which, has the potential to overturn how we make many public decisions, large and small. The ultimate effect of these schemes rolled out in their full form *could be as deeply transformative as the industrial-era advent of mostly-free markets and constitutional democracy*. But now, you may be thinking: "These are large promises. What do these new governance technologies have that justifies such claims?"

Private goods, private markets...

To understand what is going on, let us first consider an existing social technology: money, and property rights - the invisible social technology that generally hides behind money. Money and private property are extremely powerful social technologies, for all the reasons classical economists have been stating for over a hundred years. If Bob is producing apples, and Alice wants to buy apples, we can economically model the interaction between the two, and the results seem to make sense:



Alice keeps buying apples until the marginal value of the next apple to her is less than the cost of producing it, which is pretty much exactly the optimal thing that could happen. And if the cost of producing the apples is greater than their value to Alice, then Alice just doesn't buy any:



This is all formalized in results such as the "[fundamental theorems of welfare economics](#) (https://en.wikipedia.org/wiki/Fundamental_theorems_of_welfare_economics)". Now, those of you who have learned some economics may be screaming, but what about [imperfect competition](#) (https://en.wikipedia.org/wiki/Imperfect_competition)? [Asymmetric information](#) (https://en.wikipedia.org/wiki/Information_asymmetry)? [Economic inequality](#) (https://en.wikipedia.org/wiki/Economic_inequality)? [Public goods](#) (https://en.wikipedia.org/wiki/Public_good)? [Externalities](#) (<https://en.wikipedia.org/wiki/Externality>)? Many activities in the real world, including those that are key to the progress of human civilization, benefit (or harm) many people in complicated ways. These activities and the consequences that arise from them often cannot be neatly decomposed into sequences of distinct trades between two parties.

But since when do we expect a single package of technologies to solve every problem anyway? "What about oceans?" isn't an argument against cars, it's an argument against *car maximalism*, the position that we need cars and nothing else. Much like how private property and markets deal with private goods, can we try to use economic means to deduce what kind of social technologies would work well for encouraging production of the public goods that we need?

... Public goods, public markets

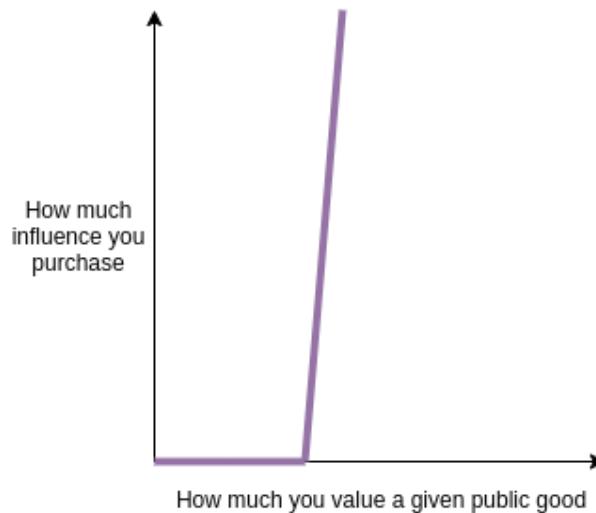
Private goods (eg. apples) and public goods (eg. public parks, air quality, scientific research, this article...) are different in some key ways. When we are talking about private goods, production for multiple people (eg. the same farmer makes apples for both Alice and Bob) can be decomposed into (i) the farmer making some apples for Alice, and (ii) the farmer making some other apples for Bob. If Alice wants apples but Bob does not, then the farmer makes Alice's apples, collects payment from Alice, and leaves Bob alone. Even complex collaborations (the "[I, Pencil](#)" essay (<https://fee.org/resources/i-pencil/>) popular in libertarian circles comes to mind) can be decomposed into a series of such interactions. When we are talking about public goods, however, *this kind of decomposition is not possible*. When I write this blog article, it can be read by both Alice and Bob (and everyone else). I could put it behind a paywall, but if it's popular enough it will inevitably get mirrored on third-party sites,

and paywalls are in any case annoying and not very effective. Furthermore, making an article available to ten people is not ten times cheaper than making the article available to a hundred people; rather, *the cost is exactly the same*. So I either produce the article for everyone, or I do not produce it for anyone at all.

So here comes the challenge: how do we aggregate together people's preferences? Some private and public goods are worth producing, others are not. In the case of private goods, the question is easy, because we can just decompose it into a series of decisions for each individual. Whatever amount each person is willing to pay for, that much gets produced for them; the economics is not especially complex. In the case of public goods, however, you cannot "decompose", and so we need to add up people's preferences in a different way.

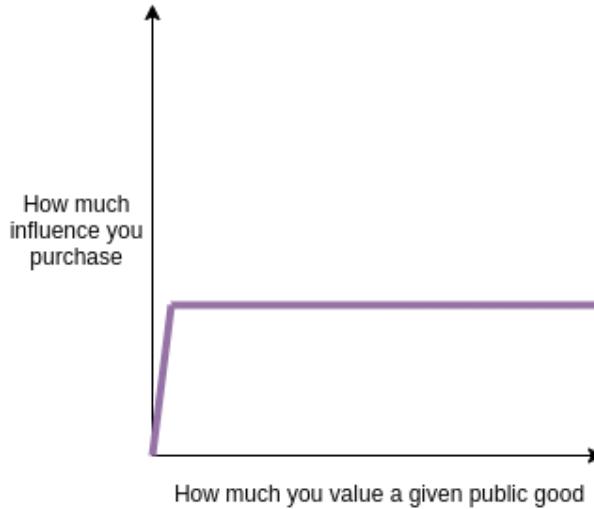
First of all, let's see what happens if we just put up a plain old regular market: I offer to write an article as long as at least \$1000 of money gets donated to me (fun fact: I literally did this back in 2011 (<https://bitcointalk.org/index.php?topic=28681.msg360909#msg360909>)). Every dollar donated increases the probability that the goal will be reached and the article will be published; let us call this "marginal probability" p . At a cost of \$ k , you can increase the probability that the article will be published by $k * p$ (though eventually the gains will decrease as the probability approaches 100%). Let's say to you personally, the article being published is worth \$ v . Would you donate? Well, donating a dollar increases the probability it will be published by p , and so gives you an expected $p * v$ of value. If $p * v > 1$, you donate, and quite a lot, and if $p * v < 1$ you don't donate at all.

Phrased less mathematically, either you value the article enough (and/or are rich enough) to pay, and if that's the case it's in your interest to keep paying (and influencing) quite a lot, or you don't value the article enough and you contribute nothing. Hence, the only blog articles that get published would be articles where some single person is willing to basically pay for it themselves (<https://en.wikipedia.org/wiki/Patronage>). (in my experiment in 2011, this prediction was experimentally verified: in most (<https://bitcointalk.org/index.php?topic=23934.msg306437#msg306437>) rounds (<https://bitcointalk.org/index.php?topic=28681.msg360909#msg360909>), over half of the total contribution came from a single donor).



Note that *this reasoning applies for any kind of mechanism that involves "buying influence" over matters of public concern*. This includes paying for public goods, shareholder voting in corporations, public advertising, bribing politicians, and much more. The little guy has too little influence (not quite zero, because in the real world things like altruism exist) and the big guy has too much. If you had an intuition that markets work great for buying apples, but money is corrupting in "the public sphere", this is basically a simplified mathematical model that shows why.

We can also consider a different mechanism: one-person-one-vote. Let's say you can either vote that I deserve a reward for writing this article, or you can vote that I don't, and my reward is proportional to the number of votes in my favor. We can interpret this as follows: your first "contribution" costs only a small amount of effort, so you'll support an article if you care about it enough, but after that point there is no more room to contribute further; your second contribution "costs" infinity.

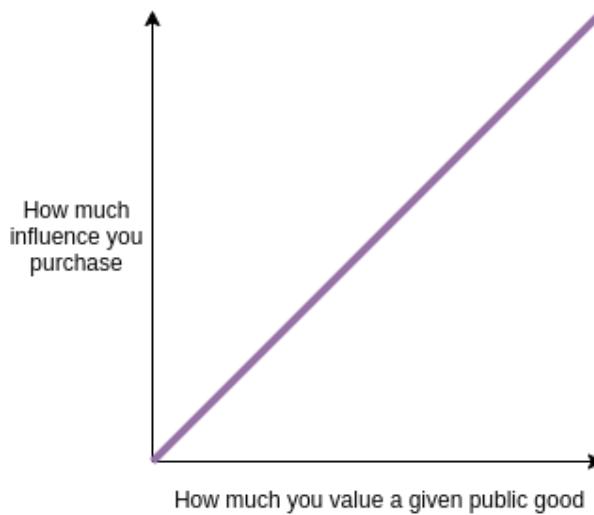


Now, you might notice that neither of the graphs above look quite right. The first graph over-privileges people who care *a lot* (or are wealthy), the second graph over-privileges people who care *only a little*, which is also a problem. The single sheep's desire to live is more important than the two wolves' desire to have a tasty dinner.

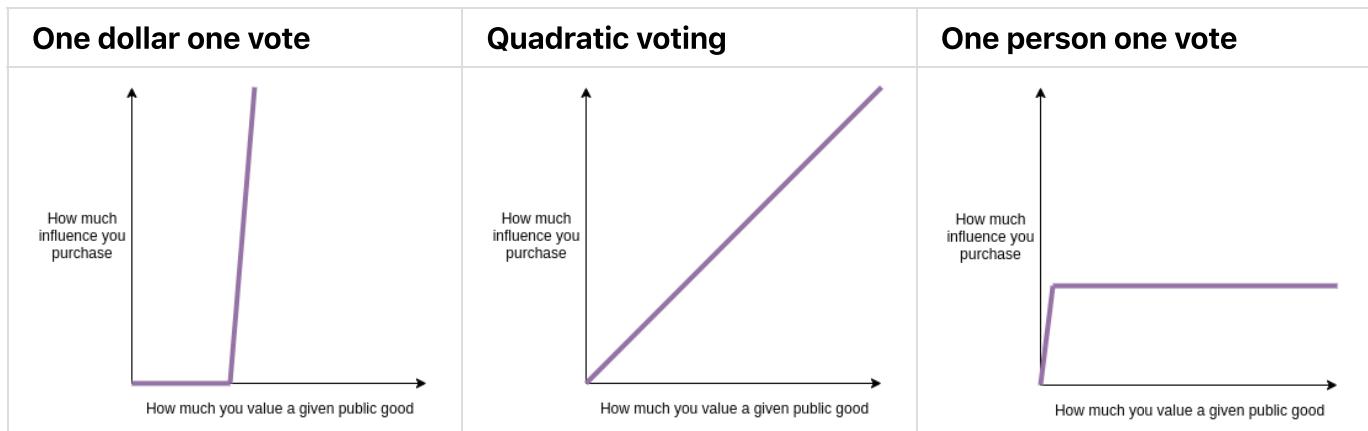
But what do we actually want? Ultimately, we want a scheme where *how much influence you "buy" is proportional to how much you care*. In the mathematical lingo above, we want your k to be proportional to your v . But here's the problem: your v determines how much you're willing to pay for *one unit of influence*. If Alice were willing to pay \$100 for the article if she had to fund it herself, then she would be willing to pay \$1 for an increased 1% chance it will get written, and if Bob were only willing to pay \$50 for the article then he would only be willing to pay \$0.5 for the same "unit of influence".

So how do we match these two up? The answer is clever: *your n'th unit of influence costs you \$n*. That is, for example, you could buy your first vote for \$0.01, but then your second would cost \$0.02, your third \$0.03, and so forth. Suppose you were Alice in the example above; in such a system she would keep buying units of influence until the cost of the next one got to \$1, so she would buy 100 units. Bob would similarly buy until the cost got to \$0.5, so he would buy 50 units. Alice's 2x higher valuation turned into 2x more units of influence purchased.

Let's draw this as a graph:

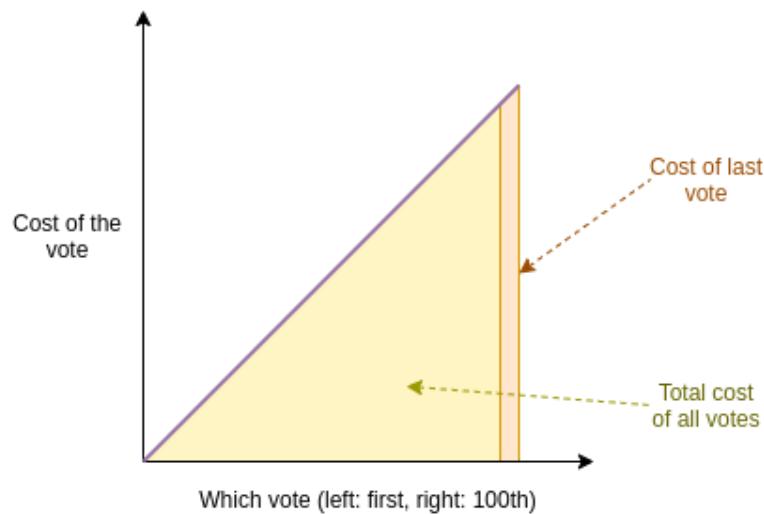


Now let's look at all three beside each other:



Notice that only quadratic voting has this nice property that the amount of influence you purchase is proportional to how much you care; the other two mechanisms either over-privilege concentrated interests or over-privilege diffuse interests.

Now, you might ask, where does the *quadratic* come from? Well, the *marginal* cost of the n^{th} vote is $\$n$ (or $\$0.01 \cdot n$), but the *total* cost of n votes is $\approx \frac{n^2}{2}$. You can view this geometrically as follows:



The total cost is the area of a triangle, and you probably learned in math class that area is base * height / 2. And since here base and height are proportionate, that basically means that total cost is proportional to number of votes squared - hence, "quadratic". But honestly it's easier to think "your n'th unit of influence costs \$n".

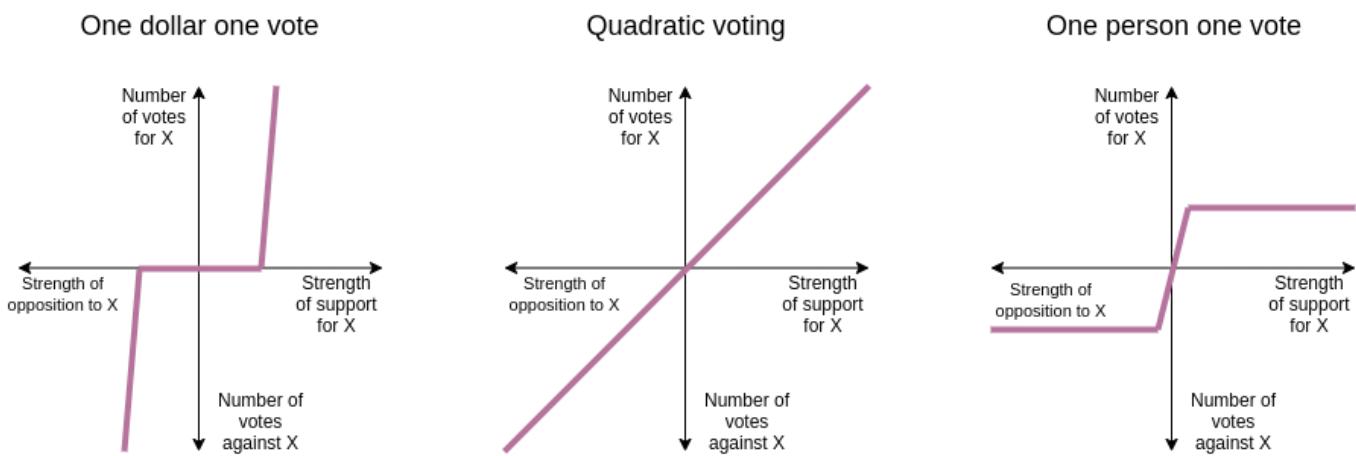
Finally, you might notice that above I've been vague about what "one unit of influence" actually means. This is deliberate; it can mean different things in different contexts, and the different "flavors" of quadratic payments reflect these different perspectives.

Quadratic Voting

See also the original paper: <https://papers.ssrn.com/sol3/papers.cfm?abstract%5fid=2003531>
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2003531

Let us begin by exploring the first "flavor" of quadratic payments: quadratic voting. Imagine that some organization is trying to choose between two choices for some decision that affects all of its members. For example, this could be a company or a nonprofit deciding which part of town to make a new office in, or a government deciding whether or not to implement some policy, or an internet forum deciding whether or not its rules should allow discussion of cryptocurrency prices. Within the context of the organization, the choice made is a public good (or public bad, depending on whom you talk to): everyone "consumes" the results of the same decision, they just have different opinions about how much they like the result.

This seems like a perfect target for quadratic voting. The goal is that option A gets chosen if in total people like A more, and option B gets chosen if in total people like B more. With simple voting ("one person one vote"), the distinction between stronger vs weaker preferences gets ignored, so on issues where one side is of very high value to a few people and the other side is of low value to more people, simple voting is likely to give wrong answers. With a private-goods market mechanism where people can buy as many votes as they want at the same price per vote, the individual with the strongest preference (or the wealthiest) carries everything. Quadratic voting, where you can make n votes in either direction at a cost of n^2 , is right in the middle between these two extremes, and creates the perfect balance.



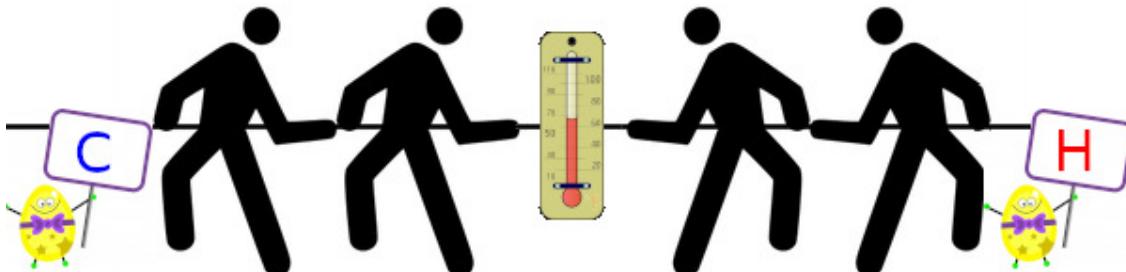
Note that in the voting case, we're deciding two options, so different people will favor A over B or B over A; hence, unlike the graphs we saw earlier that start from zero, here voting and preference can both be positive or negative (which option is considered positive and which is negative doesn't matter; the math works out the same way)

As shown above, because the n^{th} vote has a cost of n , the number of votes you make is proportional to how much you value one unit of influence over the decision (the value of the decision multiplied by the probability that one vote will tip the result), and hence proportional to how much you care about A being chosen over B or vice versa. Hence, we once again have this nice clean "preference adding" effect.

We can extend quadratic voting in multiple ways. First, we can allow voting between more than two options. While traditional voting schemes inevitably fall prey to various kinds of "strategic voting" issues because of [Arrow's theorem](https://en.wikipedia.org/wiki/Arrow%27s_impossibility_theorem) (https://en.wikipedia.org/wiki/Arrow%27s_impossibility_theorem) and [Duverger's law](https://en.wikipedia.org/wiki/Duverger%27s_law) (https://en.wikipedia.org/wiki/Duverger%27s_law), quadratic voting [continues to be optimal](#) (<http://www.econ.msu.edu/seminars/docs/QuadMultAltshort19.pdf>) in contexts with more than two choices.

The intuitive argument for those interested: suppose there are established candidates A and B and new candidate C. Some people favor $C > A > B$ but others $C > B > A$. In a regular vote, if both sides think C stands no chance, they decide may as well vote their preference between A and B, so C gets no votes, and C's failure becomes a self-fulfilling prophecy. In quadratic voting the former group would vote $[A +10, B -10, C +1]$ and the latter $[A -10, B +10, C +1]$, so the A and B votes cancel out and C's popularity shines through.

Second, we can look not just at voting between discrete options, but also at voting on the setting of a thermostat: anyone can push the thermostat up or down by 0.01 degrees n times by paying a cost of n^2 .



Plot twist: the side wanting it colder only wins when they convince the other side that "C" stands for "caliente".

Quadratic funding

See also the original paper: <https://papers.ssrn.com/sol3/papers.cfm?abstract%5fid=3243656> (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3243656).

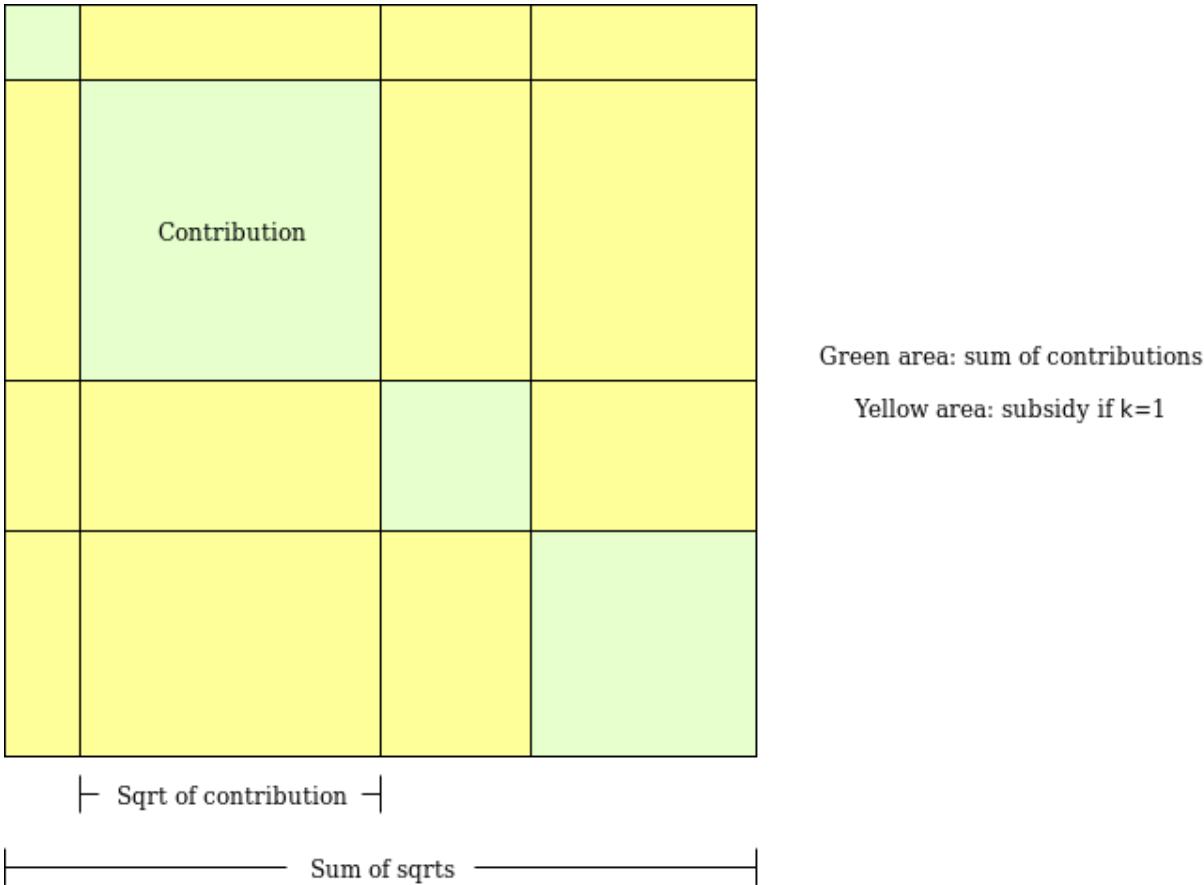
Quadratic voting is optimal when you need to make some fixed number of collective decisions. But one weakness of quadratic voting is that it doesn't come with a built-in mechanism for deciding what goes on the ballot in the first place. Proposing votes is potentially a source of considerable power if not handled with care: a malicious actor in control of it can repeatedly propose some decision that a majority weakly approves of and a minority strongly disapproves of, and keep proposing it until the minority runs out of voting tokens (if you do the math you'll see that the minority would burn through tokens much faster than the majority). Let's consider a flavor of quadratic payments that does not run into this issue, and makes the choice of decisions itself endogenous (ie. part of the mechanism itself). In this case, the mechanism is specialized for one particular use case: individual provision of public goods.

Let us consider an example where someone is looking to produce a public good (eg. a developer writing an open source software program), and we want to figure out whether or not this program is worth funding. But instead of just thinking about one single public good, let's create a mechanism where *anyone* can raise funds for what they claim to be a public good project. Anyone can make a contribution to any project; a mechanism keeps track of

these contributions and then at the end of some period of time the mechanism calculates a payment to each project. The way that this payment is calculated is as follows: for any given project, take the square root of each contributor's contribution, add these values together, and take the square of the result. Or in math speak:

$$\left(\sum_{i=1}^n \sqrt{c_i} \right)^2$$

If that sounds complicated, here it is graphically:

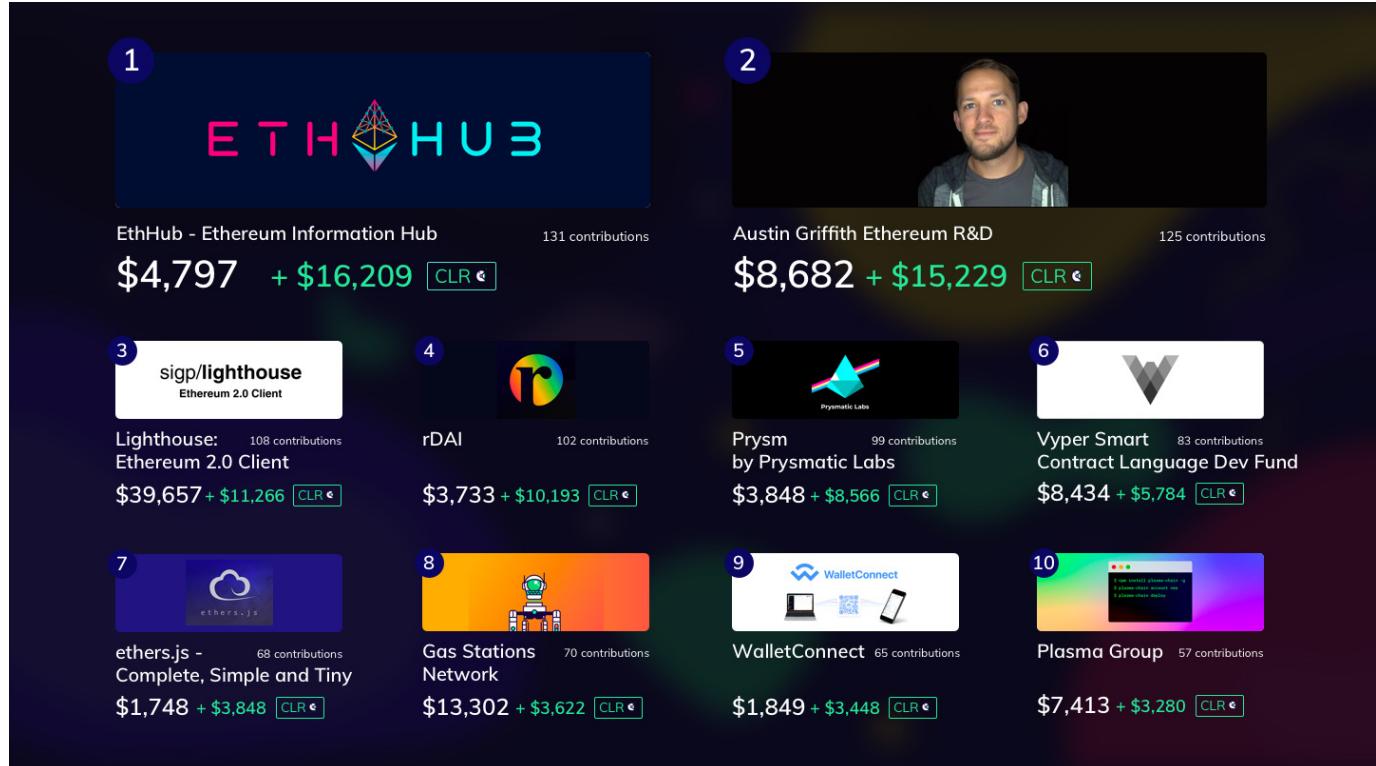


In any case where there is more than one contributor, the computed payment is greater than the raw sum of contributions; the difference comes out of a central subsidy pool (eg. if ten people each donate \$1, then the sum-of-square-roots is \$10, and the square of that is \$100, so the subsidy is \$90). Note that if the subsidy pool is not big enough to make the full required payment to every project, we can just divide the subsidies proportionately by whatever constant makes the totals add up to the subsidy pool's budget; **you can prove that this solves the tragedy-of-the-commons problem as well as you can with that subsidy budget.**

There are two ways to intuitively interpret this formula. First, one can look at it through the "fixing market failure" lens, a surgical fix to the [tragedy of the commons](https://en.wikipedia.org/wiki/Tragedy_of_the_commons) (https://en.wikipedia.org/wiki/Tragedy_of_the_commons) problem. In any situation where Alice contributes to a project and Bob also contributes to that same project, Alice is making a contribution to something that is valuable not only to herself, but also to Bob. When deciding *how much to contribute*, Alice was only taking into account the benefit to herself, not Bob, whom she most likely does not even know. The quadratic funding mechanism adds a subsidy to compensate for this effect, determining how much Alice "would have" contributed if she also took into account the benefit her contribution brings to Bob. Furthermore, we can separately calculate the subsidy for each pair of people (nb. if there are N people there are $N * (N-1) / 2$ pairs), and add up all of these subsidies together, and give Bob the combined subsidy from all pairs. And it turns out that this gives exactly the quadratic funding formula.

Second, one can look at the formula through a quadratic voting lens. We interpret the quadratic funding as being a *special case* of quadratic voting, where the contributors to a project are voting for that project and there is one imaginary participant voting against it: the subsidy pool. Every "project" is a motion to take money from the subsidy pool and give it to that project's creator. Everyone sending c_i of funds is making $\sqrt{c_i}$ votes, so there's a total of $\sum_{i=1}^n \sqrt{c_i}$ votes in favor of the motion. To kill the motion, the subsidy pool would need to make more than $\sum_{i=1}^n \sqrt{c_i}$ votes against it, which would cost it more than $(\sum_{i=1}^n \sqrt{c_i})^2$. Hence, $(\sum_{i=1}^n \sqrt{c_i})^2$ is the maximum transfer from the subsidy pool to the project that the subsidy pool would not vote to stop.

Quadratic funding is starting to be explored as a mechanism for funding public goods already; [Gitcoin grants](https://vitalik.ca/general/2019/10/24/gitcoin.html) (<https://vitalik.ca/general/2019/10/24/gitcoin.html>) for funding public goods in the Ethereum ecosystem is currently the biggest example, and the most recent round led to results that, in my own view, did a quite good job of making a fair allocation to support projects that the community deems valuable.

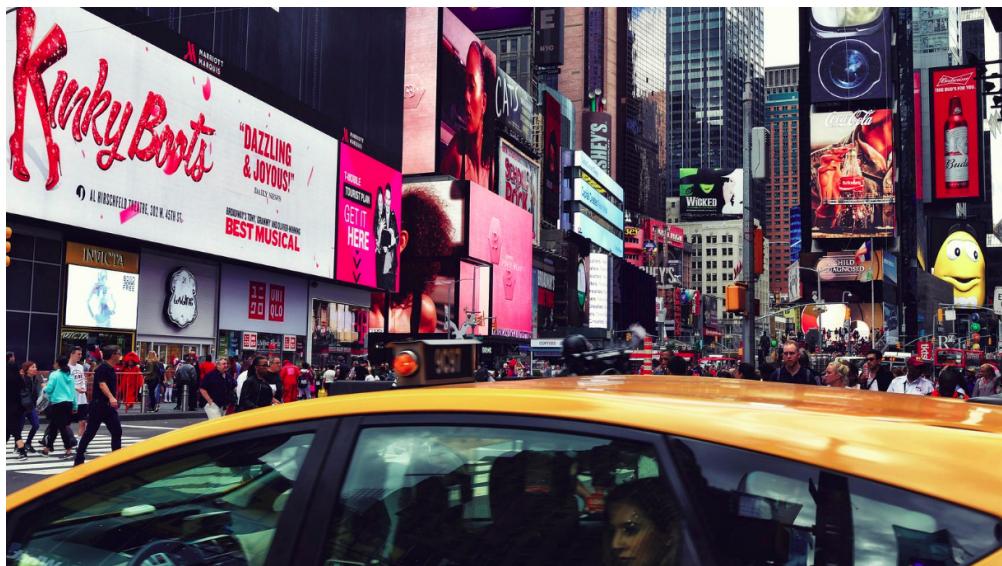


Numbers in white are raw contribution totals; numbers in green are the extra subsidies.

Quadratic attention payments

See also the original post: <https://kortina.nyc/essays/speech-is-free-distribution-is-not-a-tax-on-the-purchase-of-human-attention-and-political-power/> (<https://kortina.nyc/essays/speech-is-free-distribution-is-not-a-tax-on-the-purchase-of-human-attention-and-political-power/>).

One of the defining features of modern capitalism that people love to hate is ads. Our cities have ads:



Source: <https://www.flickr.com/photos/argonavigo/36657795264>. (<https://www.flickr.com/photos/argonavigo/36657795264>).

Our subway turnstiles have ads:



Source: https://commons.wikimedia.org/wiki/File:NYC_subway_ad_on_Prince_St.jpg.
(https://commons.wikimedia.org/wiki/File:NYC_subway_ad_on_Prince_St.jpg).

Our politics are dominated by ads:



Source:

https://upload.wikimedia.org/wikipedia/commons/e/e3/Billboard_Challenging_the_validity_of_Barack_Obama%27s_Birth_Certificate.JPG
https://upload.wikimedia.org/wikipedia/commons/e/e3/Billboard_Challenging_the_validity_of_Barack_Obama%27s_Birth_Certificate.JF

And even the rivers and the skies have ads (<https://newyork.cbslocal.com/2018/11/13/are-led-boat-advertisements-on-the-hudson-river-going-a-step-too-far/>). Now, there are some places that seem to not have this problem:

t Comrade Natalie Retweeted

Comrade Natalie
@NatalieRevolts

The DPRK is like having adblock in real life 😊❤️

8:30 AM · Nov 2, 2018 · Twitter Web Client

1.1K Retweets 4.4K Likes

But really they just have a different kind of ads:



Now, recently there are attempts to move beyond this in some cities

(<https://www.theguardian.com/cities/2015/aug/11/can-cities-kick-ads-ban-urban-billboards>). And on Twitter (<https://twitter.com/jack/status/1189634360472829952>). But let's look at the problem systematically and try to see what's going wrong. The answer is actually surprisingly simple: public advertising is the evil twin of public goods production. In the case of public goods production, there is one actor that is taking on an expenditure to produce some product, and this product benefits a large number of people. Because these people cannot effectively coordinate to pay for the public goods by themselves, we get much less public goods than we need, and the ones we do get are those favored by wealthy actors or centralized authorities. Here, there is one actor that reaps a large *benefit* from forcing other people to look at some image, and this action *harms* a large number of people. Because these people cannot effectively coordinate to buy out the slots for the ads, we get ads we don't want to see, that are favored by... wealthy actors or centralized authorities.

So how do we solve this dark mirror image of public goods production? With a bright mirror image of quadratic funding: quadratic fees! Imagine a billboard where anyone can pay \$1 to put up an ad for one minute, but if they want to do this multiple times the prices go up: \$2 for the second minute, \$3 for the third minute, etc. Note that you can pay to extend the lifetime of *someone else's* ad on the billboard, and this also costs you only \$1 for the first minute, even if *other people already paid to extend the ad's lifetime many times*. We can once again interpret this as being a special case of quadratic voting: it's basically the same as the "voting on a thermostat" example above, but where the thermostat in question is the number of seconds an ad stays up.

This kind of payment model could be applied in cities, on websites, at conferences, or in many other contexts, if the goal is to optimize for putting up things that people want to see (or things that people want other people to see, but even here it's much more democratic than simply buying space) rather than things that wealthy people and centralized institutions want people to see.

Complexities and caveats

Perhaps the biggest challenge to consider with this concept of quadratic payments is the practical implementation issue of identity and bribery/collusion (<https://vitalik.ca/general/2019/04/03/collusion.html>). Quadratic payments in any form require a model of identity where individuals cannot easily get as many identities as they want: if they could, then they could just keep getting new identities and keep paying \$1 to influence some decision as many times as they want, and the mechanism collapses into linear vote-buying. Note that the identity system does *not* need to be airtight (in the sense of preventing multiple-identity acquisition), and indeed there are good civil-liberties reasons why identity systems probably should *not* try to be airtight. Rather, it just needs to be robust enough that manipulation is not worth the cost.

