ATMcash

Pre-development technical presentation white paper.

(This paper was written before actual devleopment, and another method was ultimately used. This is at least most of the math and explanation that was eventually made use of, and explanations therein. The ultimate design chosen follows all the mentioned rules and is superior to what is described here.)

A Cryptocurrency Based on Proofs of capacity

(Sunoo Park , Krzysztof Pietrzak, Albert Kwon , Joel Alwen , Georg Fuchsbauer, Peter Gazi, with additions, clarification and formatting by ATMcash technical team post-development)

We propose a decentralized cryptocurrency calledATMcash, which is based on a blockchain ledger similar to hatof Bitcoin, but where the wasteful proofs of work are replacedby efficient proofs of capacity , recently introduced by Dziembowskiet al. Instead of requiring that a majority of the network's computing power is controlled by honest miners (as in Bitcoin), our currency requires that honest miners dedicate more net disk
capacity than a potential adversary.

In ATMcash, once a miner has dedicated and initialized some capacity , participating in the mining process is very cheap. A new block is added to the chain every fixed period of time, and in every period a miner just has to make a small number of lookups to the stored capacity to check if she "wins", and thus can efficiently add the next block to the chain and get the mining reward. In this paper, we detail the construction of ATMcash, analyze its security and game-theoretic properties, and study its
performance. Our prototype shows that it takes approximately 25 seconds to prove over a terabyte of capacity , and it takes a fraction of a second to verify the proof.

I. INTRODUCTION

Bitcoin is a decentralized digital currency which was introduced
in 2009 and now is by far the most successful digital
currency ever deployed. The currency's decentralized bookkeeping
depends on maintaining a public ledger recording all
transactions that occur. This ledger is implemented by a block
chain: that is, a sequence of blocks each of which contains
transaction records and some auxiliary information, which
are generated by participants in the network. To encourage
participants to contribute blocks, those who add a block to the
chain are rewarded with some newly minted Bitcoin.

A principal difficulty when designing a digital currency is
to provide security against double-spending: that is, the owner
of a coin must be able to spend it exactly once. To prevent
double-spending, it must be enforced that all parties in the
network agree on the same blockchain (except possibly for
the most recent few blocks). Before accepting a transaction,
a recipient should wait until the transaction has been in the
chain long enough that she can be reasonably sure it will stay
there forever (that is, consensus has been reached).
The Bitcoin protocol achieves consensus by making it
computationally hard to add a block to the chain: currently,

1. Accordingly, a Bitcoin block is considered to be
a proof of work: that is, a proof that a certain amount of
computational resources were invested.

The Bitcoin network mines a block approximately every
10 minutes, and so it consumes computational resources –
and associated natural resources, primarily in the form of
electricity – at a massive scale. Current network usage is
estimated to be several 100 MW of power; moreover, most
mining is currently done by dedicated hardware, which has
no use beyond mining Bitcoins. For these reasons, Bitcoin is
considered an "environmental disaster" by some.

The original idea behind basing Bitcoin mining on computational
power was that anyone could participate in the
network by dedicating their spare CPU cycles, which incurs
little marginal cost in that it uses the idle time of already existing
personal computers. However, the dynamics of modern
Bitcoin mining have become very different: the majority of
successful mining is done by large-scale mining farms, often
in collaboration with electricity producers. Without specialized
mining ASIC hardware, you don't have a chance: mining
with your spare CPU cycles will lose money, due to overhead
electricity costs. The nonlinearity of mining rewards in Bitcoin
relative to resources invested has been quantified by Designing a
cryptocurrency where the expected reward is more proportional to
invested resources would be desirable for a number of reasons,
including that the presence of "small players" can be important for
the stability and decentralization of the currency.

To address these issues, in this paper we propose a cryptocurrency, ATMcash, which replaces the costly proofs of work underlying Bitcoin with proofs of capacity. In ATMcash, in order to mine blocks (and thereby mint coins),miners must invest disk capacity, rather than computational power.

2. Miners who dedicate more disk capacity have a proportionally higher expectation of successfully mining a block and reaping the reward. These days, tremendous amounts of disk capacity are lying around unused, and all of this storage capacity carries potential for mining. We note that in a capacity-based scheme, it is clear that miners will be incentivized to invest in hard drive capacity, just as Bitcoin miners are incentivized to invest in electricity. However, we highlight a couple of key differences:

1) In ATMcash, the investment is in the form of capital expenditure, and the mining process after the hard drives are bought incurs negligible overhead cost (both in terms of monetary and natural resources). In contrast, in Bitcoin, the mining process requires perpetual energy expenditure from miners.

2) In Bitcoin, resources are "used up" by mining: electricity is a depletable resource which once used is gone, and Bitcoin mining hardware is specialized, single-purpose resource that is not useful for anything once the need for Bitcoin mining is removed. In contrast, the resource consumed by ATMcash is recyclable, in that it can be used over and over, and multi-purpose, since hard drives have intrinsic value in their ability to store useful data.1
The distinction between capital expenditure and recurring overhead costs is significant also in that it changes the trade off between expected reward and mining resources invested. More concretely, due to the low marginal cost of mining a block, the shape of the curve depicted in Figure 1 will be much flatter for ATMcash than Bitcoin, and thus the profitable area would be spread more evenly over the horizontal axis.

A. Background and challenges

A "greener" cryptocurrency. The community has looked

for alternative decentralized consensus protocols and found a
potentially promising candidate in proofs of stake (PoStake).
In such schemes, the probability that a party mines the next
block is proportional to the fraction of coins (out of all coins
belonging to participating miners) that it holds. This idea is
very appealing as no resources (like energy, hardware, etc.)
are wasted, but unfortunately, making this approach actually
work turns out much more delicate than for schemes based on
proof of work (PoW).

Trying to adapt Bitcoin in a straightforward way by replacing
PoW by PoStake, one runs into at least three major
problems which are outlined below. Intuitively, the first two
problems are related to the fact that producing a PoStake
proof is computationally cheap and thus opens up potential for
cheating in ways which are not possible in Bitcoin. We will see
that analogous challenges arise from the computational ease
of mining a block by proof of capacity , too; and our ATMcash
construction will propose ways to resolve these challenges.

1) Multiple chains: In Bitcoin, a rational miner will always
work towards extending the longest chain of which he is
aware, as working on any other chain would only lower the
probability that his mined block will end up in the block
chain. When using PoStake instead of PoW, checking whether
One may ask: won't ATMcash spur development of specialized types of
storage which are tailored for mining, and thus end up in the same position
as Bitcoin in this respect? We argue that this is unlikely; see Section IX.
one can extend a chain is very cheap, and thus miners may
try extending many different chains in parallel. This impedes
quick consensus finding, unlike in Bitcoin where all rational
miners concentrate on the longest chain, and thus it always
grows faster than others

• **Mining multiple chains:**

Our approach is based on penalizing
miners who work on more than one branch. It is
important to discourage miners not only from announcing
blocks on multiple chains, but also from "trying out"
many different chains and choosing only the best one
to announce. We suggest three variant solutions. In the
first two schemes, a miner's block quality is fixed for any

given time-step, so that trying many chains does not yield any benefit; and then we penalize miners who announce blocks on multiple chains. The third scheme takes a different approach, and introduces interaction between miners during the mining process, thus enabling detection of miners who try many chains.

