Publicly Verifiable Proofs of Space

Markus Jakobsson, Agari

***Abstract*—We introduce a simple and practical Proof of Space (PoS) with applicability to ledger-based payment schemes[1]. It has a dramatically simpler structure than previous proposals, and with that, becomes very easy to analyze. A proof can be as short as a few hundred bits, and can be publicly verified using only two hash function computations.**

I. Introduction

While hardly anybody beyond cryptographers paid much attention to Nakamoto's 2008 Bitcoin publication [1] as it was published, only ten years later, it is hard to find somebody who has not heard about the methods described in the paper. Along with the growth of interest in BitCoin has also come a dramatic increase in the computational requirements associated with mining. This, in turn, has caused notable increases in power consumption.

While not all Bitcoin variants impose the same energy costs, the variations are relatively minor, prompting calls for alternative methods with the same principal functionality, but with *dramatically* lower power consumption impact. A promising approach replaces the underlying Proof of Work (PoW) – which is intrinsically related to power consumption – with a Proof of Space (PoS).

The last few years have, accordingly, seen research into PoS-based payment schemes. Ateniese et al. [2] and Dziembowski et al. [3] independently of each other introduced the notion of PoS, based on pebbling of a tree-based structure, for which an interactive proof of knowledge is added for purposes of verification. While the need for interaction arising from the use of challenge-response protocols makes these approaches impractical for use in payment schemes, this drawback can be avoided using standard random oracle based techniques. However, the use of non-interactive proofs of knowledge results in very large proofs.

The first large-scale implementation of a payment scheme relying on a PoS was put forth by Burstcoin [4]. However, as pointed out by Park et al. [5], Burstcoin does not disincentivize computationally bound mining, and requires a large amount of storage to be *accessed* – not just configured and possessed – for the mining to work. Park et al. introduce a solution that improves on Burstcoin, but which still relies on a challenge-response structure to obtain public verifiability.

We introduce a solution that, by relying on a much more straightforward data structure dramatically simplifies the approach, resulting in a PoS solution with a *tiny* proof – just above 300 bits in one configuration. The proof can be efficiently verified and does not rely on a challenge-response method. Our solution is backwards compatible with typical PoWs used in payment schemes, but for a minor difference in the transaction scheduling (that we think is *necessary* for any true PoS to be secure). The maybe most notable feature of our scheme is its simplicity, among other things making our security claims straightforward – including the proofs relating to the probability of success for a miner with limited storage resources.

II. Solution

As for any PoS or PoW with applications to payments, there are four parties:

- An authority $\mathcal{A}$ that publishes a set of algorithms and set of parameters, defined by an initial ledger $\mathcal{L}_0$.
- A miner $\mathcal{M}$ that attempts to derive a new entry $\mathcal{E}_{j+1}$ to the ledger $\mathcal{L}_j$ by solving a task; this is referred to as *mining*.
- A verifier $\mathcal{V}$ that wishes to verify a tentative new entry $\mathcal{E}_{j+1}$ to ledger $\mathcal{L}_j$. Typically, any miner would want to do this before embarking on a mining effort.
- A payer who creates a transaction, comprising a message with a digital signature. To record transactions (e.g., avoid double-spending) transactions are posted to the ledger, and are not considered completed until the ledger has incorporated them.

We will now describe the basic version of our solution:

Configuration. $\mathcal{A}$ specifies the mining/verification algorithms (detailed below), selects a *root* R and a parameter b that dictates the difficulty of mining. $\mathcal{A}$ then creates an initial *ledger* $\mathcal{L}_0$ = (b, h($\mathcal{V}$)). Here, h is a hash function and $\mathcal{V}$ is a specification of how to verify a ledger entry. $\mathcal{A}$ *closes* $\mathcal{L}_0$ by concatenating an entry $\mathcal{E}_1$ generated using the mining operation specified below, using $(\mathcal{E}_{-1}, \mathcal{E}_0) = (0, 0)$ for boostrapping. The resulting new ledger $\mathcal{L}_1 = \mathcal{L}_0|\mathcal{E}_1$ is made public.

Here, R may be a 256 bit pseudo-random number and $b = 58$. The hash function h may be selected as SHA256.