Collusion is also tricky. If we can't prevent people from selling their votes, the mechanisms once again collapse into one-dollar-one-vote. We don't just need votes to be anonymous and private (while still making the final result provable and public); **we need votes to be so private that even the person who made the vote can't prove to anyone else what they voted for.** This is difficult. Secret ballots do this well in the offline world, but secret ballots are a nineteenth century technology, far too inefficient for the sheer amount of quadratic voting and funding that we want to see in the twenty first century.

Fortunately, there are technological means that can help (<https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413>), combining together zero-knowledge proofs, encryption and other cryptographic technologies to achieve the precise desired set of privacy and verifiability properties. There's also proposed techniques (<https://twitter.com/phildajan/status/1181822995993681921>) to verify that private keys actually are in an individual's possession and not in some hardware or cryptographic system that can restrict how they use those keys. However, these techniques are all untested and require quite a bit of further work.

Another challenge is that quadratic payments, being a payment-based mechanism, continues to favor people with more money. Note that because the cost of votes is quadratic, this effect is dampened: someone with 100 times more money only has 10 times more influence, not 100 times, so the extent of the problem goes down by 90% (and even more for ultra-wealthy actors). That said, it may be desirable to mitigate this inequality of power further. This could be done either by denominating quadratic payments in a separate token of which everyone gets a fixed number of units, or giving each person an allocation of funds that can only be used for quadratic-payments use cases: this is basically Andrew Yang's "democracy dollars" (<https://www.yang2020.com/policies/democracydollars/>) proposal.



A third challenge is the "rational ignorance (https://en.wikipedia.org/wiki/Rational_ignorance)" and "rational irrationality (https://en.wikipedia.org/wiki/Rational_irrationality)" problems, which is that decentralized public decisions have the weakness that any single individual has very little effect on the outcome, and so little motivation to make sure they are supporting the decision that is best for the long term; instead, pressures such as tribal affiliation may dominate. There are many strands of philosophy that emphasize the ability of large crowds to be very wrong despite (or because of!) their size, and quadratic payments in any form do little to address this.

Quadratic payments do better at mitigating this problem than one-person-one-vote systems, and these problems can be expected to be less severe for medium-scale public goods than for large decisions that affect many millions of people, so it may not be a large challenge at first, but it's certainly an issue worth confronting. One approach is combining quadratic voting with elements of sortition (<https://ethresear.ch/t/quadratic-voting-with-sortition/6065>). Another, potentially more long-term durable, approach is to combine quadratic voting with another economic technology that is much more specifically targeted toward rewarding the "correct contrarianism" that can dispel mass delusions: prediction markets (https://en.wikipedia.org/wiki/Prediction_market). A

simple example would be a system where quadratic funding is done *retrospectively*, so people vote on which public goods were valuable some time ago (eg. even 2 years), and projects are funded up-front by selling shares of the results of these deferred votes; by buying shares people would be both funding the projects and betting on which project would be viewed as successful in 2 years' time. There is a large design space to experiment with here.

Conclusion

As I mentioned at the beginning, quadratic payments do not solve every problem. They solve the problem of governing resources that affect large numbers of people, but they do not solve many other kinds of problems. A particularly important one is information asymmetry and low quality of information in general. For this reason, I am a fan of techniques such as prediction markets (see electionbettingodds.com (<https://electionbettingodds.com/>)) for one example) to solve information-gathering problems, and many applications can be made most effective by combining different mechanisms together.

One particular cause dear to me personally is what I call "entrepreneurial public goods": public goods that in the present only a few people believe are important but in the future many more people will value. In the 19th century, contributing to abolition of slavery may have been one example; in the 21st century I can't give examples that will satisfy every reader because it's the nature of these goods that their importance will only become common knowledge later down the road, but I would point to [life extension](https://www.sens.org/) (<https://www.sens.org/>) and [AI risk research](https://intelligence.org/) (<https://intelligence.org/>) as two possible examples.

That said, we don't need to solve every problem today. Quadratic payments are an idea that has only become popular in the last few years; we still have not seen more than small-scale trials of quadratic voting and funding, and quadratic attention payments have not been tried at all! There is still a long way to go. But if we can get these mechanisms off the ground, there is a lot that these mechanisms have to offer!

Christmas Special

2019 Dec 24

[See all posts \(/\)](#)

Since it's Christmas time now, and we're theoretically supposed to be enjoying ourselves and spending time with our families instead of waging endless holy wars on Twitter, this blog post will offer some games that you can play with your friends that will help you have fun *and* at the same time understand some spooky mathematical concepts!

1.58 dimensional chess

**Emin Gün Sirer**

@el33th4xor

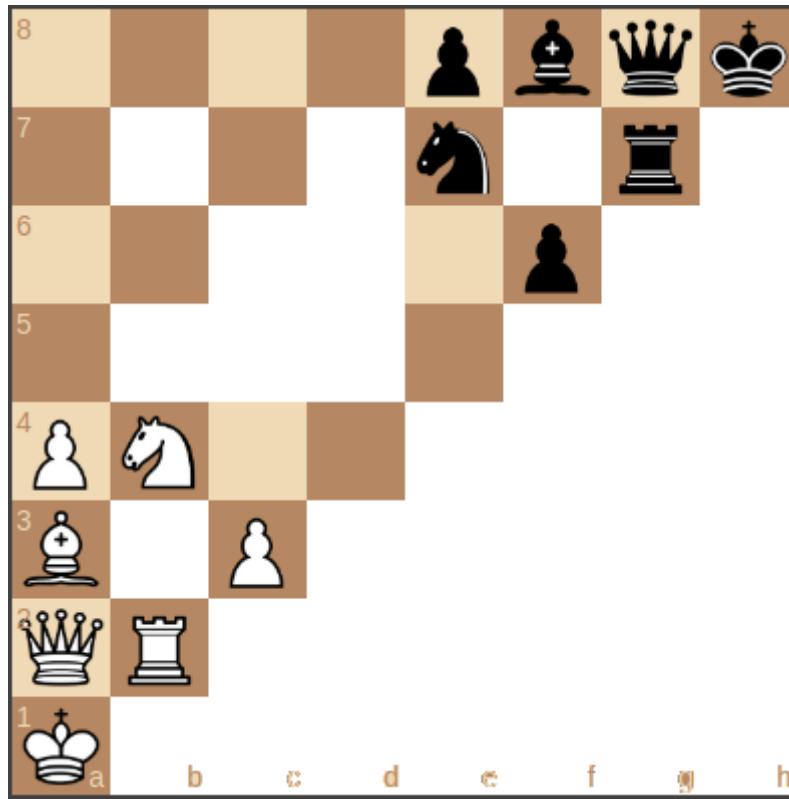
▼

A vignette from the IC3 Bootcamp, where people unwind, among other things, by playing "1.58 dimensional chess," a game of Vitalik's invention that's surprisingly fun.



(<https://twitter.com/el33th4xor/status/1138777837320716288>).

This is a variant of chess where the board is set up like this:



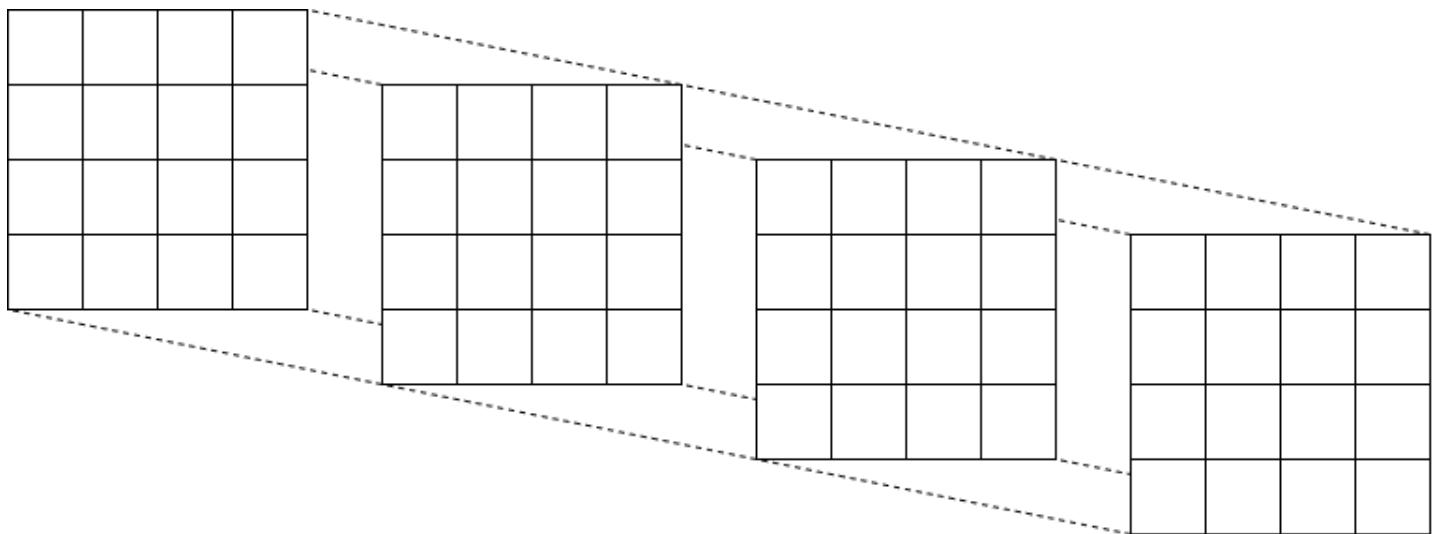
The board is still a normal 8x8 board, but there are only 27 open squares. The other 37 squares should be covered up by checkers or Go pieces or anything else to denote that they are inaccessible. The rules are the same as chess, with a few exceptions:

- White pawns move up, black pawns move left. White pawns take going left-and-up or right-and-up, black pawns take going left-and-down or left-and-up. White pawns promote upon reaching the top, black pawns promote upon reaching the left.
- No en passant, castling, or two-step-forward pawn jumps.
- Chess pieces cannot move onto or *through* the 37 covered squares. Knights cannot move onto the 37 covered squares, but don't care what they move "through".

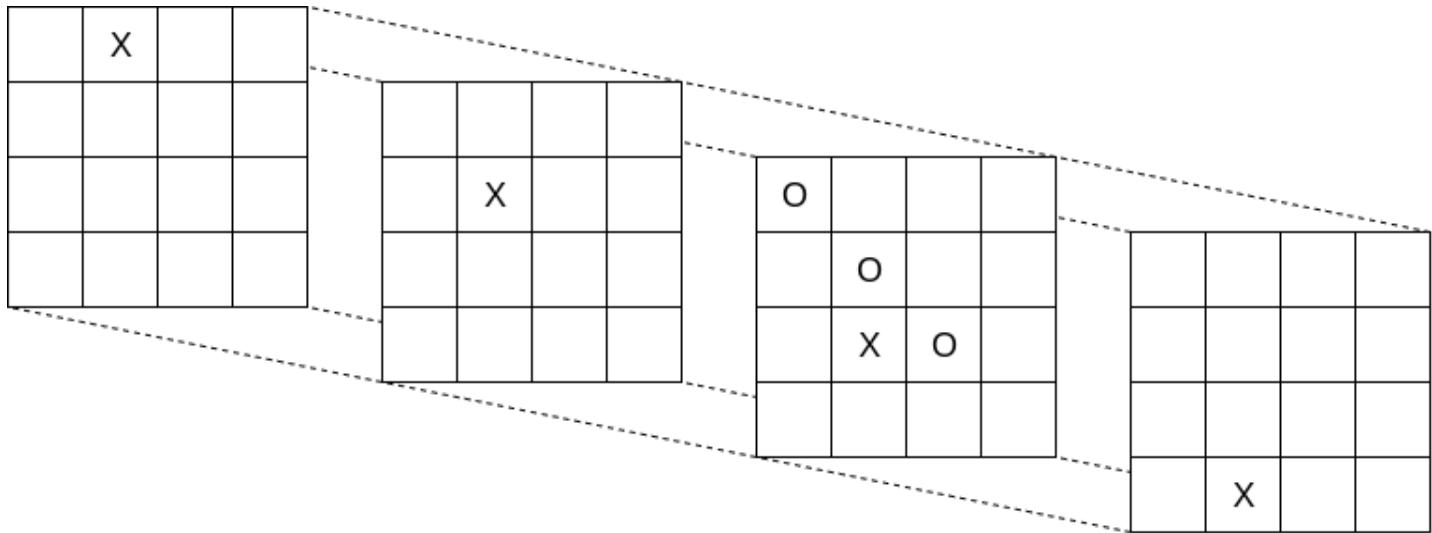
The game is called 1.58 dimensional chess because the 27 open squares are chosen according to a pattern based on the [Sierpinski triangle](https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle) (https://en.wikipedia.org/wiki/Sierpi%C5%84ski_triangle). You start off with a single open square, and then every time you double the width, you take the shape at the end of the previous step, and copy it to the top left, top right and bottom left corners, but leave the bottom right corner inaccessible. Whereas in a one-dimensional structure, doubling the width increases the space by $2x$, and in a two-dimensional structure, doubling the width increases the space by $4x$ ($4 = 2^2$), and in a three-dimensional structure, doubling the width increases the space by $8x$ ($8 = 2^3$), here doubling the width increases the space by $3x$ ($3 = 2^{1.58496}$), hence "1.58 dimensional" (see [Hausdorff dimension](https://en.wikipedia.org/wiki/Hausdorff_dimension) (https://en.wikipedia.org/wiki/Hausdorff_dimension) for details).

The game is substantially simpler and more "tractable" than full-on chess, and it's an interesting exercise in showing how in [lower-dimensional spaces](https://en.wikipedia.org/wiki/Flatland) (<https://en.wikipedia.org/wiki/Flatland>), defense becomes much easier than offense. Note that the relative value of different pieces may change here, and new kinds of endings become possible (eg. you can checkmate with just a bishop).

3 dimensional tic tac toe



The goal here is to get 4 in a straight line, where the line can go in any direction, along an axis or diagonal, including between planes. For example in this configuration X wins:



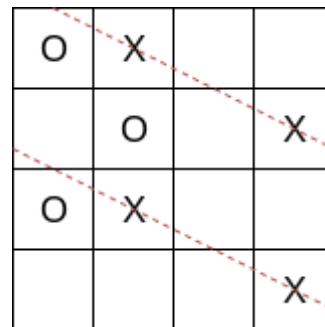
It's considerably harder than [traditional 2D tic tac toe](https://www.quora.com/Is-there-a-way-to-never-lose-at-Tic-Tac-Toe) (<https://www.quora.com/Is-there-a-way-to-never-lose-at-Tic-Tac-Toe>), and hopefully much more fun!

Modular tic-tac-toe

Here, we go back down to having two dimensions, except we allow lines to wrap around:

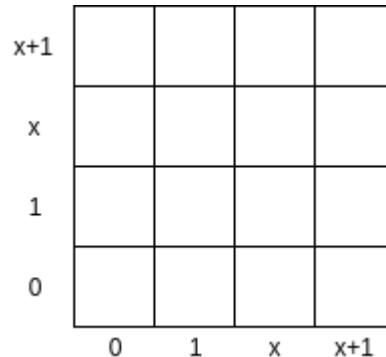
	X		
X	O	O	O
			X
		X	

Note that we allow diagonal lines with any slope, as long as they pass through all four points. Particularly, this means that lines with slope $+/ - 2$ and $+/ - 1/2$ are admissible:



Mathematically, the board can be interpreted as a 2-dimensional vector space over integers modulo 4 (https://en.wikipedia.org/wiki/Modular_arithmetic), and the goal being to fill in a line that passes through four points over this space. Note that there exists at least one line passing through any two points.

Tic tac toe over the 4-element binary field



Here, we have the same concept as above, except we use an even spookier mathematical structure, the 4-element field (https://en.wikipedia.org/wiki/Finite_field#Field_with_four_elements) of polynomials over \mathbb{Z}_2 modulo $x^2 + x + 1$. This structure has pretty much no reasonable geometric interpretation, so I'll just give you the addition and multiplication tables:

Addition

Multiplication

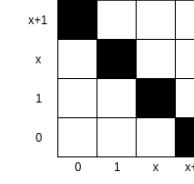
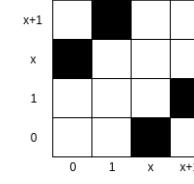
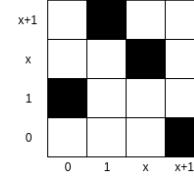
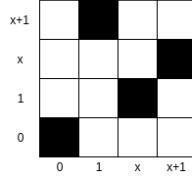
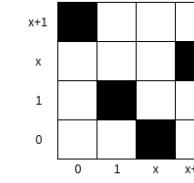
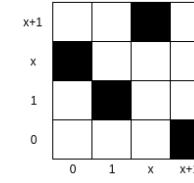
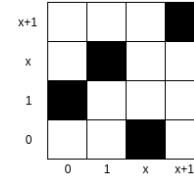
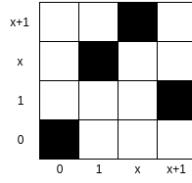
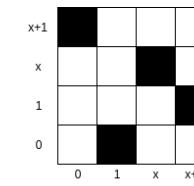
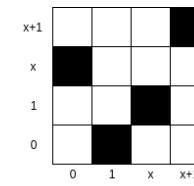
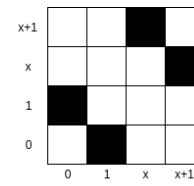
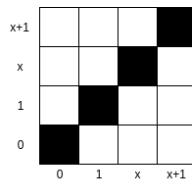
x+1	x+1	x	1	0
x	x	x+1	0	1
1	1	0	x+1	x
0	0	1	x	x+1

0 1 x x+1

x+1	0	x+1	1	x
x	0	x	x+1	1
1	0	1	x	x+1
0	0	0	0	0

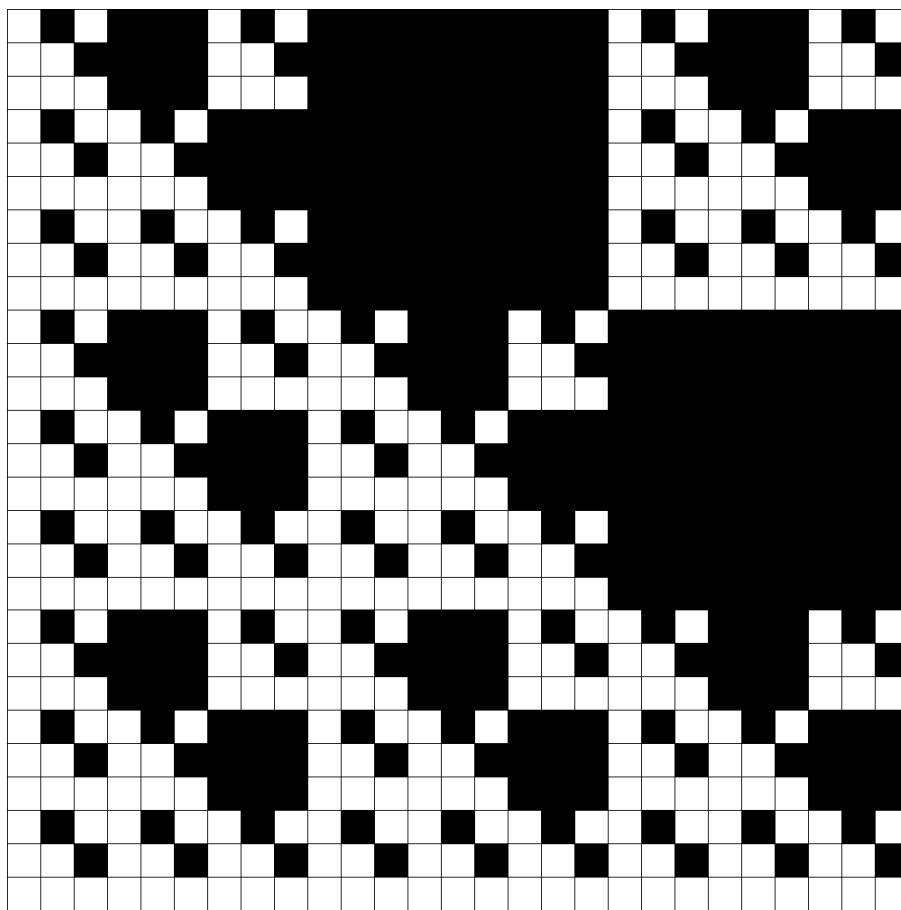
0 1 x x+1

OK fine, here are all possible lines, excluding the horizontal and the vertical lines (which are also admissible) for brevity:



The lack of geometric interpretation does make the game harder to play; you pretty much have to memorize the twenty winning combinations, though note that they are *basically* rotations and reflections of the same four basic shapes (axial line, diagonal line, diagonal line starting in the middle, that weird thing that doesn't look like a line).

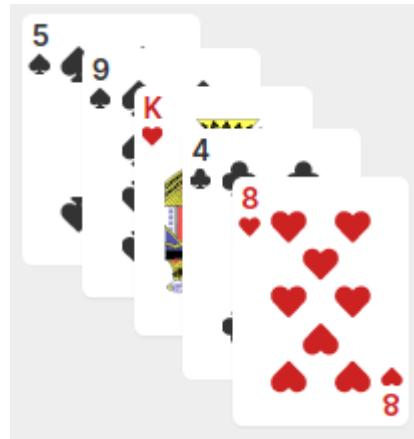
Now play 1.77 dimensional connect four. I dare you.



Modular poker

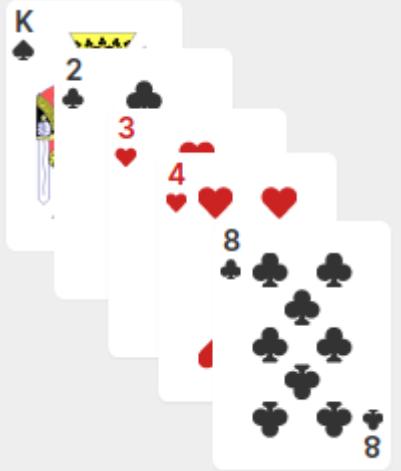
Everyone is dealt five (you can use whatever variant poker rules you want here in terms of how these cards are dealt and whether or not players have the right to swap cards out). The cards are interpreted as: jack = 11, queen = 12, king = 0, ace = 1. A hand is stronger than another hand, if it contains a longer sequence, with any constant difference between consecutive cards (allowing wraparound), than the other hand.

Mathematically, this can be represented as, a hand is stronger if the player can come up with a line $L(x) = mx + b$ such that they have cards for the numbers $L(0), L(1) \dots L(k)$ for the highest k .



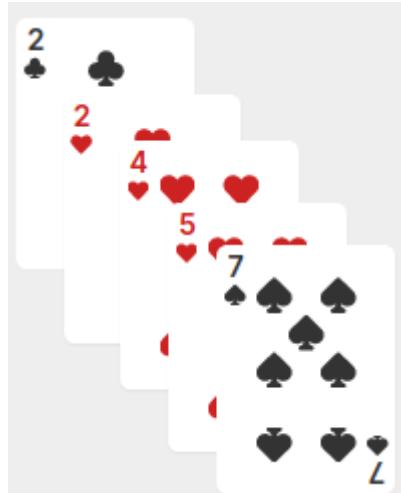
Example of a full five-card winning hand. $y = 4x + 5$.

To break ties between equal maximum-length sequences, count the number of distinct length-three sequences they have; the hand with more distinct length-three sequences wins.



This hand has four length-three sequences: K 2 4, K 4 8, 2 3 4, 3 8 K. This is rare.

Only consider lines of length three or higher. If a hand has three or more of the same denomination, that counts as a sequence, but if a hand has two of the same denomination, any sequences passing through that denomination only count as one sequence.



This hand has no length-three sequences.

If two hands are completely tied, the hand with the higher highest card (using J = 11, Q = 12, K = 0, A = 1 as above) wins.

Enjoy!

Base Layers And Functionality Escape Velocity

2019 Dec 26

[See all posts \(/\)](#)

One common strand of thinking in blockchain land goes as follows: blockchains should be maximally simple, because they are a piece of infrastructure that is difficult to change and would lead to great harms if it breaks, and more complex functionality should be built on top, in the form of layer 2 protocols: [state channels](https://www.jeffcoleman.ca/state-channels/) (<https://www.jeffcoleman.ca/state-channels/>), [Plasma](https://ethresear.ch/t/minimal-viable-plasma/426) (<https://ethresear.ch/t/minimal-viable-plasma/426>), [rollup](https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477) (<https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477>), and so forth. Layer 2 should be the site of ongoing innovation, layer 1 should be the site of stability and maintenance, with large changes only in emergencies (eg. a one-time set of serious breaking changes to prevent the base protocol's cryptography from falling to quantum computers would be okay).

This kind of layer separation is a very nice idea, and in the long term I strongly support this idea. However, this kind of thinking misses an important point: while layer 1 cannot be *too* powerful, as greater power implies greater complexity and hence greater brittleness, layer 1 must also be *powerful enough* for the layer 2 protocols-on-top that people want to build to actually be possible in the first place. Once a layer 1 protocol has achieved a certain level of functionality, which I will term "functionality escape velocity", then yes, you can do everything else on top without further changing the base. But if layer 1 is not powerful enough, then you can talk about filling in the gap with layer 2 systems, but the reality is that there is no way to actually build those systems, without reintroducing a whole set of trust assumptions that the layer 1 was trying to get away from. This post will talk about some of what this minimal functionality that constitutes "functionality escape velocity" is.

A programming language

It must be possible to execute custom user-generated scripts on-chain. This programming language can be simple, and actually does not need to be high-performance, but it needs to at least have the level of functionality required to be able to verify arbitrary things that might need to be verified. This is important because the layer 2 protocols that are going to be built on top need to have some kind of verification logic, and this verification logic must be executed by the blockchain somehow.

You may have heard of [Turing completeness](https://en.wikipedia.org/wiki/Turing_completeness) (https://en.wikipedia.org/wiki/Turing_completeness); the "layman's intuition" for the term being that if a programming language is Turing complete then it can do anything that a computer theoretically could do. Any program in one Turing-complete language can be translated into an equivalent program in any other Turing-complete language. However, it turns out that we only need something slightly lighter: it's okay to restrict to programs without loops, or programs which are [guaranteed to terminate](https://en.wikipedia.org/wiki/Total_functional_programming) (https://en.wikipedia.org/wiki/Total_functional_programming) in a specific number of steps.

Rich Statefulness

It doesn't just matter that a programming language *exists*, it also matters precisely how that programming language is integrated into the blockchain. Among the more constricted ways that a language could be integrated is if it is used for pure transaction verification: when you send coins to some address, that address represents a computer program P which would be used to verify a transaction that sends coins *from* that

address. That is, if you send a transaction whose hash is h , then you would supply a signature S , and the blockchain would run $P(h, S)$, and if that outputs TRUE then the transaction is valid. Often, P is a verifier for a cryptographic signature scheme, but it could do more complex operations. Note particularly that in this model P does not have access to the destination of the transaction.

However, this "pure function" approach is not enough. This is because this pure function-based approach is not powerful enough to implement many kinds of layer 2 protocols that people actually want to implement. It can do channels (and channel-based systems like the Lightning Network), but it cannot implement other scaling techniques with stronger properties, it cannot be used to bootstrap systems that do have more complicated notions of state, and so forth.

To give a simple example of what the pure function paradigm cannot do, consider a savings account with the following feature: there is a cryptographic key k which can initiate a withdrawal, and if a withdrawal is initiated, within the next 24 hours that same key k can cancel the withdrawal. If a withdrawal remains uncancelled within 24 hours, then anyone can "poke" the account to finalize that withdrawal. The goal is that if the key is stolen, the account holder can prevent the thief from withdrawing the funds. The thief could of course prevent the legitimate owner from getting the funds, but the attack would not be profitable for the thief and so they would probably not bother with it (see [the original paper](#) (<http://hackingdistributed.com/2016/02/26/how-to-implement-secure-bitcoin-vaults/>), for an explanation of this technique).

Unfortunately this technique cannot be implemented with just pure functions. The problem is this: there needs to be some way to move coins from a "normal" state to an "awaiting withdrawal" state. But the program P does not have access to the destination! Hence, any transaction that could authorize moving the coins to an awaiting withdrawal state could also authorize just stealing those coins immediately; P can't tell the difference. The ability to change the state of coins, without completely setting them free, is important to many kinds of applications, including layer 2 protocols. Plasma itself fits into this "authorize, finalize, cancel" paradigm: an exit from Plasma must be approved, then there is a 7 day challenge period, and within that challenge period the exit could be cancelled if the right evidence is provided. Rollup also needs this property: coins inside a rollup must be controlled by a program that keeps track of a state root R , and changes from R to R' if some verifier $P(R, R', \text{data})$ returns TRUE - but it only changes the state to R' in that case, it does not set the coins free.

This ability to authorize state changes without completely setting all coins in an account free, is what I mean by "rich statefulness". It can be implemented in many ways, some UTXO-based, but without it a blockchain is not powerful enough to implement most layer 2 protocols, without including trust assumptions (eg. a set of functionaries who are collectively trusted to execute those richly-stateful programs).

Note: yes, I know that if P has access to h then you can just include the destination address as part of S and check it against h , and restrict state changes that way. But it is possible to have a programming language that is too resource-limited or otherwise restricted to actually do this; and surprisingly this often actually is the case in blockchain scripting languages.

Sufficient data scalability and low latency

It turns out that plasma and channels, and other layer 2 protocols that are fully off-chain have some fundamental weaknesses that prevent them from fully replicating the capabilities of layer 1. I go into this in detail [here](https://vitalik.ca/general/2019/08/28/hybrid_layer_2.html) (https://vitalik.ca/general/2019/08/28/hybrid_layer_2.html); the summary is that these protocols need to have a way of adjudicating situations where some parties maliciously fail to provide data that they promised to provide, and because data publication is not globally verifiable (you don't know when data was published unless you already downloaded it yourself) these adjudication games are not game-theoretically stable. Channels and Plasma cleverly get around this instability by adding additional assumptions, particularly

assuming that for every piece of state, there is a single actor that is interested in that state not being incorrectly modified (usually because it represents coins that they own) and so can be trusted to fight on its behalf. However, this is far from general-purpose; systems like [Uniswap](http://uniswap.exchange) (<http://uniswap.exchange>), for example, include a large "central" contract that is not owned by anyone, and so they cannot effectively be protected by this paradigm.

There is one way to get around this, which is layer 2 protocols that publish very small amounts of data on-chain, but do computation entirely off-chain. If data is guaranteed to be available, then computation being done off-chain is okay, because games for adjudicating who did computation correctly and who did it incorrectly are game-theoretically stable (or could be replaced entirely by [SNARKs](https://vitalik.ca/general/2017/02/01/zk_snarks.html) (https://vitalik.ca/general/2017/02/01/zk_snarks.html)) or [STARKs](https://vitalik.ca/general/2017/11/09/starks_part_1.html) (https://vitalik.ca/general/2017/11/09/starks_part_1.html)). This is the logic behind [ZK rollup](https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477) (<https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477>) and [optimistic rollup](https://medium.com/plasma-group/ethereum-smart-contracts-in-l2-optimistic-rollup-2c1cef2ec537) (<https://medium.com/plasma-group/ethereum-smart-contracts-in-l2-optimistic-rollup-2c1cef2ec537>). If a blockchain allows for the publication and guarantees the availability of a reasonably large amount of data, even if its capacity for *computation* remains very limited, then the blockchain can support these layer-2 protocols and achieve a high level of scalability and functionality.

Just how much data does the blockchain need to be able to process and guarantee? Well, it depends on what TPS you want. With a rollup, you can compress most activity to ~10-20 bytes per transaction, so 1 kB/sec gives you 50-100 TPS, 1 MB/sec gives you 50,000-100,000 TPS, and so forth. Fortunately, internet bandwidth [continues to grow quickly](http://www.circleid.com/posts/20191119_nielsens_law_of_internet_bandwidth/) (http://www.circleid.com/posts/20191119_nielsens_law_of_internet_bandwidth/), and does not seem to be slowing down the way Moore's law for computation is, so increasing scaling for data without increasing computational load is quite a viable path for blockchains to take!

Note also that it is not just data capacity that matters, it is also data latency (ie. having low block times). Layer 2 protocols like rollup (or for that matter Plasma) only give any guarantees of security when the data actually is published to chain; hence, the time it takes for data to be reliably included (ideally "finalized") on chain is the time that it takes between when Alice sends Bob a payment and Bob can be confident that this payment will be included. The block time of the base layer sets the latency for anything whose confirmation depends on things being included in the base layer. This could be worked around with on-chain security deposits, aka "bonds", at the cost of high capital inefficiency, but such an approach is inherently imperfect because a malicious actor could trick an unlimited number of different people by sacrificing one deposit.

Conclusions

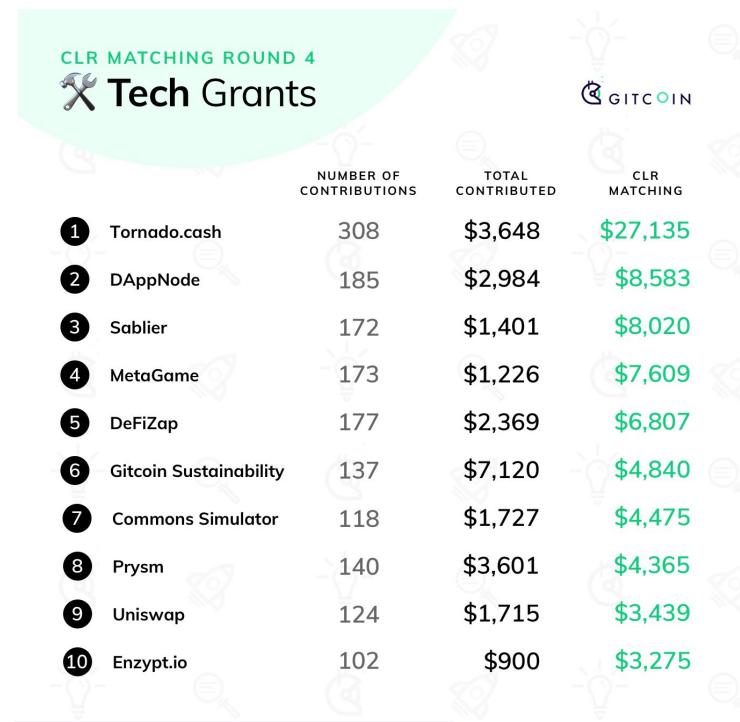
"Keep layer 1 simple, make up for it on layer 2" is NOT a universal answer to blockchain scalability and functionality problems, because it fails to take into account that layer 1 blockchains themselves must have a sufficient level of scalability and functionality for this "building on top" to actually be possible (unless your so-called "layer 2 protocols" are just trusted intermediaries). However, it is true that beyond a certain point, any layer 1 functionality *can* be replicated on layer 2, and in many cases it's a good idea to do this to improve upgradeability. Hence, we need [layer 1 development in parallel with layer 2 development in the short term, and more focus on layer 2 in the long term](https://vitalik.ca/general/2018/08/26/layer_1.html) (https://vitalik.ca/general/2018/08/26/layer_1.html).