Finally, we perform a game-theoretic analysis of ATMcash and find that it has at least as strong equilibrium properties as Bitcoin. We model the ATMcash protocol as an extensive game, and prove that miners are not incentivized to deviate from the rules as long as there is an honest majority. More formally, we prove that the protocol is a sequentially rational Nash equilibrium, which is the standard equilibrium concept for games which happen over many time-steps. Prior work related to equilibria in Bitcoin has given only an informal treatment of the problem: notably, [20] presents a thorough, but still informal, analysis of equilibrium strategies in Bitcoin, and concludes that honest mining is a Nash equilibrium in Bitcoin (if there is an honest majority).

Contribution. In summary, our contribution is as follows:

• **Cryptocurrency from proofs-of-capacity** : ATMcash is a cryptocurrency based purely on proofs of capacity , and thus avoids the major drawbacks of existing proof-of-work based schemes as discussed in this section.

• **Addressing the “nothing-at-stake” problem**: We propose novel approaches to the known problems of grinding and mining multiple chains in non-proof-of-work based systems. Our solutions can also be extended to the proof of-stake setting where these issues were first encountered.
• Evaluation of ATMcash and proofs-of-capacity : We implemented a proof-of-capacity library and a prototype of ATMcash. It takes less than 20 seconds to prove over 1 TB of capacity , and a fraction of a second to verify.

• **Game theory of ATMcash**: Our game-theoretic analysis models ATMcash as an extensive game proves that the adhering to the protocol is a sequential Nash equilibrium.

II. RELATED WORK

Proofs of storage/retrievability. Other concepts similar to
proofs of capacity are proofs of storage and proofs of retrievability.
These are proof systems where a verifier sends a file to a prover,
and later the prover can convince the verifier that it really
stored or received the file. Proving that one stores a (random)
file certainly shows that one dedicates capacity , but these proof
systems are not proofs of capacity because the verifier sends
the entire file to the prover, whereas an important property
of POC is that the verifier's computation (and thus also
communication) is at most polylogarithmic in the size of
storage dedicated.

('Proofs of secure erasure' is Another type of proof system
which is related to POC.)

III..PROOFS OF capacity

As briefly discussed in Section I, the goal of proof of capacity
is for a prover to prove to a verifier that it is storing a certain
amount of capacity . In this section, we first discuss two straw man
approaches that do not work, and then present our variant of
POC for the cryptocurrency setting.
A. Two simple approaches that don't work
Storing a function table. A tempting "solution" is to
Have a random-looking function, sorted by the output. The prover's
challenge would be to invert the function on value f(x) for
some random x ? an honest prover can do this in time
log by binary search. Unfortunately, this doesn't work
due to time/memory trade-offs, which allow a cheating
prover to only store roughly ⅔ input/output pairs and still linvert the function in time.
Storing a random file. Another simple idea would be for
to send(pseudo)random bits during initialization, and
simply query back for a random subsets of these bits during execution.
However, this requires bits of communication, whereas
a POC requires that the verifier's efficiency depends on
some security parameter, but must be basically independent
of– and this property is crucial for all applications of
POC discussed.

IV. OVERVIEW OF ATMcash

A. High-level protocol description

Transactions. Transactions are performed basically identically to Bitcoin: each coin “belongs” to some public key pk. The block chain acts as a ledger that keeps track of which coins belong to which keys (but to prevent grinding, we propose a new design for the blockchain in Section VI where the transactions are decoupled from the proofs). To transfer a coin from pk to pk0, a transaction specifying this must be signed by sk (the secret key for pk), and then be added to the block. The nonce ensures that the same capacity cannot be used for two different proofs (this will be discussed more later).

We also add special transactions to initialize miners (plotting), and a special type of transaction which penalizes a miner who extends two different chains using the same proof of capacity (which Hasn’t been implemented yet, but there are methods to verify and Remove incorrect chains.)

Incentivize mining. Like in Bitcoin, there are two ways to incentivize miners to contribute resources (disk capacity in ATMcash, computing power in Bitcoin.)

(1) a reward for adding blocks
(2) transaction fees.

Reward: For adding a block to the chain, a miner receives some freshly minted coins. The reward size is specified as part of the protocol, and typically depends on the block index.

> In Bitcoin, the reward was initially 50 Bitcoins, but it halves roughly every 4 years, and is currently at 25.
>
> In ATMcash the reward is a slowly declining curve that changes 10% per month, allowing for a release schedule that is not as drastic in cutting the reward to the miners. We believe this is a more adequate approach than that of Bitcoin and other currencies with use ‘halving’.

Transaction fees: When generating a transaction, one can dedicate some (usually very small) amount of the transferred coins to the miner who adds the transaction to the blockchain.

Initialize miner. If a miner wants to contribute N bits of
capacity to the mining effort, she samples a public/secret key pair
(pk, sk) and runs the POC initialization procedure. (Being
in a non-interactive setting, there is no verifier to generate the
unique nonce μ, so we simply use pk for this.)

(?, S?) := Init(pk, N) .

The miner stores (S?, sk) and generates a special transaction
which just contains (pk, ?). Once this transaction is in the
blockchain the miner can start mining as described next.
Mining. Blocks are added to the blockchain every fixed time
period (say, every minute), and we require that all parties have
a clock that is roughly synchronized. To add a block in time
period i, the miner retrieves the hash value of the last block
in the best chain so far (this chain has i - 1 blocks), and
also a challenge c. This c is used as randomness to sample kp
challenges cp and kcv challenges ccv.

Where, depending on which of the Pocapacity discussed in the last section
we use, kp = O(1), kcv = log(n) or kp = O(1), kcv = ? · log(n). These
challenges can be sampled by first using c as a seed to generate a sufficient
amount of randomness (rp, rcv) := hash(c) for the challenge sampling algorithm
to get cp := Challenge(n, kp, rp), ccv := Challenge(n, kcv, rcv).

How to derive the challenge c is the main difficulty we face.
In our simplest solution we assume an unpredictable beacon
that broadcasts a fresh random (or at least unpredictable) value
from which the challenge is derived every minute (we also
propose two solutions without assuming a beacon). The miner
then computes the POC answer from cp:
a := Answer(pk, S?, cp) .
For two valid proofs (pk, ?, c, a) and (pk0, ?0, c0, a0) we denote
with (recall that N? is the size of the capacity committed by ?)
(a0, N?0) ? (a, N?)
that the proof a is better than a0. We postpone the discussion
on the precise definition of this ordering to Section V-B. For
now, we only mention that the ordering should satisfy
Pr[(a0, N0) ? (a, N)] = NN + N0
.
That is, the probability that a wins is proportional to its
fraction of the total capacity . The probability is taken over the
choice of random oracle used to compute the proof quality.

If the answer a found by a miner is so good that there is a
realistic chance of it being the best answer found by any miner,
the miner creates a block and sends it out to the network in the
hope that it will end up in the chain. A block must contain
transactions, the POC proof a and also the commitment
verification output computed as acv := Answer(pk, S?, ccv).
Note that the commitment verification need not be executed
unless the miner has found an exceptionally good proof: thus,
the computation of the vast majority of miners in the network
will be very low, since they need only check the quality of their
proof, and most of them will not proceed with verification.
For the remainder of the one-minute time period, the miner need not do anything. As mining only requires a small amount of work (computation, communication and random access to the storage) in every time period, it can be run on any computer that has some free disk capacity and is connected to the internet, without incurring any noticeable slowdown.