Mining Setup. As illustrated in figure 1, a miner $\mathcal{M}$ selects a public key $pk_{\mathcal{M}}$, computes $id = h(pk_{\mathcal{M}})$; then generates and stores n pairs (leaf_i, i), where i is will be referred to as a *witness*, and $\text{leaf}_i = h(R, id, i)$. These pairs are stored, lexicographically ordered by leaf_i.

Transactions. A *transaction* is an event that needs to be recorded; an example transaction is a payment. The transactions are recorded by a ledger entry being added, where the ledger entry is a function of the transactions in question.

The selection of what transactions to include in the generation of a task in the mining operation depends on whether the mining operation uses a PoW or a PoS, as explained in the

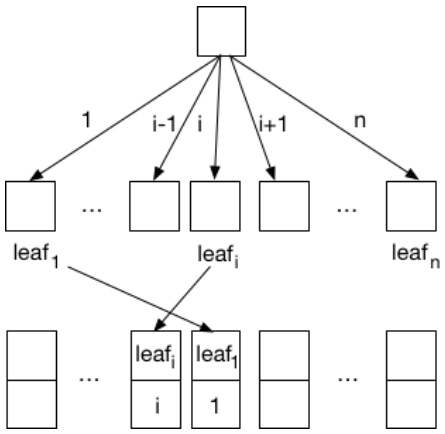

Figure 1. The figure shows the mining setup process, wherein miner $\mathcal{M}$ generates leaf values leaf_i and corresponding witnesses i from root R, by computing a hash of (R, id, i), where R is the public root value, id is a hash of public key $pk_{\mathcal{M}}$, and i is a counter. The leaves and witnesses are stored, sorted with respect to the value of leaf_i.

Appendix. For PoW-based mining, a new ledger entry is based on the previous ledger entry and a collection of transactions posted after the old ledger entry [1].

A slightly more complicated transaction selection structure, shown in figure 2, is necessary for any scheme based on PoS-based mining to avoid the abuse detailed in the Appendix. Here, the transactions recorded between the two most recent ledger entries – along with these entries – are used to generate the challenge for a new ledger entry.

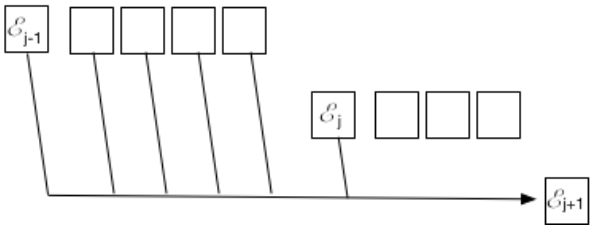

Figure 2. The figure shows how a new ledger entry $\mathcal{E}_{j+1}$ depends on recorded transactions and previous ledger entries $\mathcal{E}_{j-1}$ and $\mathcal{E}_j$ in a PoS-based scheme. This structure prevents a cheating miner from increasing his success probability by converting the PoS to a PoW, as described in the Appendix.

Mining. The miner $\mathcal{M}$ computes a *task* $t = h(\mathcal{E}_{j-1}, \mathcal{E}_j, T_j)$, where T_j is a hash value of a list of transactions, selected as described above, and ordered based on the time they were posted to the ledger. Here, $\mathcal{E}_j$ is the most recently posted ledger entry. $\mathcal{M}$ then determines whether it has stored a value leaf_i that *matches* t, i.e., for which the b least significant bits are the same. (See fig. 3.) If so, $\mathcal{M}$ publishes a new ledger entry $\mathcal{E}_{j+1} = (id, i)$, where i is the witness associated with leaf_i.

Verification. A verifier $\mathcal{V}$ verifies a new ledger entry $\mathcal{E}_{j+1} = (id, i)$ by computing $t = h(\mathcal{E}_{j-1}, \mathcal{E}_j, \mathcal{T}_j)$ and comparing the b least significant bits of t and $h(R, id, i)$; if these match, then $\mathcal{V}$ concludes that $\mathcal{L}_{j+1} \leftarrow \mathcal{L}_j | \mathcal{E}_{j+1}$, where $|$ denotes concatenation.