Review of Gitcoin Quadratic Funding Round 4

2020 Jan 28

[See all posts \(/\)](#)

Round 4 of Gitcoin Grants quadratic funding has just completed, and here are the results:



		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Tornado.cash	308	\$3,648	\$27,135
2	DAppNode	185	\$2,984	\$8,583
3	Sablier	172	\$1,401	\$8,020
4	MetaGame	173	\$1,226	\$7,609
5	DeFiZap	177	\$2,369	\$6,807
6	Gitcoin Sustainability	137	\$7,120	\$4,840
7	Commons Simulator	118	\$1,727	\$4,475
8	Prysm	140	\$3,601	\$4,365
9	Uniswap	124	\$1,715	\$3,439
10	Enzypt.io	102	\$900	\$3,275



		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	Week in Ethereum News	140	\$3,191	\$13,536
2	@antiprosynth	134	\$2,420	\$11,393
3	EthHub	139	\$2,150	\$11,365
4	Bankless (Scholarships)	104	\$2,093	\$5,733
5	David Hoffman	101	\$1,399	\$5,414
6	Wizards of DApps	86	\$2,088	\$5,258
7	Cryptorado	75	\$1,475	\$4,682
8	Ethereum Magicians	69	\$633	\$3,745
9	Zero Knowledge Podcast	60	\$503	\$2,998
10	DeFi by Chris Blec	76	\$2,975	\$2,876

The main distinction between round 3 and round 4 was that while round 3 had only one category, with mostly tech projects and a few outliers such as EthHub, in round 4 there were two separate categories, one with a \$125,000 matching pool for tech projects, and the other with a \$75,000 matching pool for "media" projects. Media could include documentation, translation, community activities, news reporting, theoretically pretty much anything in that category. And while the tech section went about largely without incident, in the new media section **the results proved to be much more interesting than I could have possibly imagined, shedding a new light on deep questions in institutional design and political science.**

Tech: quadratic funding worked great as usual

In the tech section, the main changes that we see compared to round 3 are (i) the rise of [Tornado Cash](https://tornado.cash/) (<https://tornado.cash/>) and (ii) the decline in importance of eth2 clients and the rise of "utility applications" of various forms. Tornado Cash is a trustless smart contract-based Ethereum mixer. It became popular quickly in recent months, as the Ethereum community was swept by worries about the blockchain's current [low levels of privacy](https://www.cryptopolitan.com/jeffrey-wilcke-sold-92k-eth/) (<https://www.cryptopolitan.com/jeffrey-wilcke-sold-92k-eth/>) and wanted solutions. Tornado Cash amassed an incredible \$31,200. If they continue receiving such an amount every two months then this would allow them to pay two people \$7,800 per month each - meaning that the hoped-for milestone of seeing the first "quadratic freelancer" may have already been reached! The other major winners included tools like [Dappnode](https://dappnode.io/) (<https://dappnode.io/>), a software package to help people run nodes, [Sablier](https://www.sablier.finance/) (<https://www.sablier.finance/>), a payment streaming service, and [DefiZap](https://gitcoin.co/grants/235/defizap) (<https://gitcoin.co/grants/235/defizap>), which makes DeFi services easy to use. The [Gitcoin Sustainability Fund](https://gitcoin.co/grants/86/gitcoin-sustainability-fund) (<https://gitcoin.co/grants/86/gitcoin-sustainability-fund>) got over \$13,000, conclusively resolving my complaint from [last round](https://vitalik.ca/general/2019/10/24/gitcoin.html) (<https://vitalik.ca/general/2019/10/24/gitcoin.html>) that they were under-supported. All in all, valuable grants for valuable projects that provide services that the community genuinely needs.

We can see one major shift this round compared to the previous rounds. Whereas in previous rounds, the grants went largely to projects like eth2 clients that were already well-supported, this time the largest grants shifted toward having a different focus from the grants given by the Ethereum Foundation. The EF has not given grants to tornado.cash, and generally limits its grants to application-specific tools, Uniswap being a notable exception. The Gitcoin Grants quadratic fund, on the other hand, is supporting DefiZap, Sablier, and many other tools that are valuable to the community. This is arguably a positive development, as it allows Gitcoin Grants and the Ethereum Foundation to complement each other rather than focusing on the same things.

The one proposed change to the quadratic funding implementation for tech that I would favor is a user interface change, that makes it easier for users to commit funds for multiple rounds. This would increase the stability of contributions, thereby increasing the stability of projects' income - very important if we want "quadratic freelancer" to actually be a viable job category!

Media: The First Quadratic Twitter Freelancer

Now, we get to the new media section. In the first few days of the round, the leading recipient of the grants was "@antiprosynth Twitter account activity": an Ethereum community member who is [very active on twitter](https://twitter.com/antiprosynth) (<https://twitter.com/antiprosynth>) promoting Ethereum and refuting misinformation from Bitcoin maximalists, asking for help from the Gitcoin QF crowd to.... fund his tweeting activities. At its peak, the projected matching going to @antiprosynth exceeded \$20,000. This naturally proved to be controversial, with many criticizing this move and questioning whether or not a Twitter account is a legitimate public good:

**Chris Hitchcott**

@hitchcott

Replying to @gitcoin and @sassalox

I massively respect what you've done with @gitcoin

But it seems like such a mistake to let twitter influencers raise (and get quadratic funding from EF) on your platform.

What have they got to do with git?

Where do you draw the line at what's acceptable to be funded?

10:30 AM · Jan 13, 2020 · Twitter Web App

5 Likes



(<https://twitter.com/hitchcott/status/1216548157238079490>).

On the surface, it does indeed seem like someone getting paid \$20,000 for operating a Twitter account is ridiculous. But it's worth digging in and questioning exactly *what*, if anything, is actually wrong with this outcome. After all, maybe this is what effective marketing in 2020 actually looks like, and it's our expectations that need to adapt.

There are two main objections that I heard, and both lead to interesting criticisms of quadratic funding in its current implementation. First, there was criticism of **overpayment**. Twittering is a fairly "trivial" activity; it does not require *that* much work, lots of people do it for free, and it doesn't provide nearly as much long-term value as something more substantive like EthHub (<https://ethhub.io/>) or the Zero Knowledge Podcast (<https://gitcoin.co/grants/329/zero-knowledge-podcast-2>). Hence, it feels wrong to pay a full-time salary for it.

antiprosynthesis.eth Retweeted

antiprosynthesis.eth 69K Tweets

#ethereum #optimisticrollups @ethereum @jadler0



What is Fuel?
Fuel uses optimistic rollups to scale Ethereum. Learn more at <https://fuel.sh/> Fund them at ...
youtube.com

2 4 12

antiprosynthesis.eth Retweeted

Figo.ismoney.eth @FigoFinozeros · 4h
"With erasure code data availability proofs, TPS could be 10s of thousands to millions TPS"

According to Fuellabs possible with current #Ethereum 1.x and Optimistic Rollups (and mentioned proofs).

Thint @Thintandgrow · 6h
What is @fuellabs_?

#ethereum #optimisticrollups @ethereum @jadler0
youtube.com/watch?v=IUUHE...

4 7

antiprosynthesis.eth Retweeted

phil.eth @CryptoPhil · 4h
Went through some pain and compiled @prylabs \$ETH 2 client prysm on my raspberry pi 4. I am now on the initial sync. Estimated remaining time 70h with 0.5 blocks/s processing. Oo

3 2 10

[Show this thread](#)

antiprosynthesis.eth @antiprosynth · 9h
Replying to @antiprosynth and @JosephTodaro_
So, yes, Bitcoin was a massive breakthrough. But we can only hope that it won't be the end point.

antiprosynthesis.eth @antiprosynth · Sep 8, 2019
#Bitcoin \$BTC is one of the most important breakthroughs of this century, much like the steam machine, vacuum tube and telephone before it.

But today we're driving electric cars, and transistors power the general purpose, programmable computers we call phones.

#Ethereum \$ETH

4

antiprosynthesis.eth @antiprosynth · 9h
Replying to @antiprosynth and @JosephTodaro_
So what's left is digital gold, but with a fixed supply schedule. Which one could argue is worthy on its own.

Except then you learn that this supply schedule is not sustainable and will inevitably have to be lifted through social consensus to be able to secure the network.

1

antiprosynthesis.eth @antiprosynth · 9h
Replying to @JosephTodaro_
That's because it's not censorship resistant for anyone but a small set of wealthy, centralized entities.

This is currently masked by the fact that nobody uses it, but it is the reality without base layer scalability.

2 11

Examples of @antiprosynth's recent tweets

If we accept the metaphor of quadratic funding as being like a market for public goods (<https://vitalik.ca/general/2019/12/07/quadratic.html>), then one could simply extend the metaphor, and reply to the concern with the usual free-market argument. People voluntarily paid their own money to support @antiprosynth's twitter activity, and that itself signals that it's valuable. Why should we trust you with your mere words and protestations over a costly signal of real money on the table from dozens of people?

The most plausible answer is actually quite similar to one that you often hear in discussions about financial markets: markets can give skewed results when you can express an opinion *in favor of* something but cannot express an opinion *against* it. When short selling is not possible, financial markets are often much more inefficient (<https://ethresear.ch/t/token-sales-and-shorting/376>), because instead of reflecting the *average* opinion on an asset's true value, a market may instead reflect the inflated expectations of an asset's few rabid supporters. In this version of quadratic funding, **there too is an asymmetry, as you can donate in support of a project but you cannot donate to oppose it**. Might this be the root of the problem?

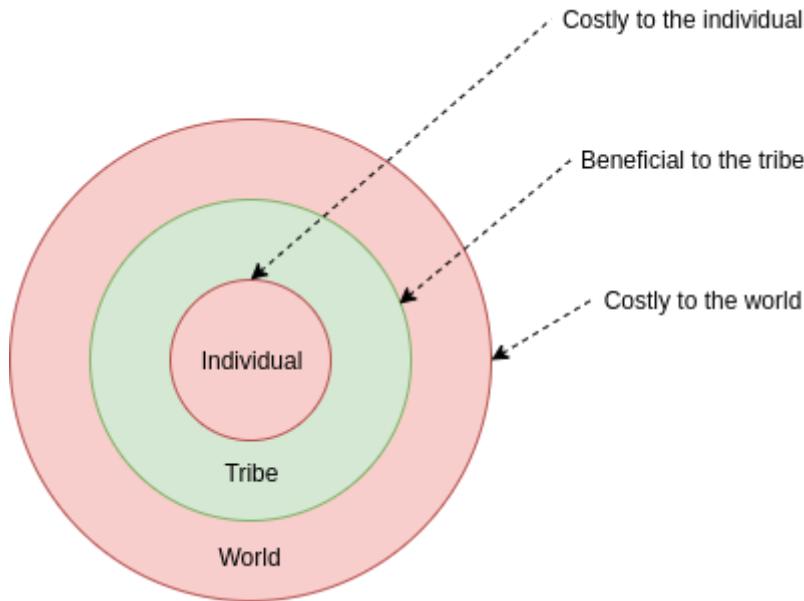
One can go further and ask, why might overpayment happen to this particular project, and not others? I have heard a common answer: twitter accounts *already have a high exposure*. A client development team like Nethermind (<https://nethermind.io/>) does not gain much publicity through their work directly, so they need to separately market themselves, whereas a twitter account's "work" is self-marketing by its very nature. Furthermore, the most prominent twitterers get quadratically more matching out of their exposure, amplifying their outsized advantage further - a problem I alluded to in my review of round 3 (<https://vitalik.ca/general/2019/10/24/gitcoin.html>).

Interestingly, in the case of vanilla quadratic voting there was an argument (<http://www.econ.msu.edu/seminars/docs/QuadMultAltshort19.pdf>) made by Glen Weyl for why economies-of-scale effects of traditional voting, such as Duverger's law (https://en.wikipedia.org/wiki/Duverger%27s_law), don't apply to quadratic voting: a project becoming more prominent increases the incentive to give it both positive and negative votes, so on net the effects cancel out. But notice once again, that **this argument relies on negative votes being a possibility**.

Good for the tribe, but is it good for the world?

The particular story of @antipro synth had what is in my opinion a happy ending: over the next ten days, more contributions came in to other candidates, and @antipro synth's match reduced to \$11,316, still a respectably high amount but on par with EthHub and below Week in Ethereum. However, even a quadratic matching grant of this size still raises to the next criticism: **is twittering a public good or public bad anyway?**

Traditionally, public goods of the type that Gitcoin Grants quadratic funding is trying to support were selected and funded by governments. The motivation of @antipro synth's tweets is "aggregating Ethereum-related news, fighting information asymmetry and fine-tuning/signaling a consistent narrative for Ethereum (and ETH)": essentially, fighting the good fight against anti-Ethereum misinformation by bitcoin maximalists (<https://blog.ethereum.org/2014/11/20/bitcoin-maximalism-currency-platform-network-effects/>). And, lo and behold, governments too have a rich history of sponsoring social media participants (https://en.wikipedia.org/wiki/State-sponsored_Internet_propaganda) to argue on their behalf. And it seems likely that most of these governments see themselves as "fighting the good fight against anti-[X] misinformation by [Y] {extremists, imperialists, totalitarians}", just as the Ethereum community feels a need to fight the good fight against maximalist trolls. From the inside view of each individual country (and in our case the Ethereum community) organized social media participation seems to be a clear public good (ignoring the possibility of blowback effects, which are real and important). But from the outside view of the entire world, it can be viewed as a zero-sum game.



This is actually a common pattern to see in politics, and indeed there are many instances of larger-scale coordination that are precisely intended to undermine smaller-scale coordination that is seen as "good for the tribe but bad for the world": antitrust law, free trade agreements, state-level pre-emption of local zoning codes, anti-militarization agreements... the list goes on. A broad environment where public subsidies are generally viewed suspiciously also does quite a good job of limiting many kinds of malign local coordination. But as public goods become more important, and we discover better and better ways for communities to coordinate on producing them, that strategy's efficacy becomes more limited, and properly grappling with these discrepancies between what is good for the tribe and what is good for the world becomes more important.

That said, internet marketing and debate is not a zero-sum game, and there are plenty of ways to engage in internet marketing and debate that are good for the world. Internet debate in general serves to help the public learn what things are true, what things are not true, what causes to support, and what causes to oppose. Some tactics are clearly not truth-favoring, but other tactics are quite truth-favoring. Some tactics are clearly offensive, but others are defensive. And in the Ethereum community, there is [widespread](https://twitter.com/safetyth1rd/status/1221126919019749376) ([sentiment](https://twitter.com/wmougayar/status/1192130181445640192) ([that there is not enough resources going into marketing of some kind](https://twitter.com/ErikNummer8/status/1220420334962335744), and I personally agree with this sentiment).

What kind of marketing is positive-sum (good for tribe and good for world) and what kind of marketing is zero-sum (good for tribe but bad for world) is another question, and one that's worth the community debating. I naturally hope that the Ethereum community continues to value maintaining a moral high ground. Regarding the case of @antiprosynth himself, I cannot find any tactics that I would classify as bad-for-world, especially when compared to outright misinformation ("it's impossible to run a full node") that we often see used against Ethereum - but I am pro-Ethereum and hence biased, hence the need to be careful.

Universal mechanisms, particular goals

The story has another plot twist, which reveals yet another feature (or bug?) or quadratic funding. Quadratic funding was originally described (<https://arxiv.org/abs/1809.06421>) as "Formal Rules for a Society Neutral among Communities", the intention being to use it at a very large, potentially even global, scale. Anyone can participate as a project or as a participant, and projects that support public goods that are good for *any*

"public" would be supported. In the case of Gitcoin Grants, however, the matching funds are coming from Ethereum organizations, and so there is an expectation that the system is there to support Ethereum projects. But there is nothing in the rules of quadratic funding that privileges Ethereum projects and prevents, say, Ethereum Classic projects from seeking funding using the same platform! And, of course, this is exactly what happened:



ACTIVITY	DESCRIPTION	TRANSACTIONS (7)	STATS	UPDATES (0)
CONTRIBUTORS (6)				

I've been a supporter of the original Ethereum project for a long time and always preach the gospels of immutability on Twitter and other social media to the unwashed masses whenever I can.

However, protecting the original vision of Ethereum for so long can take its toll on me so I hope with this grant I can be more incentivized to talk about the original Ethereum more and more. We want to keep things classic in the Twitter space.

I also promote the fundamentals of DePi or Decentralized Pizza which is very important to ensure censorship-resistant pizza is accessible to all!

Yazanator Twitter Account Activity

 <https://twitter.com/Yazanator>



[0x31cA6CA7f7A3298Bc6c5103Aa45847f34e382a1C](https://twitter.com/Yazanator/status/131cA6CA7f7A3298Bc6c5103Aa45847f34e382a1C)

 Grant accepts DAI

[Fund this Grant](#)

CLR MATCH ROUND 4

TOTAL FUNDED

24 DAI

ESTIMATED

38 DAI

(<https://gitcoin.co/grants/324/yazanator-twitter-account-activity>).

So now the result is, \$24 of funding from Ethereum organizations will be going toward supporting an Ethereum Classic promoter's twitter activity. To give people outside of the crypto space a feeling for what this is like, imagine the USA holding a quadratic funding raise, using government funding to match donations, and the result is that some of the funding goes to someone explicitly planning to use the money to talk on Twitter about how great Russia is (or vice versa). The matching funds are coming from Ethereum sources, and there's an implied expectation that the funds should support Ethereum, but nothing actually prevents, or even discourages, non-Ethereum projects from organizing to get a share of the matched funds on the platform!

Solutions

There are two solutions to these problems. One is to modify the quadratic funding mechanism to support negative votes in addition to positive votes. The mathematical theory behind quadratic voting already implies that it is the "right thing" to do to allow such a possibility (every positive number has a negative square root as

well as a positive square root). On the other hand, there are social concerns that allowing for negative voting would cause more animosity and lead to other kinds of harms. After all, mob mentality is at its worst when it is against something rather than for something. Hence, it's my view that it's not certain that allowing negative contributions will work out well, but there is enough evidence that it might that it is definitely worth trying out in a future round.

The second solution is to use two separate mechanisms for identifying relative goodness of good projects and for screening out bad projects. For example, one could use a challenge mechanism followed by a majority ETH coin vote, or even at first just a centralized appointed board, to screen out bad projects, and then use quadratic funding as before to choose between good projects. This is less mathematically elegant, but it would solve the problem, and it would at the same time provide an opportunity to mix in a separate mechanism to ensure that chosen projects benefit Ethereum specifically.

But even if we adopt the first solution, defining boundaries for the quadratic funding itself may also be a good idea. There is intellectual precedent for this. In Elinor Ostrom's [eight principles for governing the commons](https://www.onthecommons.org/magazine/elinor-ostroms-8-principles-managing-commons) (<https://www.onthecommons.org/magazine/elinor-ostroms-8-principles-managing-commons>), defining clear boundaries about who has the right to access the commons is the first one. Without clear boundaries, Ostrom writes, "local appropriators face the risk that any benefits they produce by their efforts will be reaped by others who have not contributed to those efforts." In the case of Gitcoin Grants quadratic funding, one possibility would be to set the maximum matching coefficient for any pair of users to be proportional to the geometric average of their ETH holdings, using that as a proxy for measuring membership in the Ethereum community (note that this avoids being plutocratic because 1000 users with 1 ETH each would have a maximum matching of $\approx k * 500,000$ ETH, whereas 2 users with 500 ETH each would only have a maximum matching of $k * 1,000$ ETH).

Collusion

Another issue that came to the forefront this round was the issue of collusion. The math behind quadratic funding, which compensates for tragedies of the commons by magnifying individual contributions based on the total number and size of other contributions to the same project, only works if there is an actual tragedy of the commons limiting natural donations to the project. If there is a "quid pro quo", where people get something individually in exchange for their contributions, the mechanism can easily over-compensate. The long-run solution to this is something like [MACI](https://github.com/barryWhiteHat/macj) (<https://github.com/barryWhiteHat/macj>), a cryptographic system that ensures that contributors have no way to prove their contributions to third parties, so any such collusion would have to be done by honor system. In the short run, however, the rules and enforcement has not yet been set, and this has led to vigorous debate about what kinds of quid pro quo are legitimate:



Richard Burton
@ricburton

Replying to @travis_willmann and @AlokVasudev

I only share that stuff with my private paid group 😂

It's all on [@gitcoin](#)

7:36 AM · Jan 16, 2020 · Twitter for iPhone

3 Likes



antiprosynthesis.eth @antiprosynth · Jan 17

Replying to @ricburton @travis_willmann and 2 others

Dude, Gitcoin grants are meant to direct funding to public goods. A private paid group is quite literally the opposite of that.



13



(<https://twitter.com/ricburton/status/1217591449908404225>).

[Update 2020.01.29: the above was ultimately a result of a miscommunication from Gitcoin (<https://twitter.com/owocki/status/1222419312851353602?s=21>); a member of the Gitcoin team had okayed Richard Burton's proposal to give rewards to donors without realizing the implications. So Richard himself is blameless here; though the broader point that we underestimated the need for explicit guidance about what kinds of quid pro quo are acceptable is very much real.]

Currently, the position is that quid pro quo are disallowed (<https://twitter.com/owocki/status/1217993123311177728>), though there is a more nuanced feeling that informal social quid pro quo ("thank yous" of different forms) are okay, whereas formal and especially monetary or product rewards are a no-no. This seems like a reasonable approach, though it does put Gitcoin further into the uncomfortable position of being a central arbiter, compromising credible neutrality (<https://nakamoto.com/credible-neutrality/>) somewhat. One positive byproduct of this whole discussion is that it has led to much more awareness in the Ethereum community of what actually is a public good (as opposed to a "private good" or a "club good"), and more generally brought public goods much further into the public discourse.

Conclusions

Whereas round 3 was the first round with enough participants to have any kind of interesting effects, round 4 felt like a true "coming-out party" for the cause of decentralized public goods funding. The round attracted a large amount of attention from the community, and even from outside actors such as the Bitcoin community. It is part of a broader trend in the last few months where public goods funding has become a dominant part (<https://medium.com/ethereum-optimism/optimism-cd9bea61a3ee>) of the crypto community discourse. Along with this, we have also seen much more discussion (<https://ethresear.ch/t/mev-auction-auctioning-transaction-ordering->

[rights-as-a-solution-to-miner-extractable-value/6788](#) of [strategies](#)

(<https://twitter.com/MuteDialog/status/1207249423715229697>) about long-term sources of funding for quadratic matching pools of larger sizes.

Discussions about funding will be important going forward: donations from large Ethereum organizations are enough to sustain quadratic matching at its current scale, but not enough to allow it to grow much further, to the point where we can have hundreds of quadratic freelancers instead of about five. At those scales, sources of funding for Ethereum public goods must rely on network effect lockin to some extent, or else they will have little more staying power than individual donations, but there are strong reasons not to embed these funding sources too deeply into Ethereum (eg. into the protocol itself, a la [the recent BCH proposal](#) (<https://twitter.com/VitalikButerin/status/1220141595846033410>)), to avoid risking the protocol's neutrality.

Approaches based on capturing transaction fees at layer 2 are surprisingly viable: currently, there are about \$50,000-100,000 per day (~\$18-35m per year) of transaction fees happening on Ethereum, roughly equal to the entire budget of the Ethereum Foundation. And there is evidence that [miner-extractable value](#) (<https://arxiv.org/abs/1904.05234>) is even higher. There are all discussions that we need to have, and challenges that we need to address, if we want the Ethereum community to be a leader in implementing decentralized, credibly neutral and market-based solutions to public goods funding challenges.

A Quick Garbled Circuits Primer

2020 Mar 21

[See all posts \(/\)](#)

Special thanks to Dankrad Feist for review

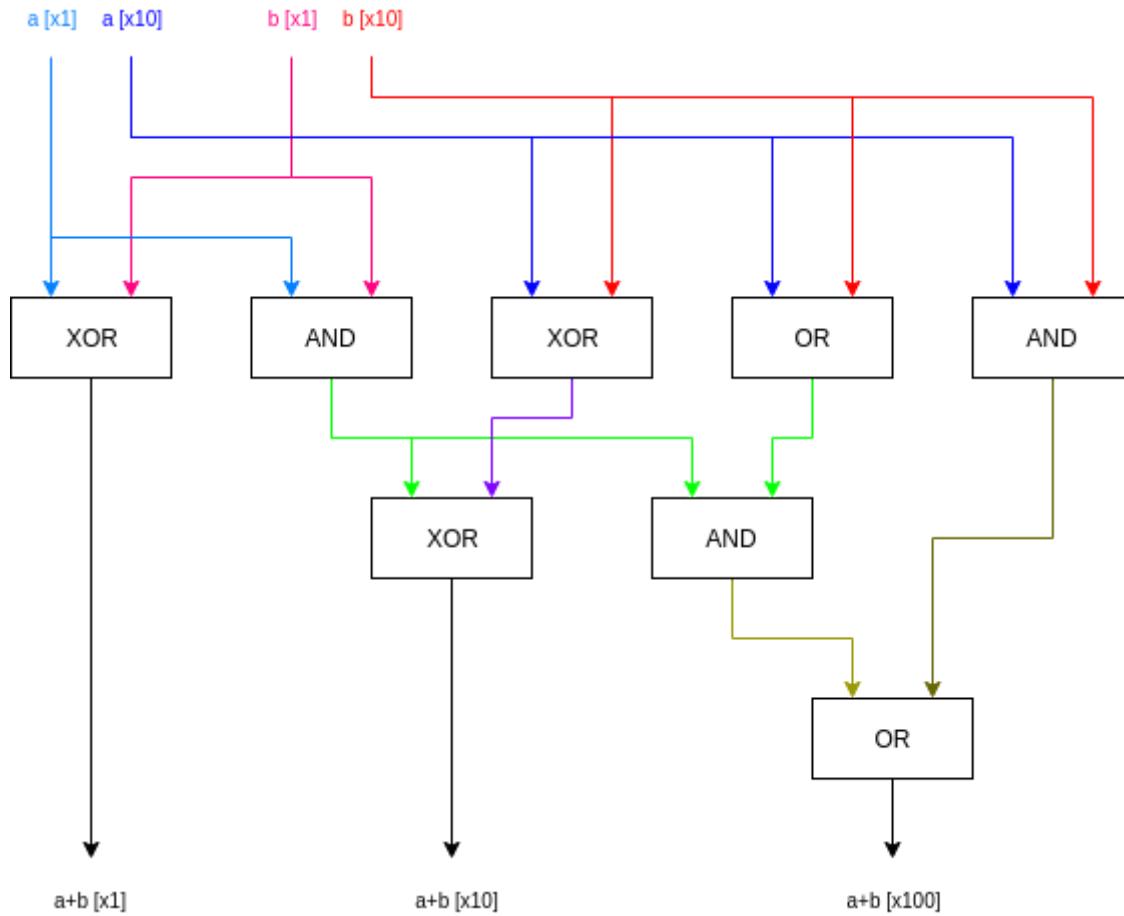
Garbled circuits (https://en.wikipedia.org/wiki/Garbled_circuit) are a quite old, and surprisingly simple, cryptographic primitive; they are quite possibly the simplest form of general-purpose "multi-party computation" (MPC) to wrap your head around.

Here is the usual setup for the scheme:

- Suppose that there are two parties, Alice and Bob, who want to compute some function `f(alice_inputs, bob_inputs)`, which takes inputs from both parties. Alice and Bob want to both learn the result of computing `f`, but Alice does not want Bob to learn her inputs, and Bob does not want Alice to learn his inputs. Ideally, they would both learn nothing except for just the output of `f`.
- Alice performs a special procedure ("garbling") to encrypt a circuit (meaning, a set of AND, OR... gates) which evaluates the function `f`. She passes along inputs, also encrypted in a way that's compatible with the encrypted circuit, to Bob.
- Bob uses a technique called "1-of-2 oblivious transfer" to learn the encrypted form of his own inputs, without letting Alice know which inputs he obtained.
- Bob runs the encrypted circuit on the encrypted data and gets the answer, and passes it along to Alice.

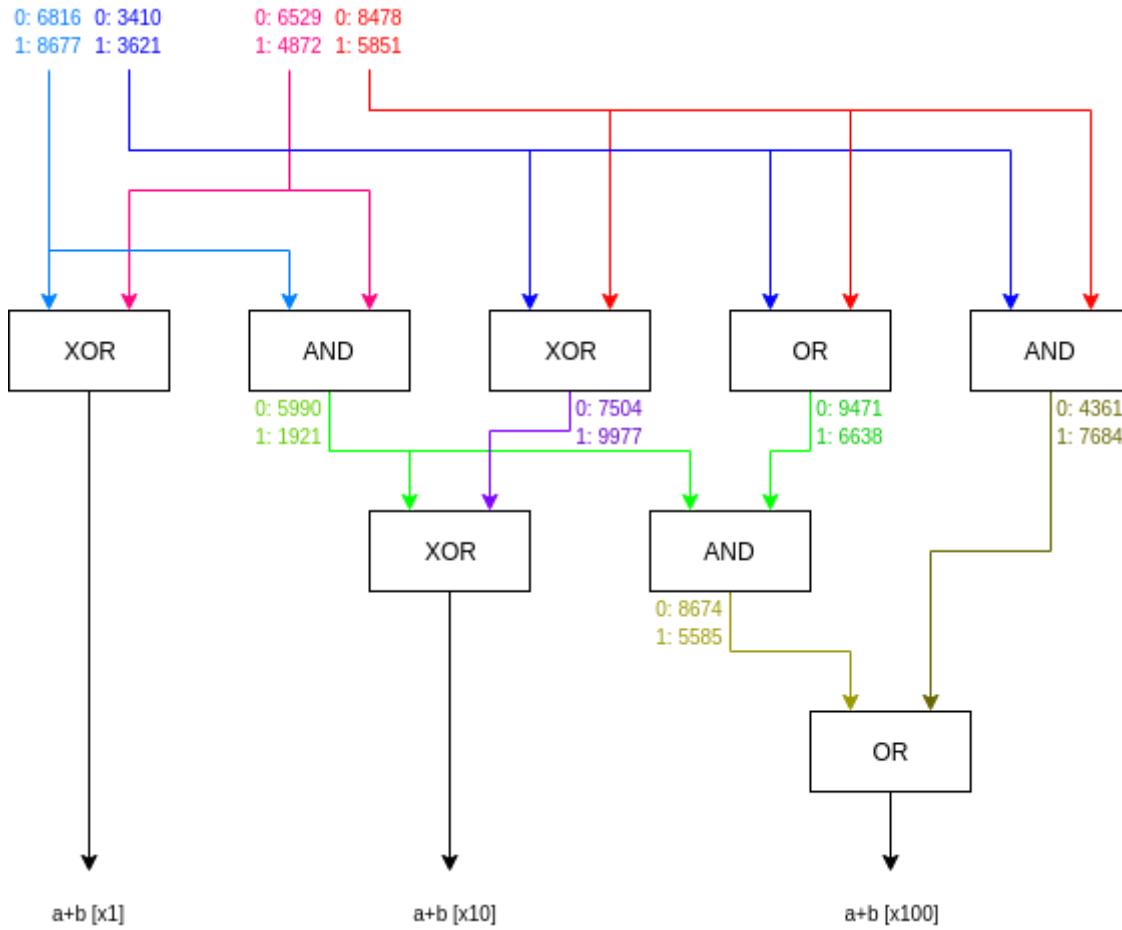
Extra cryptographic wrappings can be used to protect the scheme against Alice and Bob sending wrong info and giving each other an incorrect answer; we won't go into those here for simplicity, though it suffices to say "wrap a ZK-SNARK around everything" is one (quite heavy duty and suboptimal!) solution that works fine.

So how does the basic scheme work? Let's start with a circuit:



This is one of the simplest examples of a not-completely-trivial circuit that actually does something: it's a two-bit adder. It takes as input two numbers in binary, each with two bits, and outputs the three-bit binary number that is the sum.

Now, let's encrypt the circuit. First, for every input, we randomly generate two "labels" (think: 256-bit numbers): one to represent that input being 0 and the other to represent that input being 1. Then we also do the same for every intermediate wire, not including the output wires. Note that this data is not part of the "garbling" that Alice sends to Bob; so far this is just setup.



Now, for every gate in the circuit, we do the following. For every combination of inputs, we include in the "garbling" that Alice provides to Bob the label of the output (or if the label of the output is a "final" output, the output directly) encrypted with a key generated by hashing the input labels that lead to that output together. For simplicity, our encryption algorithm can just be `enc(out, in1, in2) = out + hash(k, in1, in2)` where `k` is the index of the gate (is it the first gate in the circuit, the second, the third?). If you know the labels of both inputs, and you have the garbling, then you can learn the label of the corresponding output, because you can just compute the corresponding hash and subtract it out.

Here's the garbling of the first XOR gate:

Inputs	Output	Encoding of output
00	0	0 + hash(1, 6816, 6529)
01	1	1 + hash(1, 6816, 4872)
10	1	1 + hash(1, 8677, 6529)
11	0	0 + hash(1, 8677, 4872)

Notice that we are including the (encrypted forms of) 0 and 1 directly, because this XOR gate's outputs are directly final outputs of the program. Now, let's look at the leftmost AND gate:

Inputs	Output	Encoding of output
00	0	5990 + hash(2, 6816, 6529)
01	0	5990 + hash(2, 6816, 4872)
10	0	5990 + hash(2, 8677, 6529)
11	1	1921 + hash(2, 8677, 4872)

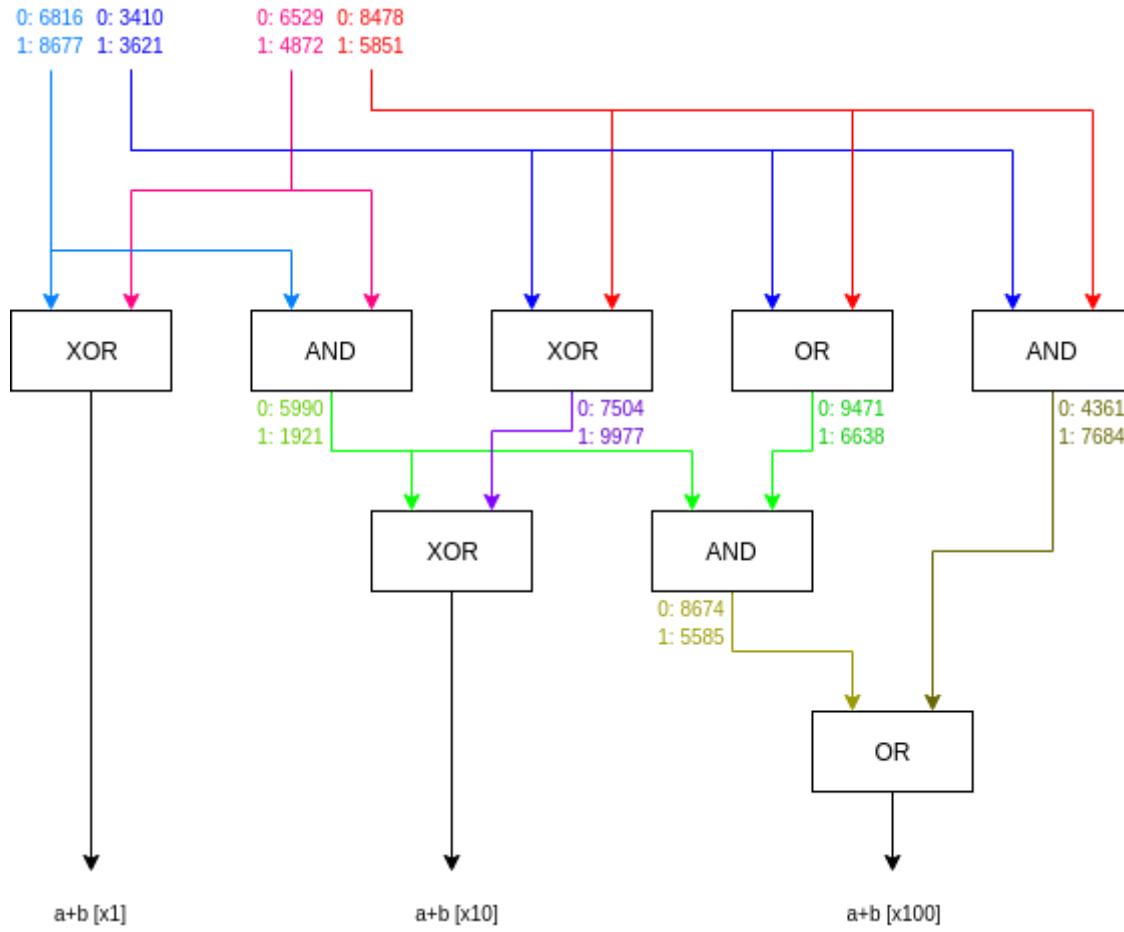
Here, the gate's outputs are just used as inputs to other gates, so we use labels instead of bits to hide these intermediate bits from the evaluator.

The "garbling" that Alice would provide to Bob is just everything in the third column for each gate, with the rows of each gate re-ordered (to avoid revealing whether a given row corresponds to a 0 or a 1 in any wire). To help Bob learn which value to decrypt for each gate, we'll use a particular order: for each gate, the first row becomes the row where both input labels are even, in the second row the second label is odd, in the third row the first label is odd, and in the fourth row both labels are odd (we deliberately chose labels earlier so that each gate would have an even label for one output and an odd label for the other). We garble every other gate in the circuit in the same way.

All in all, Alice sends to Bob four ~256 bit numbers for each gate in the circuit. It turns out that four is far from optimal; see [here](http://web.mit.edu/sonka89/www/papers/2017ygc.pdf) (<http://web.mit.edu/sonka89/www/papers/2017ygc.pdf>) for some optimizations on how to reduce this to three or even two numbers for an AND gate and zero (!!) for an XOR gate. Note that these optimizations do rely on some changes, eg. using XOR instead of addition and subtraction, though this should be done anyway for security.

