I’ve been thinking blockchains and the general web suffer from similar problems: they both try to make the creation of false identities difficult. Blockchains use computational and/or economic costs. Whereas, websites opt towards using visual challenges (captchas) to filter out bots. The problem with using captchas is the heavy degradation of usability. Services like ‘cloudflare’ now serve to guard websites from bots and force annoying puzzles on everyone. If there were a way to solve this problem without using captchas it would be extremely beneficial for users of the web.

If we look at ‘hashcash’ it was originally designed as a computational stamp for sending emails. Its aim was to prevent spam in emails which is very similar to the aim of captcha. But using hashcash in this way would make it easy to parallelize. Attackers with botnets would still have the upper hand. There are some small changes that might make the idea workable though. Let’s throw out the idea of having individual proofs for actions and move towards a single, long-running verifiable delay function.

account = VDF(…)

The aim of the algorithm will be to replace the need for manual account registration, especially the need to fill out captcha prompts by utilizing long-running VDFs that serve as a computational cost for reserving resources on other systems.

A registration server will distribute a random IV to the user to run the VDF on. The user will give their intended run_time_speed for the VDF to the server which is accepted as long as it’s fast enough for modern hardware.

account = VDF(server_iv, run_time_speed)

Initial resources (say for that day) can be assigned to the user after a set period. In order for this to happen the user breaks up their VDF into checkpoints that the server can verify in parallel. E.g. see the construct of Gwern’s serial hashing for time-lock encryption. If the checkpoints are correct then the user must have kept running the VDF and they’re given their first resource allocation.

resources = elapsed_time * run_time_speed * account_limit

Since this scheme requires ‘bootstrapping’ to wait for an initial allocation of resources. It could be side-stepped by requiring a larger initial commitment of resources. Such as in a Storj-style file allocation where proofs would be made against files on disk. These files could then be reduced overtime as incurred computational cost offsets disk space resources.

One useful consequence of using VDFs for accounts is complex, registration flows are no longer required. A service could manually check if the user has enough resources. Or they could outsource it to a specialized service who sends back signed vouches. If the user’s machine is offline they don’t automatically accrue more resources to spend on other systems. Their allocations are only refreshed while they’re active with their machines.

Another problem that these account-based VDFs solves is it makes it easier to block abusive users. Traditional registration systems are often plagued by never-ending attacks because attackers have access to large pools of IPs. Simple ban-by-IP approaches don’t work here with VPNs leading to less privacy on the web. With account-based VDFs it would be harder to create sock puppets. One ban would invalidate an entire sequential VDF and VPNs could still be used. You could even require VDFs that were a certain maturity before they attained full permissions.

The main requirements for the VDF to be useful:

1. It should not be possible to paralyze it
2. It should not degrade the performance of the machine it runs on
3. One machine should ideally only be able to run one VDF

Requirements (1) and (2) can be satisfied by using sequential, memory-based hashing. The difficulty for which should be chosen with respect to the ‘average consumer hardware’ at the time. Even though in reality it may be possible to run multiple VDFs on a single machine. The parameters should be chosen so that doing so would significantly degrade performance. At least so far as there’s a cost that regular users would not enjoy (this would also enable people infected with bots to detect infections since their machines would get marginally slower.)

Requirement (3) is something that would be ‘nice to have’ but I suspect impossible to design algorithmically. If an attacker has access to hardware that could run many VDFs (it might have many cores or lots of memory), then they would be in a better position to launch attacks. I think that this might be an area where trusted computing may be useful. As an example – an Intel CPU could use trusted computing to vouch for its serial number and model ID. The registration server could then use this information to help reduce impersonation.

These are my thoughts for now. Let me know what you think.