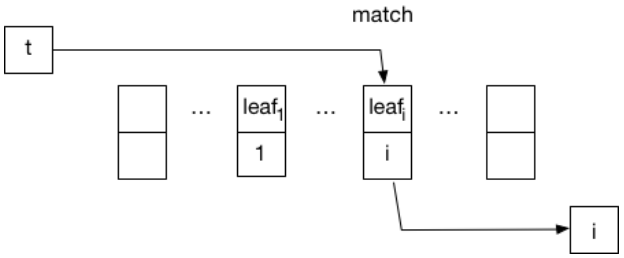

Figure 3. The figure shows the mining process. The miner computes a task t and compares it to stored values leaf_i. If the b least significant bits are the same, it is a match, and the witness i is output. Given lexicographic ordering of the pairs (leaf_i, i) with respect to the value of leaf_i, the search operation takes $O(log\, n)$ memory accesses, where n is the number of pairs the miner stores.

Generating a transaction. A miner $\mathcal{M}$ with a public key $pk_{\mathcal{M}}$ that is a hash of a value id in a ledger entry can compute a digital signature s on a message m and output $(m, s, pk_{\mathcal{M}}, j)$ where j indicates the entry $\mathcal{E}_j$ associated with $pk_{\mathcal{M}}$. This is verified by computing $id = h(pk_{\mathcal{M}})$, verifying that id is part of a valid ledger entry $\mathcal{E}_j$; and then verifying that s is a valid signature on m with respect to the public key $pk_{\mathcal{M}}$.

Here, the message m may include another public key $pk_{\mathcal{P}}$ associated with a second party $\mathcal{P}$, who may use *this* key to generate transactions, etc. Such signatures are verified by checking that the chain of signatures is valid, and that the "original" public key $pk_{\mathcal{M}}$ is part of a valid ledger entry.

To avoid double-spending of financial resources, and to timestamp agreements, transactions are recorded in the ledger, as described above.

Analysis. It is easy to see that the probability of success of one mining operation is upper bounded by $n/2^b$ for a miner $\mathcal{M}$ storing n leaf/witness pairs, since h acts as a random oracle.

Here, the storage requirements are roughly $n \times (log\, n + b)$ bits, assuming only the b bits of the leaf value that will be used for matching are stored. In other words, if $n = 2^{38}$ and $b = 58$ then this requires approximately 3.3 Terabytes, and results in a success probability of about one in a million. The time to perform the mining is approximately that of 38 memory accesses, which on a typical high-performance hard drive takes approximately 0.1s. The size of a ledger entry is the size of the hash function output plus the size of the witness.

Fairness. It can be seen that this type of PoS is *fair* in that a party with twice the storage capacity allocated to mining would have twice the success probability. To favor larger players – or promote massive collaboration – one can require that a valid ledger entry $\mathcal{E}_j$ contains *several* witnesses corresponding to multiple, statistically independent, task values[2]. This has the advantage of tightening up the requirements on the amount of storage needed; for example, having less than half of the "required" storage in a scheme requiring 20 witnesses results in a probability of success of one in a million, whereas

[2]For example, a collection of statistically independent tasks t_c can be computed as $h(\mathcal{L}, c)$ where c is a counter.

an entity in possession of the required storage is essentially *guaranteed* success.

Consensus. Our PoS proposal is compatible with existing ledger-based schemes, and potential race conditions are addressed using consensus-based methods, just like for BitCoin [1]. An important aspect relating to consensus is to determine that all selected transactions were, indeed, posted *before* the last ledger entry, due to the abusive strategy described in the Appendix. As a result, verifiers observing a transaction near-simultaneously to a ledger entry benefit from assuming that the transaction came in last, as this minimizes abuse, and abuse results in a relative loss for parties not *performing* the abuse. This, in turn, means that verifiers in doubt will choose the strategy that most others will choose, as consensus benefits those supporting it.

Verifiers will also use consensus to choose the ledger entry among competing (i.e., near-simultaneous) entries. If race conditions between posted ledger entries are starting to become common or the average arrival time between ledger entries fall below a threshold then an authority can make mining more difficult. T

Difficulty. The difficulty of mining can simply be raised by increasing the value for b. This is best done by $\mathcal{A}$ by a signed announcement, relating to a *future*[3] ledger state, such as a statement corresponding to "after another valid entry has been made to the ledger, increase b by one." Increasing b by 1, of course, increases the difficulty of mining by a factor 2 for a party that stores less than 2^b unique leaf values[4].