When Bob receives the circuit, he asks Alice for the labels corresponding to her input, and he uses a protocol called "1-of-2 oblivious transfer" to ask Alice for the labels corresponding to his own input without revealing to Alice what his input is. He then goes through the gates in the circuit one by one, uncovering the output wires of each intermediate gate.

Suppose Alice's input is the two left wires and she gives (0, 1), and Bob's input is the two right wires and he gives (1, 1). Here's the circuit with labels again:



- At the start, Bob knows the labels 6816, 3621, 4872, 5851
- Bob evaluates the first gate. He knows 6816 and 4872, so he can extract the output value corresponding to (1, 6816, 4872) (see the table above) and extracts the first output bit, 1
- Bob evaluates the second gate. He knows 6816 and 4872, so he can extract the output value corresponding to (2, 6816, 4872) (see the table above) and extracts the label 5990
- Bob evaluates the third gate (XOR). He knows 3621 and 5851, and learns 7504
- Bob evaluates the fourth gate (OR). He knows 3621 and 5851, and learns 6638
- Bob evaluates the fifth gate (AND). He knows 3621 and 5851, and learns 7684
- Bob evaluates the sixth gate (XOR). He knows 5990 and 7504, and learns the second output bit, 0
- Bob evaluates the seventh gate (AND). He knows 5990 and 6638, and learns 8674
- Bob evaluates the eighth gate (OR). He knows 8674 and 7684, and learns the third output bit, 1

And so Bob learns the output: 101. And in binary $10 + 11$ actually equals 101 (the input and output bits are both given in smallest-to-greatest order in the circuit, which is why Alice's input 10 is represented as (0, 1) in the circuit), so it worked!

Note that addition is a fairly pointless use of garbled circuits, because Bob knowing 101 can just subtract out his own input and get $101 - 11 = 10$ (Alice's input), breaking privacy. However, in general garbled circuits can be used for computations that are not reversible, and so don't break privacy in this way (eg. one might imagine a computation where Alice's input and Bob's input are their answers to a personality quiz, and the output is a single bit that determines whether or not the algorithm thinks they are compatible; that one bit of information won't let Alice or Bob know anything about each other's individual quiz answers).

1 of 2 Oblivious Transfer

Now let us talk more about 1-of-2 oblivious transfer, this technique that Bob used to obtain the labels from Alice corresponding to his own input. The problem is this. Focusing on Bob's first input bit (the algorithm for the second input bit is the same), Alice has a label corresponding to 0 (6529), and a label corresponding to 1 (4872). Bob has his desired input bit: 1. Bob wants to learn the correct label (4872) without letting Alice know that his input bit is 1. The trivial solution (Alice just sends Bob both 6529 and 4872) doesn't work because Alice only wants to give up one of the two input labels; if Bob receives both input labels this could leak data that Alice doesn't want to give up.

Here is a [fairly simple protocol](https://crypto.stanford.edu/pbc/notes/crypto/ot.html) (<https://crypto.stanford.edu/pbc/notes/crypto/ot.html>) using elliptic curves:

1. Alice generates a random elliptic curve point, H .
2. Bob generates two points, P_1 and P_2 , with the requirement that $P_1 + P_2$ sums to H . Bob chooses either P_1 or P_2 to be $G * k$ (ie. a point that he knows the corresponding private key for). Note that the requirement that $P_1 + P_2 = H$ ensures that Bob has no way to generate P_1 and P_2 such that he knows the corresponding private key for. This is because if $P_1 = G * k_1$ and $P_2 = G * k_2$ where Bob knows both k_1 and k_2 , then $H = G * (k_1 + k_2)$, so that would imply Bob can extract the discrete logarithm (or "corresponding private key") for H , which would imply all of elliptic curve cryptography is broken.
3. Alice confirms $P_1 + P_2 = H$, and encrypts v_1 under P_1 and v_2 under P_2 using some standard public key encryption scheme (eg. [El-Gamal](#) (https://en.wikipedia.org/wiki/ElGamal_encryption)). Bob is only able to decrypt one of the two values, because he knows the private key corresponding to at most one of (P_1, P_2) , but Alice does not know which one.

This solves the problem; Bob learns one of the two wire labels (either 6529 or 4872), depending on what his input bit is, and Alice does not know which label Bob learned.

Applications

Garbled circuits are potentially useful for many more things than just 2-of-2 computation. For example, you can use them to make multi-party computations of arbitrary complexity with an arbitrary number of participants providing inputs, that can run in a constant number of rounds of interaction. Generating a garbled circuit is completely parallelizable; you don't need to finish garbling one gate before you can start garbling gates that depend on it. Hence, you can simply have a large multi-party computation with many participants compute a garbling of all gates of a circuit and publish the labels corresponding to their inputs. The labels themselves are random and so reveal nothing about the inputs, but anyone can then execute the published garbled circuit and learn the output "in the clear". See [here](#) (<https://eprint.iacr.org/2017/189.pdf>) for a recent example of an MPC protocol that uses garbling as an ingredient.

Multi-party computation is not the only context where this technique of splitting up a computation into a parallelizable part that operates on secret data followed by a sequential part that can be run in the clear is useful, and garbled circuits are not the only technique for accomplishing this. In general, the literature on [randomized encodings](#) (<https://eprint.iacr.org/2017/385.pdf>) includes many more sophisticated techniques. This branch of math is also useful in technologies such as [functional encryption](#) (https://en.wikipedia.org/wiki/Functional_encryption) and [obfuscation](#) (https://en.wikipedia.org/wiki/Indistinguishability_obfuscation).

Gitcoin Grants Round 5 Retrospective

2020 Apr 30

[See all posts \(/\)](#)

Special thanks to Kevin Owocki and Frank Chen for help and review

Round 5 of Gitcoin Grants has just finished, with \$250,000 of matching split between tech, media, and the new (non-Ethereum-centric) category of "public health". In general, it seems like the mechanism and the community are settling down into a regular rhythm. People know what it means to contribute, people know what to expect, and the results emerge in a relatively predictable pattern - even if which specific grants get the most funds is not so easy to predict.

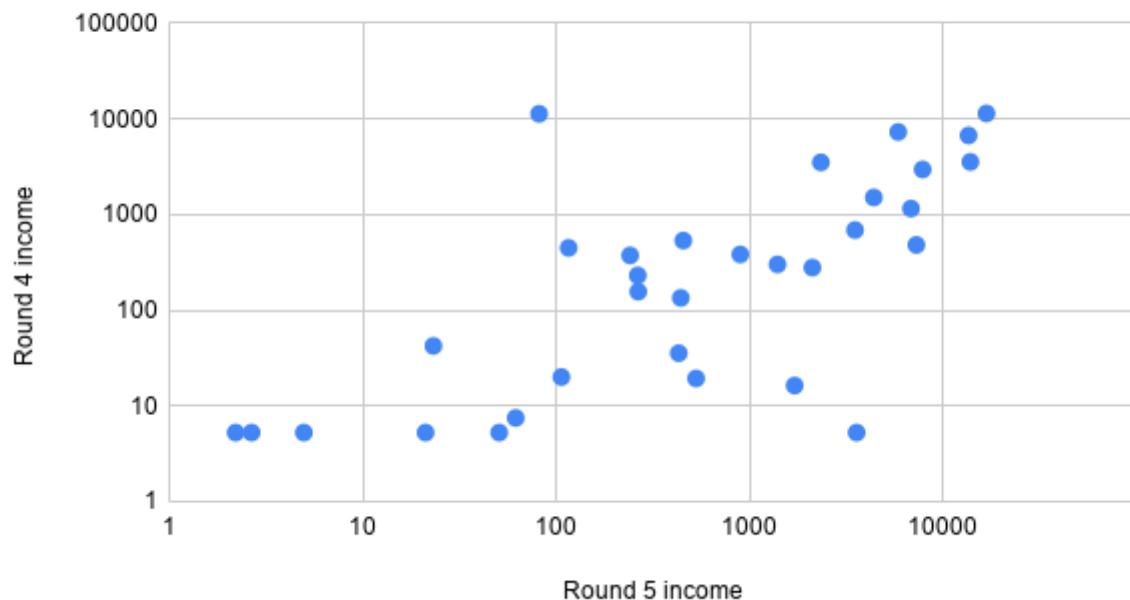
CLR MATCHING ROUND 5				CLR MATCHING ROUND 5					
Tech Grants		Media Grants							
	NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING		
1	White Hat Hacking	234	\$4,687	\$15,704	1	Week in Ethereum News	157	\$2,141	\$10,054
2	Arboreum	178	\$7,038	\$9,046	2	Chris Blec	143	\$1,390	\$5,716
3	1inch.exchange	180	\$50,712	\$7,893	3	EthHub	123	\$1,566	\$5,148
4	The Commons Stack	145	\$3,061	\$6,497	4	The Defiant	116	\$3,748	\$4,886
5	POAP	128	\$2,940	\$5,139	5	MetaCartel	99	\$610	\$3,232
6	Tornado.cash	142	\$1,178	\$3,496	6	Transaction Pending	41	\$4,358	\$2,547
7	DAppNode	107	\$1,993	\$3,352	7	Bankless (Translations)	81	\$1,096	\$2,167
8	Rotki	111	\$1,578	\$3,165	8	@antiprosynth	68	\$1,576	\$1,897
9	Gitcoin Grants Dev	116	\$2,688	\$3,087	9	DeFi Dad Tutorials	73	\$465	\$1,793
10	Prysm by Prysmatic Labs	119	\$1,939	\$3,055	10	Interspace.chat	54	\$1,007	\$1,755

Stability of income

So let's go straight into the analysis. One important property worth looking at is stability of income across rounds: do projects that do well in round N also tend to do well in round N+1? Stability of income is very important if we want to support an ecosystem of "quadratic freelancers": we want people to feel comfortable relying on their income knowing that it will not completely disappear the next round. On the other hand, it would be harmful if some recipients became completely entrenched, with no opportunity for new projects to come in and compete for the pot, so there is a need for a balance.

On the media side, we do see some balance between stability and dynamism:

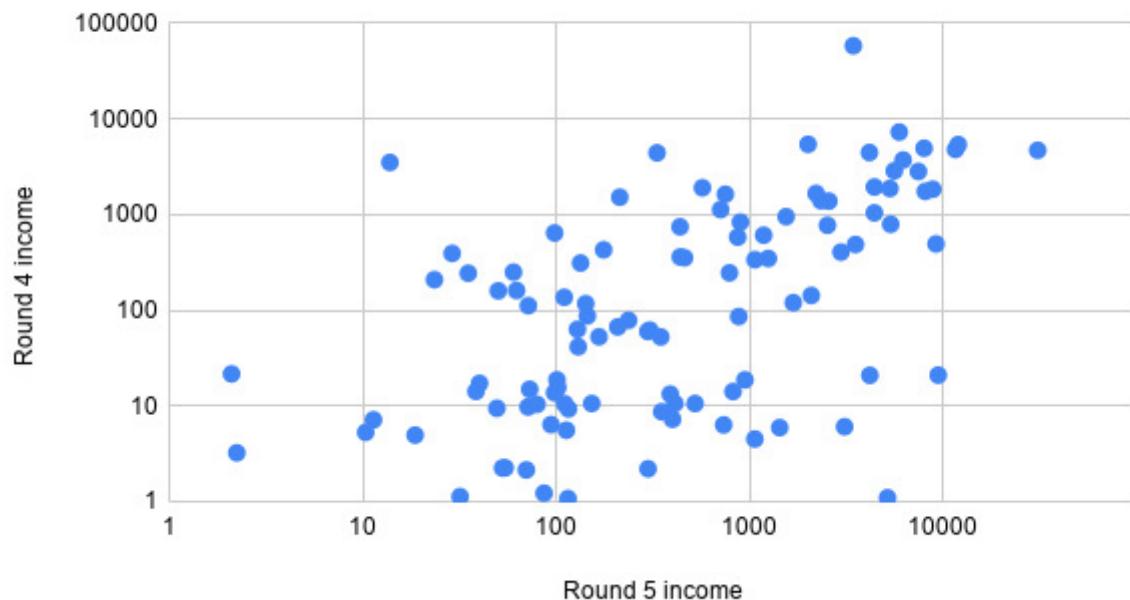
Round 4 vs Round 5 income (media)



Week in Ethereum had the highest total amount received in both [the previous round](https://vitalik.ca/general/2020/01/28/round4.html) (<https://vitalik.ca/general/2020/01/28/round4.html>) and the current round. EthHub and Bankless are also near the top in both the current round and the previous round. On the other hand, Antiprosynthesis, the (beloved? notorious? famous?) Twitter info-warrior, has decreased from \$13,813 to \$5,350, while [Chris Blec's YouTube channel](https://gitcoin.co/grants/174/defi-educational-videos-by-chris-blec) (<https://gitcoin.co/grants/174/defi-educational-videos-by-chris-blec>) has increased from \$5,851 to \$12,803. So some churn, but also some continuity between rounds.

On the tech side, we see much more churn in the winners, with a less clear relationship between income last round and income this round:

Round 4 vs Round 5 income (tech)



Last round, the winner was Tornado Cash, claiming \$30,783; this round, they are down to \$8,154. This round, the three roughly-even winners are [Samczsun](https://gitcoin.co/grants/444/white-hat-hacking) (\$4,631 contributions + \$15,704 match = \$20,335 total), [Arboreum](https://gitcoin.co/grants/618/arboreum) (\$16,084 contributions + \$9,046 match = \$25,128 total) and [1inch.exchange](https://gitcoin.co/grants/246/1split) (\$58,566 contributions + \$7,893 match = \$66,459 total), in the latter case the bulk coming from one contribution:

tgerring
47,500.0000 DAI
(+2500.0 DAI optional tip to Gitcoin)

<https://gitcoin.co/tgerring>

In the previous round, those three winners were not even in the top ten, and in some cases not even part of Gitcoin Grants at all.

These numbers show us two things. First, large parts of the Gitcoin community seem to be in the mindset of treating grants not as a question of "how much do you deserve for your last two months of work?", but rather as a one-off reward for years of contributions in the past. This was one of the strongest rebuttals that I received to my [criticism of Antiprosynthesis receiving \\$13,813 in the last round](https://vitalik.ca/general/2020/01/28/round4.html) (<https://vitalik.ca/general/2020/01/28/round4.html>): that the people who contributed to that award did not see it as two months' salary, but rather as a reward for years of dedication and work for the Ethereum ecosystem. In the next round, contributors were content that the debt was sufficiently repaid, and so they moved on to give a similar gift of appreciation and gratitude to Chris Blec.

That said, not everyone contributes in this way. For example, Prysm got \$7,966 last round and \$8,033 this round, and Week in Ethereum is consistently well-rewarded (\$16,727 previous, \$12,195 current), and EthHub saw less stability but still kept half its income (\$13,515 previous, \$6,705 current) even amid a 20% drop to the matching pool size as some funds were redirected to public health. So there definitely are some contributors that are getting almost a reasonable monthly salary from Gitcoin Grants (yes, even these amounts are all serious underpayment, but remember that the pool of funds Gitcoin Grants has to distribute in the first place is quite small, so there's no allocation that would *not* seriously underpay most people; the hope is that in the future we will find ways to make the matching pot grow bigger).

Why didn't more people use recurring contributions?

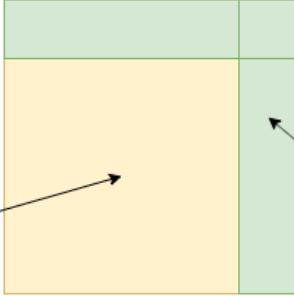
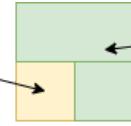
One feature that was tested this round to try to improve stability was recurring contributions: users could choose to split their contribution among multiple rounds. However, the feature was not used often: out of over 8,000 total contributions, only 120 actually made recurring contributions. I can think of three possible explanations for this:

1. People just don't want to give recurring contributions; they genuinely prefer to freshly rethink who they are supporting every round.
2. People would be willing to give more recurring contributions, but there is some kind of "market failure" stopping them; that is, it's collectively optimal for everyone to give more recurring contributions, but it's not any individual contributor's interest to be the first to do so.
3. There's some UI inconveniences or other "incidental" obstacles preventing recurring contributions.

In a recent call with the Gitcoin team, hypothesis (3) was mentioned frequently. A specific issue was that people were worried about making recurring contributions because they were concerned whether or not the money that they lock up for a recurring contribution would be safe. Improving the payment system and notification workflow may help with this. Another option is to move away from explicit "streaming" and instead simply have the UI provide an option for repeating the last round's contributions and making edits from there.

Hypothesis (1) also should be taken seriously; there's genuine value in preventing ossification and allowing space for new entrants. But I want to zoom in particularly on hypothesis (2), the coordination failure hypothesis.

My explanation of hypothesis (2) starts, interestingly enough, with a defense of (1): why ossification is genuinely a risk. Suppose that there are two projects, A and B, and suppose that they are equal quality. But A already has an established base of contributors; B does not (we'll say for illustration it only has a few existing contributors). Here's how much matching you are contributing by participating in each project:

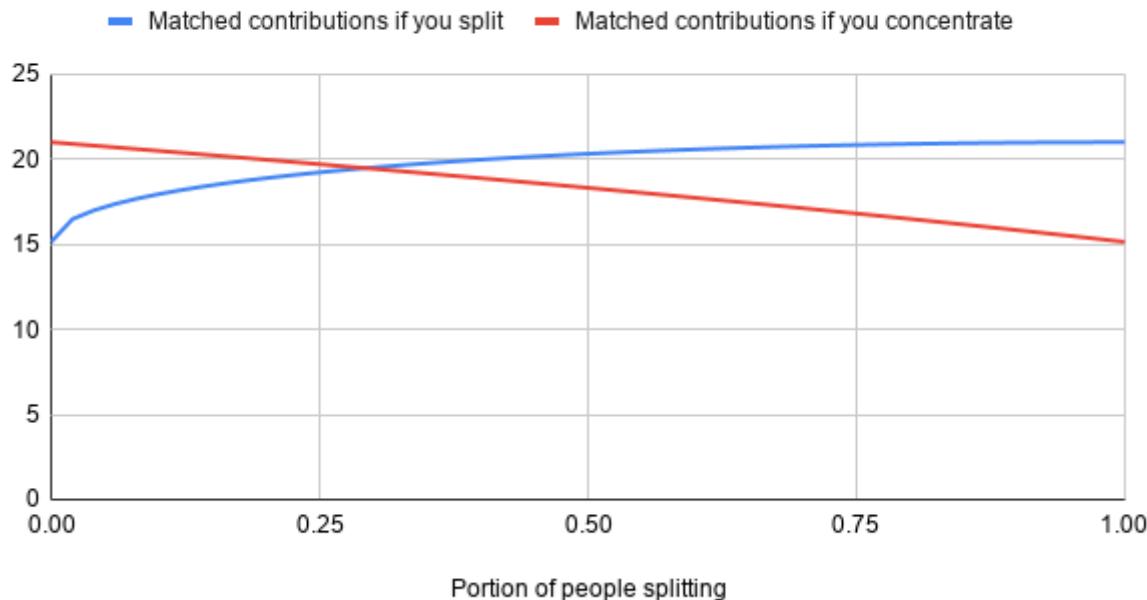
Contributing to A	Contributing to B
 Existing contributions + match → Your contribution + match	 Existing contributions + match → Your contribution + match

Clearly, you have more impact by supporting A, and so A gets even more contributors and B gets fewer; the rich get richer. Even if project B was *somewhat better*, the greater impact from supporting A could still create a lock-in that reinforces A's position. The current everyone-starts-from-zero-in-each-round mechanism greatly limits this type of entrenchment, because, well, everyone's matching gets reset and starts from zero.

However, a very similar effect also is the cause behind the market failure preventing stable recurring contributions, and the every-round-reset *actually exacerbates it*. Look at the same picture above, except instead of thinking of A and B as *two different projects*, think of them as *the same project in the current round and in the next round*.

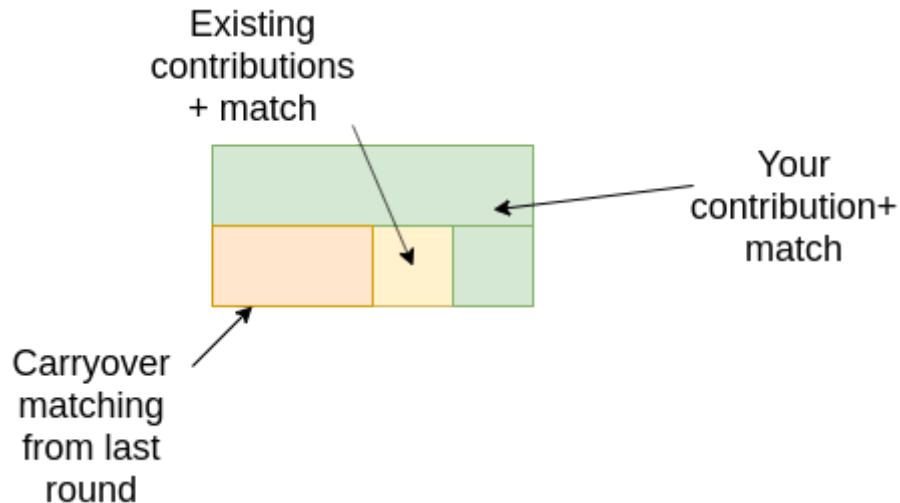
We simplify the model as follows. An individual has two choices: contribute \$10 in the current round, or contribute \$5 in the current round and \$5 in the next round. If the matchings in the two rounds were equal, then the latter option would actually be more favorable: because the matching is proportional to the square root of the donation size, the former might give you eg. a \$200 match now, but the latter would give you \$141 in the current round + \$141 in the next round = \$282. But if you see a large mass of people contributing in the current round, and you expect much fewer people to contribute in the second round, then the choice is not \$200 versus \$141 + \$141, it might be \$200 versus \$141 + \$5. And so you're better off joining the current round's frenzy. We can mathematically analyze the equilibrium:

Split between current and next round, vs go all in current round



So there is a substantial region within which the bad equilibrium of everyone concentrating is sticky: if more than about 3/4 of contributors are expected to concentrate, it seems in your interest to also concentrate. A mathematically astute reader may note that there is always some intermediate strategy ([./images/round5/split2.png](#)) that involves splitting but at a ratio different from 50/50, which you can prove performs better than either full concentrating or the even split, but here we get back to hypothesis (3) above: the UI doesn't offer such a complex menu of choices, it just offers the choice of a one-time contribution or a recurring contribution, so people pick one or the other.

How might we fix this? One option is to add a bit of continuity to matching ratios: when computing pairwise matches, match against not just the current round's contributors but, say, 1/3 of the previous round's contributors as well:



This makes some philosophical sense: the objective of quadratic funding is to subsidize contributions to projects that are detected to be public goods because multiple people have contributed to them, and contributions in the previous round are certainly also evidence of a project's value, so why not reuse those? So here, moving away from everyone-starts-from-zero toward this partial carryover of matching ratios would mitigate the round concentration effect - but, of course, it would exacerbate the risk of entrenchment. Hence, some experimentation and balance may be in order. A broader philosophical question is, is there really a deep inherent tradeoff between risk of entrenchment and stability of income, or is there some way we could get both?

Responses to negative contributions

This round also introduced negative contributions, a feature proposed in my [review of the previous round](https://vitalik.ca/general/2020/01/28/round4.html) (<https://vitalik.ca/general/2020/01/28/round4.html>). But as with recurring contributions, very few people made negative contributions, to the point where their impact on the results was negligible. Also, there was [active opposition](https://twitter.com/evan_van_ness/status/1250152866519621632?s=20) (https://twitter.com/evan_van_ness/status/1248390335048216576?s=20) to [negative contributions](https://twitter.com/ljxie/status/1250178888946176000?s=20) (<https://twitter.com/ljxie/status/1250178888946176000?s=20>), [\(<https://twitter.com/josephdelong/status/1250175753372807170?s=20>\):](https://twitter.com/josephdelong/status/1250175753372807170?s=20)



Source: honestly I have no idea, someone else sent it to me and they forgot where they found it. Sorry :(

The main source of opposition was basically what I predicted in the previous round. Adding a mechanism that allows people to penalize others, even if deservedly so, can have tricky and easily harmful social consequences. Some people even opposed the negative contribution mechanism to the point where they took care to give positive contributions to everyone who received a negative contribution.

How do we respond? To me it seems clear that, in the long run, *some* mechanism of filtering out bad projects, and ideally compensating for overexcitement into good projects, will have to exist. It doesn't necessarily need to be integrated as a symmetric part of the QF, but there does need to be a filter of some form. And this mechanism, whatever form it will take, invariably opens up the possibility of the same social dynamics. So there is a challenge that will have to be solved no matter how we do it.

One approach would be to hide more information: instead of just hiding *who* made a negative contribution, outright hide the fact that a negative contribution was made. Many opponents of negative contributions explicitly indicated that they would be okay (or at least more okay) with such a model. And indeed (see the next section), this is a direction we will have to go anyway. But it would come at a cost - effectively hiding negative contributions would mean not giving as much real-time feedback into what projects got how much funds.

Stepping up the fight against collusion

This round saw much larger-scale attempts at collusion:



Ewoki, GITer of Coins

@owocki

Today I have a major decision to announce.

TLDR - We have identified an instance of collusion in the health round and are counting contributions we identify as colluding as being from the same account.

Next time we will have a more robust identity system.

Details 

12:23 PM · Apr 14, 2020 · [Twitter Web App](#)

15 Retweets 78 Likes

(<https://twitter.com/owocki/status/1250097472694702080>).

It does seem clear that, at current scales, stronger protections against manipulation are going to be required. The first thing that can be done is adding a stronger identity verification layer than Github accounts; this is something that the Gitcoin team is already working on. There is definitely a complex tradeoff between security and inclusiveness to be worked through, but it is not especially difficult to implement a first version. And if the identity problem is solved to a reasonable extent, that will likely be enough to prevent collusion at current scales. But in the longer term, we are going to need protection not just against manipulating the system by making many fake accounts, but also against collusion via bribes (explicit and implicit).

MACI (https://github.com/barryWhiteHat/macI/blob/master/specs/01_introduction.md) is the solution that I proposed (and Barry Whitehat and co are implementing) to solve this problem. Essentially, MACI is a cryptographic construction that allows for contributions to projects to happen on-chain in a privacy-preserving, encrypted form, that allows anyone to cryptographically verify that the mechanism is being implemented *correctly*, but prevents participants from being able to prove to a third party that they made any particular contribution. Unprovability means that if someone tries to bribe others to contribute to their project, the bribe recipients would have no way to prove that they actually contributed to that project, making the bribe unenforceable. Benign "collusion" in the form of friends and family supporting each other would still happen, as people would not easily lie to each other at such small scales, but any broader collusion would be very difficult to maintain.

However, we do need to think through some of the second-order consequences that integrating MACI would introduce. The biggest blessing, and curse, of using MACI is that contributions become hidden. Identities necessarily become hidden, but even the exact timing of contributions would need to be hidden to prevent deanonymization through timing (to prove that *you* contributed, make the total amount jump up between 17:40 and 17:42 today). Instead, for example, totals could be provided and updated once per day. Note that as a corollary negative contributions would be hidden as well; they would only appear if they exceeded all positive contributions for an entire day (and if even that is not desired then the mechanism for when balances are updated could be tweaked to further hide downward changes).

The challenge with hiding contributions is that we lose the "social proof" motivator for contributing: if contributions are unprovable you can't as easily publicly brag about a contribution you made. My best proposal for solving this is for the mechanism to publish one extra number: the *total* amount that a particular participant contributed (counting only projects that have received at least 10 contributors to prevent inflating one's number by self-dealing). Individuals would then have a generic "proof-of-generosity" that they contributed some specific *total* amount, and could publicly state (without proof) what projects it was that they supported. But this is all a significant change to the user experience that will require multiple rounds of experimentation to get right.

Conclusions

All in all, Gitcoin Grants is establishing itself as a significant pillar of the Ethereum ecosystem that more and more projects are relying on for some or all of their support. While it has a relatively low amount of funding at present, and so inevitably underfunds almost everything it touches, we hope that over time we'll continue to see larger sources of funding for the matching pools appear. One option is MEV auctions (<https://ethresear.ch/t/mev-auction-auctioning-transaction-ordering-rights-as-a-solution-to-miner-extractable-value/6788>), another is that new or existing token projects looking to do airdrops could provide the tokens to a matching pool. A third is transaction fees of various applications. With larger amounts of funding, Gitcoin Grants could serve as a more significant funding stream - though to get to that point, further iteration and work on fine-tuning the mechanism will be required.

CLR MATCHING ROUND 5
 **Health Grants**



		NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1	CIC (COVID-19) Kenyan Crisis Aid	85	\$2,767	\$19,802
2	Open Source Covid Ventilators + Masks	89	\$2,472	\$19,628
3	Mask + Test Kit Mutual Aid Fund	87	\$2,447	\$16,718
4	CuraDAO COVID-19 Campaign	75	\$1,056	\$9,884
5	Giveth & Coz: Giving to COVID-19 Causes	57	\$6,027	\$6,785
6	African Angels	48	\$1,970	\$5,947
7	Decentralised supply chain of 3D printed protective gear	49	\$700	\$4,322
8	Tracy	93	\$3,186	\$4,058
9	Save the Children	52	\$6,963	\$3,344
10	Collab19	35	\$175	\$1,844

Additionally, this round saw Gitcoin Grants' first foray into applications beyond Ethereum with the health section. There is growing interest in quadratic funding from local government bodies and other non-blockchain groups, and it would be very valuable to see quadratic funding more broadly deployed in such contexts. That said, there are unique challenges there too. First, there's issues around onboarding people who do not already have cryptocurrency. Second, the Ethereum community is naturally expert in the needs of the Ethereum community, but neither it nor average people are expert in, eg. medical support for the coronavirus pandemic. We should expect quadratic funding to perform worse when the participants are not experts in the domain they're being asked to contribute to. Will non-blockchain uses of QF focus on domains where there's a clear local community that's expert in its own needs, or will people try larger-scale deployments soon? If we do see larger-scale deployments, how will those turn out? There's still a lot of questions to be answered.

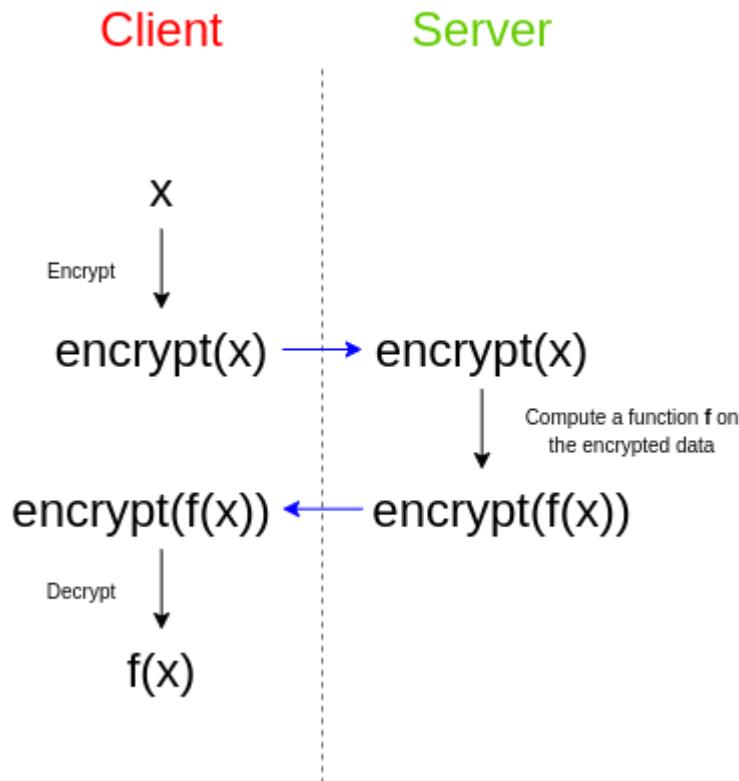
Exploring Fully Homomorphic Encryption

2020 Jul 20

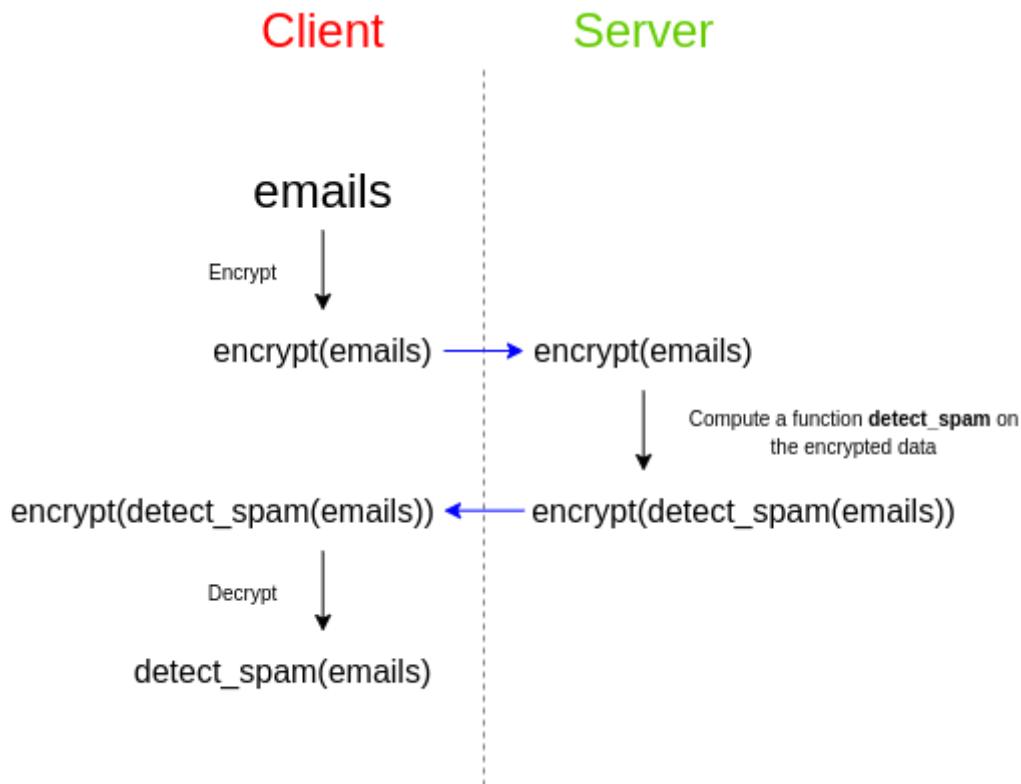
[See all posts \(/\)](#)

Special thanks to Karl Floersch and Dankrad Feist for review

Fully homomorphic encryption has for a long time been considered one of the holy grails of cryptography. The promise of fully homomorphic encryption (FHE) is powerful: it is a type of encryption that allows a third party to perform computations on encrypted data, and get an encrypted result that they can hand back to whoever has the decryption key for the original data, *without* the third party being able to decrypt the data or the result themselves.



As a simple example, imagine that you have a set of emails, and you want to use a third party spam filter to check whether or not they are spam. The spam filter has a desire for *privacy of their algorithm*: either the spam filter provider wants to keep their source code closed, or the spam filter depends on a very large database that they do not want to reveal publicly as that would make attacking easier, or both. However, you care about the *privacy of your data*, and don't want to upload your unencrypted emails to a third party. So here's how you do it:



Fully homomorphic encryption has many applications, including in the blockchain space. One key example is that can be used to implement privacy-preserving light clients (the light client hands the server an encrypted index i , the server computes and returns $\text{data}[0] * (i = 0) + \text{data}[1] * (i = 1) + \dots + \text{data}[n] * (i = n)$, where $\text{data}[i]$ is the i 'th piece of data in a block or state along with its Merkle branch and $(i = k)$ is an expression that returns 1 if $i = k$ and otherwise 0; the light client gets the data it needs and the server learns nothing about what the light client asked).

It can also be used for:

- More efficient stealth address protocols (<https://ethresear.ch/t/open-problem-improving-stealth-addresses/7438>), and more generally scalability solutions to privacy-preserving protocols that today require each user to personally scan the entire blockchain for incoming transactions
- Privacy-preserving data-sharing marketplaces that let users allow some specific computation to be performed on their data while keeping full control of their data for themselves
- An ingredient in more powerful cryptographic primitives, such as more efficient multi-party computation protocols and perhaps eventually obfuscation

And it turns out that fully homomorphic encryption is, conceptually, not that difficult to understand!

Partially, Somewhat, Fully homomorphic encryption

First, a note on definitions. There are different kinds of homomorphic encryption, some more powerful than others, and they are separated by what kinds of functions one can compute on the encrypted data.

