Poster: Proofs of Space

Stefan Dziembowski*, Sebastian Faust†, Vladimir Kolmogorov‡, Krzysztof Pietrzak‡
*University of Warsaw and *Sapienza* University of Rome, *faculty* †EPFL Lausanne, *post-doc* ‡IST Austria, *faculty*

***Abstract*—Proofs of work (PoW) have been suggested by Dwork and Naor (Crypto'92) as protection to a shared resource. The basic idea is to ask the service requestor to dedicate some non-trivial amount of computational work to every request. The original applications included prevention of spam and protection against denial of service attacks. More recently, PoWs have been used to prevent double spending in the Bitcoin digital currency system.**

In this work, we put forward an alternative concept for PoWs – so-called *proofs of space* (PoS), where a service requestor must dedicate a significant amount of disk space as opposed to computation. We construct secure PoS schemes in the random oracle model, using graphs with high "pebbling complexity" and Merkle hash-trees.

I. Proofs of Work (PoW).

Dwork and Naor [3] suggested "proofs of work" (PoW) to address the problem of junk emails (aka. Spam). The basic idea is to require that an email be accompanied with some value related to that email that is moderately hard to compute but which can be verified very efficiently. Such a proof could for example be a value σ such that the hash value $\mathcal{H}(\mathsf{Email}, \sigma)$ starts with t zeros. If we model the hash function $\mathcal{H}$ as a random oracle, then the sender must compute an expected 2^t hashes until she finds such an σ.[1] A useful property of this PoW is that there is no speedup when one has to find many proofs, i.e., finding s proofs requires $s2^t$ evaluations. The value t should be chosen such that it is not much of a burden for a party sending out a few emails per day (say, it takes 10 seconds to compute), but is expensive for a Spammer trying to send millions of messages. Verification on the other hand is extremely efficient, the receiver will accept σ as a PoW for Email, if the hash $\mathcal{H}(\mathsf{Email}, \sigma)$ starts with t zeros, i.e., it requires only one evaluation of the hash funciton. PoWs have many applications, and are in particular used to prevent double spending in the Bitcoin digital currency system which has become widely popular by now.

Despite many great applications, the PoWs suffer from certain drawbacks. Firstly, running the PoWs costs energy – especially if they are used on a massive scale, like in the Bitcoin system. Actually, for this reason the Bitcoin has even been called by some an "environmental disaster". Secondly, they give advantage to users who use dedicated hardware, as doing computation in hardware is always much more efficient than in software. Therefore, the parameters in the PoW-based schemes need to be chosen in such a way that a system is secure even against an adversary equipped with special-purpose hardware. This may, in some cases, mean that the computing effort spent by the honest (software-based) users becomes high.

II. Proofs of Space (PoS).

From a more abstract point of view, a proof of work is simply a means of showing that one invested a non-trivial amount of effort related to some statement. This general principle also works with resources other than computation like real money in micropayment systems or human attention in CAPTCHAs. In this paper we put forward the concept of *proofs of space* where the resource in question is the disk space. Some computational work is needed only in the initialization phase whose goal is to fill-in the disk space of the user with some data. Later, the user can engage in the PoS proofs using very small computation. This holds provided he did not erase the data produced during the initialization. On the other hand, if this data is not on his disk, then running the proof requires substantial computational effort. As a consequence, an adversary that wants to run s proofs efficiently needs to have disk space that is s times larger than the disk space of the honest users.

Our idea is motivated by an observation that for many users using this system will be essentially "for free", as anyway they have too much disk space (a standard laptop comes with a 500 GB hard disk)[2]. This is in contrast with the computing power, whose usage is associated with energy consumption even if one contributes only the CPU time of processors that would otherwise be idle. Moreover, in case of the PoS's the to users with "dedicated hardware" have no advantage compared to the "software users", as there is no such thing as "dedicated disk space" for solving a particular problem. Another advantage is that the disks can be reused for other applications — if a user decides to stop using our PoS he can even sell his disks on the second hand market. This is not true for the hardware used to solve some of the PoW (e.g. the Bitcoin ones), which can be used only for one purpose and is useless otherwise.

Our approach borrows ideas from [1], [2], [4], who suggest to construct PoW using functions whose computation requires accessing memory many times. This is motivated by the fact that different memories typically do not differ much in terms of the access time, which is in contrast with the computing speeds of CPU's that differ significantly. Hence, it is argued that this type of a proof of work would be more fair in the sense that having extremely fast CPUs does not help much, because the running time is to a large extend determined by the number of cache misses (which require "slow" memory accesses). This type of PoW can be seen as some kind of PoS, but it lacks

An extended version of this work appears in [5]. The first author was supported by the WELCOME/2010-4/2 grant founded within the framework of the EU Innovative Economy (National Cohesion Strategy) Operational Programme.

[1]The hashed Email should also contain the receiver of the email, and maybe also a timestamp, so that the sender has to search for a fresh σ for each receiver, and also when resending the email at a later point in time.

[2]This is, of course, not true for all the users. Therefore in some applications it would make sense to give users a choice between using a PoS and a PoW.

several properties we expect from a PoS, most importantly, the verifier needs to use the same amount of memory as the prover. This is not a problem in the original application of PoWs as here the space just needs to be larger than the cache of a potential malicious prover[3]. When this approach is used as a PoS, where the main resource dedicated is space, this is not an option, as here we want the verifier to use a tiny fraction of the space dedicated by the prover.