However, smaller increments of the mining difficulty are also possible, and can be made in a practical manner. For example, $\mathcal{A}$ can announce three bit positions (p_1, p_2, p_3) of witnesses and an associated disallowed pattern – such as $(0, 0, 1)$. This effectively makes $1/8$th of all witnesses temporarily invalid, thereby increasing the level of difficulty correspondingly.

Privacy. The description above has, for simplicity of denotation, focused on only *one* id per miner. However, it is straightforward to generalize this to multiple public keys and associated ids. A miner can either partition its storage in terms of what id is used (which increases the number of read operations of the mining) or store a tag indicating the id used for each leaf/witness pair (requiring a slight increase in storage requirements). Matching to a task will be done to the leaves of all ids. If any one results in a witness, that witness and corresponding id are posted. A miner that successfully produces a ledger entry $\mathcal{E}_j$ can replace all entries in its storage corresponding to the id of $\mathcal{E}_j$ to avoid repeatedly using the corresponding public key.

[3]Making the announcement relative to a current state introduces a potential divergence risk, where some miners are using one set of guidelines and others use another.

[4]This suggests a storage strategy in which miners would store *at least* the b bits currently required to verify a potential match.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008, https://bitcoin.org/bitcoin.pdf.

[2] G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi, "Proofs of Space: When Space is of the Essence," 2013, https://eprint.iacr.org/2013/805.pdf.

[3] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak, "Proofs of Space," 2013, https://eprint.iacr.org/2013/796.pdf.

[4] "BURSTCOIN Celebrates Birthday With Release Of New Energy Efficient HDD Mining Wallet," Aug 11, 2015, http://us.newsbtc.com/burstcoin-celebrates-birthday-with-release-of-new-energy-efficient-hdd-mining-wallet/.

[5] S. Park, A. Kwon, J. Alwen, G. Fuchsbauer, P. Gazi, and K. Pietrzak, "SpaceMint: A Cryptocurrency Based on Proofs of Space," 2015, https://eprint.iacr.org/2015/528.pdf.

APPENDIX

PoW-based mining uses challenges that are set as a function of the current ledger (or most recent ledger entry,) combined with a representation of a list of transactions posted after the last transaction included in the previous ledger entry. This is possible because in PoW-based mining, every challenge is equally difficult to solve as another, which means that there is no incentive for miners to include transactions simply for the reason of changing the challenge. That, however, is *not* true in PoS-based mining schemes.

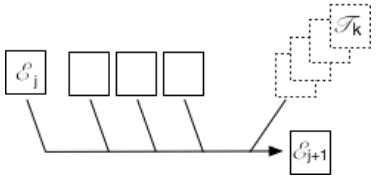

Figure 4. The figure shows how a cheating miner generates (but does not post) a large number of *potential* transactions, then determining whether the posted list of transactions combined with any of the potential transactions results in a task that the miner can answer. If so, then the miner posts the successful potential transaction and the corresponding ledger entry in rapid succession.

If the selection of transactions of a PoW scheme were to be used in a PoS scheme, a miner in possession of a certified public key could therefore create a very large number p of *potential* transactions, which we will denote $\tau_1 ... \tau_p$, each one corresponding to a unique valid digital signature on a message (see Figure 4.) It would then determine, for each τ_k, what challenge c_k would result from posting τ_k to the ledger. If the miner has a witness that matches this challenge c_k, then it would post τ_k and, in rapid succession, the ledger entry that contains a valid witness with respect to the set of posted transactions. See Figure 4. Since each PoS takes only fractions of a second to generate, this would be a practical approach for miners to dramatically increase their success by effectively turning the PoS into a memory-bound PoW.

To stop such abuse, it is necessary to change the method of selecting transactions so that transactions need to be posted before they can be selected. The most practical way of achieving that is to use new ledger entries as "punctuation" – where the most recent entry also effectively serves as a randomizer of the challenge that results from it and the transactions posted before it, as illustrated in Figure 2. Consensus-based methods will provide assurance that the ordering is adhered to.