Quality of a Pocapacity Proof Consider some valid proofs
(pk1, ?1, c1, a1), . . . ,(pkm, ?m, cm, am) for capacity s of
size N1, . . . , Nm. We want to assign a quality to a proof
(which will only be a function of ai and Ni), such that the
probability (over the choice of the random oracle hash) that
the ith proof has the best "quality" corresponds to its fraction
of the total capacity , i.e.
Pr hash[?j 6= i : (aj , Nj) ? (ai, Ni)] = P Ni m j=1 Nj
.
We observe that in order to achieve this, it is sufficient to
achieve this for any pair of commitments, i.e.,
Pr hash [(aj , Nj) ? (ai, Ni)] = Ni Ni + Nj

If all the Ni were of the same size N, we could simply define
(aj , N) ? (ai, N) ?? hash(aj) = hash(ai)
That is, we map every ai to a random value hash(ai), and
whichever value is largest wins. We want to allow for different
Ni values, so miners who want to contribute capacity N0 only
need one capacity commitment, and do not have to split it up in
N0/N capacity commitments of size N, and then run a proof for
each chunk separately.

For this, we define a distribution DN , N ? N which is
defined by sampling N values in [0, 1] at random, and then
outputting the largest of them.

DN ~ max {r1, . . . , rN : ri ? [0, 1], i = 1, . . . , N} (2)
With DN (t) we denote a sample of DN (or rather, a distribution which is very close to it) using randomness t to sample.
We now say that (ai, Ni) is of higher quality than (aj , Nj)
if (aj , Nj) ? (ai, Ni) ? DNj (hash(aj)) = DNi (hash(ai)).

It remains to show how to efficiently sample from the distribution DN for a given N. Recall that if FX denotes the cumulative distribution function (CDF) of some random variable X over [0, 1] and the inverse F-1 X exists, then F-1 X (U) for U uniform over [0, 1] has the same distribution as X. The random variable X sampled according to the distribution DN has CDF FX(z) = zN , since this is the probability that all N values ri considered in (2) end up being below z (and hence also their maximum). Therefore, if we want to sample from the distribution DN , we can simply sample F-1X (U) for U uniform over [0, 1], which is U1/N . In we want to sample DNi using randomness hash(ai), and hash outputs bit strings in {0, 1}256 instead of values in [0, 1], so we have to normalize:
DNi (hash(ai)) :=hash(a)/2 2561/N

Note that this introduces a tiny imprecision due to the fact that hash(a)/2 256 is uniform over a discrete set instead of the continuous interval [0, 1], but this can be safely disregarded.
Remark. Notice that the quality function described above has the property that the quality of block that a given miner pk can produce in a given time-step is fixed, regardless of which chain he chooses to extend. This property will be important to prevent the “mining multiple chains” attack which was described in Section I, as explained in the next section.

VI. THE BLOCKCHAIN FORMAT

A block chain is a sequence of blocks ß0, ß1, . . . Each block ßi = (fi, si, ti) is created by a miner and consists of three main parts, which we call “sub-blocks”. Each subblock starts with the index i that specifies its position in the block chain. Below, we outline the remaining components of the three sub-blocks of a block ßi, i > 0. The genesis block ß0 necessarily has a somewhat different format as it cannot depend on previous blocks:

• The HASH sub-block fi contains:

  – A 256-bit hash hash(fi-1) of the HASH sub-block from the previous block in the chain.

  – A “capacity proof” containing the miner’s identity pk (more details on this are given below).

• The TRANSACTION sub-block ti contains:
– A list of transactions (defined in more detail below).

• The SIGNATURE sub-block si contains:

  – The miner’s signature Sign(sk, ti) on the TRANSACTION sub-block ti associated with this block.

  – The miner’s signature Sign(sk, si-1) on the SIGNATURE sub-block si-1 associated with the previous block.

The links between consecutive blocks in the blockchain are illustrated in Figure 2. We will also refer to the hash subblocks as the proof chain, and the signature sub-blocks with the transactions as the signature chain. Solid arrows represent hashes, and dotted arrows represent signatures. Notice that while the signature and transaction sub-blocks are all linked together, the hash sub-blocks are only linked to each other and not to any signature or transaction sub-blocks.

A. Solution to the grinding problem:

By decoupling proofs from transactions we achieve security against grinding: for any capacity commitment (pk, ?), the miner pk cannot generate two (or more) correctly formatted hash blocks to be added to the proof chain.

The signature chain binds the transactions to the proof chain. If an honest miner (honest to be defined below) adds the ith block, the transactions corresponding to this proof
To prove this, we require that it is computationally hard to find two distinct accepting transcripts for the same challenge. The POC of satisfies this property (finding two accepting transcripts in their schemes amount to breaking collision-resistance of the underlying hash function).
The chain up to block i cannot be changed any more, even if an adversary controls all secret keys from miners that added the first - 1 blocks. Here the miner being honest means that she only signs a single block of transactions using the secretkey sk corresponding to her identity pk, and moreover keeps

sk secret. To see this, note that if we want to change the
transactions in block j < i while keeping the current proof
chain up to block i, then the signatures for blocks j, . . . , i
must be re-computed, which requires sk.

B. Transactions ATMcash is based on a secure 12 signature scheme

S = (SigParamGen, SigKeyGen, Sign, SigVerify)
and a POC protocol ? = (Init, Challenge, Answer, Verify) .
In the following we specify the three types of transactions
(for payments, capacity commitments and punishments) that we
allow in ATMcash. Payments. Coins are held and transfered by parties identified
by a verification key in the support of SigKeyGen.

A transaction transfers coins from benefactors
to n beneficiaries and has the formctx = (payment, txId,in, ~ out ~).

In order for a transaction to be considered valid, the
following conditions must be satisfied:

• txId: A unique, arbitrary transaction identifier. That is,
no two transactions in a blockchain can have the same
Identifier.

• out ~ : A list of beneficiaries and the amount they receive.
Specifically, out ~ = (out1, . . . , outm) with outi =(pki, vi), where:
– pki is in the support of SigKeyGen and specifies a
beneficiary, and – viis the number of coins that pki
is to be paid.

• A list of input coins to the transaction. Specifically,
in~ = (in1, . . . , inn), a list of n benefactors, each comprised
of a triple: inj = (txIdj , kj , sigj), where:

– txIdj is the identifier of a past transaction

– kj is an index that specifies a particular beneficiary
pkkj of the transaction txIdj

– sigj is a signature of (txId, txIdj , kj , out ~), which
verifies under key pkkj proving ownership of the the
beneficiary of transaction txId and binding the
coin to the beneficiaries.

In Bitcoin, the specification of payments is more general:

instead of specifying beneficiaries via their verification keys, recipients are specified by writing a script in a special (non-Turing-complete) scripting language called Bitcoin Script. The output coins of a transaction can then be redeemed by any party which can produce inputs which "satisfies" the script scr. In practice, ATMcash can be straightforwardly modified to accommodate such scripting; but in this work, for clarity of exposition, we assume that each payment recipient is specified by a verification key.

That is the kj th beneficiary of transaction txIdj is the jth benefactor of transaction txId.

txId is signed in order to avoid transaction malleability [https://en.bitcoin.it/wiki/Transaction Malleability](https://en.bitcoin.it/wiki/Transaction_Malleability)

1) No benefactor is referenced by more than one transaction in the blockchain (to prevent double-spending).

2) The sum of the input values to the transaction (i.e. the sum of the amounts provided by each benefactor) is at least the sum of the amounts paid to beneficiaries.

Note that some of the beneficiary identities may belong to the creator of the transaction, who may thus transfer money back to himself as "change": e.g. if the sum of the input values exceeds the total payment amount he wants to transfer to other parties.