- **Partially homomorphic encryption** allows evaluating only a very limited set of operations on encrypted data: either just additions (so given $\text{encrypt}(a)$ and $\text{encrypt}(b)$ you can compute $\text{encrypt}(a+b)$), or just multiplications (given $\text{encrypt}(a)$ and $\text{encrypt}(b)$ you can compute $\text{encrypt}(a*b)$).
- **Somewhat homomorphic encryption** allows computing additions as well as a *limited* number of multiplications (alternatively, polynomials up to a limited degree). That is, if you get $\text{encrypt}(x_1) \dots$

`encrypt(xn)` (assuming these are "original" encryptions and not already the result of homomorphic computation), you can compute `encrypt(p(x1 ... xn))`, as long as `p(x1 ... xn)` is a polynomial with degree < D for some specific degree bound D (D is usually very low, think 5-15).

- **Fully homomorphic encryption** allows unlimited additions and multiplications. Additions and multiplications let you replicate any binary circuit gates (`AND(x, y) = x*y`, `OR(x, y) = x+y-x*y`, `XOR(x, y) = x+y-2*x*y` or just `x+y` if you only care about even vs odd, `NOT(x) = 1-x` ...), so this is sufficient to do arbitrary computation on encrypted data.

Partially homomorphic encryption is fairly easy; eg. RSA has a multiplicative homomorphism: $\text{enc}(x) = x^e$, $\text{enc}(y) = y^e$, so $\text{enc}(x) * \text{enc}(y) = (xy)^e = \text{enc}(xy)$. Elliptic curves can offer similar properties with addition. Allowing both addition and multiplication is, it turns out, significantly harder.

A simple somewhat-HE algorithm

Here, we will go through a somewhat-homomorphic encryption algorithm (ie. one that supports a limited number of multiplications) that is surprisingly simple. A more complex version of this category of technique was used by Craig Gentry to create [the first-ever fully homomorphic scheme](#) (<https://crypto.stanford.edu/craig/craig-thesis.pdf>) in 2009. More recent efforts have switched to using different schemes based on vectors and matrices, but we will still go through this technique first.

We will describe all of these encryption schemes as *secret-key* schemes; that is, the same key is used to encrypt and decrypt. Any secret-key HE scheme can be turned into a public key scheme easily: a "public key" is typically just a set of many encryptions of zero, as well as an encryption of one (and possibly more powers of two). To encrypt a value, generate it by adding together the appropriate subset of the non-zero encryptions, and then adding a random subset of the encryptions of zero to "randomize" the ciphertext and make it infeasible to tell what it represents.

The secret key here is a large prime, p (think of p as having hundreds or even thousands of digits). The scheme can only encrypt 0 or 1, and "addition" becomes XOR, ie. $1 + 1 = 0$. To encrypt a value m (which is either 0 or 1), generate a large random value R (this will typically be even larger than p) and a smaller random value r (typically much smaller than p), and output:

$$\text{enc}(m) = R * p + r * 2 + m$$

To decrypt a ciphertext ct, compute:

$$\text{dec}(ct) = (ct \bmod p) \bmod 2$$

To add two ciphertexts ct_1 and ct_2 , you simply, well, add them: $ct_1 + ct_2$. And to multiply two ciphertexts, you once again... multiply them: $ct_1 * ct_2$. We can prove the homomorphic property (that the sum of the encryptions is an encryption of the sum, and likewise for products) as follows.

Let:

$$ct_1 = R_1 * p + r_1 * 2 + m_1$$

$$ct_2 = R_2 * p + r_2 * 2 + m_2$$

We add:

$$ct_1 + ct_2 = R_1 * p + R_2 * p + r_1 * 2 + r_2 * 2 + m_1 + m_2$$

Which can be rewritten as:

$$(R_1 + R_2) * p + (r_1 + r_2) * 2 + (m_1 + m_2)$$

Which is of the exact same "form" as a ciphertext of $m_1 + m_2$. If you decrypt it, the first mod p removes the first term, the second mod 2 removes the second term, and what's left is $m_1 + m_2$ (remember that if $m_1 = 1$ and $m_2 = 1$ then the 2 will get absorbed into the second term and you'll be left with zero). And so, voila, we have additive homomorphism!

Now let's check multiplication:

$$ct_1 * ct_2 = (R_1 * p + r_1 * 2 + m_1) * (R_2 * p + r_2 * 2 + m_2)$$

Or:

$$\begin{aligned} & (R_1 * R_2 * p + r_1 * 2 + m_1 + r_2 * 2 + m_2) * p + \\ & (r_1 * r_2 * 2 + r_1 * m_2 + r_2 * m_1) * 2 + \\ & (m_1 * m_2) \end{aligned}$$

This was simply a matter of expanding the product above, and grouping together all the terms that contain p, then all the remaining terms that contain 2, and finally the remaining term which is the product of the messages. If you decrypt, then once again the mod p removes the first group, the mod 2 removes the second group, and only $m_1 * m_2$ is left.

But there are two problems here: first, the size of the ciphertext itself grows (the length roughly doubles when you multiply), and second, the "noise" (also often called "error") in the smaller $\backslash * 2$ term also gets quadratically bigger. Adding this error into the ciphertexts was necessary because the security of this scheme is based on the [approximate GCD problem](https://oeis.org/wiki/Greatest_common_divisor#Approximate_GCD_problem) (https://oeis.org/wiki/Greatest_common_divisor#Approximate_GCD_problem):

Approximate GCD problem

In the **approximate GCD problem** you are given n "near" multiples $\{m_1, m_2, \dots, m_n\}$ of some unknown positive integer p , i.e.

$$m_1 = p \cdot q_1 + r_1, \quad 0 \leq r_1 < p,$$

$$m_2 = p \cdot q_2 + r_2, \quad 0 \leq r_2 < p,$$

...,

$$m_n = p \cdot q_n + r_n, \quad 0 \leq r_n < p,$$

and you need to find out p , without the knowledge of any q_i and r_i .

Had we instead used the "exact GCD problem", breaking the system would be easy: if you just had a set of expressions of the form $p * R_1 + m_1, p * R_2 + m_2, \dots$, then you could use the [Euclidean algorithm](https://en.wikipedia.org/wiki/Euclidean_algorithm) (https://en.wikipedia.org/wiki/Euclidean_algorithm) to efficiently compute the greatest common divisor p . But if the ciphertexts are only *approximate* multiples of p with some "error", then extracting p quickly becomes impractical, and so the scheme can be secure.

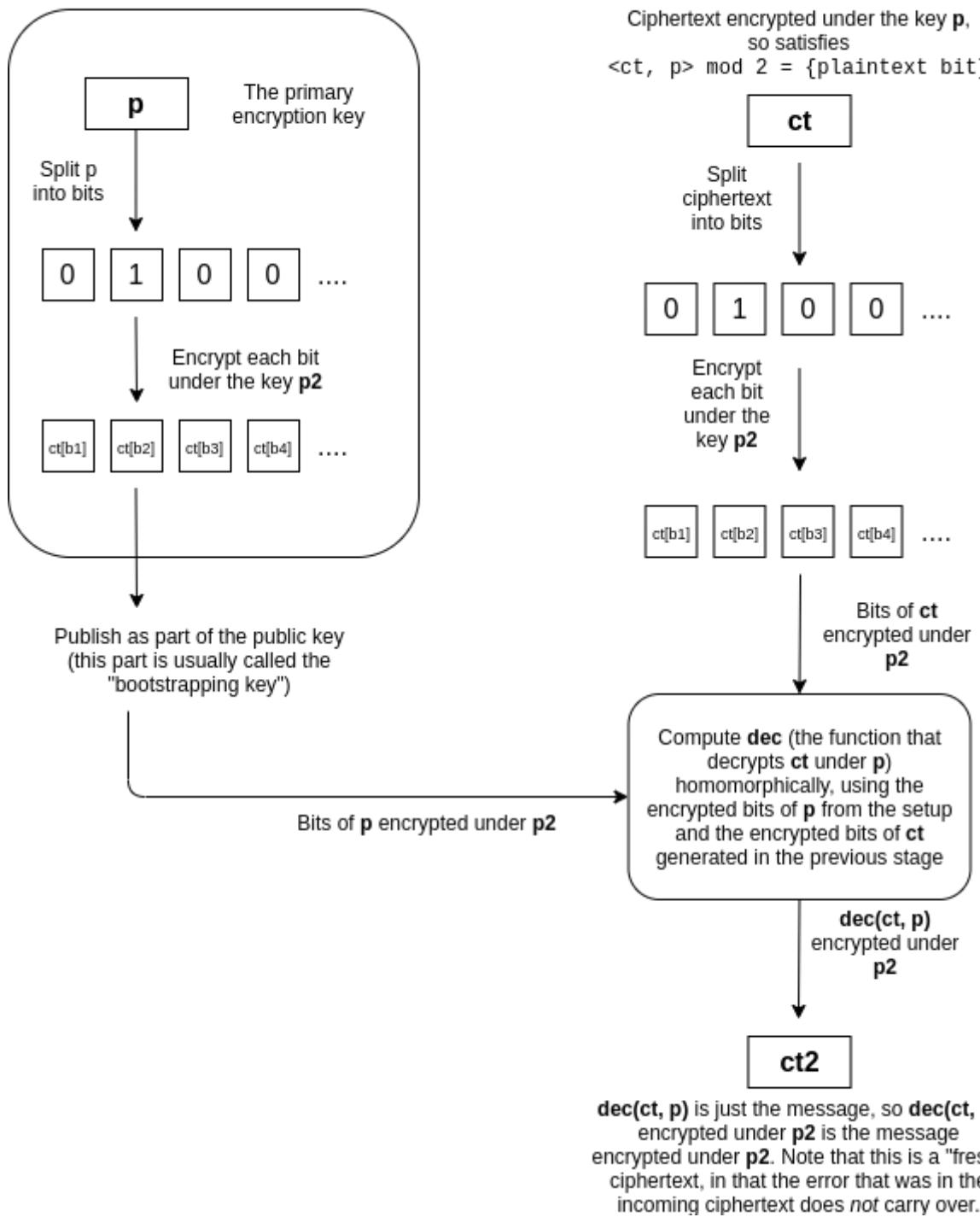
Unfortunately, the error introduces the inherent limitation that if you multiply the ciphertexts by each other enough times, the error eventually grows big enough that it exceeds p , and at that point the mod p and mod 2 steps "interfere" with each other, making the data unextractable. This will be an inherent tradeoff in all of these homomorphic encryption schemes: extracting information from *approximate* equations "with errors" is much harder than extracting information from exact equations, but any error you add quickly increases as you do computations on encrypted data, bounding the amount of computation that you can do before the error becomes overwhelming. And **this is why these schemes are only "somewhat" homomorphic**.

Bootstrapping

There are two classes of solution to this problem. First, in many somewhat homomorphic encryption schemes, there are clever tricks to make multiplication only increase the error by a constant factor (eg. 1000x) instead of squaring it. Increasing the error by 1000x still sounds by a lot, but keep in mind that if p (or its equivalent in other schemes) is a 300-digit number, that means that you can multiply numbers by each other 100 times, which is enough to compute a very wide class of computations. Second, there is Craig Gentry's technique of "bootstrapping".

Suppose that you have a ciphertext ct that is an encryption of some m under a key p , that has a lot of error. The idea is that we "refresh" the ciphertext by turning it into a new ciphertext of m under another key p_2 , where this process "clears out" the old error (though it will introduce a fixed amount of new error). The trick is quite clever. The holder of p and p_2 provides a "bootstrapping key" that consists of an encryption of *the bits of* p under the key p_2 , as well as the public key for p_2 . Whoever is doing computations on data encrypted under p would then take the bits of the ciphertext ct , and individually encrypt these bits under p_2 . They would then *homomorphically compute the decryption under* p using these ciphertexts, and get out the single bit, which would be m encrypted under p_2 .

Setup



This is difficult to understand, so we can restate it as follows. The decryption procedure $\text{dec}(ct, p)$ is itself a computation, and so it can itself be implemented as a circuit that takes as input the bits of ct and the bits of p , and outputs the decrypted bit $m \in \{0, 1\}$. If someone has a ciphertext ct encrypted under p , a public key for p_2 , and the bits of p encrypted under p_2 , then they can compute $\text{dec}(ct, p) = m$ "homomorphically", and get out m encrypted under p_2 . Notice that the decryption procedure itself washes away the old error; it just outputs 0 or 1. The decryption procedure is itself a circuit, which contains additions or multiplications, so it will introduce new error, but this new error does not depend on the amount of error in the original encryption.

(Note that we can avoid having a distinct new key p_2 (and if you want to bootstrap multiple times, also a p_3, p_4, \dots) by just setting $p_2 = p$. However, this introduces a new assumption, usually called "circular security"; it becomes more difficult (https://link.springer.com/content/pdf/10.1007%2F978-3-642-36594-2_32.pdf). to formally prove security if you do this,

though many cryptographers think it's fine and circular security poses no significant risk in practice)

But.... there is a catch. In the scheme as described above (using circular security or not), the error blows up so quickly that even the decryption circuit of the scheme itself is too much for it. That is, the new m encrypted under p_2 would already have so much error that it is unreadable. This is because each AND gate doubles the bit-length of the error, so a scheme using a d -bit modulus p can only handle less than $\log(d)$ multiplications (in series), but decryption requires computing $\bmod p$ in a circuit made up of these binary logic gates, which requires... more than $\log(d)$ multiplications.

Craig Gentry came up with clever techniques to get around this problem, but they are arguably too complicated to explain; instead, we will skip straight to newer work from 2011 and 2013, that solves this problem in a different way.

Learning with errors

To move further, we will introduce a [different type of somewhat-homomorphic encryption](#) (<https://eprint.iacr.org/2011/344.pdf>) introduced by Brakerski and Vaikuntanathan in 2011, and show how to bootstrap it. Here, we will move away from keys and ciphertexts being *integers*, and instead have keys and ciphertexts be *vectors*. Given a key $k = k_1, k_2, \dots, k_n$, to encrypt a message m , construct a vector $c = c_1, c_2, \dots, c_n$ such that the inner product (or "[dot product](#) (https://en.wikipedia.org/wiki/Dot_product)")

$\langle c, k \rangle = c_1 k_1 + c_2 k_2 + \dots + c_n k_n$, modulo some fixed number p , equals $m + 2e$ where m is the message (which must be 0 or 1), and e is a small (much smaller than p) "error" term. A "public key" that allows encryption but not decryption can be constructed, as before, by making a set of encryptions of 0; an encryptor can randomly combine a subset of these equations and add 1 if the message they are encrypting is 1. To decrypt a ciphertext c knowing the key k , you would compute $\langle c, k \rangle \bmod p$, and see if the result is odd or even (this is the same "mod p mod 2" trick we used earlier). Note that here the $\bmod p$ is typically a "symmetric" mod, that is, it returns a number between $-\frac{p}{2}$ and $\frac{p}{2}$ (eg. $137 \bmod 10 = -3$, $212 \bmod 10 = 2$); this allows our error to be positive or negative. Additionally, p does not necessarily have to be prime, though it does need to be odd.

Key	3	14	15	92	65
Ciphertext	2	71	82	81	8

The key and the ciphertext are both vectors, in this example of five elements each.

In this example, we set the modulus $p = 103$. The dot product is $3 * 2 + 14 * 71 + 15 * 82 + 92 * 81 + 65 * 8 = 10202$, and $10202 = 99 * 103 + 5$. 5 itself is of course $2 * 2 + 1$, so the message is 1. Note that in practice, the first element of the key is often set to 1; this makes it easier to generate ciphertexts for a particular value (see if you can figure out why).

The security of the scheme is based on an assumption known as "[learning with errors](#) (https://en.wikipedia.org/wiki/Learning_with_errors)" (LWE) - or, in more jargony but also more understandable terms, the hardness of *solving systems of equations with errors*.

LWE. The LWE problem asks to recover a secret $\mathbf{s} \in \mathbb{Z}_q^n$ given a sequence of ‘approximate’ random linear equations on \mathbf{s} . For instance, the input might be

$$\begin{aligned} 14s_1 + 15s_2 + 5s_3 + 2s_4 &\approx 8 \pmod{17} \\ 13s_1 + 14s_2 + 14s_3 + 6s_4 &\approx 16 \pmod{17} \\ 6s_1 + 10s_2 + 13s_3 + 1s_4 &\approx 3 \pmod{17} \\ 10s_1 + 4s_2 + 12s_3 + 16s_4 &\approx 12 \pmod{17} \\ 9s_1 + 5s_2 + 9s_3 + 6s_4 &\approx 9 \pmod{17} \\ 3s_1 + 6s_2 + 4s_3 + 5s_4 &\approx 16 \pmod{17} \end{aligned}$$

where each equation is correct up to some small additive error (say, ± 1), and our goal is to recover \mathbf{s} . (The answer in this case is $\mathbf{s} = (0, 13, 9, 11)$.)

If not for the error, finding \mathbf{s} would be very easy: after about n equations, we can recover \mathbf{s} in polynomial time using Gaussian elimination. Introducing the error seems to make the problem significantly more difficult. For instance, the Gaussian elimination algorithm takes linear combinations of n equations, thereby amplifying the error to unmanageable levels, leaving essentially no information in the resulting equations.

(<https://cims.nyu.edu/~regev/papers/lwesurvey.pdf>).

A ciphertext can itself be viewed as an equation: $k_1c_1 + \dots + k_nc_n \approx 0$, where the key $k_1 \dots k_n$ is the unknowns, the ciphertext $c_1 \dots c_n$ is the coefficients, and the equality is only approximate because of both the message (0 or 1) and the error ($2e$ for some relatively small e). The LWE assumption ensures that even if you have access to many of these ciphertexts, you cannot recover \mathbf{k} .

Note that in some descriptions of LWE, $\langle c, k \rangle$ can equal any value, but this value must be provided as part of the ciphertext. This is mathematically equivalent to the $\langle c, k \rangle = m+2e$ formulation, because you can just add this answer to the end of the ciphertext and add -1 to the end of the key, and get two vectors that when multiplied together just give $m+2e$. We'll use the formulation that requires $\langle c, k \rangle$ to be near-zero (ie. just $m+2e$) because it is simpler to work with.

Multiplying ciphertexts

It is easy to verify that the encryption is additive: if $\langle ct_1, k \rangle = 2e_1 + m_1$ and $\langle ct_2, k \rangle = 2e_2 + m_2$, then $\langle ct_1 + ct_2, k \rangle = 2(e_1 + e_2) + m_1 + m_2$ (the addition here is modulo p). What is harder is multiplication: unlike with numbers, there is no natural way to multiply two length- n vectors into another length- n vector. The best that we can do is the [outer product](https://en.wikipedia.org/wiki/Outer_product) (https://en.wikipedia.org/wiki/Outer_product): a vector containing the products of each possible pair where the first element comes from the first vector and the second element comes from the second vector. That is, $a \otimes b = a_1b_1 + a_2b_1 + \dots + a_nb_1 + a_1b_2 + \dots + a_nb_2 + \dots + a_nb_n$. We can “multiply ciphertexts” using the convenient mathematical identity $\langle a \otimes b, c \otimes d \rangle = \langle a, c \rangle * \langle b, d \rangle$.

Given two ciphertexts c_1 and c_2 , we compute the outer product $c_1 \otimes c_2$. If both c_1 and c_2 were encrypted with k , then $\langle c_1, k \rangle = 2e_1 + m_1$ and $\langle c_2, k \rangle = 2e_2 + m_2$. The outer product $c_1 \otimes c_2$ can be viewed as an encryption of $m_1 * m_2$ under $k \otimes k$; we can see this by looking what happens when we try to decrypt with $k \otimes k$:

$$\begin{aligned} &\langle c_1 \otimes c_2, k \otimes k \rangle \\ &= \langle c_1, k \rangle * \langle c_2, k \rangle \\ &= (2e_1 + m_1) * (2e_2 + m_2) \end{aligned}$$

$$= 2(e_1m_2 + e_2m_1 + 2e_1e_2) + m_1m_2$$

So this outer-product approach works. But there is, as you may have already noticed, a catch: the size of the ciphertext, and the key, grows quadratically.

Relinearization

We solve this with a **relinearization** procedure. The holder of the private key k provides, as part of the public key, a "relinearization key", which you can think of as "noisy" encryptions of $k \otimes k$ under k . The idea is that we provide these encrypted pieces of $k \otimes k$ to anyone performing the computations, allowing them to compute the equation $\langle c_1 \otimes c_2, k \otimes k \rangle$ to "decrypt" the ciphertext, but only in such a way that the output comes back encrypted under k .

It's important to understand what we mean here by "noisy encryptions". Normally, this encryption scheme only allows encrypting $m \in \{0, 1\}$, and an "encryption of m " is a vector c such that $\langle c, k \rangle = m + 2e$ for some small error e . Here, we're "encrypting" arbitrary $m \in \{0, 1, 2, \dots, p-1\}$. Note that the error means that you can't fully recover m from c ; your answer will be off by some multiple of 2. However, it turns out that, for this specific use case, this is fine.

The relinearization key consists of a set of vectors which, when inner-producted (modulo p) with the key k , give values of the form $k_i * k_j * 2^d + 2e \pmod{p}$, one such vector for every possible triple (i, j, d) , where i and j are indices in the key and d is an exponent where $2^d < p$ (note: if the key has length n , there would be $n^2 * \log(p)$ values in the relinearization key; make sure you understand why before continuing).

$\text{enc}(k_1 * k_1)$	$\text{enc}(k_1 * k_1 * 2)$	$\text{enc}(k_1 * k_1 * 4)$	$\text{enc}(k_1 * k_1 * 8)$
$\text{enc}(k_1 * k_2)$	$\text{enc}(k_1 * k_2 * 2)$	$\text{enc}(k_1 * k_2 * 4)$	$\text{enc}(k_1 * k_2 * 8)$
$\text{enc}(k_2 * k_1)$	$\text{enc}(k_2 * k_1 * 2)$	$\text{enc}(k_2 * k_1 * 4)$	$\text{enc}(k_2 * k_1 * 8)$
$\text{enc}(k_2 * k_2)$	$\text{enc}(k_2 * k_2 * 2)$	$\text{enc}(k_2 * k_2 * 4)$	$\text{enc}(k_2 * k_2 * 8)$

Example assuming $p = 15$ and k has length 2. Formally, $\text{enc}(x)$ here means "outputs $x+2e$ if inner-producted with k ".

Now, let us take a step back and look again at our goal. We have a ciphertext which, if decrypted with $k \otimes k$, gives $m_1 * m_2$. We want a ciphertext which, if decrypted with k , gives $m_1 * m_2$. We can do this with the relinearization key. Notice that the decryption equation $\langle ct_1 \otimes ct_2, k \otimes k \rangle$ is just a big sum of terms of the form $(ct_{1,i} * ct_{2,j}) * k_p * k_q$.

And what do we have in our relinearization key? A bunch of elements of the form $2^d * k_p * k_q$, noisy-encrypted under k , for every possible combination of p and q ! Having all the powers of two in our relinearization key allows us to generate any $(ct_{1,i} * ct_{2,j}) * k_p * k_q$ by just adding up $\leq \log(p)$ powers of two (eg. $13 = 8 + 4 + 1$) together for each (p, q) pair.

For example, if $ct_1 = [1, 2]$ and $ct_2 = [3, 4]$, then $ct_1 \otimes ct_2 = [3, 4, 6, 8]$, and $\text{enc}(\langle ct_1 \otimes ct_2, k \otimes k \rangle) = \text{enc}(3k_1k_1 + 4k_1k_2 + 6k_2k_1 + 8k_2k_2)$ could be computed via:

$$\begin{aligned} &\text{enc}(k_1 * k_1) + \text{enc}(k_1 * k_1 * 2) + \text{enc}(k_1 * k_1 * 4) + \\ &\text{enc}(k_2 * k_1 * 2) + \text{enc}(k_2 * k_1 * 4) + \text{enc}(k_2 * k_1 * 8) \end{aligned}$$

Note that each noisy-encryption in the relinearization key has some even error $2e$, and the equation $\langle ct_1 \otimes ct_2, k \otimes k \rangle$ itself has some error: if $\langle ct_1, k \rangle = 2e_1 + m_1$ and $\langle ct_2, k \rangle = 2e_2 + m_2$, then $\langle ct_1 \otimes ct_2, k \otimes k \rangle = \langle ct_1, k \rangle * \langle ct_2, k \rangle = 2(2e_1e_2 + e_1m_2 + e_2m_1) + m_1m_2$. But this total error is still (relatively)

small ($2e_1e_2 + e_1m_2 + e_2m_1$ plus $n^2 * \log(p)$ fixed-size errors from the realinearization key), and the error is even, and so the result of this calculation still gives a value which, when inner-producted with k , gives $m_1 * m_2 + 2e'$ for some "combined error" e' .

The broader technique we used here is a common trick in homomorphic encryption: provide pieces of the key encrypted under the key itself (or a different key if you are pedantic about avoiding circular security assumptions), such that someone computing on the data can compute the decryption equation, but only in such a way that the output itself is still encrypted. It was used in bootstrapping above, and it's used here; it's best to make sure you mentally understand what's going on in both cases.

This new ciphertext has considerably more error in it: the $n^2 * \log(p)$ different errors from the portions of the relinearization key that we used, plus the $2(2e_1e_2 + e_1m_2 + e_2m_1)$ from the original outer-product ciphertext. Hence, the new ciphertext still does have quadratically larger error than the original ciphertexts, and so we still haven't solved the problem that the error blows up too quickly. To solve this, we move on to another trick...

Modulus switching

Here, we need to understand an important algebraic fact. A ciphertext is a vector ct , such that $\langle ct, k \rangle = m + 2e$, where $m \in \{0, 1\}$. But we can also look at the ciphertext from a different "perspective": consider $\frac{ct}{2}$ (modulo p). $\langle \frac{ct}{2}, k \rangle = \frac{m}{2} + e$, where $\frac{m}{2} \in \{0, \frac{p+1}{2}\}$. Note that because (modulo p) $(\frac{p+1}{2}) * 2 = p + 1 = 1$, division by 2 (modulo p) maps 1 to $\frac{p+1}{2}$; this is a very convenient fact for us.

x	Modular division by 2, mod 9	Regular rounded-down division by 2
0	0	0
1	5	0
2	1	1
3	6	1
4	2	2
5	7	2
6	3	3
7	8	3
8	4	4

The scheme in this section uses both modular division (ie. multiplying by the [modular multiplicative inverse](#) (https://en.wikipedia.org/wiki/Modular_multiplicative_inverse)) and regular "rounded down" integer division; make sure you understand how both work and how they are different from each other.

That is, the operation of dividing by 2 (modulo p) converts small even numbers into small numbers, and it converts 1 into $\frac{p}{2}$ (rounded up). So if we look at $\frac{ct}{2}$ (modulo p) instead of ct , decryption involves computing $\langle \frac{ct}{2}, k \rangle$ and seeing if it's closer to 0 or $\frac{p}{2}$. This "perspective" is much more robust to certain kinds of errors, where you know the error is small but can't guarantee that it's a multiple of 2.

Now, here is something we can do to a ciphertext.

1. Start: $\langle ct, k \rangle = \{0 \text{ or } 1\} + 2e \pmod{p}$
2. Divide ct by 2 (modulo p): $\langle ct', k \rangle = \{0 \text{ or } \frac{p}{2}\} + e \pmod{p}$
3. Multiply ct' by $\frac{q}{p}$ using "regular rounded-down integer division": $\langle ct'', k \rangle = \{0 \text{ or } \frac{q}{2}\} + e' + e_2 \pmod{q}$
4. Multiply ct'' by 2 (modulo q): $\langle ct''', k \rangle = \{0 \text{ or } 1\} + 2e' + 2e_2 \pmod{q}$

Step 3 is the crucial one: it converts a ciphertext under modulus p into a ciphertext under modulus q . The process just involves "scaling down" each element of ct' by multiplying by $\frac{q}{p}$ and rounding down, eg.

$$\text{f loor}(56 * \frac{15}{103}) = \text{f loor}(8.15533\dots) = 8.$$

The idea is this: if $\langle ct', k \rangle = m * \frac{p}{2} + e \pmod{p}$, then we can interpret this as $\langle ct', k \rangle = p(z + \frac{m}{2}) + e$ for some integer z . Therefore, $\langle ct' * \frac{q}{p}, k \rangle = q(z + \frac{m}{2}) + e * \frac{p}{q}$. Rounding adds error, but only a little bit (specifically, up to the size of the values in k , and we can make the values in k small without sacrificing security). Therefore, we can say $\langle ct' * \frac{q}{p}, k \rangle = m * \frac{q}{2} + e' + e_2 \pmod{q}$, where $e' = e * \frac{q}{p}$, and e_2 is a small error from rounding.

What have we accomplished? We turned a ciphertext with modulus p and error $2e$ into a ciphertext with modulus q and error $2(\text{f loor}(e * \frac{p}{q}) + e_2)$, where the new error is *smaller* than the original error.

Let's go through the above with an example. Suppose:

- ct is just one value, [5612]
- $k = [9]$
- $p = 9999$ and $q = 113$

$\langle ct, k \rangle = 5612 * 9 = 50508 = 9999 * 5 + 2 * 256 + 1$, so ct represents the bit 1, but the error is fairly large ($e = 256$).

Step 2: $ct' = \frac{ct}{2} = 2806$ (remember this is modular division; if ct were instead 5613, then we would have $\frac{ct}{2} = 7806$). Checking: $\langle ct', k \rangle = 2806 * 9 = 25254 = 9999 * 2.5 + 256.5$

Step 3: $ct'' = \text{f loor}(2806 * \frac{113}{9999}) = \text{f loor}(31.7109\dots) = 31$. Checking: $\langle ct'', k \rangle = 279 = 113 * 2.5 - 3.5$

Step 4: $ct''' = 31 * 2 = 62$. Checking: $\langle ct''', k \rangle = 558 = 113 * 5 - 2 * 4 + 1$

And so the bit 1 is preserved through the transformation. The crazy thing about this procedure is: *none of it requires knowing k* . Now, an astute reader might notice: you reduced the *absolute* size of the error (from 256 to 2), but the *relative* size of the error remained unchanged, and even slightly increased: $\frac{256}{9999} \approx 2.5\%$ but $\frac{4}{113} \approx 3.5\%$. Given that it's the relative error that causes ciphertexts to break, what have we gained here?

The answer comes from what happens to error when you multiply ciphertexts. Suppose that we start with a ciphertext x with error 100, and modulus $p = 10^{16} - 1$. We want to repeatedly square x , to compute $((x^2)^2)^2 = x^{16}$. First, the "normal way":

Power of x	Error	Modulus
x	100	$10^{16}-1$
x^2	10^4	$10^{16}-1$
x^4	10^8	$10^{16}-1$
x^8	10^{16}	$10^{16}-1$
x^{16}	10^{32}	$10^{16}-1$

The error blows up too quickly for the computation to be possible. Now, let's do a modulus reduction after every multiplication. We assume the modulus reduction is imperfect and increases error by a factor of 10, so a 1000x modulo reduction only reduces error from 10000 to 100 (and not to 10):

Power of x	Error	Modulus
x	100	$10^{16}-1$
x^2	10^4	$10^{16}-1$
x^4	100	$10^{13}-1$
x^8	10^4	$10^{13}-1$
x^{16}	100	$10^{10}-1$
x^8	10^4	$10^{10}-1$
x^8	100	10^7-1
x^{16}	10^4	10^7-1

The key mathematical idea here is that the *factor* by which error increases in a multiplication depends on the absolute size of the error, and not its relative size, and so if we keep doing modulus reductions to keep the error small, each multiplication only increases the error by a constant factor. And so, with a d bit modulus (and hence $\approx 2^d$ room for "error"), we can do $O(d)$ multiplications! This is enough to bootstrap.

Another technique: matrices

Another technique (see [Gentry, Sahai, Waters \(2013\)](https://eprint.iacr.org/2013/340.pdf)) for fully homomorphic encryption involves matrices: instead of representing a ciphertext as ct where $\langle ct, k \rangle = 2e + m$, a ciphertext is a matrix, where $k * CT = k * m + e$ (k , the key, is still a vector). The idea here is that k is a "secret near-eigenvector" - a secret vector which, if you multiply the matrix by it, returns something very close to either zero or the key itself.

The fact that addition works is easy: if $k * CT_1 = m_1 * k + e_1$ and $k * CT_2 = m_2 * k + e_2$, then $k * (CT_1 + CT_2) = (m_1 + m_2) * k + (e_1 + e_2)$. The fact that multiplication works is also easy:

$$k * CT_1 * CT_2 = (m_1 * k + e_1) * CT_2 = m_1 * k * CT_2 + e_1 * CT_2 = m_1 * m_2 * k + m_1 * e_2 + e_1 * CT_2$$

The first term is the "intended term"; the latter two terms are the "error". That said, notice that here error does blow up quadratically (see the $e_1 * CT_2$ term; the size of the error increases by the size of each ciphertext element, and the ciphertext elements also square in size), and you do need some clever tricks for avoiding this. Basically, this involves turning ciphertexts into matrices containing their constituent bits before multiplying, to avoid multiplying by anything higher than 1; if you want to see how this works in detail I recommend looking at my code:

https://github.com/vbuterin/research/blob/master/matrix_fhe/matrix_fhe.py#L121

(https://github.com/vbuterin/research/blob/master/matrix_fhe/matrix_fhe.py#L121).

In addition, the code there, and also

https://github.com/vbuterin/research/blob/master/tensor_fhe/homomorphic_encryption.py#L186

(https://github.com/vbuterin/research/blob/master/tensor_fhe/homomorphic_encryption.py#L186), provides simple examples of useful circuits that you can build out of these binary logical operations; the main example is for adding numbers that are represented as multiple bits, but one can also make circuits for comparison ($<$, $>$, $=$), multiplication, division, and many other operations.

Since 2012-13, when these algorithms were created, there have been many optimizations, but they all work on top of these basic frameworks. Often, polynomials are used instead of integers; this is called [ring LWE](#) (https://en.wikipedia.org/wiki/Ring_learning_with_errors). The major challenge is still efficiency: an operation involving a single bit involves multiplying entire matrices or performing an entire relinearization computation, a very high overhead. There are tricks that allow you to perform many bit operations in a single ciphertext operation, and this is actively being worked on and improved.

We are quickly getting to the point where many of the applications of homomorphic encryption in privacy-preserving computation are starting to become practical. Additionally, research in the more advanced applications of the lattice-based cryptography used in homomorphic encryption is rapidly progressing. So this is a space where some things can already be done today, but we can hopefully look forward to much more becoming possible over the next decade.

Gitcoin Grants Round 6 Retrospective

2020 Jul 22

[See all posts \(/\)](#)

Round 6 of Gitcoin Grants has just finished, with \$227,847 in contributions from 1,526 contributors and \$175,000 in matched funds distributed across 695 projects. This time around, we had three categories: the two usual categories of "tech" and "community" (the latter renamed from "media" to reflect a desire for a broad emphasis), and the round-6-special category Crypto For Black Lives.

First of all, here are the results, starting with the tech and community sections:

NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING	NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1 EIP-1559 Community Fund	\$29,963	\$35,579	1 Week in Ethereum News	\$1,873	\$8,621
2 White Hat Hacking	\$3,472	\$5,728	2 EthHub	\$1,597	\$6,764
3 DAppNode	\$2,820	\$5,003	3 Bankless	\$1,021	\$5,382
4 Tornado.cash	\$2,961	\$4,965	4 The Defiant	\$1,114	\$3,776
5 Prysm by Prysmatic Labs	\$3,127	\$4,631	5 DeFi Dad Tutorial	\$1,191	\$3,228
6 Gitcoin Grants Dev Fund	\$4,518	\$4,631	6 Meta Gamma Delta	\$1,639	\$2,925
7 1inch.exchange	\$4,008	\$3,281	7 MetaCartel	\$383	\$2,211
8 ethers.js	\$2,857	\$2,626	8 @antipro synth	\$549	\$1,790
9 Rotki	\$2,344	\$2,508	9 David Hoffman	\$835	\$1,531
10 The Commons Stack	\$5,832	\$2,427	10 The Daily Gwei	\$960	\$1,498

Stability of income

In the last round, one [concern I raised](#) (<https://vitalik.ca/general/2020/04/30/round5.html>) was stability of income. People trying to earn a livelihood off of quadratic funding grants would want to have some guarantee that their income isn't going to completely disappear in the next round just because the hive mind suddenly gets excited about something else.

