Why Proof-of-Work (PoW)

https://devSteve.com/blog/posts/why-proof-of-work.html

September 03, 2025

Preface

This post is **not** related to Bitcoin or any other cryptocurrency. The focus of this post is on the proof-of-work algorithm itself (not the cryptocurrencies that use it).

Assertion

You should be using the proof-of-work algorithm in your software.

Algorithm

The proof-of-work algorithm is surprisingly simple. Start with an input value and convert it to a string (let's call this the "input string"). Next append the number 0 (zero) cast as a string to the end of the input string. Now generate a hash of the input string with the appended zero string. Did the hash begin with X number of zeros (where X is the level of difficulty)? If so you've solved the proof-of-work.

If the hash did *not* begin with the desired number of zeros, then increment the appended 0 to a 1, cast it as a string, and append the one string to the original input string. Now generate a hash of the input string with the appended one string. Did the hash begin with the desired number of zeros? No? Now increment the 1 to a 2 and try again.

Keep doing this until you find a number that, when appended to the original input string, will generate a hash that starts with the desired number of zeros. This is how the proof-of-work algorithm works. Here's the proof-of-work algorithm expressed as code:

<code snippet redacted>

If you want to generate a sha256 hash that begins with three zeros (`000`) for the input `{ value: "hello world!" }` then you'll need to append the string `692`. Anyone who knows how to generate a sha256 hash can verify the proof-of-work by generating a single hash.

<code snippet redacted>

The beauty of proof-of-work is that the difficulty only affects the *solver*. It only took 693 tries (we started at 0 so we used 692 + 1 tries) to find a hash that started with three zeros. If the difficulty were to be raised to five or six, it could take orders of magnitude longer to solve the proof-of-work.

Verifying a proof-of-work hash, on the other hand, only ever takes **one** try. You take the input and the nonce (the number appended to the input), and you run it through the same hash function that was used to generate the nonce. Did you get the hash you wanted (ie: a hash that starts with three zeros)? Congratulations, you just verified the work in one try.

If this difficulty imbalance between solving and verifying sounds familiar, it might be because you like playing Sudoku puzzles. A Sudoku puzzle is a perfect analogy for the proof-of-work algorithm because a Sudoku can be incredibly difficult to solve, but every solution is extremely easy to verify since the rules of Sudoku are so simple.

But Why?

A prominent sales executive at my first employer introduced me to the term "cooking the books". In short, "cooking the books" means committing fraud by illegally modifying company records. When a practice - in this case, committing fraud - is so common in our society it warrants countless slang expressions, it might be time to acknowledge we have a problem.

In the retail world there's this notion of "shrinkage", which means "loss of product", which usually boils down to "theft". What's the top cause of shrinkage/theft? You guessed it, not the customers but the store's own employees!

I like to believe that most people are good most of the time. It would be great if we lived in a world without any liars, cheaters, or crooks, but unfortunately that's not the case - and wishing won't make it so.

The point of all this is that proof-of-work offers at least one line of defense against fraud. How, you ask? Well for one, no human can solve a proof-of-work hash without using a computer. Go ahead, try to even generate a single sha256 hash by hand, I'll wait.

This immediately reduces your attack surface from "any nefarious human" to "any nefarious human that knows how to code software to solve a specific proof-of-work algorithm". That's a pretty steep drop-off in attack surface.

Consider where most company data is stored. Excel spreadsheets? Databases where too many people have elevated write privileges? Even in places where the data is reasonably secure, every employee with the root/admin credentials - which usually turns out to be quite a few because most IT tasks require root/admin privileges - has the ability to go in and update database records.

Anyone with elevated data privileges can update sales figures, inventory counts, test results, literally anything that's stored in the database. With root/admin level access some of these operations won't even be recorded in the server logs, making it even more difficult to detect the fraudulent behavior.

Data stores that include proof-of-work hashes and/or blockchains along with their data reduce the threat of internal actors with elevated privileges "cooking the books". If implemented correctly, any attempted fraudulent change would be detected by the broken proof-of-work chain.

It's important to note that nothing is 100% safe and secure. The goal is to get as close to 100% as possible.

Example

If you want to see proof-of-work in action, look no further than the comment section at the end of this blog post. If you try to submit a comment, you'll be greeted with a simple proof-of-work captcha.

<video redacted>

Each number in the chain is used when generating the next number. While the work for each input is minimal, the cumulative work of the chain should limit form submissions to a maximum of one comment every three seconds (per client).

If one submission every three seconds still sounds like too much, you can increase the level of difficulty so that each number in the chain takes longer to generate. Careful though, if the difficulty level is set too high, some clients might not be able to solve the proof-of-work.

The captcha system for the comments goes a step further by penalizing incorrect entries. If an incorrect value is entered into a field then a new proof-of-work code will need to be generated.

<video redacted>

But Why?

If you're still asking yourself why someone would want to use proof-of-work for a captcha system (or anything else), then I'll try to sum it up as follows:

1. The only way to solve the captcha is by using this website and entering the numbers, or by writing a sophisticated bot that can use this website and enter the numbers.
2. Any comment submission that does not use one of the two methods listed in item 1 will be rejected outright.
3. The difficulty level can be increased for IP addresses with nefarious users.
4. This approach uses a simple open-source algorithm and does not require a third-party like Google or some other corporate tech behemoth.

Environment

Anyone familiar with proof-of-work or blockchain technology has undoubtedly heard the concerns about the environmental impact of having machines doing all of this "work". Unfortunately, for reasons already covered earlier in this post, the work is required because it helps to reduce human fraud.

The day people stop ripping each other off is the day we won't need to spend extra electricity to help make our systems more secure.

If you're that concerned with CO2 emissions then I encourage you to contact your local politicians and ask them to expedite the transition to green energy. The fact we're still burning things like coal to produce electricity is not the fault of proof-of-work.

Conclusion

This post is my initial attempt (don't worry, there will be more) at persuading more people to start using proof-of-work in their software. Though not perfect, it can greatly increase the difficulty of committing fraudulent acts. I'd be skeptical of anyone who argues *against* using proof-of-work (the only obvious motive is that they want to rip you off).