VII. INSTANTIATION (plotting)

In this section we describe the concrete steps required for setting up, mining and paying in ATMcash. We give the instantiation for the second scheme (challenge from the past), outlined in Section V-C. The first (random beacon) scheme is almost identical, except that the challenge c is derived from the random beacon (and not by hashing a block from the chain).
16?0 is a Merkle-hash of all the labels in a hard to pebble graph. We can change ?0 to another value by simply changing a single label, which will not be noticed in the execution phase unless this particular label with its children is requested.

At setup we have to fix the security parameter ? to
be used for the signature and POC scheme. Moreover, we
must specify parameters and functions:

• time ? N specifies the length of a timeslot in minutes. It
should be sufficiently larger than the network propagation
time, but otherwise as small as possible. time = 1 seems
like a reasonable choice here.

• d ? N specifies that the challenge for block i is a function
of block i - d. A reasonable value is d = 120.

• Reward is a function such that Reward(i) specifies the
amount of coins a miner gets for mining the i
th block.

• Quality is function that takes as input a capacity commitment
(pk, ?) for capacity of size N together with a
challenge/answer pair (c, a). If Verify(pk, ?, c, a) 6= 1
(i.e., it is not a valid POC proof transcript), the
Quality function outputs -8. Otherwise the output is
(with DN as defined in Section V-B):

Quality(pk, ?, c, a) = DN (hash(a)) .
In order to decide which of two given proof chains is the
"better" one, we also need define the quality of a proof chain
f0, . . . , fi, which we'll denote with QualityPC(f0, . . . , fi).
Each hash block fj contains a proof (pkj , ?j , cj , aj), and we
let vj = DNj (aj) denote the quality of the jth proof in the
chain. For any quality v ? [0, 1], we denote with N(v) = min{N ? N : Pr[v ? w | w ? DN] = 1/2}
the capacity required to get a better proof than v on a random
challenge with probability 1/2.

Note that N(vj) will usually be around the total storage of all miners that were active
when the block was mined. With this definition, a natural
measure for the quality of the chain would be simply the sum17
Pi j=1 N(vj).

The problem with this measure is that if some
miner finds an extremely good proof, say N(v) is 1000 times
larger than the total storage (this will happen roughly every

1000 blocks), then the miner could withhold his proof, and
1000 blocks later generate a fork using this proof followed
by 999 arbitrarily bad proofs for the remaining blocks. To
avoid such deep forks, we cap proofs that are too good by
saying that vj cannot contribute more to the sum than, say 10
times the median of the last 101 blocks (the median gives a
good approximation of the total capacity that is dedicated towards
mining).

Formally, let Nˆ(vj) be recursively defined as
Nˆ(vj) = max{N(vj) , 10·median(N(vj-101), . . . , N(vj-1)}

Another reason why defining the quality simply as
Pi j=1 N(vj) is problematic, is that the total contributed capacity
can increase drastically over time. In this case, in order to
come up with a chain whose quality is better than the quality
of the real chain it is sufficient to dedicate much less than the
total capacity that is currently devoted towards mining. For this
reason, we only take the last 1000 blocks into account when
computing the quality:

QualityPC(f0, . . . , fi) = X j=max{1,i-1000} Nˆ(vj)

We start summing with j = 1, not j = 0, as the genesis block (still to
be defined) will not contain a proof.

Finally, a genesis block ß0 = (f0, s0, t0) is generated and
published; it has a format different from other blocks. The
transactions block contains only one capacity commitment t0 =
(commit, txId,(pk0, ?0)), the hash block f0 contains only
some random string,18 and the signature block s0 contains
the signature Sign(sk0, t0) of the transactions block (but not
of the previous signature block, as there is none).

Initialize Mining. In order to dedicate N bits of storage for
mining, a party generates an identity and a capacity commitment
(pk, sk) ? SigKeyGen , (?, S?) := Init(pk, N) .

It stores S? (of size N) and sk locally. The miner then generates and
publishes a transaction ctx = (commit, txId,(pk, ?)). Once ctx has been
added asa transaction to the hash chain, the miner can start mining as described next.

Mining:

As we enter time slot i, the miner retrieves the
so-far-best blockchain ß0, . . . , ßi-1 (that is, the chain maximizing
Quality PC(f0, . . . , fi-1). We assume that the miner
“honestly” stores capacity S? and the corresponding commitment
(pk, ?) has been added to some transcription block tj , j = i-1
in this chain.

Next, the miner computes the randomness for the
challenge sampling by hashing the hash block that is d blocks
in the past c := hash(pk, fi-d).

From this c we then compute the challenges cp, ccv. The miner
computes the POC answer a := Answer(pk, S?, cp) .

If q := Quality(pk, ?, c, a) is very high, so it has a realistic
chance to end up as the best answer of the entire network, the
miner generates a hash block fi = (i, hash(fi-1), pi), where
Pi is 20 (pk, ?, c, j, q, a, acv), where acv := Answer(pk, S?, ccv) .
is the output of the commitment verification (for efficiency reasons we
only execute commitmentverification at this point).

Then the miner retrieves transactions (typically, giving
priority to the ones paying the highest fees), checks
their correctness, and adds the valid ones to a transaction
Block.

. It then computes the signature block si =
(Sign(sk, si-1), Sign(sk, ti)) and publishes block ßi =(fi, si
, ti), hoping that it will end up in the blockchain,earning the
miner Reward(i) coins, plus the transactions fees of the
transactions in the block.

Transaction. Any party can generate a transaction and publish
it. If it is correctly generated, it should ultimately end up
in the blockchain. We have already described the format and
semantics of the three types of transactions.

Or better, some kind of timestamp like a sentence from a newspaper of
the day, as is done in Bitcoin, to show that the genesis block was not generated
before some date “publishes” and “retrieves” we mean that a party sends or downloads
something from the network. Typically, there would be some servers that
organize the data, i.e., keep track of the best chains and collect transactions,

so a miner would only interact with one or a few such servers it trusts.

To evaluate ATMcash, we have implemented a prototype
in Go, using SHA3 in 256-bit mode as the hash function. The
prototype uses the graphs from [24], and forces a cheating
prover to store at least ?(N/ log(N)) bits in order to effi-
ciently generate proofs. Given that the network infrastructure
is very similar to Bitcoin, we are mainly interested in three
quantities: time to initialize the capacity (graph), size of the proof,
and time to generate and verify the proof. The experiments
were conducted on a server equipped with an Intel i5-4690K
Haswell CPU and 8GB of memory. We used an off-the-shelf
hard disk drive, with 2TB of capacity and 64 MB of cache.
Our proof-of-capacity library and ATMcash prototype are
available at: https://github.com/kwonalbert/ATMcash.
Time to Initialize.

To start mining ATMcash, the clients must first initialize their capacity.
This involves computing all the hashes of the nodes, and
computing the Merkle tree over the hashes (plotting). In Figure 3,
we show the initialization time for capacity s of size 8 KB to
1.3 TB. As expected the time to initialize grows linearly with
the size of the capacity ; at 1.3 TB, it takes approximately 41
hours to commit the graph. While expensive, we note that this
procedure is done only once when the miner first joins the
ATMcash network, and will use the initialized capacity over and
over again. In fact, we require capacity initialization to non-trivial
time, because an extremely fast capacity initialization would
make re-using the same capacity for different commitments a
viable strategy (Section V-C).

Size of the Proof. A proof (i.e., a full solution to the puzzle)
in ATMcash consists of the hashes of the challenge nodes
and their parents, in the Merkle inclusion proofs

Time to Generate/Verify the Proof. Unlike Bitcoin, generating
an answer for a puzzle (i.e., generating a proof-of-capacity)
takes little time.