Round 6 had two mechanisms to try to provide more stability of income:

1. A "shopping cart" interface for giving many contributions, with an explicit "repeat your contributions from the last round" feature
2. A rule that the matching amounts are calculated using not just contributions from this round, but also "carrying over" 1/3 of the contributions from the previous round (ie. if you made a \$10 grant in the previous round, the matching formula would pretend you made a \$10 grant in the previous round and also a \$3.33 grant this round)

1. was clearly successful at one goal: increasing the total number of contributions. But its effect in ensuring stability of income is hard to measure. The effect of (2), on the other hand, is easy to measure, because we have stats for the actual matching amount as well as what the matching amount "would have been" if the 1/3 carry-over rule was not in place.

First from the tech category:

Project	Round 5 match	Round 6 match (with carryover)	Round 6 match (if no carryover)
White Hat Hacking	\$15,704	\$5,728	\$3,676
Arboreum	\$9,046	\$1,143	\$336
1inch.exchange	\$7,893	\$2,626	\$2,429
The Commons Stack	\$6,497	\$2,426	\$1,711
EIP-1559 Community Fund	\$0	\$35,578	\$45,030

Now from the community category:

Project	Round 5 match	Round 6 match (with carryover)	Round 6 match (if no carryover)
Week in Ethereum News	\$10,054	\$8,621	\$7,659
Chris Blec	\$5,716	-	-
EthHub	\$5,148	\$6,764	\$6,437
The Defiant	\$4,886	\$3,776	\$3,205
MetaCartel	\$3,232	\$2,211	\$1,797

Clearly, the rule helps reduce volatility, pretty much exactly as expected. That said, one could argue that this result is trivial: you could argue that all that's going on here is something very similar to grabbing part of the revenue from round N (eg. see how the new EIP-1559 Community Fund earned less than it otherwise would have) and moving it into round N+1. Sure, numerically speaking the revenues are more "stable", but individual projects could have just provided this stability to themselves by only spending 2/3 of the pot from each round, and using the remaining third later when some future round is unexpectedly low. Why should the quadratic funding mechanism significantly increase its complexity just to achieve a gain in stability that projects could simply provide for themselves?

My instinct says that it would be best to try the next round with the "repeat last round" feature but *without* the 1/3 carryover, and see what happens. Particularly, note that the numbers seem to show that the media section would have been "stable enough" even without the carryover. The tech section was more volatile, but only because of the sudden entrance of the EIP 1559 community fund; it would be part of the experiment to see just how common that kind of situation is.

About that EIP 1559 Community fund...

The big unexpected winner of this round was the EIP 1559 community fund. EIP 1559 ([EIP here](https://github.com/ethereum/EIPs/issues/1559) (<https://github.com/ethereum/EIPs/issues/1559>), [FAQ here](https://notes.ethereum.org/Wjr1SnW-QaST7phX9C5wkg?view) (<https://notes.ethereum.org/Wjr1SnW-QaST7phX9C5wkg?view>), original paper [here](https://ethresear.ch/t/draft-position-paper-on-resource-pricing/2838) (<https://ethresear.ch/t/draft-position-paper-on-resource-pricing/2838>)) is a major fee market reform proposal which far-reaching consequences; it aims to improve the user experience of sending Ethereum transactions, reduce economic inefficiencies, provide an accurate in-protocol gas price oracle and burn a portion of fee revenue.

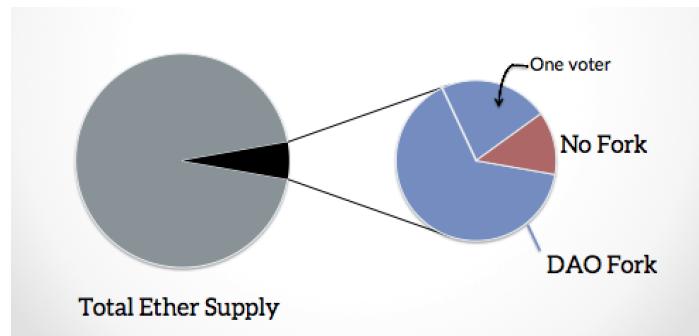
Many people in the Ethereum community are very excited about this proposal, though so far there has been fairly little funding toward getting it implemented. This gitcoin grant was a large community effort toward fixing this.

The grant had quite a few very large contributions, including roughly \$2,400 each from myself and Eric Conner, early on. Early in the round, one could clearly see the EIP 1559 community grant having an abnormally low ratio of matched funds to contributed funds; it was somewhere around \$4k matched to \$20k contributed. This was because while the amount contributed was large, it came from relatively few wealthier donors, and so the matching amount was less than it would have been had the same quantity of funds come from more diverse sources - the quadratic funding formula working as intended. However, a social media push advertising the grant then led to a large number of smaller contributors following along, which then quickly raised the match to its currently very high value (\$35,578).

Quadratic signaling

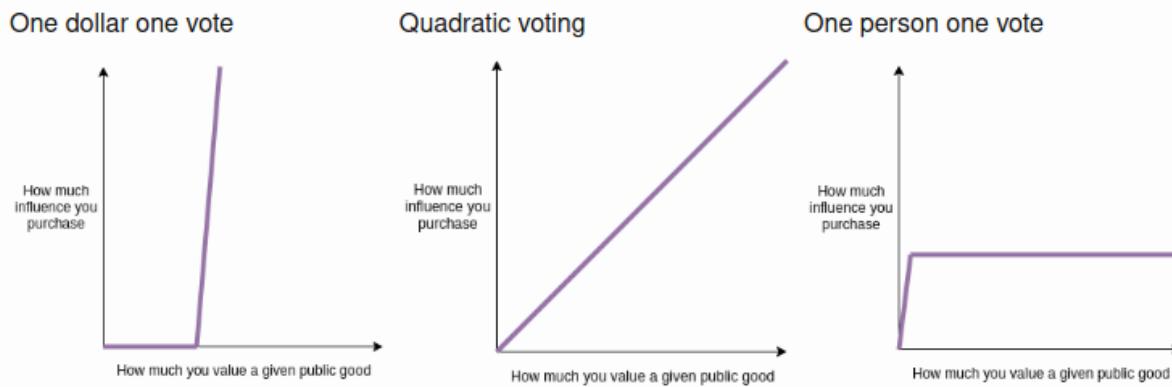
Unexpectedly, this grant proved to have a double function. First, it provided \$65,473 of much-needed funding to EIP 1559 implementation. Second, it served as a credible community signal of the level of demand for the proposal. The Ethereum community has [long been struggling](https://vitalik.ca/general/2017/12/17/voting.html) (<https://vitalik.ca/general/2017/12/17/voting.html>), to find effective ways to determine what "the community" supports, especially in cases of controversy.

Coin votes have been [used in the past](https://www.etherchain.org/coinvote) (<https://www.etherchain.org/coinvote>), and have the advantage that they come with an answer to the key problem of determining who is a "real community member" - the answer is, your membership in the Ethereum community is proportional to how much ETH you have. However, they are plutocratic; in the famous DAO coin vote, a single "yes" voter voted with more ETH than all "no" voters put together (~20% of the total).



The alternative, looking at github, reddit and twitter comments and votes to measure sentiment (sometimes derided as "proof of social media") is egalitarian, but it is easily exploitable, comes with no skin-in-the-game, and frequently falls under criticisms of "foreign interference" (are those *really* ethereum community members disagreeing with the proposal, or just those dastardly bitcoiners coming in from across the pond to stir up trouble?).

Quadratic funding falls perfectly in the middle: the need to contribute monetary value to vote ensures that the votes of those who *really* care about the project count more than the votes of less-concerned outsiders, and the square-root function ensures that the votes of individual ultra-wealthy "whales" cannot beat out a poorer, but broader, coalition.



A diagram from my post on quadratic payments (<https://vitalik.ca/general/2019/12/07/quadratic.html>), showing how quadratic payments is "in the middle" between the extremes of voting-like systems and money-like systems, and avoids the worst flaws of both.

This raises the question: might it make sense to try to use explicit quadratic voting (with the ability to vote "yes" or "no" to a proposal) as an additional signaling tool to determine community sentiment for ethereum protocol proposals?

How well are "guest categories" working?

Since round 5, Gitcoin Grants has had three categories per round: tech, community (called "media" before), and some "guest" category that appears only during that specific round. In round 5 this was COVID relief; in round 6, it's Crypto For Black Lives.



By far the largest recipient was Black Girls CODE, claiming over 80% of the matching pot. My guess for why this happened is simple: Black Girls CODE is an established project that has been participating in the grants for several rounds already, whereas the other projects were new entrants that few people in the Ethereum community knew well. In addition, of course, the Ethereum community "understands" the value of helping people code more than it understands chambers of commerce and bail funds.

This raises the question: is Gitcoin's current approach of having a guest category each round actually working well? The case for "no" is basically this: while the individual causes (empowering black communities, and fighting covid) are certainly admirable, the Ethereum community is by and large not experts at these topics, and we're certainly not experts on *those specific projects* working on those challenges.

If the goal is to try to bring quadratic funding to causes beyond Ethereum, the natural alternative is a separate funding round marketed specifically to those communities; <https://downtownstimulus.com/> (<https://downtownstimulus.com/>) is a great example of this. If the goal is to get the Ethereum community interested in other causes, then perhaps running more than one round on each cause would work better. For example, "guest categories" could last for three rounds (~6 months), with \$8,333 matching per round (and there could be two or three guest categories running simultaneously). In any case, it seems like some revision of the model makes sense.

Collusion

Now, the bad news. This round saw an unprecedented amount of attempted collusion and other forms of fraud. Here are a few of the most egregious examples.

Blatant attempted bribery:

[←](#) **Tweet**

 **arablockchain**
@arablockchain1

any one need free 0.01 \$ETH contact me need github account plz youyou dont have dont try #Airdrops

1:34 PM · Jun 27, 2020 · [Twitter Web App](#)

Impersonation:



Dapp University
@DappUniversity

⚠️ SCAM ALERT! (?)

I *DID NOT* set up a [@gitcoin](#) grant for Dapp University.

← [Thread](#)

Either:

(1) Someone set this up for me to help (if so, please reach out ASAP)

OR:

(2) They're trying to run a scam

[gitcoin.co/grants/847/dap...](https://gitcoin.co/grants/847/dapp-university)

WATCH OUT!



Dapp University | Grants

Hey there, welcome to Dapp University! I am Gregory McCubbin and I just released a new article: Master ...
[🔗 gitcoin.co](https://gitcoin.co)

12:15 PM · Jul 3, 2020 · [Twitter Web App](#)

Many contributions with funds clearly coming from a single address:

 gitcoin disputes
@GitcoinDisputes

Here's a proof of collusion for this grant quicknote.io/7fea75e0-be4e-..... Most of the funds came from a single whale account. Contributions made by fake accounts were already withdrawn by the grant owner through small transactions and funds ...

 "LatAm Cartel" | Grants
This is a call to action, to everyone who wants to be part of this wonderful story. This is a story from unknown times. It'...
gitcoin.co

9:26 AM · Jul 6, 2020 · [Twitter Web App](#)

1 Retweet and comment 1 Like

The big question is: how much fraudulent activity can be prevented in a fully automated/technological way, without requiring detailed analysis of each and every case? If quadratic funding cannot survive such fraud without needing to resort to expensive case-by-case judgement, then regardless of its virtues in an ideal world, in reality it would not be a very good mechanism!

Fortunately, there is a lot that we can do to reduce harmful collusion and fraud that we are not yet doing. Stronger identity systems is one example; in this round, Gitcoin added optional SMS verification, and it seems like the in this round the detected instances of collusion were mostly github-verified accounts and not SMS-verified accounts. In the next round, making some form of extra verification beyond a github account (whether SMS or something more decentralized, eg. BrightID) seems like a good idea. To limit bribery, [MACI](https://github.com/appliedzkp/mac) (<https://github.com/appliedzkp/mac>) can help, by making it impossible for a briber to tell who actually voted for any particular project.

Impersonation is not really a quadratic funding-specific challenge; this could be solved with manual verification, or if one wishes for a more decentralized solution one could try using [Kleros](https://kleros.io/) (<https://kleros.io/>) or some similar system. One could even imagine incentivized reporting: anyone can lay down a deposit and flag a project as fraudulent, triggering an investigation; if the project turns out to be legitimate the deposit is lost but if the project turns out to be fraudulent, the challenger gets half of the funds that were sent to that project.

Conclusion

The best news is the unmentioned news: many of the positive behaviors coming out of the quadratic funding rounds have stabilized. We're seeing valuable projects get funded in the tech and community categories, there has been less social media contention this round than in previous rounds, and people are getting better and better at understanding the mechanism and how to participate in it.

That said, the mechanism is definitely at a scale where we are seeing the kinds of attacks and challenges that we would realistically see in a larger-scale context. There are some challenges that we have not yet worked through (one that I am particularly watching out for is: matched grants going to a project that one part of the community supports and another part of the community thinks is very harmful). That said, we've gotten as far as we have with fewer problems than even I had been anticipating.

I recommend holding steady, focusing on security (and scalability) for the next few rounds, and coming up with ways to increase the size of the matching pots. And I continue to look forward to seeing valuable public goods get funded!

A Philosophy of Blockchain Validation

2020 Aug 17

[See all posts \(/\)](#)

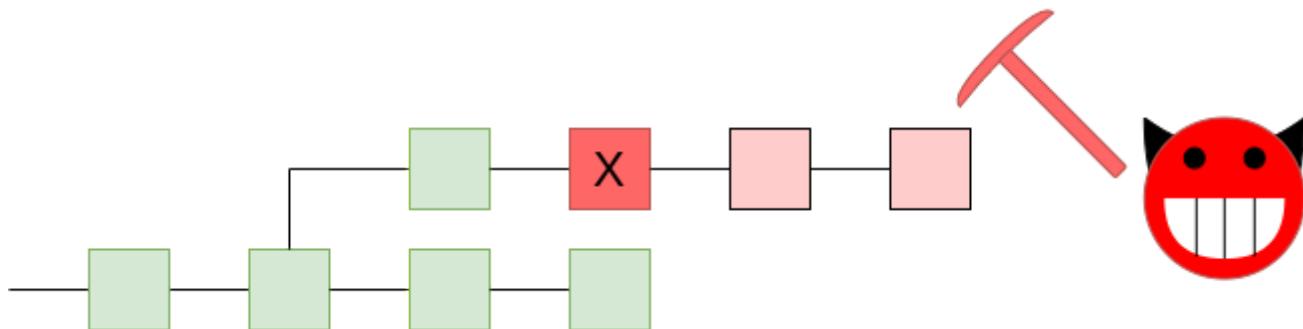
See also:

- [A Proof of Stake Design Philosophy](https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51) (<https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51>).
- [The Meaning of Decentralization](https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274) (<https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274>)
- [Engineering Security through Coordination Problems](https://vitalik.ca/general/2017/05/08/coordination_problems.html) (https://vitalik.ca/general/2017/05/08/coordination_problems.html).

One of the most powerful properties of a blockchain is the fact that every single part of the blockchain's execution can be independently validated. Even if a great majority of a blockchain's miners (or validators in PoS) get taken over by an attacker, if that attacker tries to push through invalid blocks, the network will simply reject them. Even those users that were not verifying blocks at that time can be (potentially automatically) warned by those who were, at which point they can check that the attacker's chain is invalid, and automatically reject it and coordinate on accepting a chain that follows the rules.

But how much validation do we actually need? Do we need a hundred independent validating nodes, a thousand? Do we need a culture where the average person in the world runs software that checks every transaction? It's these questions that are a challenge, and a very important challenge to resolve especially if we want to build blockchains with consensus mechanisms better than the single-chain "Nakamoto" proof of work that the blockchain space originally started with.

Why validate?



A 51% attack pushing through an invalid block. We want the network to reject the chain!

There are two main reasons why it's beneficial for a user to validate the chain. First, it maximizes the chance that the node can correctly determine and say on the **canonical chain** - the chain that the community accepts as legitimate. Typically, the canonical chain is defined as something like "the valid chain that has the most miners/validators supporting it" (eg. the "longest valid chain" in Bitcoin). Invalid chains are rejected by

definition, and if there is a choice between multiple valid chains, the chain that has the most support from miners/validators wins out. And so if you have a node that verifies all the validity conditions, and hence detects which chains are valid and which chains are not, that maximizes your chances of correctly detecting what the canonical chain actually is.

But there is also another deeper reason why validating the chain is beneficial. Suppose that a powerful actor tries to push through a change to the protocol (eg. changing the issuance), and has the support of the majority of miners. If no one else validates the chain, this attack can very easily succeed: everyone's clients will, *by default*, accept the new chain, and by the time anyone sees what is going on, it will be *up to the dissenters* to try to coordinate a rejection of that chain. But if average users are validating, then the coordination problem falls on the other side: it's now the responsibility of whoever is trying to change the protocol to convince the users to actively download the software patch to accept the protocol change.

If enough users are validating, then **instead of defaulting to victory, a contentious attempt to force a change of the protocol will default to chaos**. Defaulting to chaos still causes a lot of disruption, and would require out-of-band social coordination to resolve, but it places a much larger barrier in front of the attacker, and makes attackers much less confident that they will be able to get away with a clean victory, making them much less motivated to even try to start an attack. If *most* users are validating (directly or indirectly), and an attack has *only* the support of the majority of miners, then the attack will outright **default to failure** - the best outcome of all.

The definition view versus the coordination view

Note that this reasoning is very different from a different line of reasoning that we often hear: that a chain that changes the rules is somehow "by definition" not the correct chain, and that no matter how many other users accept some new set of rules, what matters is that you personally can stay on the chain with the old rules that you favor.

Here is one example of the "by definition" perspective from Gavin Andresen (<http://gavinandresen.ninja/a-definition-of-bitcoin>):

I'd like to propose this big-picture technical definition of "Bitcoin":

"Bitcoin" is the ledger of not-previously-spent, validly signed transactions contained in the chain of blocks that begins with the genesis block (hash 00000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f), follows the 21-million coin creation schedule, and has the most cumulative double-SHA256-proof-of-work.¹

Here's another from the Wasabi wallet (<https://docs.wasabiwallet.io/using-wasabi/BitcoinFullNode.html#the-importance-of-running-a-full-node>); this one comes even more directly from the perspective of explaining why full nodes are valuable:

When running a Bitcoin full node, you define the precise monetary rules that you voluntarily agree on. Nobody else forces this choice upon you. Thus any sovereign individual who wants to claim financial independence must run a full node. Once your own rules are firmly established, your software discovers other nodes in the Bitcoin peer-to-peer network which do not break your rules. These peers send you transactions and blocks which are valid according to their set of rules, and you verify for yourself if they are also correct for you. If one of the proposed transactions breaks your own rules, then you mark it as invalid, disconnect from and ban the node who sent you the malicious transaction.

Claim your monetary sovereignty

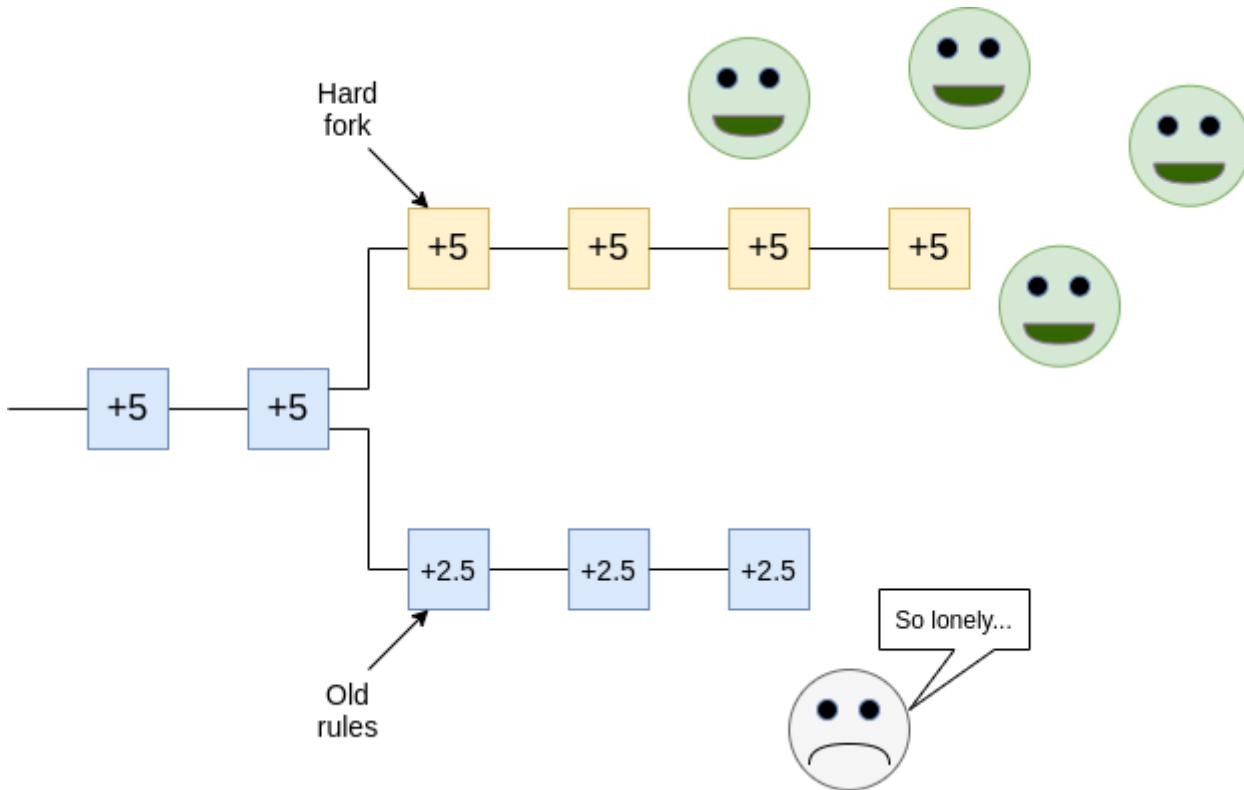
With your full node you define, verify, and enforce the rules of your sound money Bitcoin.

Notice two core components of this view:

1. A version of the chain that does not accept the rules that you consider fundamental and non-negotiable is *by definition* not bitcoin (or not ethereum or whatever other chain), not matter how many other people accept that chain.
2. What matters is that *you* remain on a chain that has rules that *you* consider acceptable.

However, I believe this "individualist" view to be very wrong. To see why, let us take a look at the scenario that we are worried about: the vast majority of participants accept some change to protocol rules that you find unacceptable. For example, imagine a future where transaction fees are very low, and to keep the chain secure, almost everyone else agrees to change to a new set of rules that increases issuance. You stubbornly keep running a node that continues to enforce the old rules, and you fork off to a different chain than the majority.

From your point of view, you still have your coins in a system that runs on rules that you accept. But so what? Other users will not accept your coins. Exchanges will not accept your coins. Public websites may show the price of the new coin as being some high value, but they're referring to the coins on the majority chain; *your* coins are valueless. Cryptocurrencies and blockchains are fundamentally social constructs; without other people believing in them, they mean nothing.



So what is the alternative view? The core idea is to look at blockchains as [engineering security through coordination problems](https://vitalik.ca/general/2017/05/08/coordination_problems.html) (https://vitalik.ca/general/2017/05/08/coordination_problems.html).

Normally, coordination problems in the world are a bad thing: while it would be better for most people if the English language got rid of its highly complex and irregular spelling system and made a phonetic one, or if the United States switched to metric, or if we could immediately drop all prices and wages by ten percent in the event of a recession (<http://www.interfluidity.com/v2/6088.html>), in practice this requires everyone to agree on the switch at the same time, and this is often very very hard.

With blockchain applications, however, we *are using coordination problems to our advantage*. We are using the friction that coordination problems create as a bulwark against malfeasance by centralized actors. We can build systems that have property X, and we can guarantee that they will preserve property X because changing the rules from X to not-X would require a whole bunch of people to agree to update their software at the same time. Even if there is an actor that could force the change, doing so would be hard - much much harder than it would be if it were instead the responsibility of *users* to actively coordinate dissent to resist a change.

Note one particular consequence of this view: it's emphatically *not* the case that the purpose of your full node is just to protect *you*, and in the case of a contentious hard fork, people with full nodes are safe and people without full nodes are vulnerable. Rather, the perspective here is much more one of **herd immunity**: the more people are validating, the more safe everyone is, and even if only some portion of people are validating, everyone gets a high level of protection as a result.

Looking deeper into validation

We now get to the next topic, and one that is very relevant to topics such as light clients and sharding: what are we actually accomplishing by validating? To understand this, let us go back to an earlier point. If an attack happens, I would argue that we have the following preference order over how the attack goes:

default to failure > default to chaos > default to victory

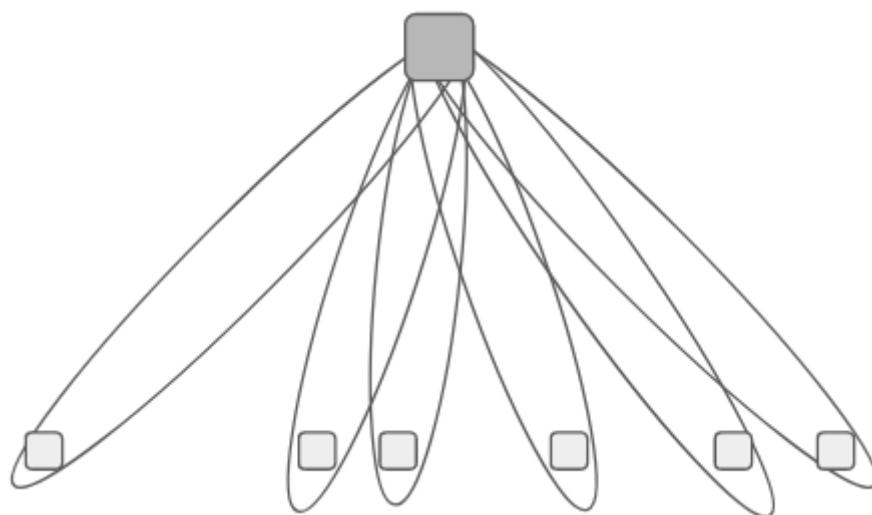
The ">" here of course means "better than". The best is if an attack outright fails; the second best is if an attack leads to confusion, with everyone disagreeing on what the correct chain is, and the worst is if an attack succeeds. Why is chaos so much better than victory? This is a matter of incentives: chaos raises costs for the attacker, and denies them the certainty that they will even win, discouraging attacks from being attempted in the first place. A default-to-chaos environment means that an attacker needs to win *both* the blockchain war of making a 51% attack *and* the "social war" of convincing the community to follow along. This is much more difficult, and much less attractive, than just launching a 51% attack and claiming victory right there.

The goal of validation is then to move away from default to victory to (ideally) default to failure or (less ideally) default to chaos. If you have a fully validating node, and an attacker tries to push through a chain with different rules, then the attack fails. If some people have a fully validating node but many others don't, the attack leads to chaos. But now we can think: are there other ways of achieving the same effect?

Light clients and fraud proofs

One natural advancement in this regard is **light clients with fraud proofs**. Most blockchain light clients that exist today work by simply validating that the majority of miners support a particular block, and not bothering to check if the other protocol rules are being enforced. The client runs on the trust assumption that the majority of miners is honest. If a contentious fork happens, the client follows the majority chain by default, and it's up to users to take an active step if they want to follow the minority chain with the old rules; hence, today's light clients under attack default to victory. But with fraud proof technology, the situation starts to look very different.

A fraud proof in its simplest form works as follows. Typically, a single block in a blockchain only touches a small portion of the blockchain "state" (account balances, smart contract code....). If a fully verifying node processes a block and finds that it is invalid, they can generate a package (the fraud proof) containing the block along with just enough data from the blockchain state to process the block. They broadcast this package to light clients. Light clients can then take the package and use that data to verify the block themselves, even if they have no other data from the chain.



A single block in a blockchain touches only a few accounts. A fraud proof would contain the data in those accounts along with Merkle proofs proving that that data is correct.

This technique is also sometimes known as [stateless validation](https://ethresear.ch/t/the-stateless-client-concept/172) (<https://ethresear.ch/t/the-stateless-client-concept/172>): instead of keeping a full database of the blockchain state, clients can keep only the block headers, and they can verify any block in real time by asking other nodes for a Merkle proof for any desired state entries that block validation is accessing.

The power of this technique is that **light clients can verify individual blocks only if they hear an alarm** (and alarms are verifiable, so if a light client hears a false alarm, they can just stop listening to alarms from that node). Hence, under normal circumstances, the light client is still light, checking only which blocks are supported by the majority of miners/validators. But under those exceptional circumstances where the majority chain contains a block that the light client would not accept, **as long as there is at least one honest node verifying the fraudulent block, that node will see that it is invalid, broadcast a fraud proof, and thereby cause the rest of the network to reject it.**

Sharding

Sharding is a natural extension of this: in a sharded system, there are too many transactions in the system for most people to be verifying directly all the time, but if the system is well designed then any individual invalid block can be detected and its invalidity proven with a fraud proof, and that proof can be broadcasted across the entire network. A sharded network can be summarized as everyone being a light client. And as long as each shard has some minimum threshold number of participants, the network has herd immunity.

In addition, the fact that in a sharded system block *production* (and not just block *verification*) is highly accessible and can be done even on consumer laptops is also very important. The lack of dependence on high-performance hardware at the core of the network ensures that there is a low bar on dissenting minority chains being viable, making it even harder for a majority-driven protocol change to "win by default" and bully everyone else into submission.

This is what auditability usually means in the real world: not that everyone is verifying everything all the time, but that (i) there are enough eyes on each specific piece that if there is an error it will get detected, and (ii) when an error is detected that fact that be made clear and visible to all.

That said, in the long run blockchains can certainly improve on this. One particular source of improvements is ZK-SNARKs (or "validity proofs"): efficiently verifiably cryptographic proofs that allow block producers to prove to clients that blocks satisfy some arbitrarily complex validity conditions. [Validity proofs are stronger than fraud proofs](https://medium.com/starkware/validity-proofs-vs-fraud-proofs-4ef8b4d3d87a) (<https://medium.com/starkware/validity-proofs-vs-fraud-proofs-4ef8b4d3d87a>), because they do not depend on an interactive game to catch fraud. Another important technology is [data availability checks](https://arxiv.org/pdf/1809.09044.pdf) (<https://arxiv.org/pdf/1809.09044.pdf>), which can protect against blocks whose data is not fully published. Data availability checks do rely on a very conservative assumption that there exists at least some small number of honest nodes somewhere in the network continues to apply, though the good news is that this minimum honesty threshold is low, and does not grow even if there is a very large number of attackers.

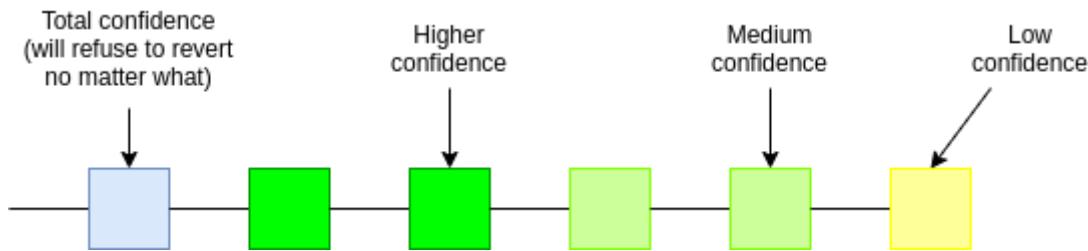
Timing and 51% attacks

Now, let us get to the most powerful consequence of the "default to chaos" mindset: 51% attacks themselves. The current norm in many communities is that if a 51% attack wins, then that 51% attack is necessarily the valid chain. This norm is often stuck to quite strictly; and a recent [51% attack on Ethereum Classic](https://blog.bitquery.io/attacker-stole-807k-etc-in-ethereum-classic-51-attack) (<https://blog.bitquery.io/attacker-stole-807k-etc-in-ethereum-classic-51-attack>) illustrated this quite well. The attacker

reverted more than 3000 blocks (stealing 807,260 ETC in a double-spend in the process), which pushed the chain farther back in history than one of the two ETC clients (OpenEthereum) was technically able to revert; as a result, Geth nodes went with the attacker's chain but OpenEthereum nodes stuck with the original chain.

We can say that the attack did in fact default to chaos, though this was an accident and not a deliberate design decision of the ETC community. Unfortunately, the community then elected to accept the (longer) attack chain as canonical, a move described by the eth classic twitter (https://twitter.com/eth_classic/status/1289637659351031809) as "following Proof of Work as intended". Hence, *the community norms actively helped the attacker win*.

But we could instead agree on a definition of the canonical chain that works differently: particularly, imagine a rule that once a client has accepted a block as part of the canonical chain, and that block has more than 100 descendants, the client will from then on never accept a chain that does not include that block. Alternatively, in a finality-bearing proof of stake setup (which eg. Ethereum 2.0 is), imagine a rule that once a block is finalized it can never be reverted.



5 block revert limit only for illustration purposes; in reality the limit could be something longer like 100-1000 blocks.

To be clear, this introduces a significant change to how canonicalness is determined: instead of clients just looking at the data they receive by itself, clients also look at *when* that data was received. This introduces the possibility that, because of network latency, clients disagree: what if, because of a massive attack, two conflicting blocks A and B finalize at the same time, and some clients see A first and some see B first? But I would argue that this is good: it means that **instead of defaulting to victory, even 51% attacks that just try to revert transactions default to chaos**, and out-of-band emergency response can choose which of the two blocks the chain continues with. If the protocol is well-designed, forcing an escalation to out-of-band emergency response should be very expensive: in proof of stake, such a thing would require 1/3 of validators sacrificing their deposits and getting slashed.

Potentially, we could expand this approach. We could try to make 51% attacks that censor transactions (<https://ethresear.ch/t/censorship-detectors-via-99-fault-tolerant-consensus/2878>) default to chaos too. Research on timeliness detectors (<https://ethresear.ch/t/timeliness-detectors-and-51-attack-recovery-in-blockchains/6925>) pushes things further in the direction of attacks of all types defaulting to failure, though a little chaos remains because timeliness detectors cannot help nodes that are not well-connected and online.

For a blockchain community that values immutability, implementing revert limits of this kind are arguably the superior path to take. It is difficult to honestly claim that a blockchain is immutable when no matter how long a transaction has been accepted in a chain, there is always the possibility that some unexpected activity by powerful actors can come along and revert it. Of course, I would claim that even BTC and ETC do *already* have a revert limit at the extremes; if there was an attack that reverted weeks of activity, the community would likely adopt a user-activated soft fork to reject the attackers' chain. But more definitively agreeing on and formalizing this seems like a step forward.

Conclusion

There are a few "morals of the story" here. First, if we accept the legitimacy of social coordination, and we accept the legitimacy of indirect validation involving "1-of-N" trust models (that is, assuming that there exists one honest person in the network somewhere; NOT the same as assuming that one specific party, eg. Infura, is honest), then we can create blockchains that are much more scalable.

Second, client-side validation is extremely important for all of this to work. A network where only a few people run nodes and everyone else really does trust them is a network that can easily be taken over by special interests. But avoiding such a fate does *not* require going to the opposite extreme and having everyone always validate everything! Systems that allow each individual block to be verified in isolation, so users only validate blocks if someone else raises an alarm, are totally reasonable and serve the same effect. But this requires accepting the "coordination view" of *what validation is for*.

Third, if we allow the definition of canonicalness includes timing, then we open many doors in improving our ability to reject 51% attacks. The easiest property to gain is weak subjectivity

(<https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/>): the idea that if clients are required to log on at least once every eg. 3 months, and refuse to revert longer than that, then we can add slashing to proof of stake and make attacks very expensive. But we can go further: we can reject chains that revert finalized blocks and thereby protect immutability, and even protect against censorship. Because the network is unpredictable, relying on timing does imply attacks "defaulting to chaos" in some cases, but the benefits are very much worth it.

With all of these ideas in mind, we can avoid the traps of (i) over-centralization, (ii) overly redundant verification leading to inefficiency and (iii) misguided norms accidentally making attacks easier, and better work toward building more resilient, performant and secure blockchains.