III. Formalizing Proofs of Space

In this work we consider *interactive* PoS between a *single* prover P and a verifier V. As an illustrative application for such an PoS, suppose that the verifier V is an organization that offers a free email service. To prevent that someone registers a huge number of fake-addresses for spamming, V might require users to dedicate some nontrivial amount of disk space, say 100GB, for every address registered. Occasionally, V will run a PoS to verify that the user really dedicates this space. The simplest solution would be for the verifier V to generate a random (and thus incompressible) 100GB file $\mathcal{F}$ and send it to the prover P during an initialization phase. Later, V can ask P to send back some bits of $\mathcal{F}$ at random positions, making sure V stores (at least a large fraction of) $\mathcal{F}$. A slightly better solution is to have V generate $\mathcal{F}$ using a pseudorandom function, in which case V does not have to store the entire $\mathcal{F}$, but just a short PRF key. Unfortunately, with this solution, V still has to send a huge 100GB file to P, which makes this approach pretty much useless in practice.

We want a PoS where the computation, storage requirement and communication complexity of the verifier V during initialization and execution of the PoS is very small, in particular, at most polylogarithmic in the storage requirement N of the prover P and polynomial in some security parameter γ. In order to achieve small communication complexity, we must let the prover P generate a large file $\mathcal{F}$ locally during an initialization phase, which takes some time I. Note that I must be at least linear in N, our constructions will basically achieve this lower bound. Later, P and V can run executions of the PoS which will be very cheap for V, and also for P, assuming the later has stored $\mathcal{F}$.

Unfortunately, unlike in the trivial solution (where P sends $\mathcal{F}$ to V), now there is no way we can force a potentially cheating prover $\tilde{\mathsf{P}}$ to store $\mathcal{F}$ in-between the initialization and the execution of the PoS: $\tilde{\mathsf{P}}$ can delete $\mathcal{F}$ after initialization, and instead only store the (short) communication with V during the initialization phase. Later, before an execution of the PoS, P reconstructs $\mathcal{F}$ (in time I), runs the PoS, and deletes $\mathcal{F}$ once it is done.

We will thus consider a security definition where one requires that a cheating prover $\tilde{\mathsf{P}}$ can only make V accept with non-negligible probability if $\tilde{\mathsf{P}}$ either uses N_0 bits of storage in-between executions of the PoS *or* if $\tilde{\mathsf{P}}$ invests time T for every execution. Here $N_0 \leq N$ and $T \leq I$ are parameters, and ideally we want them to be not much smaller than N and I, respectively. Our actual security definition is more fine-grained, and besides the storage N_0 that $\tilde{\mathsf{P}}$ uses in-between initialization and execution, we also consider a bound N_1 on the total storage used by $\tilde{\mathsf{P}}$ during execution (including N_0, so $N_1 \geq N_0$).

IV. Our constructions

We provide two constructions of the PoS's with different (and incomparable) parameters. Informally, our first construction proves an $\Omega(N/\log N)$ bound on the storage required by a malicious prover. Moreover, no matter how much time he is willing to spend during the execution of the protocol, he is forced to use at least $\Omega(N/\log N)$ storage when executing the protocol. Our second construction gives a stronger bound on the storage. In particular, a successful malicious prover either has to dedicate $\Theta(N)$ storage (i.e., almost as much as the N stored by the honest prover) or otherwise it has to use $\Theta(N)$ time with every execution of the PoS (after the initialization is completed).

Our proofs use a standard technique for proving lower bounds on the space complexity of computational problems, called "pebbling". Typically, the lower bounds shown using this method are obtained via the *pebbling games* played on a directed graph. During the game a player can place pebbles on some vertices. The game starts with some pebbles already on the graph. Informally, placing a pebble on a vertex v corresponds to the fact that an algorithm keeps the label of a vertex v in his memory. Removing a pebble from a vertex corresponds therefore to deleting the vertex label from the memory. A pebble can be placed on a vertex v only if the vertices in-going to v have pebbles, which corresponds to the fact that computing v's label is possible only if the algorithm keeps in his memory the labels of the in-going vertices (in our case this will be achieved by defining the label of v to be a hash of the labels of its in-going vertices). The goal of the player is to pebble a certain vertex of the graph. This technique was used in cryptography already before [4].

Acknowledgements: We would like to thank Moni Naor for pointing out the problem with our previous construction of a PoS.

References

[1] Martín Abadi, Michael Burrows, and Ted Wobber. Moderately hard and memory-bound functions. In *NDSS 2003*. The Internet Society, February 2003.

[2] Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 426–444. Springer, August 2003.

[3] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 139–147. Springer, August 1993.

[4] Cynthia Dwork, Moni Naor, and Hoeteck Wee. Pebbling and proofs of work. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 37–54. Springer, August 2005.

[5] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. Cryptology ePrint Archive, Report 2013/796, 2013. http://eprint.iacr.org/.

[3]The analysis of [1], [2], [4] takes into account the fact that a typical processor has a small *cache* memory that can be accessed quickly, and hence what counts is the number of times the processor needs to access the large "non-cache" memory, called the "cash misses"