In Bitcoin, the miner is expected to work through most of the epoch in
an attempt to find a preimage of a hash with sufficient difficulty.
In ATMcash, assuming a mineris storing the capacity correctly,
the miner needs to only perform (1) lookups in the disk to find their solution

which takes fraction of a second.
For instance, it takes < 1 msto read a single hash from the disk.
Only if the miner believes its answer is of very good quality will it generate the full proof, but even this takes seconds, not minutes.

As outlined above, our proofs are substantially bigger than Bitcoin's, and require more than just one hash evaluation to verify. However, for an active currency, we can still expect the size and verification time for the proofs added with every block to be marginal compared to the size of the transaction added with every block, and the time required to verify that the transactions are consistent. This indeed shows that though it may take seconds to generate the proof, verification takes a fraction of a second.

Energy:

Though our prototype was evaluated using a full CPU which wastes a lot of energy, one could in principle run the prover and the verifier on a energy-efficient devices such as Raspberry Pi [3]. An efficient microcontroller consumes less than 10 W of power, and most miners will only open one node per time-step since the quality of their answers will likely be bad. To get an upper bound on the power requirement, let us assume that there are 100,000 miners, each with 1 TB of capacity , and about 1% of the miners mine "good" answers which they will want to generate a full answer. Then we have
$10W \cdot 100000 \cdot 0.01s + 10W \cdot 1000 \cdot 20s = 210000J/block$
which translates to 210 kJ/min if we add one block a minute.
In contrast, Bitcoin on average uses 100 MW, so it consumes 6 GJ/min, which is several orders of magnitude larger. We note that this 1% figure is a very conservative bound, so the difference could be even larger in practice.

GAME THEORY OF ATMcash

The miners in a cryptocurrency are strategic agents who seek to maximize the reward that they get for mining blocks. As such, it is a crucial property of a cryptocurrency that "following the rules" is an equilibrium strategy: in other words, it is important that the protocol rules are designed in such a way that miners never find themselves in a situation where "cheating" and deviating from the rules yields more expected profit than mining honestly.

Intuitively, ATMcash mining is modeled by the following
n-player strategic game. Game-play occurs over a series of
discrete time steps, each of which corresponds to a block
being added to the blockchain. At each time step, each player
(miner) must choose a strategy, specified by:

• which blocks to extend (if any), which transactions to include in the new blocks, and
• which extended blocks to publish (if any).

We present the details of our game-theoretic analysis in the
unpredictable-beacon model, and remark that the analysis can
be extended to cover the other models too.

A. Game-theoretic preliminaries

The standard game-theoretic notion for a strategic game
which occurs over multiple time steps (rather than in “one
shot”) is the extensive game. In order to accurately model the
probabilistic aspects of the ATMcash protocol (e.g. the unpredictable
beacon), we consider extensive games with chancemoves:
this is the standard game-theoretic notion to capture
extensive games which involve exogenous uncertainty. The
uncertainty is modeled by an additional player called Chance
which behaves according to a known probability distribution.
In the ATMcash setting, every player (including Chance)
makes an action at every time step. A player’s action consists
of choosing whether and how to extend the blockchain, and
the action of Chance determines the value of the unpredictable
beacon for the next time step.

An extensive game is commonly visualized as a game tree,
with the root node representing the start of the game. Each
node represents a state of the game, and the outward edges
from any given node represent the actions that players can
take at that node. Leaf nodes represent terminal states: once
a leaf is reached, the game is over. In accordance with the
literature, we refer to paths in the game tree (starting at the
root) as histories; and histories which end at a leaf node are
called terminal histories.

Definition X.1 (Extensive game). An extensive game G =hN, H, fC, I~, ~ui is defined by:

• [N], a finite set of players.

- H, the set of all possible histories, which must satisfy the following two properties:

– the empty sequence () is in H, and

– if (a1, . . . , aK) ? H then for all L = K, it holds that (a1, . . . , aL) ? H.

We write Z ? H to denote the subset consisting of all terminal histories. For any history h,A(h) = {a : (h, a) ? H} = ×i?[N]Ai(h) denotes the set of action profiles that can occur at that history, and Ai(h) denotes the set of actions that are available to player i at history h.

- f(·, h) is a probability measure on AC(h), where h ? H and C denotes the Chance player.

- I~ = (I1, . . . , IN), where each Ii is a partition of H into disjoint information sets, such that Ai(h) = Ai(h0) whenever h and h0 are in the same information set I ? Ii. Let Ai(I) denote the set of actions that are available to player i at any history in information set I.

- ~u = (u1, . . . , uN), where each ui: Z ? R is the utility function of player i.

Imperfect information and information sets. An extensive game is said to have perfect information if at any point during game-play, every player is perfectly informed of all actions taken so far by every other player. In the context of ATMcash, players are only aware of each others' announced actions: for example, if Alice tries extending several blocks and then only announces one of them, then Bob does not know about the other blocks that Alice tried to extend. Thus, ATMcash is a game of imperfect information.

The information that players do not know about other players' actions is modeled by the partitions I~ = (I1, . . . , IN) in Definition X.1. Each Ii is a partition of H into disjoint information sets, and for each i ? [N] and any pair of histories h, h0 ? I in a particular information set I ? Ii, player i cannot tell the difference between game-play at h and at h0.

Example X.2 (“Match my number” game).

Consider a simple two-player game in two rounds: in the first round, player 1
chooses a number a ? {0, 1, 2}. In the second round, player
2 chooses a number b ? {0, 1, 2}. Player 2 wins if b = a, and
player 1 wins otherwise. Clearly, player 2 can always win if
he knows a.

However, we consider a game of imperfect information
where player 2 must choose b without knowing a: in particular,
suppose player 2 only learns whether a = 0. Then, the
histories (a = 1) and (a = 2) are in the same information
set in the partition A strategy of a player in an extensive game
is defined by specifying how the player decides his next
move at any given history. In games of imperfect information,
the player may not know which history he is at, so we
instead specify how the player decides his next move at any
information set.

Definition X.3 (Strategy profile).

A strategy profile ~a =(a1, . . . , aN) of an extensive game
G = hN, H, fC, I~, ~ui specifies for each player i ? [N] and each
information set I ? Ii a probability distribution ai(I) over the action set
Ai(I). We say that ai is the strategy of player i.

Let I(h) denote the information set in which history h lies.

The probability that a history h occurs under strategy profile
a is denoted by Pr~a[h], and the probability that a history h0
occurs given that h occurred is denoted by Pr~a[h0 |h].

Recall that the utility functions u1, . . . , uN were originally
defined on inputs in Z, the set of terminal histories. For each ?
[N], we now define ui(~a) to be the expected utility of
player i given the strategy profile ~a. That is,ui(~a) = Xh ?Z ui(h) · Pr ~a[h].

Moreover, we define ui(~a|h) to be the expected utility of
player i given ~a and given that history h has already occurred.
That is,ui(~a|h) = X h0?Z ui(h 0) · Pr ~a[h 0 |h].

Equilibrium notions. The most widely known equilibrium concept for a strategic game is the Nash equilibrium [23], given in Definition X.4. Intuitively, in a Nash equilibrium, each player's strategy is a best response to the strategies of the other players.