Trust Models

2020 Aug 20

[See all posts \(/\)](#)

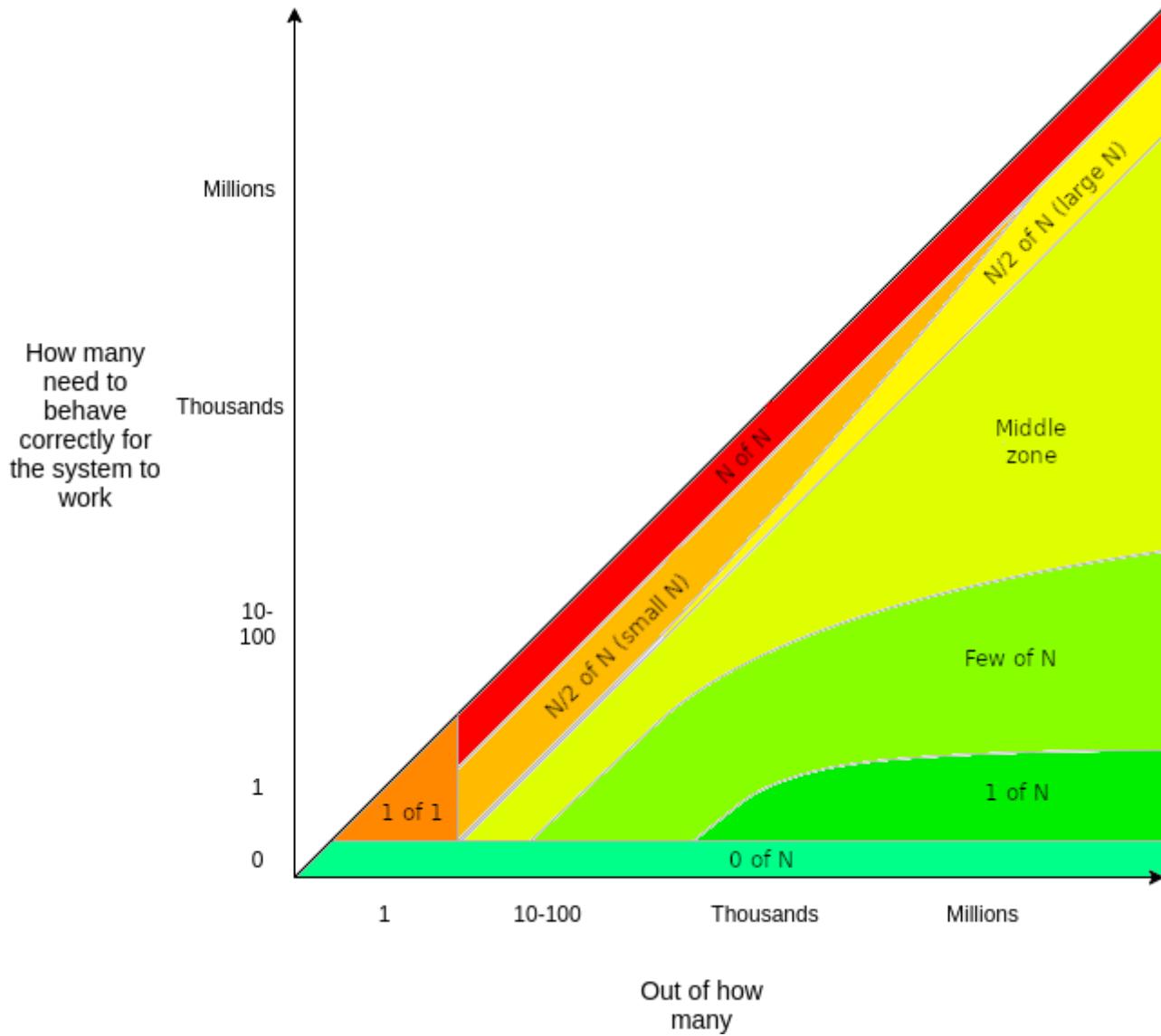
One of the most valuable properties of many blockchain applications is *trustlessness*: the ability of the application to continue operating in an expected way without needing to rely on a specific actor to behave in a specific way even when their interests might change and push them to act in some different unexpected way in the future. Blockchain applications are never *fully* trustless, but some applications are much closer to being trustless than others. If we want to make practical moves toward trust minimization, we want to have the ability to compare different degrees of trust.

First, my simple one-sentence definition of trust: **trust is the use of any assumptions about the behavior of other people**. If before the pandemic you would walk down the street without making sure to keep two meters' distance from strangers so that they could not suddenly take out a knife and stab you, that's a kind of trust: both trust that people are very rarely completely deranged, and trust that the people managing the legal system continue to provide strong incentives against that kind of behavior. When you run a piece of code written by someone else, you trust that they wrote the code honestly (whether due to their own sense of decency or due to an economic interest in maintaining their reputations), or at least that *there exist* enough people checking the code that a bug would be found. Not growing your own food is another kind of trust: trust that enough people will realize that it's in *their* interests to grow food so they can sell it to you. You can trust different sizes of groups of people, and there are different kinds of trust.

For the purposes of analyzing blockchain protocols, I tend to break down trust into four dimensions:

- How many people do you need to behave as you expect?
- Out of how many?
- What kinds of motivations are needed for those people to behave? Do they need to be altruistic, or just profit seeking? Do they need to be uncoordinated (https://vitalik.ca/general/2017/05/08/coordination_problems.html)?
- How badly will the system fail if the assumptions are violated?

For now, let us focus on the first two. We can draw a graph:



The more green, the better. Let us explore the categories in more detail:

- **1 of 1**: there is exactly one actor, and the system works if (and only if) that one actor does what you expect them to. This is the traditional "centralized" model, and it is what we are trying to do better than.
- **N of N**: the "dystopian" world. You rely on a whole bunch of actors, *all* of whom need to act as expected for everything to work, with no backups if any of them fail.
- **N/2 of N**: this is how blockchains work - they work if the majority of the miners (or PoS validators) are honest. Notice that N/2 of N becomes significantly more valuable the larger the N gets; a blockchain with a few miners/validators dominating the network is much less interesting than a blockchain with its miners/validators widely distributed. That said, we want to improve on even this level of security, hence the concern around surviving 51% attacks (<https://vitalik.ca/general/2020/08/17/philosophy.html>).
- **1 of N**: there are many actors, and the system works as long as at least one of them does what you expect them to. Any system based on fraud proofs falls into this category, as do trusted setups though in that case the N is often smaller. Note that you do want the N to be as large as possible!
- **Few of N**: there are many actors, and the system works as long as at least some small fixed number of them do what you expect them to. Data availability checks (<https://arxiv.org/abs/1809.09044>) fall into this category.

- **0 of N:** the system works as expected without any dependence whatsoever on external actors. Validating a block by checking it yourself falls into this category.

While all buckets other than "0 of N" can be considered "trust", they are very different from each other!

Trusting that one particular person (or organization) will work as expected is very different from trusting that *some single person anywhere* will do what you expect them to. "1 of N" is arguably much closer to "0 of N" than it is to "N/2 of N" or "1 of 1". A 1-of-N model might perhaps feel like a 1-of-1 model because it feels like you're going through a single actor, but the reality of the two is *very* different: in a 1-of-N system, if the actor you're working with at the moment disappears or turns evil, you can just switch to another one, whereas in a 1-of-1 system you're screwed.

Particularly, note that even the correctness of the software you're running typically depends on a "few of N" trust model to ensure that if there's bugs in the code someone will catch them. With that fact in mind, trying really hard to go from 1 of N to 0 of N on some other aspect of an application is often like making a reinforced steel door for your house when the windows are open.

Another important distinction is: how does the system fail if your trust assumption is violated? In blockchains, two most common types of failure are **liveness failure** and **safety failure**. A liveness failure is an event in which you are temporarily unable to do something you want to do (eg. withdraw coins, get a transaction included in a block, read information from the blockchain). A safety failure is an event in which something actively happens that the system was meant to prevent (eg. an invalid block gets included in a blockchain).

Here are a few examples of trust models of a few blockchain layer 2 protocols. I use "**small N**" to refer to the set of participants of the layer 2 system itself, and "**big N**" to refer to the participants of the blockchain; the assumption is always that the layer 2 protocol has a smaller community than the blockchain itself. I also limit my use of the word "liveness failure" to cases where coins are stuck for a significant amount of time; no longer being able to use the system but being able to near-instantly withdraw does not count as a liveness failure.

- **Channels** (incl state channels, lightning network): 1 of 1 trust for liveness (your counterparty can temporarily freeze your funds, though the harms of this can be mitigated if you split coins between multiple counterparties), N/2 of big-N trust for safety (a blockchain 51% attack can steal your coins)
- **Plasma** (assuming centralized operator): 1 of 1 trust for liveness (the operator can temporarily freeze your funds), N/2 of big-N trust for safety (blockchain 51% attack)
- **Plasma** (assuming semi-decentralized operator, eg. DPOS): N/2 of small-N trust for liveness, N/2 of big-N trust for safety
- **Optimistic rollup:** 1 of 1 or N/2 of small-N trust for liveness (depends on operator type), N/2 of big-N trust for safety
- **ZK rollup:** 1 of small-N trust for liveness (if the operator fails to include your transaction, you can withdraw, and if the operator fails to include your withdrawal immediately they cannot produce more batches and you can self-withdraw with the help of any full node of the rollup system); no safety failure risks
- **ZK rollup** ([light-withdrawal enhancement](https://ethresear.ch/t/efficient-unassisted-exit-witness-generation-from-rollups/7776) (<https://ethresear.ch/t/efficient-unassisted-exit-witness-generation-from-rollups/7776>)): no liveness failure risks, no safety failure risks

Finally, there is the question of incentives: does the actor you're trusting need to be very altruistic to act as expected, only slightly altruistic, or is being rational enough? Searching for fraud proofs is "by default" slightly altruistic, though just how altruistic it is depends on the complexity of the computation (see [the verifier's dilemma](https://eprint.iacr.org/2015/702.pdf) (<https://eprint.iacr.org/2015/702.pdf>)), and there are ways to modify the game to make it rational.

Assisting others with withdrawing from a ZK rollup is rational if we add a way to micro-pay for the service, so there is *really* little cause for concern that you won't be able to exit from a rollup with any significant use.

Meanwhile, the greater risks of the other systems can be alleviated if we agree as a community.

(<https://vitalik.ca/general/2020/08/17/philosophy.html>) to not accept 51% attack chains

(<https://ethresear.ch/t/timeliness-detectors-and-51-attack-recovery-in-blockchains/6925>) that revert too far in history or censor blocks for too long.

Conclusion: when someone says that a system "depends on trust", ask them in more detail what they mean! Do they mean 1 of 1, or 1 of N, or N/2 of N? Are they demanding these participants be altruistic or just rational? If altruistic, is it a tiny expense or a huge expense? And what if the assumption is violated - do you just need to wait a few hours or days, or do you have assets that are stuck forever? Depending on the answers, your own answer to whether or not you want to use that system might be very different.

Coordination, Good and Bad

2020 Sep 11

[See all posts \(/\)](#)

Special thanks to Karl Floersch and Jinglan Wang for feedback and review

See also:

- [On Collusion](https://vitalik.ca/general/2019/04/03/collusion.html) (<https://vitalik.ca/general/2019/04/03/collusion.html>).
- [Engineering Security Through Coordination Problems](https://vitalik.ca/general/2017/05/08/coordination_problems.html) (https://vitalik.ca/general/2017/05/08/coordination_problems.html).
- [Trust Models](https://vitalik.ca/general/2020/08/20/trust.html) (<https://vitalik.ca/general/2020/08/20/trust.html>).
- [The Meaning Of Decentralization](https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274) (<https://medium.com/@VitalikButerin/the-meaning-of-decentralization-a0c92b76a274>).

Coordination, the ability for large groups of actors to work together for their common interest, is one of the most powerful forces in the universe. It is the difference between a king comfortably ruling a country as an oppressive dictatorship, and the people coming together and overthrowing him. It is the difference between the global temperature going up [3-5°C](https://www.reuters.com/article/us-climate-change-un/global-temperatures-on-track-for-3-5-degree-rise-by-2100-u-n-idUSKCN1NY186) (<https://www.reuters.com/article/us-climate-change-un/global-temperatures-on-track-for-3-5-degree-rise-by-2100-u-n-idUSKCN1NY186>) and the temperature going up by a much smaller amount if we work together to stop it. And it is the factor that makes companies, countries and any social organization larger than a few people possible at all.

Coordination can be improved in many ways: faster spread of information, better norms that identify what behaviors are classified as cheating along with more effective punishments, stronger and more powerful organizations, tools like smart contracts that allow interactions with reduced levels of trust, governance technologies (voting, shares, decision markets...), and much more. And indeed, we as a species are getting better at all of these things with each passing decade.

But there is also a very philosophically counterintuitive dark side to coordination. **While it is emphatically true that "everyone coordinating with everyone" leads to much better outcomes than "every man for himself", what that does NOT imply is that each individual step toward more coordination is necessarily beneficial.** If coordination is improved in an unbalanced way, the results can easily be harmful.

We can think about this visually as a map, though in reality the map has many billions of "dimensions" rather than two:

More coordination between A and B

Unbalanced coordination (oppression, coordinated attacks...)

Total coordination ("Utopia")

Realistically achievable middle

Every man for himself ("Hobbesian jungle")

Unbalanced coordination (oppression, coordinated attacks)

More coordination between A and C

The bottom-left corner, "every man for himself", is where we don't want to be. The top-right corner, total coordination, is ideal, but likely unachievable. But the landscape in the middle is far from an even slope up, with many reasonably safe and productive places that it might be best to settle down in and many deep dark caves to avoid.

Now what are these dangerous forms of partial coordination, where someone coordinating with some fellow humans but not others leads to a deep dark hole? It's best to describe them by giving examples:

- Citizens of a nation valiantly sacrificing themselves for the greater good of their country in a war.... when that country turns out to be WW2-era Germany or Japan
- A lobbyist giving a politician a bribe in exchange for that politician adopting the lobbyist's preferred policies
- Someone selling their vote in an election

- All sellers of a product in a market colluding to raise their prices at the same time
- Large miners of a blockchain colluding to launch a 51% attack

In all of the above cases, we see a group of people coming together and cooperating with each other, but to the great detriment of some group that is outside the circle of coordination, and thus to the net detriment of the world as a whole. In the first case, it's all the people that were the victims of the aforementioned nations' aggression that are outside the circle of coordination and suffer heavily as a result; in the second and third cases, it's the people affected by the decisions that the corrupted voter and politician are making, in the fourth case it's the customers, and in the fifth case it's the non-participating miners and the blockchain's users. It's not an individual defecting against the group, it's a group defecting against a broader group, often the world as a whole.

This type of partial coordination is often called "collusion", but it's important to note that the range of behaviors that we are talking about is quite broad. In normal speech, the word "collusion" tends to be used more often to describe relatively symmetrical relationships, but in the above cases there are plenty of examples with a strong asymmetric character. Even extortionate relationships ("vote for my preferred policies or I'll publicly reveal your affair") are a form of collusion in this sense. In the rest of this post, we'll use "collusion" to refer to "undesired coordination" generally.

Evaluate Intentions, Not Actions (!!)

One important property of especially the milder cases of collusion is that one cannot determine whether or not an action is part of an undesired collusion just by looking at the action itself. The reason is that the actions that a person takes are a combination of that person's internal knowledge, goals and preferences together with externally imposed incentives on that person, and so the actions that people take when colluding, versus the actions that people take on their own volition (or coordinating in benign ways) often overlap.

For example, consider the case of collusion between sellers (a type of antitrust violation (<https://en.wikipedia.org/wiki/Antitrust>)). If operating independently, each of three sellers might set a price for some product between \$5 and \$10; the differences within the range reflect difficult-to-see factors such as the seller's internal costs, their own willingness to work at different wages, supply-chain issues and the like. But if the sellers collude, they might set a price between \$8 and \$13. Once again, the range reflects different possibilities regarding internal costs and other difficult-to-see factors. If you see someone selling that product for \$8.75, are they doing something wrong? Without knowing whether or not they coordinated with other sellers, you can't tell! Making a law that says that selling that product for more than \$8 would be a bad idea; maybe there are legitimate reasons why prices have to be high at the current time. But making a law against collusion, and successfully enforcing it, gives the ideal outcome - you get the \$8.75 price if the price has to be that high to cover sellers' costs, but you don't get that price if the factors driving prices up naturally are low.

This applies in the bribery and vote selling cases too: it may well be the case that some people vote for the Orange Party legitimately, but others vote for the Orange Party because they were paid to. From the point of view of someone determining the rules for the voting mechanism, they don't know ahead of time whether the Orange Party is good or bad. But what they do know is that a vote where people vote based on their honest internal feelings works reasonably well, but a vote where voters can freely buy and sell their votes works terribly. This is because vote selling has a tragedy-of-the-commons: each voter only gains a small portion of the benefit from voting correctly, but would gain the full bribe if they vote the way the briber wants, and so the required bribe to lure each individual voter is far smaller than the bribe that would actually compensate the population for the costs of whatever policy the briber wants. Hence, votes where vote selling is permitted quickly collapse into plutocracy (<https://vitalik.ca/general/2019/04/03/collusion.html>).

Understanding the Game Theory

We can zoom further out and look at this from the perspective of game theory. In the version of game theory that focuses on individual choice - that is, the version that assumes that each participant makes decisions independently and that does not allow for the possibility of groups of agents working as one for their mutual benefit, there are mathematical proofs (https://en.wikipedia.org/wiki/Nash_equilibrium#Proof_of_existence) that at least one stable Nash equilibrium must exist in any game. In fact, mechanism designers have a very wide latitude to "engineer" games (https://en.wikipedia.org/wiki/Mechanism_design) to achieve specific outcomes. But in the version of game theory that allows for the possibility of coalitions working together (ie. "colluding"), called *cooperative game theory*, we can prove that (https://en.wikipedia.org/wiki/Bondareva%20%93Shapley_theorem), there are large classes of games that do not have any stable outcome (called a "core" ([https://en.wikipedia.org/wiki/Core_\(game_theory\)](https://en.wikipedia.org/wiki/Core_(game_theory)))). In such games, whatever the current state of affairs is, there is always some coalition that can profitably deviate from it.

One important part of that set of inherently unstable games is *majority games*. A majority game is formally described (<https://web.archive.org/web/20180329012328/https://www.math.mcgill.ca/vetta/CS764.dir/Core.pdf>) as a game of agents where any subset of more than half of them can capture a fixed reward and split it among themselves - a setup eerily similar to many situations in corporate governance, politics and many other situations in human life. That is to say, if there is a situation with some fixed pool of resources and some currently established mechanism for distributing those resources, and it's unavoidably possible for 51% of the participants can conspire to seize control of the resources, no matter what the current configuration is there is always some conspiracy that can emerge that would be profitable for the participants. However, that conspiracy would then in turn be vulnerable to potential new conspiracies, possibly including a combination of previous conspirators and victims... and so on and so forth.

Round	A	B	C
1	1/3	1/3	1/3
2	1/2	1/2	0
3	2/3	0	1/3
4	0	1/3	2/3

This fact, the instability of majority games under cooperative game theory, is arguably highly underrated as a simplified general mathematical model of why there may well be no "end of history" in politics and no system that proves fully satisfactory; I personally believe it's much more useful than the more famous Arrow's theorem (https://en.wikipedia.org/wiki/Arrow%27s_impossibility_theorem), for example.

Note once again that the core dichotomy here is not "individual versus group"; for a mechanism designer, "individual versus group" is surprisingly easy to handle. It's "group versus broader group" that presents the challenge.

Decentralization as Anti-Collusion

But there is another, brighter and more actionable, conclusion from this line of thinking: if we want to create mechanisms that are stable, then we know that one important ingredient in doing so is finding ways to make it more difficult for collusions, especially large-scale collusions, to happen and to maintain themselves. In the

case of voting, we have the [secret ballot](https://en.wikipedia.org/wiki/Secret_ballot) (https://en.wikipedia.org/wiki/Secret_ballot) - a mechanism that ensures that voters have no way to prove to third parties how they voted, even if they want to prove it ([MACI](https://github.com/appliedzkp/mac) (<https://github.com/appliedzkp/mac>)) is one project trying to use cryptography to extend secret-ballot principles to an online context). This disrupts trust between voters and bribers, heavily restricting undesired collusions that can happen. In that case of antitrust and other corporate malfeasance, we often rely on whistleblowers and even [give them rewards](https://www.mahanyertl.com/2019/antitrust-whistleblower-reward/) (<https://www.mahanyertl.com/2019/antitrust-whistleblower-reward/>), explicitly incentivizing participants in a harmful collusion to defect. And in the case of public infrastructure more broadly, we have that oh-so-important concept: **decentralization**.

One naive view of why decentralization is valuable is that it's about reducing risk from single points of technical failure. In traditional "enterprise" distributed systems, this is often actually true, but in many other cases we know that this is not sufficient to explain what's going on. It's instructive here to look at blockchains. A large mining pool publicly showing how they have internally distributed their nodes and network dependencies doesn't do much to calm community members scared of mining centralization. And pictures like these, showing 90% of Bitcoin hashpower at the time being capable of showing up to the same conference panel, do quite a bit to scare people:



But why is this image scary? From a "decentralization as fault tolerance" view, large miners being able to talk to each other causes no harm. But if we look at "decentralization" as being the presence of barriers to harmful collusion, then the picture becomes quite scary, because it shows that those barriers are not nearly as strong as we thought. Now, in reality, the barriers are still far from zero; the fact that those miners can easily perform technical coordination and likely are all in the same Wechat groups does *not*, in fact, mean that Bitcoin is "in practice little better than a centralized company".

So what are the remaining barriers to collusion? Some major ones include:

- **Moral Barriers.** In [Liars and Outliers](https://books.google.com.sg/books/about/Liars_and_Outliers.html?id=IPsbhIUexoOC&redir_esc=y) (https://books.google.com.sg/books/about/Liars_and_Outliers.html?id=IPsbhIUexoOC&redir_esc=y), Bruce Schneier reminds us that many "security systems" (locks on doors, warning signs reminding people of punishments...) also serve a moral function, reminding potential misbehavers that they are about to conduct a serious transgression and if they want to be a good person they should not do that. Decentralization arguably serves that function.
- **Internal negotiation failure.** The individual companies may start demanding concessions in exchange for participating in the collusion, and this could lead to negotiation stalling outright (see "[holdout problems](https://en.wikipedia.org/wiki/Holdout_problem)" (https://en.wikipedia.org/wiki/Holdout_problem) in economics).
- **Counter-coordination.** The fact that a system is decentralized makes it easy for participants not participating in the collusion to make a fork that strips out the colluding attackers and continue the system from there. Barriers for users to join the fork are low, and the *intention* of decentralization creates moral pressure in favor of participating in the fork.
- **Risk of defection.** It still is much harder for five companies to join together to do something widely considered to be bad than it is for them to join together for a non-controversial or benign purpose. The five companies do not know each other *too well*, so there is a risk that one of them will refuse to participate and blow the whistle quickly, and the participants have a hard time judging the risk. Individual employees within the companies may blow the whistle too.

Taken together, these barriers are substantial indeed - often substantial enough to stop potential attacks in their tracks, even when those five companies are simultaneously perfectly capable of quickly coordinating to do something legitimate. Ethereum blockchain miners, for example, are perfectly capable of coordinating [increases to the gas limit](https://etherscan.io/chart/gaslimit) (<https://etherscan.io/chart/gaslimit>), but that does not mean that they can so easily collude to attack the chain.

The blockchain experience shows how designing protocols as institutionally decentralized architectures, even when it's well-known ahead of time that the bulk of the activity will be dominated by a few companies, can often be a very valuable thing. This idea is not limited to blockchains; it can be applied in other contexts as well (eg. see [here](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3597399) (https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3597399) for applications to antitrust).

Forking as Counter-Coordination

But we cannot always effectively prevent harmful collusions from taking place. And to handle those cases where a harmful collusion does take place, it would be nice to make systems that are more robust against them - more expensive for those colluding, and easier to recover for the system.

There are two core operating principles that we can use to achieve this end: (1) **supporting counter-coordination** and (2) **skin-in-the-game**. The idea behind counter-coordination is this: we know that we cannot design systems to be *passively* robust to collusions, in large part because there is an extremely large number of ways to organize a collusion and there is no passive mechanism that can detect them, but what we can do is *actively* respond to collusions and strike back.

In digital systems such as blockchains (this could also be applied to more mainstream systems, eg. DNS), a major and crucially important form of counter-coordination is **forking**.



If a system gets taken over by a harmful coalition, the dissidents can come together and create an alternative version of the system, which has (mostly) the same rules except that it removes the power of the attacking coalition to control the system. Forking is very easy in an open-source software context; the main challenge in creating a successful fork is usually gathering the **legitimacy** (game-theoretically viewed as a form of "[common knowledge](https://en.wikipedia.org/wiki/Common_knowledge_(logic)) ([https://en.wikipedia.org/wiki/Common_knowledge_\(logic\)](https://en.wikipedia.org/wiki/Common_knowledge_(logic))))") needed to get all those who disagree with the main coalition's direction to follow along with you.

This is not just theory; it has been accomplished successfully, most notably in the [Steem community's rebellion](https://decrypt.co/38050/steem-steemit-tron-justin-sun-cryptocurrency-war) (<https://decrypt.co/38050/steem-steemit-tron-justin-sun-cryptocurrency-war>) against a hostile takeover attempt, leading to a new blockchain called Hive in which the original antagonists have no power.

Markets and Skin in the Game

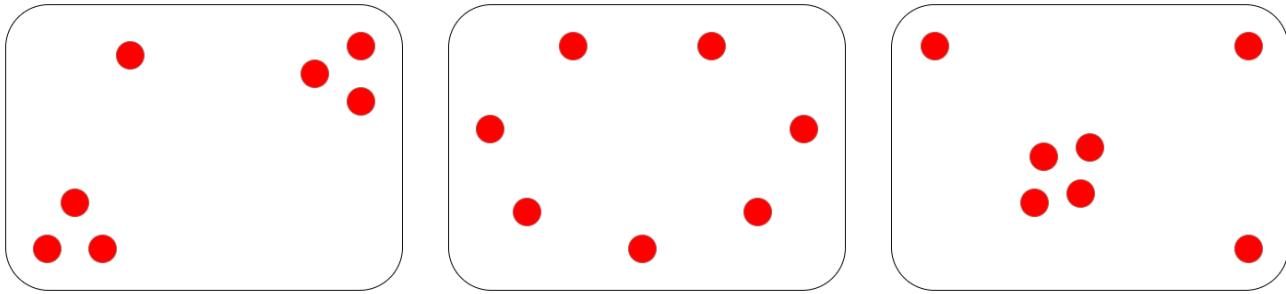
Another class of collusion-resistance strategy is the idea of **skin in the game**. Skin in the game, in this context, basically means any mechanism that holds individual contributors in a decision individually accountable for their contributions. If a group makes a bad decision, those who approved the decision must suffer more than those who attempted to dissent. This avoids the "tragedy of the commons" inherent in voting systems.

Forking is a powerful form of counter-coordination precisely because it introduces skin in the game. In Hive, the community fork of Steem that threw off the hostile takeover attempt, the coins that were used to vote in favor of the hostile takeover were largely deleted in the new fork. The key individuals who participated in the attack individually suffered as a result.

Markets are in general very powerful tools precisely because they maximize skin in the game. **Decision markets** (https://en.wikipedia.org/wiki/Prediction_market), used to guide decisions; also called **futarchy** (<https://blog.ethereum.org/2014/08/21/introduction-futarchy/>) are an attempt to extend this benefit of markets to organizational decision-making. That said, decision markets can only solve some problems; in particular, they cannot tell us what variables we should be optimizing for in the first place.

Structuring Coordination

This all leads us to an interesting view of what it is that people building social systems do. One of the goals of building an effective social system is, in large part, determining *the structure of coordination*: which groups of people and in what configurations can come together to further their group goals, and which groups cannot?



Different coordination structures, different outcomes

Sometimes, more coordination is good: it's better when people can work together to collectively solve their problems. At other times, more coordination is dangerous: a subset of participants could coordinate to disenfranchise everyone else. And at still other times, more coordination is necessary for another reason: to enable the broader community to "strike back" against a collusion attacking the system.

In all three of those cases, there are different mechanisms that can be used to achieve these ends. Of course, it is very difficult to prevent communication outright, and it is very difficult to make coordination perfect. But there are many options in between that can nevertheless have powerful effects.

Here are a few possible coordination structuring techniques:

- Technologies and norms that protect privacy
- Technological means that make it difficult to prove how you behaved (secret ballots, MACI and similar tech)
- Deliberate decentralization, distributing control of some mechanism to a wide group of people that are known to not be well-coordinated
- Decentralization in physical space, separating out different functions (or different shares of the same function) to different locations (eg. see [Samo Burja on connections between urban decentralization and political decentralization](#) (<https://www.city-journal.org/urban-politics-shape-national-politics>))
- Decentralization between role-based constituencies, separating out different functions (or different shares of the same function) to different types of participants (eg. in a blockchain: "core developers", "miners", "coin holders", "application developers", "users")
- [Schelling points](#) ([https://en.wikipedia.org/wiki/Focal_point_\(game_theory\)](https://en.wikipedia.org/wiki/Focal_point_(game_theory))), allowing large groups of people to quickly coordinate around a single path forward. Complex Schelling points could potentially even be implemented in code (eg. [recovery from 51% attacks](#) (<https://ethresear.ch/t/timeliness-detectors-and-51-attack-recovery-in-blockchains/6925>) can benefit from this).

- Speaking a common language (or alternatively, splitting control between multiple constituencies who speak different languages)
- Using per-person voting instead of per-(coin/share) voting to greatly increase the number of people who would need to collude to affect a decision
- Encouraging and relying on defectors to alert the public about upcoming collusions

None of these strategies are perfect, but they can be used in various contexts with differing levels of success. Additionally, these techniques can and should be combined with mechanism design that attempts to make harmful collusions less profitable and more risky to the extent possible; skin in the game is a very powerful tool in this regard. Which combination works best ultimately depends on your specific use case.

Gitcoin Grants Round 7 Retrospective

2020 Oct 18

[See all posts \(/\)](#)

Round 7 of Gitcoin Grants has successfully completed! This round has seen an unprecedented growth in interest and contributions, with \$274,830 in contributions and \$450,000 in matched funds distributed across 857 projects.

The category structure was once again changed; this time was had a split between "dapp tech", "infrastructure tech" and "community". Here are the results:

CLR MATCHING ROUND 7 dApp tech Grants

	NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1 BrightID	291	\$9,102	\$26,204
2 Rotki	230	\$6,967	\$21,721
3 EPNS-Ethereum Push Notification Service	214	\$1,914	\$14,154
4 Snapshot	187	\$6,362	\$11,055
5 Yearn.finance	185	\$5,893	\$10,935
6 Gitcoin Grants Round 8 + Dev Fund	1044	\$7,703	\$9,576
7 POAP (Proof of Attendance Protocol)	173	\$3,266	\$8,078
8 The Commons Stack	137	\$1,503	\$4,462
9 YieldFarming.info	117	\$3,046	\$3,774
10 DeFi Saver	105	\$2,993	\$3,387

CLR MATCHING ROUND 7 Infra Grants

	NUMBER OF CONTRIBUTIONS	TOTAL CONTRIBUTED	CLR MATCHING
1 DAppNode - Panvala League	217	\$5,514	\$22,089
2 White Hat Hacking	199	\$17,462	\$21,796
3 Turbo-Geth	181	\$12,383	\$14,794
4 WalletConnect	169	\$6,462	\$13,313
5 ArchiveNode.io - The Public Access Ethereum Archive Node	160	\$9,486	\$10,767
6 Fuel Labs	147	\$8,273	\$8,838
7 Ethereum on ARM	136	\$2,676	\$6,350
8 Nethermind	116	\$4,249	\$5,054
9 Create Eth App	98	\$1,275	\$4,138
10 Ethereum Swarm	100	\$888	\$3,763



Defi joins the matching!

In this round, we were able to have much higher matching values than before. This was because the usual matchings, provided by the Ethereum Foundation and a few other actors, were supplemented for the first time by a high level of participation from various defi projects:



The matchers were:

- [Chainlink](https://chain.link/), a smart contract oracle project
- [Optimism](https://optimism.io/), a layer-2 optimistic rollup
- The [Ethereum Foundation](http://ethereum.org)
- [Balancer](https://balancer.exchange/), a decentralized exchange
- [Synthetix](https://synthetix.io/), a synthetic assets platform
- [Yearn](https://yearn.finance/), a collateralized-lending platform
- [Three Arrows Capital](https://www.threearrowscap.com/about-us/), an investment fund
- [Defiance Capital](https://twitter.com/defiancecapital), another investment fund
- [Future Fund](https://twitter.com/future_fund_), which is totally not an investment fund! (/s)
- \$MEME, a memecoin
- [Yam](https://yam.finance/), a defi project

- Some individual contributors: [ferretpatrol, bantg_](https://twitter.com/bantg/)(<https://twitter.com/bantg/>), [Mariano Conti](https://twitter.com/nanexcool/)(<https://twitter.com/nanexcool/>), [Robert Leshner](https://twitter.com/rleshner/)(<https://twitter.com/rleshner/>), [Eric Conner](https://twitter.com/econoar/)(<https://twitter.com/econoar/>), [10b576da0](https://twitter.com/10b576da0)(<https://twitter.com/10b57e6da0>).

The projects together contributed a large amount of matching funding, some of which was used this round and some of which is reserved as a "rainy day fund" for future rounds in case future matchers are less forthcoming.

This is a significant milestone for the ecosystem because it shows that Gitcoin Grants is expanding beyond reliance on a very small number of funders, and is moving toward something more sustainable. But it is worth exploring, what exactly is driving these matchers to contribute, and is it sustainable?

There are a few possible motivations that are likely all in play to various extents:

1. People are naturally altruistic to some extent, and this round defi projects got unexpectedly wealthy for the first time due to a rapid rise in interest and token prices, and so donating some of that windfall felt like a natural "good thing to do"
2. Many in the community are critical of defi projects by default, viewing them as unproductive casinos that create a negative image of what Ethereum is supposed to be about. Contributing to public goods is an easy way for a defi project to show that they want to be a positive contributor to the ecosystem and make it better
3. Even in the absence of such negative perceptions, defi is a competitive market that is heavily dependent on community support and network effects, and so it's very valuable to a project to win friends in the ecosystem
4. The largest defi projects capture enough of the benefit from these public goods that it's in their own interest to contribute
5. There's a high degree of common-ownership between defi projects (holders of one token also hold other tokens and hold ETH), and so even if it's not strictly in a *project's* interest to donate a large amount, *token holders of that project* push the project to contribute because they as holders benefit from the gains to both that project but also to the other projects whose tokens they hold.

The remaining question is, of course: how sustainable will these incentives be? Are the altruistic and public-relations incentives only large enough for a one-time burst of donations of this size, or could it become more sustainable? Could we reliably expect to see, say, \$2-3 million per year spent on quadratic funding matching from here on? If so, it would be excellent news for public goods funding diversification and democratization in the Ethereum ecosystem.

Where did the troublemakers go?

One curious result from the previous round and this round is that the "controversial" community grant recipients from previous rounds seem to have dropped in prominence on their own. In theory, we should have seen them continue to get support from their supporters with their detractors being able to do nothing about it. In practice, though, the top media recipients this round appear to be relatively uncontroversial and universally beloved mainstays of the Ethereum ecosystem. Even the [Zero Knowledge Podcast](https://www.zeroknowledge.fm/)(<https://www.zeroknowledge.fm/>), an excellent podcast but one aimed for a relatively smaller and more highly technical audience, has received a large contribution this round.

What happened? Why did the distribution of media recipients improve in quality all on its own? Is the mechanism perhaps more self-correcting than we had thought?

Overpayment

This round is the first round where top recipients on all sides received quite a large amount. On the infrastructure side, the White Hat Hacking project (basically a fund to donate to [samczsun](https://samczsun.com/) (<https://samczsun.com/>)) received a total of \$39,258, and the [Bankless podcast](http://podcast.banklesshq.com/) (<http://podcast.banklesshq.com/>) got \$47,620. We could ask the question: are the top recipients getting too much funding?

To be clear, I do think that it's very improper to try to create a moral norm that public goods contributors should only be earning salaries up to a certain level and should not be able to earn much more than that. People launching coins earn huge windfalls all the time; it is completely natural and fair for public goods contributors to also get that possibility (and furthermore, the numbers from this round translate to about ~\$200,000 per year, which is not even that high).

However, one can ask a more limited and pragmatic question: given the current reward structure, is putting an extra \$1 into the hands of a top contributor *less valuable* than putting \$1 into the hands of one of the other very valuable projects that's still underfunded? Turbogeth, Nethermind, RadicalXChange and many other projects could still do quite a lot with a marginal dollar. For the first time, the matching amounts are high enough that this is actually a significant issue.