For a strategy profile ~a, we write ~a-i to denote (aj)j?N,j6=i, that is, the profile of strategies of all players other than i; and we use (a 0 i, ~a-i) to denote the action profile where player i's strategy is a 0 i and all other players' actions are as in ~a.
Definition X.4 (Nash equilibrium of an extensive game). Let G = hN, H, f, I~, ~ui be an extensive game. A strategy profile ~a is a Nash equilibrium of G if for every player i ? [N] and every strategy a 0 i of player i, ui(~a) = ui(a 0 i, ~a-i).

The Nash equilibrium concept was originally formulated for one-shot games, and it is known to have some shortcomings in the setting of extensive games. Informally, the Nash equilibrium does not account for the possibility of players changing their strategy partway through the game: in particular, there exist Nash equilibria that are not "stable" in the sense that given the ability to change strategies during the game, no rational player would stick with his equilibrium strategy all the way to the end of the game.

Example X.5 ("Unstable" game). Consider a simple two player game in two rounds: in the first round, player 1 chooses either strategy A or B. In the second round, player 2 chooses either strategy C or D. The game tree is given below, where the notation (x, y) at the leaves denotes that player 1 gets payoff x and player 2 gets payoff y if that leaf is reached.

To address these shortcomings of the Nash equilibrium concept for extensive games, an alternative (stronger) notion has been proposed: the sequentially rational Nash equilibrium. This stronger concept ensures that players are making the best decision possible at any point during game-play. In a game with imperfect information, it is necessary to consider not only the strategy profile, but the players' beliefs at any point in time about how game-play arrived at the current information set. A strategy profile which takes into account players' beliefs is called an assessment.

Definition X.6 (Assessment). An assessment in an extensive

game is a pair (~a, ~μ) where ~a = (a1, . . . , aN) is a strategy
profile and ~μ = (μ1, . . . , μN) is a belief system, in which each μi
is a function that assigns to every information set in
Ii a probability measure on histories in the information set.

In Definition X.6, μi(I)(h) represents the probability that
player i assigns to the history h ? I having occurred,
conditioned on the information set I ? Ii having been reached.
For each i ? [N], we now define ui((~a, ~μ)|I) to be the
expected utility of player i at the information set I ? Ii,
given the strategy profile ~a and belief system ~μ. That is,
ui((~a, ~μ)|I) = X h?I ui(~a|h) · μ(I)(h).

We write ui((~a, ~μ)) to denote ui((~a, ~μ)|{()}), that is, the
expected utility for player i at the beginning of the game.
An assessment (a, μ) is said to be sequentially rational if
for every i ? [N] and every information set I ? Ii, the strategy
of player i is a best response to the other players' strategies,
given i's beliefs at I. A formal definition follows.

Definition X.7 (Sequentially rational assessment). Let G =
hN, H, f, I~, ~ui be an extensive game. An assessment (~a, ~μ) is
sequentially rational if for every i ? [N] and every strategy
A 0 i of player i, for every information set I ? Ii , it holds that ui((~a, ~μ)|I) = ui(((a 0i, ~a-i), ~μ)|I).

Definition X.7 almost fully captures the idea players should
be making the best decision possible given their beliefs at any
point during game-play. To fully characterize a sequentially
rational Nash equilibrium, we require additionally that the
beliefs of the players be consistent with ~a. For example, if an
event occurs with zero probability in ~a, then we require that
the players also believe that it will occur with zero probability.

Definition X.8 (Consistent assessment).

Let G = hN, H, f, I~, ~ui be an extensive game. A strategy
profile ~a is said to be completely mixed if it assigns
positive probability to every action at every information set.
An assessment (~a, ~μ) is consistent if there is a sequence
((~an, ~μn))n?N of assignments that converges to (~a, ~μ) in
Euclidean capacity , where each ~a n is completely mixed and
each belief system ~μ n is derived from ~a
n using Bayes' rule.

Finally, we arrive at the definition of a sequentially rational Nash equilibrium.

Definition X.9 (Sequentially rational Nash equilibrium). An assessment is a sequentially rational Nash equilibrium if it is sequentially rational and consistent.

B. Game-theoretic analysis of ATMcash

In order to analyze the game-theoretic properties of ATMcash mining, we define an extensive game, ATMcashGame, which models the actions that miners can take, and the associated payoffs. To facilitate analysis, we simplify the action capacity of the game as much as possible while still accurately modeling the incentives of ATMcash miners. Concretely:

• We do not include the action of creating a capacity commitment because (as discussed in Section V-A under "Mining") we can assume that rational miners will commit to all the capacity they have, and nothing else.

• We do not include the action of creating transactions because such actions do not affect the rewards that players receive from mining blocks, except in the case of punishment transactions. To deal with the case of punishment transactions, we define the payoff of a player who mines multiple blocks in the same time step to be zero. This payoff function exactly captures that of a miner in the actual ATMcash protocol, because it is a dominant strategy for each other miner to create a punishment transaction (including a positive transaction fee) if she sees that a cheating player has mined multiple blocks in a time step, and hence we can assume that the cheating player will surely be punished at a later point in the protocol. Since the punishment penalizes the cheating player by the amount of the mining reward, it follows that the cheater's overall utility for the time step in which he cheated is zero.

• We do not explicitly model the amount of capacity that each player has. Instead, we study the two critical cases: in our initial analysis, we assume that no miner controls

more than 50% of the capacity committed by active miners.
Then, we discuss potential issues that arise if a miner
does control a majority of the capacity.

Let B denote set of all blocks as defined in Section VI. For any number of
players N ? N, any number of time steps K ? N, and any
reward function ?: N ? N, we define the extensive game

ATMcashGame?,K,? = hN, H, fC, I~, ~u i as follows:

• The set H of histories is defined inductively as follows:

– The action set of the Chance player AC(h) ={0, 1}
m is the same for every history h.
– The empty sequence () is in H, and Ai(()) =
{(Ø, Ø)} for each i ? [N].

– Let h = (h 0, a) be any non-terminal history where the latest action profile
a = (a1, . . . , aN , aC) consists of the actions of each player in [N] ? {C} at
Later in this section, we address what happens if a miner gains additional
capacity (or loses some capacity) during the game.

We remark that the standard way to model this would be to assign a type
to each player, representing how much capacity he has.

history h 0, and for each player i ? [N], the action ai = (Si, Ti)
is a pair of sets. Then for any i ? [N], the action set Ai(h) of player i at h is
Ai(h) = P(T) × B where T =[i?[N] Ti An action ai = (Si, Ti) can be interpreted as
Follows:

The set of blocks from the previous time step which player i attempts to extend in this
time step, and Tiis the set of extended blocks which player i announces in this time step.

• The probability measure f(·, h) is uniform over {0, 1} m.
• For each i ? [N], we define the partition Ii by an
equivalence relation ~i

. The equivalence relation ~i is defined inductively as follows (we write [h]i
to denote the equivalence class of h under ~i):

– [()]i = {()}, that is, the empty sequence is equivalent
only to itself.

– [(h,((S1, T1), . . . ,(SN , TN), aC))]i = {(h 0,((S 0 1, T0 1), . . . ,(S 0 N , T0 N), a0 C)) ? H : h ~i h 0 ? Si = S 0 i ? Ti = T 0i ? aC = a 0 C ??j 6= i, Tj = T 0 j

- where h and h 0 are histories and the pairs (Sj , Tj) and (S 0 j , T0 j) are actions of player j. That is, two histories are equivalent under ~i if they are identical except in the "first components" Sj of the actions (Sj , Tj) taken by the players other than i.

• ~u = (u1, . . . , uN), where each ui : Z ? R is defined as
described below. For a history h, let beac(h) denote the
sequence of actions taken by the Chance player in h, and
let beacj (h) denote the jth action taken by the Chance
player in h. For a block B, let B.c denote the challenge
We define Quality(B, c) = (Quality(B) if B.c = c0 otherwise.

Similarly, let QualityPC((B1, . . . , BL),(c1, . . . , cL)) = (QualityPC((B1, . . . , BL)) if ?i ? [L], Bi .c = ci 0 otherwise.

Let blocks(h) denote the sequence of "winning blocks"
at each time step in the game, defined inductively:

– blocks(()) = () – blocks(h = (h 0 ,((S1, T1), . . . ,(SN , TN), aC))) = arg maxB?T (Quality(B, beac|h|(h))), where T = ?i?[N]Ti

Let blocksj (h) denote the jth block in the blockchain.

We assume that the winning block is unique at each time
Let winners(h) denote the sequence of players who announce
the winning block at each time step in the game, defined inductively as follows:

This can be achieved by breaking ties between blocks in an arbitrary way.

Note that it is not possible for two different players to announce exactly the
same (valid) block, because each block contains the miner's identity.

– winners(()) = () – winners(h = (h 0 ,((S1, T1), . . . ,(SN , TN), aC))) = arg maxi?[N] maxB?Ti (Quality(B, beac|h|(h))).

Let winnersj (h) denote the jth winner in the sequence
winners(h). Let only one j(i, h) be an indicator variable
for the event that player i's jth action (Si, Ti) in the
history h does not mine multiple blocks, i.e. |Ti | = 1.

Finally, the players' utility functions are defined as follows:

for a terminal history h of length K, ui(h) = X j?[K] di,winnersj (h) ·onlyonej
(i, h)·?(blocksj (h)), where di,j is the Kronecker delta function26. That is, a
player’s utility is the sum of the rewards he has received
for announcing a winning block (in the time steps where
he has announced at most one block).

By Definition X.10, for any i ? [N], for any histories h, h0
in the same information set I ? Ii, it holds that blocks(h) =
blocks(h 0).
Thus, we can associate a unique blockchain with
each information set: we define blocks(I) to be equal to
blocks(h) for any h ? I. Similarly, beac(h) = beac(h 0) for
any h, h0 ? I in the same information set I, so we define
beac(I) to be equal to beac(h) for any h ?For a block B ?
B and a challenge c ? Challenge, wedefine Extendi(B, c)
to be the block generated by player i when mining the next block
after B using the POC challenge c (see Section VII for exact block format).

Theorem X.11.

For any number of players N, any number
of time steps K ? N, and any reward function ? : N ?
N, let ~a = (a1, . . . , an) be a pure strategy profile of
ATMcashGame?,K,?, defined as follows: for each i ? [N],
for any information set I ? Ii such that I 6= {()},
ai(I) (({blocksj (I)} , {Extendi(blocksj (I), beacj (I))})) = 1,
where j = 1 is the length of the histories in information set
27. That is, player i’s next action at information set I is
aˆi = ({blocksj (I)} , {Extendi(blocksj (I), beacj (I))}).

Then ~a is a Nash equilibrium of ATMcashGame?,K,?.
Proof. Take any player i ? [N]. By the definition of Extend,
for any information set I ? Ii with I 6= {()}, the quality v of the extended blockchain
v = QualityPC((blocks(I), Extendi(B, beacj (I))), beac(I))
is the same for any block B which was announced at time step
j. Therefore, no utility can be gained by choosing any block B
over any other block B0 to extend: that is, ui(~a) = ui(a 0 i, ~a-i)
for any strategy a 0 i which distributes probability over actions
of the form (S, T) where |S| = 1.

Moreover, not extending any block or extending multiple
blocks precludes a player from being the “winner” and receiving
the reward in this time step, so extending a block is preferable

to not extending any block. That is, ui(~a) = ui(a
0 i , ~a-i) for any strategy a 0 i which assigns non-zero probability to any
action of the form (S, T) where |S| 6= 1.

Kronecker delta function: di,j = 1 if i = j, and 0 otherwise.
27All histories in an information set must be of the same length.
We have shown that ui(~a) = ui(a0 i , ~a-i) for all strategies A 0 i of player i.
The theorem follows.Theorem X.12. Let ? = {Init, Challenge, Answer, Verify} be
a proof of capacity . For any number of players N, any number
of time steps K ? N, and any reward function ? : N ? N, let
(~a, ~µ) be an assessment of ATMcashGame?,K,? Where:

• ~a and aˆi are defined as in Theorem X.11, and for
each n ? N, we define ~a n to be the completely mixed
strategy profile which (at history h) assigns probability 1/|Ai(h)| n to every action except aˆi
, and assigns allremaining probability to aˆi.

• ~µ is derived from ~a using Bayes' rule in the following
way: ~µ = limn?8 ~µ n, where for each n ? N, ~µ n is
derived from ~an using Bayes' rule.

Then (~a, ~µ) is a sequentially rational Nash equilibrium of
ATMcashGame?,K,?.

Proof. Let I ? Ii be any information set of player i in
ATMcashGame?,K,?, and let L be the length of histories in
I. It follows from Definition X.10 that the expected utility of
player i at I is ui((~a, ~µ)|I) = X j?[L] di,winners · only one j
(i, h) · ?(blocksj (h)) + u 0 I ((~a, ~µ)),where u 0 i is the utility function
of player i in the game ATMcashGame?,K-L,?. Since winners,
onlyone, and blocks are invariant over histories within any given information set,
the summation term can be computed explicitly by player
i at I. Hence, in order to maximize his expected utility
at I, the player needs simply to maximize u 0 I ((~a, ~µ)). Let (~a|K-L, ~µ|K-L) denote the
assessment (~a, ~µ) for the first K - L time steps of the game. By Theorem X.11, ~a|K-L
is a Nash equilibrium of ATMcashGame?,K-L,?. Since ~µ is
derived from ~a by Bayes' rule, it follows that ui((~a, ~µ)|I) =
ui(((a 0 i, ~a-i), ~µ)|I) for any strategy a 0 i of player i.

Applying this argument for every I, we conclude that (~a, ~µ) is sequentially
rational in ATMcashGame?,K,?.

By construction, limn?8 ~a n = ~a and ~µ = limn?8 ~µ n, so

(~a, ~μ) is consistent. The theorem follows.

Parameters:.

The ATMcash Game is parametrized by N and
K. It is natural to ask: do we require that the number of
miners N is fixed in advance, or that the blockchain will end
after a certain number K of time-steps? The answer is no.

Theorem X.12 gives a sequentially rational Nash equilibrium
in which each player's strategy is independent of N, and so
it makes sense for each miner to play this strategy even if N
is unknown or changes over time. In light of this, from each
rational player's point of view, K can be considered to be
the number of time-steps that he intends to participate in the
Buying of capacity . Players' strategies in equilibrium do not
depend on the amount of capacity that (they believe) other players
possess. Also, we showed above that the equilibrium strategies
are robust to changes in N. Hence, if a player's amount of
capacity changes (e.g. he buys/sells a hard disk), then he can
simply create a new capacity commitment, and then behave as a
"new player" with the new amount of capacity.

The "51% Attack".

If a player P controls more than half of the total capacity that
belongs to active miners, then following
the protocol rules is no longer a Nash equilibrium, because
whichever branch of the blockchain P chooses to mine on
will eventually become the highest-quality chain. Thus, P can
decide arbitrary rules about which blocks to extend, and the
other players will be incentivized to adapt their strategies
accordingly. Moreover, P can prevent certain transactions
from ever getting into the blockchain, by refusing to extend
blocks which contain these transactions – as a consequence,
P can mine multiple blocks per time-step without ever being
punished. This attack was first analyzed by [20] in the context
of Bitcoin, which suffers from the same problem (with respect
to computing power rather than capacity).