Especially if matching amounts increase even further, is the ecosystem going to be able to correctly allocate funds and avoid overfunding projects? Alternatively, if it fails to avoid over-concentrating funds, is that all that bad? Perhaps the possibility of becoming the center of attention for one round and earning a \$500,000 windfall will be part of the incentive that motivates independent public goods contributors!

We don't know; but these are the yet-unknown facts that running the experiment at its new increased scale is for the first time going to reveal.

Let's talk about categories...

The concept of categories as it is currently implemented in Gitcoin Grants is a somewhat strange one. Each category has a fixed total matching amount that is split between projects within that category. What this mechanism basically says is that the community can be trusted to choose between projects *within* a category, but we need a separate technocratic judgement to judge how the funds are split between the different categories in the first place.

But it gets more paradoxical from here. In Round 7, a "collections" feature (<https://gitcoin.co/grants/collections>) was introduced halfway through the round:

**Ethereum State Media**

owocki · 12 grants

A collection of grants that are putting out the good word about Ethereum + fighting disinformation.

Add to Cart

Share Collection

**Sassel's Cart of Love**

sassal · 42 grants

Add to Cart

Share Collection

**Blockchain Education & Community**

1stcryptoking · 24 grants

Add to Cart

Share Collection

**DeFi Locals**

lemoras · 8 grants

If you want global adaptation, please support local/Non-English source of informations.

Add to Cart

Share Collection

**Full Self-Sovereign Life**

tze42 · 9 grants

This short collection combines tools that are needed to share value among your peers in a fully decentralized fashion.

Add to Cart

Share Collection

**Privacy**

ceresstation · 7 grants

Privacy isn't needed only when you have something to hide in the same way that free speech isn't needed only when you have something to say

Add to Cart

Share Collection

If you click "Add to Cart" on a collection, you immediately add everything in the collection to your cart. This is strange because this mechanism seems to send the exact *opposite* message: users that don't understand the details well can choose to allocate funds to entire categories, but (unless they manually edit the amounts) they should not be making many active decisions *within* each category.

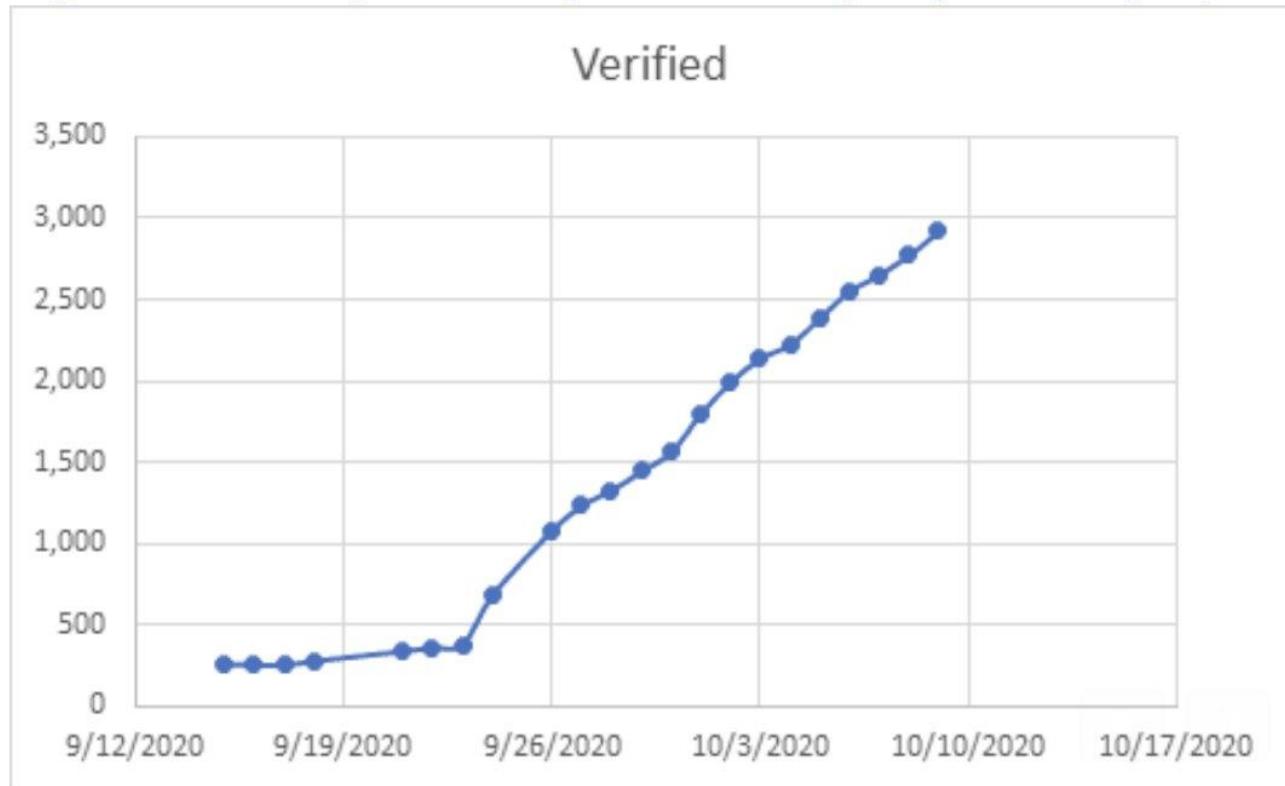
Which is it? Do we trust the radical quadratic fancy democracy to allocate within categories but not between them, do we trust it to allocate between categories but nudge people away from making fine-grained decisions within them, or something else entirely? I recommend that for Round 8 we think harder about the philosophical challenges here and come up with a more principled approach.

One option would be to have one matching pool and have *all* the categories just be a voluntary UI layer. Another would be to experiment with even *more* "affirmative action" to bootstrap particular categories: for example, we could split the Community matching into a \$25,000 matching pool for each major world region (eg. North America + Oceania, Latin America, Europe, Africa, Middle East, India, East + Southeast Asia) to try to give projects in more neglected areas a leg up. There are many possibilities here! One hybrid route is that the "focused" pools could *themselves* be quadratic funded in the previous round!

Identity verification

As collusion, fake accounts and other attacks on Gitcoin Grants have been recently increasing, Round 7 added an additional verification option with the decentralized social-graph-based [BrightID](https://www.brightid.org/) (<https://www.brightid.org/>), and single-handedly boosted the project's userbase by a factor of ten:

BrightID continues to grow like crazy thanks to the trajectory Gitcoin helped put us on.



This is good, because along with helping BrightID's growth, it also subjects the project to a trial-by-fire: there's now a large incentive to try to create a large number of fake accounts on it! BrightID is going to face a tough challenge making it reasonably easy for regular users to join but at the same time resist attacks from fake and duplicate accounts. I look forward to seeing them try to meet the challenge!

ZK rollups for scalability

Finally, Round 7 was the first round where Gitcoin Grants experimented with using the [ZkSync](https://wallet.zksync.io/) (<https://wallet.zksync.io/>) ZK rollup to decrease fees for payments:

The screenshot shows the payment interface for Gitcoin Grants Round 7. On the left, there's a summary of the total contribution: **TOTAL 0.11 ETH**. Below this, there's an option to **Hide my wallet address** (unchecked). A note states: "By using this service, you'll also be contributing 5% of your contribution". There's a link to "Adjust >>".

SUMMARY

- You are contributing **0.10 ETH**
- You are additionally contributing **0.0055 ETH** to the Gitcoin Grants Fund
- Note: The exact checkout flow will depend on whether you choose to hide your wallet address.

Save **-96%** on gas costs with zkSync

Two buttons are available: **Checkout with zkSync** (highlighted in blue) and **Standard Checkout**.

Pay with zkSync

zkSync is powered by zkRollup with a universal setup — an L2 scaling solution. Save gas fees and get faster confirmations. [Learn more](#).

To pay with zkSync, you will have to sign up to 4 messages with your Web3 wallet.

Step 1
Sign in to zkSync via Gitcoin [Sign in](#)

Step 2
Sign in to zkSync directly [Sign in](#)

Step 3
Confirm donation transaction [Confirm](#)

The main thing to report here is simply that the ZK rollup successfully did decrease fees! The user experience worked well. Many optimistic and ZK rollup projects are now looking at [collaborating with wallets](#) (<https://www.theblockcrypto.com/linked/80744/coinbase-wallet-optimisms-layer-2-rollup>) on direct integrations, which should increase the usability and security of such techniques further.

Conclusions

Round 7 has been a pivotal round for Gitcoin Grants. The matching funding has become much more sustainable. The levels of funding are now large enough to successfully fund quadratic freelancers to the point where a project getting "too much funding" is a conceivable thing to worry about! Identity verification is taking steps forward. Payments have become much more efficient with the introduction of the [ZkSync](#) (<https://wallet.zksync.io/>) ZK rollup. I look forward to seeing the grants continue for many more rounds in the future.

Why Proof of Stake (Nov 2020)

2020 Nov 06

[See all posts \(/\)](#)

There are three key reasons why PoS is a superior blockchain security mechanism compared to PoW.

PoS offers more security for the same cost

The easiest way to see this is to put proof of stake and proof of work side by side, and **look at how much it costs to attack a network per \$1 per day in block rewards.**

GPU-based proof of work

You can rent GPUs cheaply, so the cost of attacking the network is simply the cost of renting enough GPU power to outrun the existing miners. For every \$1 of block rewards, the existing miners should be spending close to \$1 in costs (if they're spending more, miners will drop out due to being unprofitable, if they're spending less, new miners can join in and take high profits). Hence, attacking the network just requires temporarily spending more than \$1 per day, and only for a few hours.

Total cost of attack: ~\$0.26 (assuming 6-hour attack), potentially reduced to zero as the attacker receives block rewards

ASIC-based proof of work

ASICs are a capital cost: you buy an ASIC once and you can expect it to be useful for ~2 years before it wears out and/or is obsoleted by newer and better hardware. If a chain gets 51% attacked, the community will likely respond by changing the PoW algorithm and your ASIC will lose its value. On average, mining is ~1/3 ongoing costs and ~2/3 capital costs (see [here](https://eth.wiki/concepts/proof-of-stake-faqs#what-about-capital-lockup-costs) (<https://eth.wiki/concepts/proof-of-stake-faqs#what-about-capital-lockup-costs>) for some sources). Hence, per \$1 per day in reward, miners will be spending ~\$0.33 per day on electricity+maintenance and ~\$0.67 per day on their ASIC. Assuming an ASIC lasts ~2 years, that's \$486.67 that a miner would need to spend on that quantity of ASIC hardware.

Total cost of attack: \$486.67 (ASICs) + \$0.08 (electricity+maintenance) = \$486.75

That said, it's worth noting that ASICs provide this heightened level of security against attacks at a high cost of centralization, as the barriers to entry to joining become very high (<https://blog.ethereum.org/2014/06/19/mining/>).

Proof of stake

Proof of stake is almost entirely capital costs (the coins being deposited); the only operating costs are the cost of running a node. Now, how much capital are people willing to lock up to get \$1 per day of rewards? **Unlike ASICs, deposited coins do not depreciate, and when you're done staking you get your coins back after a short delay. Hence, participants should be willing to pay much higher capital costs for the same quantity of rewards.**

Let's assume that a ~15% rate of return is enough to motivate people to stake (that is the expected eth2 rate of return). Then, \$1 per day of rewards will attract 6.667 years' worth of returns in deposits, or \$2433. Hardware and electricity costs of a node are small; a thousand-dollar computer can stake for hundreds of thousands of dollars in deposits, and ~\$100 per month in electricity and internet is sufficient for such an amount. But conservatively, we can say these ongoing costs are ~10% of the total cost of staking, so we only have \$0.90 per day of rewards that end up corresponding to capital costs, so we do need to cut the above figure by ~10%.

Total cost of attack: \$0.90/day * 6.667 years = \$2189

In the long run, this cost is expected to go even higher, as staking becomes more efficient and people become comfortable with lower rates of return. I personally expect this number to eventually rise to something like \$10000.

Note that the only "cost" being incurred to get this high level of security is just the inconvenience of not being able to move your coins around at will while you are staking. It may even be the case that the public knowledge that all these coins are locked up causes the value of the coin to rise, so the total amount of money floating around in the community, ready to make productive investments etc, remains the same! Whereas **in PoW, the "cost" of maintaining consensus is real electricity being burned in insanely large quantities** (<https://www.theverge.com/2019/7/4/20682109/bitcoin-energy-consumption-annual-calculation-cambridge-index-cbeci-country-comparison>).

Higher security or lower costs?

Note that there are two ways to use this 5-20x gain in security-per-cost. One is to keep block rewards the same but benefit from increased security. The other is to massively reduce block rewards (and hence the "waste" of the consensus mechanism) and keep the security level the same.

Either way is okay. I personally prefer the latter, because as we will see below, in proof of stake even a successful attack is much less harmful and much easier to recover from than an attack on proof of work!

Attacks are much easier to recover from in proof of stake

In a proof of work system, if your chain gets 51% attacked, what do you even do? So far, the only response in practice has been "wait it out until the attacker gets bored". But this misses the possibility of a much more dangerous kind of attack called a **spawn camping attack**, where the attacker attacks the chain over and over again with the explicit goal of rendering it useless.

In a GPU-based system, there is no defense, and a persistent attacker may quite easily render a chain permanently useless (or more realistically, switches to proof of stake or proof of authority). In fact, after the first few days, the attacker's costs may become very low, as honest miners will drop out since they have no way to get rewards while the attack is going on.

In an ASIC-based system, the community can respond to the first attack, but continuing the attack from there once again becomes trivial. The community would meet the first attack by hard-forking to change the PoW algorithm, thereby "bricking" all ASICs (the attacker's *and* honest miners'!). But if the attacker is willing to suffer that initial expense, after that point the situation reverts to the GPU case (as there is not enough time to build and distribute ASICs for the new algorithm), and so from there the attacker can cheaply continue the spawn camp inevitably.

In the PoS case, however, things are much brighter. For certain kinds of 51% attacks (particularly, reverting finalized blocks), there is a built-in "slashing" mechanism in [the proof of stake consensus](#) (<https://arxiv.org/abs/1710.09437>) by which a large portion of the attacker's stake (and no one else's stake) can get automatically destroyed. For other, harder-to-detect attacks (notably, a 51% coalition censoring everyone else), the community can coordinate on a **minority user-activated soft fork (UASF)** in which the attacker's funds are once again largely destroyed (in Ethereum, this is done via the "inactivity leak mechanism"). **No explicit "hard fork to delete coins" is required; with the exception of the requirement to coordinate on the UASF to select a minority block, everything else is automated and simply following the execution of the protocol rules.**

Hence, attacking the chain the first time will cost the attacker many millions of dollars, and the community will be back on their feet within days. Attacking the chain the second time will still cost the attacker many millions of dollars, as they would need to buy new coins to replace their old coins that were burned. And the third time will... cost even more millions of dollars. **The game is very asymmetric, and not in the attacker's favor.**

Proof of stake is more decentralized than ASICs

GPU-based proof of work is reasonably decentralized; it is not too hard to get a GPU. But GPU-based mining largely fails on the "security against attacks" criterion that we mentioned above. ASIC-based mining, on the other hand, requires millions of dollars of capital to get into (and if you buy an ASIC from someone else, most of the time, the manufacturing company gets the far better end of the deal).

This is also the correct answer to the common "proof of stake means the rich get richer" argument: ASIC mining *also* means the rich get richer, and that game is *even more* tilted in favor of the rich. At least in PoS the minimum needed to stake is quite low and within reach of many regular people.

Additionally, proof of stake is more censorship resistant. GPU mining and ASIC mining are both very easy to detect: they require huge amounts of electricity consumption, expensive hardware purchases and large warehouses. PoS staking, on the other hand, can be done on an unassuming laptop and even over a VPN.

Possible advantages of proof of work

There are two primary genuine advantages of PoW that I see, though I see these advantages as being fairly limited.

Proof of stake is more like a "closed system", leading to higher wealth concentration over the long term

In proof of stake, if you have some coin you can stake that coin and get more of that coin. In proof of work, you can always earn more coins, but you need some outside resource to do so. Hence, one could argue that over the long term, proof of stake coin distributions risk becoming more and more concentrated.

The main response to this that I see is simply that in PoS, the rewards in general (and hence validator revenues) will be quite low; in eth2, we are expecting annual validator rewards to equal ~0.5-2% of the total ETH supply. And the more validators are staking, the lower interest rates get. Hence, it would likely take over a century for the level of concentration to double, and on such time scales other pressures (people wanting to spend their money, distributing their money to charity or among their children, etc.) are likely to dominate.

Proof of stake requires "weak subjectivity", proof of work does not

See [here](https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/) (<https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/>) for the original intro to the concept of "weak subjectivity". Essentially, the first time a node comes online, and any subsequent time a node comes online after being offline for a very long duration (ie. multiple months), that node must find some third-party source to determine the correct head of the chain. This could be their friend, it could be exchanges and block explorer sites, the client developers themselves, or many other actors. PoW does not have this requirement.

However, arguably this is a very weak requirement; in fact, users need to trust client developers and/or "the community" to about this extent already. At the very least, users need to trust someone (usually client developers) to tell them what the protocol is and what any updates to the protocol have been. This is unavoidable in any software application. Hence, the marginal additional trust requirement that PoS imposes is still quite low.

But even if these risks do turn out to be significant, they seem to me to be much lower than the immense gains that PoS systems get from their far greater efficiency and their better ability to handle and recover from attacks.

See also: my previous pieces on proof of stake.

- [Proof of Stake FAQ](https://eth.wiki/concepts/proof-of-stake-faqs) (<https://eth.wiki/concepts/proof-of-stake-faqs>).
- [A Proof of Stake Design Philosophy](https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51) (<https://medium.com/@VitalikButerin/a-proof-of-stake-design-philosophy-506585978d51>).

Convex and Concave Dispositions

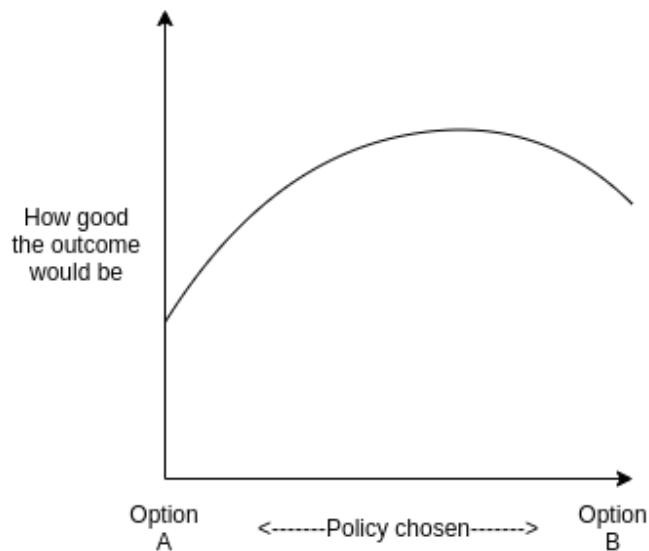
2020 Nov 08

[See all posts \(/\)](#)

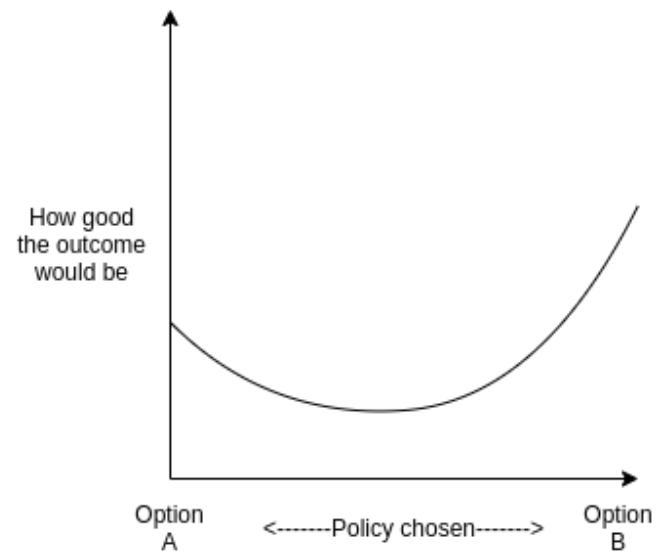
One of the major philosophical differences that I have noticed in how people approach making large-scale decisions in the world is how they approach the age-old tradeoff of compromise versus purity. Given a choice between two alternatives, often both expressed as deep principled philosophies, do you naturally gravitate toward the idea that one of the two paths should be correct and we should stick to it, or do you prefer to find a way in the middle between the two extremes?

In mathematical terms, we can rephrase this as follows: do you expect the world that we are living in, and in particular the way that it responds to the actions that we take, to fundamentally be [concave](#) (https://en.wikipedia.org/wiki/Concave_function) or [convex](#) (https://en.wikipedia.org/wiki/Convex_function)?

Concave worldview



Convex worldview



Someone with a concave disposition might say things like this:

- "Going to the extremes has never been good for us; you can die from being too hot or too cold. We need to find the balance between the two that's just right"
- "If you implement only a little bit of a philosophy, you can pick the parts that have the highest benefits and the lowest risks, and avoid the parts that are more risky. But if you insist on going to the extremes, once you've picked the low-hanging fruit, you'll be forced to look harder and harder for smaller and smaller benefits, and before you know it the growing risks might outweigh the benefit of the whole thing"

- "The opposing philosophy probably has some value too, so we should try to combine the best parts of both, and definitely avoid doing things that the opposing philosophy considers to be *extremely* terrible, just in case"

Someone with a convex disposition might say things like this:

- "We need to focus. Otherwise, we risk becoming a jack of all trades, master of none"
- "If we take even a few steps down that road, it will become slippery slope and only pull us down ever further until we end up in the abyss. There's only two stable positions on the slope: either we're down there, or we stay up here"
- "If you give an inch, they will take a mile"
- "Whether we're following this philosophy or that philosophy, we should be following *some* philosophy and just stick to it. Making a wishy-washy mix of everything doesn't make sense"

I personally find myself perenially more sympathetic to the concave approach than the convex approach, across a wide variety of contexts. If I had to choose either (i) a coin-flip between anarcho-capitalism and Soviet communism or (ii) a 50/50 compromise between the two, I would pick the latter in a heartbeat. I [argued for moderation](https://twitter.com/VitalikButerin/status/1277758769351589891) (<https://twitter.com/VitalikButerin/status/1277758769351589891>) in Bitcoin block size debates, arguing against both 1-2 MB small blocks [and 128 MB "very big blocks"](#) (<https://twitter.com/vitalikbuterin/status/1032248504603828224>). I've [argued against](#) (<https://twitter.com/VitalikButerin/status/910550101940039680>) the idea that freedom and decentralization are "you either have it or you don't" properties with no middle ground. I [argued](#) (https://www.reddit.com/r/ethereum/comments/4oj7gl/personal_statement_regarding_the_fork/) in [favor](#) (https://www.reddit.com/r/ethereum/comments/4ro2p9/options_in_the_hard_fork_slockit_blog/d52oizp/) of the DAO fork, but to many people's surprise I've argued since then against similar "state-intervention" hard forks that were proposed more recently. As I [said in 2019](#) (https://twitter.com/_sgtn/status/1089462282977824768), "support for Szabo's law [blockchain immutability] is a spectrum, not a binary".

But as you can probably tell by the fact that I needed to make those statements at all, not everyone seems to share the same broad intuition. **I would particularly argue that the Ethereum ecosystem in general has a fundamentally concave temperament, while the Bitcoin ecosystem's temperament is much more fundamentally convex.** In Bitcoin land, you can frequently hear arguments that, for example, [either you have self-sovereignty or you don't](#) (<https://twitter.com/jimmysong/status/1106308069800185874>), or that any system must have either a [fundamentally centralizing or a fundamentally decentralizing tendency](#) (<https://medium.com/hackernoon/sharding-centralizes-ethereum-by-selling-you-scaling-in-disguised-as-scaling-out-266c136fc55d>), with no possibility halfway in between.

The occasional half-joking [support for Tron](#) (<https://azcoinnews.com/udi-wertheimer-tron-will-surpass-ethereum-in-the-battle-for-decentralized-supremacy-in-2020.html>) is a key example: from my own concave point of view, if you value decentralization and immutability, you should recognize that while the Ethereum ecosystem does sometimes violate purist conceptions of these values, Tron violates them far more egregiously and without remorse, and so Ethereum is still by far the more palatable of the two options. But from a convex point of view, the extremeness of Tron's violations of these norms is a virtue: while Ethereum half-heartedly pretends to be decentralized, Tron is centralized *but at least it's proud and honest about it*.

This difference between concave and convex mindsets is not at all limited to arcane points about efficiency/decentralization tradeoffs in cryptocurrencies. It applies to politics (guess which side has more outright anarcho-capitalists), other choices in technology, and even what food you eat.

[Home](#) > [Crypto Culture](#)

The Carnivore Diet

Why are so many Bitcoiners only eating meat

SEAN by [SEAN](#) in [Crypto Culture](#)

1 0 0 0



But in all of these questions too, I personally find myself fairly consistently coming out on the side of balance.

Being concave about concavity

But it's worth noting that [even on the meta-level](https://slatestarcodex.com/2018/09/12/in-the-balance/) (<https://slatestarcodex.com/2018/09/12/in-the-balance/>), concave temperament is something that one must take great care to avoid being extreme about. There are certainly situations where policy A gives a good result, policy B gives a worse but still tolerable result, but a half-hearted mix between the two is worst of all. The coronavirus is perhaps an excellent example: a 100% effective travel ban is [far more than twice as useful](https://mobile.twitter.com/lymanstoneky/status/1321254322064236544) (<https://mobile.twitter.com/lymanstoneky/status/1321254322064236544>) as a 50% effective travel ban. An effective lockdown that pushes the R₀ of the virus down below 1 can eradicate the virus, leading to a quick recovery, but a half-hearted lockdown that only pushes the R₀ down to 1.3 leads to months of agony with little to show for it. This is one possible explanation for why many Western countries responded poorly to it: political systems designed for compromise risk falling into middle approaches even when they are not effective.

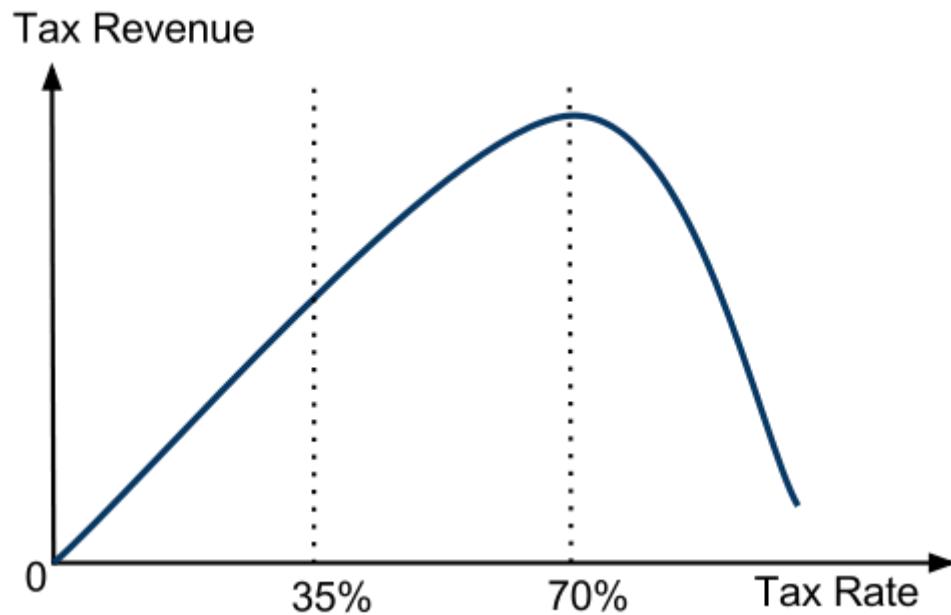
Another example is a war: if you invade country A, you conquer country A, if you invade country B, you conquer country B, but if you invade both at the same time sending half your soldiers to each one, the power of the two combined will crush you. In general, problems where the effect of a response is convex are often places where you can find benefits of some degree of centralization.

But there are also many places where a mix is clearly better than either extreme. A common example is the question of setting tax rates. In economics there is the general principle that [deadweight loss is quadratic](https://en.wikipedia.org/wiki/Deadweight_loss) (https://en.wikipedia.org/wiki/Deadweight_loss): that is, the harms from the inefficiency of a tax are proportional to the square of the tax rate. The reason why this is the case can be seen as follows. A tax rate of 2% deters very

few transactions, and even the transactions it deters are not very valuable - how valuable can a transaction be if a mere 2% tax is enough to discourage the participants from making it? A tax rate of 20% would deter perhaps ten times more transactions, but *each individual transaction that was deterred is itself ten times more valuable to its participants than in the 2% case*. Hence, a 10x higher tax may cause 100x higher economic harm. And for this reason, a low tax is generally better than a coin flip between high tax and no tax.

By similar economic logic, an outright prohibition on some behavior may cause more than twice as much harm as a tax set high enough to only deter half of people from participating. Replacing existing prohibitions with medium-high punitive taxes (a very concave-temperamental thing to do) could increase efficiency, increase freedom *and* provide valuable revenue to build public goods or help the impoverished.

Another example of effects like this in [Laffer curve](https://en.wikipedia.org/wiki/Laffer_curve) (https://en.wikipedia.org/wiki/Laffer_curve): a tax rate of zero raises no revenue, a tax rate of 100% raises no revenue because no one bothers to work, but some tax rate in the middle raises the most revenue. There are debates about what that revenue-maximizing rate is, but in general there's broad agreement that the chart looks something like this:

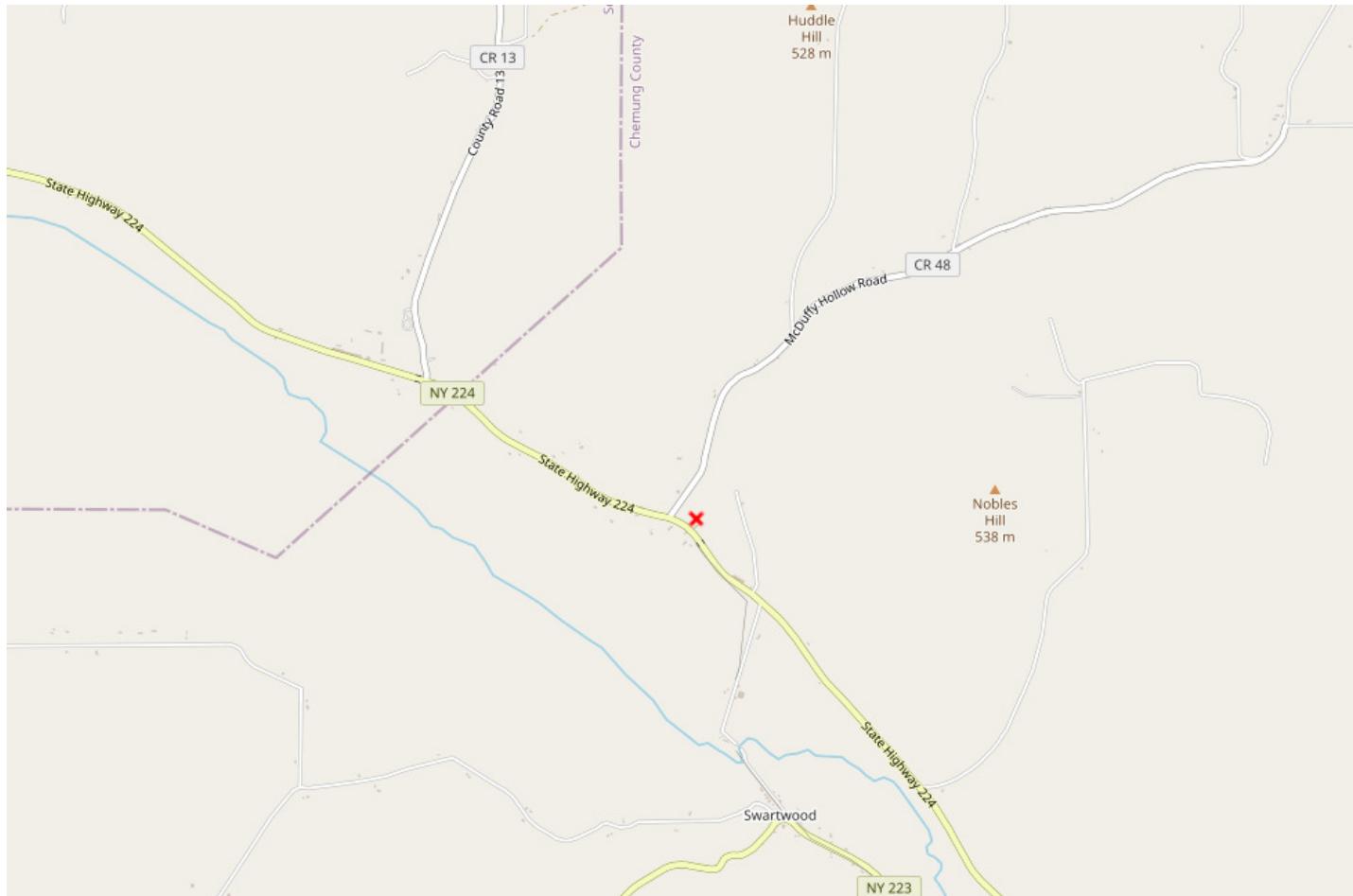


If you had to pick either the average of two proposed tax plans, or a coin-flip between them, it's obvious that the average is usually best. And taxes are not the only phenomenon that are like this; economics studies a [wide array of "diminishing returns" phenomena](https://en.wikipedia.org/wiki/Diminishing_returns) (https://en.wikipedia.org/wiki/Diminishing_returns) which occur everywhere in production, consumption and many other aspects of regular day-to-day behavior. Finally, a common flip-side of diminishing returns is accelerating costs: to give one notable example, if you take [standard economic models of utility of money](https://www.forbes.com/sites/rogerkay/2013/08/05/how-the-marginal-utility-of-money-balances-with-value/) (<https://www.forbes.com/sites/rogerkay/2013/08/05/how-the-marginal-utility-of-money-balances-with-value/>), they directly imply that double the economic inequality can cause four times the harm.

The world has more than one dimension

Another point of complexity is that in the real world, policies are not just single-dimensional numbers. There are many ways to average between two different policies, or two different philosophies. One easy example to see this is: suppose that you and your friend want to live together, but you want to live in Toronto and your friend wants to live in New York. How would you compromise between these two options?

Well, you could take the geographic compromise, and enjoy your peaceful existence at the arithmetic midpoint between the two lovely cities at....



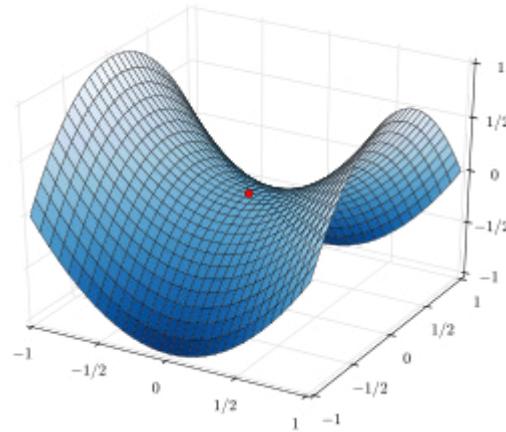
This Assembly of God church about 29km southwest of Ithaca, NY.

Or you could be even more mathematically pure, and take the straight-line midpoint between Toronto and New York without even bothering to stay on the Earth's surface. Then, you're still pretty close to that church, but you're six kilometers under it. A different way to compromise is spending six months every year in Toronto and six months in New York - and this may well be an actually reasonable path for some people to take.

The point is, when the options being presented to you are more complicated than simple single-dimensional numbers, figuring out *how* to compromise between the options well, and really take the best parts of both and not the worst parts of both, is an art, and a challenging one.

And this is to be expected: "convex" and "concave" are terms best suited to mathematical functions where the input and the output are both one-dimensional. The real world is high-dimensional - and as machine-learning researchers have now well established (<https://www.kdnuggets.com/2015/11/theoretical-deep-learning.html>), in high-

dimensional environments the most common setting that you can expect to find yourself in is not a universally convex or universally concave one, but rather a *saddle point*: a point where the local region is convex in some directions but concave in other directions.



A saddle point. Convex left-to-right, concave forward-to-backward.

This is probably the best mathematical explanation for why both of these dispositions are to some extent necessary: the world is not entirely convex, but it is not entirely concave either. But the existence of *some* concave path between any two distant positions A and B is very likely, and if you can find that path then you can often find a synthesis between the two positions that is better than both.