It may seem unrealistic that a single party would control
more than half of the total capacity that belongs to active miners
in a widely adopted currency. A more realistic concern could

be that a large group of miners (in a mining pool) may
acquire more half of the total capacity . However, under the
assumption that each miner is an individual strategic agent, we
consider it unlikely that such a mining pool could do much
damage: for this, a large group of self-interested and relatively
anonymous agents would have to coordinate and trust each
other throughout the duration of an attack. In particular, each
rational miner in the pool must be convinced that he will get
his share of the attack profits, and it seems highly unlikely
that a large group of anonymous people would all trust each
other so. The improbability of a 51% attack by a mining
pool is supported by recent events: when a large mining pool
(ghash.io) was nearing 50% of Bitcoin computing power
in 2014, self-interested miners started leaving the mining pool
in order to avoid destabilizing the currency

CONCLUSION

We have presented ATMcash, a cryptocurrency that uses
efficient proofs of capacity instead of energy-intensive proofs
of work to maintain a public ledger of all transactions. We
have described a variant of a proof-of-capacity protocol that is
more suitable for cryptocurrencies, and modified the structure
of the hash chain and transactions to address some of the
issues of other cryptocurrencies. We have also demonstrated
the feasibility of ATMcash through a prototype, and show
that maintaining a public ledger could be much more efficient
with proof-of-capacity . Finally, we do a game-theoretic analysis
of ATMcash modeled as an extensive game, and prove that
it satisfies strong equilibrium properties.

A. Proof-of-capacity Parameters

The two Pocapacity constructed in have the following
efficiency/security properties. Below thash denotes the time
17required to evaluate the underlying hash function hash :

{0, 1}* ? {0, 1} L on inputs of length 2L (to hash an input of
length m · L takes time m · thash by using Merkle-Damgard), °
For a given n, the number of nodes of the underlying graph,
an honest prover P must dedicateN = 2 · n · L bits of storage
(L · n for the labels, and almost the samefor the values required
to efficiently open the Merkle tree commitment).

ATMcash uses the Shabal256 hash function, which below
we will denote with H(·). To mine ATMcash, a miner first
initialises his disk capacity as follows: he picks a nonce µ and
an account identifier (which is a hash of a public key) Id, and
then computes iteratively 4096 values x0, x1, . . . ? {0, 1}
256 As x0 = H(Id, µ) and (4) xi+1 = H(xikxi-1k . . . kx0) for i = 0, . . . , 4095 . (5)
The miner then stores s0, . . . , s4095 where si = xi ? x4096.
Each block is called a "scoop", and the 4096 scoops
together are called a "plot". The miner is supposed to store
as many plots as he can (using different nonces) until all the
dedicated capacity is filled. To compute a plot, one must hash
4096·1+40962 ˜ 8 million 256-bit blocks. In the following we
assume for simplicity that there is just one plot s0, . . . , s4095.
Efficiency. Once every few minutes, a new block gets added
to the hash-chain. At this point the miner can compute a
designated (public) index i ? {0, . . . , 4095} and must look up
the value.

This then determines if the miner "wins" and
thus can add the next block to the block chain30. Note that this
requires accessing a constant fraction of the entire dedicated
disk capacity (i.e. one block per plot, or 0.024%), every time a
new block gets mined. Moreover, in order to verify that a miner
"won" and can add a block, it is necessary to recompute the
entire plot from the initial inputs (Id, µ), which, as mentioned
above, involves hashing over 8 · 106 blocks.

Time-memory trade-offs. We observe that ATMcash allows
for a simple time-memory trade-off: instead of storing an
entire plot s0, . . . , s4095, a miner can initially compute and
store only the value x 4096. The miner then re-computes the
required scoop si at a given time-step, but only if i is
sufficiently small (say, i = 10). This would require hashing
only at most 50 blocks. Thus, the miner will get a shot at
adding a block only at 10/4095 ˜ 0.25% of the time slots,
but now also only requires a 1/4095 ˜ 0.025% fraction of the
Note that in equation (4), a freshly computed block xi is prepended to the
previous input. This is important as Shabal256 is an iterated hash function:
appending instead of prepending would bring the number of hashes required
to compute a plot down to linear (instead of quadratic) in the length of the
plot, but at the same time would allow for much more dramatic time-memory
trade-offs than the ones outlined below.

To be precise, the miner computes x0, . . . , xi and sets si = xi ?x4096.

REFERENCES

[1] Bitcoin network graphs. http://bitcoin.sipa.be/index.html.
[2] ATMcash. http://ATMcash.info.
[3] Raspberry pi. www.raspberrypi.org.
[4] Slasher: A punitive proof-of-stake algorithm. https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm.
[5] N. Anderson. Mining Bitcoins takes power, but is it an "environmental disaster"?, April 2013. http://tinyurl.com/cdh95at.
[6] G. Ateniese, I. Bonacina, A. Faonio, and N. Galesi. Proofs of capacity : When capacity is of the essence. In M. Abdalla and R. D. Prisco, editors, SCN 14, volume 8642 of LNCS, pages 538–557. Springer, Sept. 2014.
[7] G. Ateniese, R. C. Burns, R. Curtmola, J. Herring, L. Kissner, Z. N. J. Peterson, and D. Song. Provable data possession at untrusted stores. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, ACM CCS 07, pages 598–609. ACM Press, Oct. 2007.
[8] K. D. Bowers, A. Juels, and A. Oprea. Proofs of retrievability: theory and implementation. In CCSW, pages 43–54, 2009.
[9] H. Buhrman, R. Cleve, M. Koucky, B. Loff, and F. Speelman. Com- ´ puting with a full memory: catalytic capacity . In Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014, pages 857–866, 2014.
[10] R. Di Pietro, L. Mancini, Y. W. Law, S. Etalle, and P. Havinga. Lkhw: a directed diffusion-based secure multicast scheme for wireless sensor networks. In Parallel Processing Workshops, 2003. Proceedings. 2003 International Conference on, pages 397–406, 2003.
[11] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In E. F. Brickell, editor, CRYPTO'92, volume 740 of LNCS, pages 139–147. Springer, Aug. 1993.
[12] S. Dziembowski. Proofs of capacity and a greener bitcoin, June 2013. Presentation at Workshop on Leakage, Tampering and Viruses, Warsaw. https://sites.google.com/site/warsawcryptoworkshop2013/abstracts.
[13] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of capacity . In CRYPTO 2015, 2015.
[14] S. Dziembowski, T. Kazana, and D. Wichs. One-time computable selferasing functions. In Y. Ishai, editor, TCC 2011, volume 6597 of LNCS, pages 125–143. Springer, Mar. 2011.
[15] P. Golle, S. Jarecki, and I. Mironov. Cryptographic primitives enforcing communication and storage complexity. In M. Blaze, editor, FC 2002,

volume 2357 of LNCS, pages 120–135. Springer, Mar. 2003.
[16] M. E. Hellman. A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory, 26(4):401–406, 1980.
[17] A. Juels and B. S. Kaliski Jr. Pors: proofs of retrievability for large files. In P. Ning, S. D. C. di Vimercati, and P. F. Syverson, editors, ACM CCS 07, pages 584–597. ACM Press, Oct. 2007.
[18] N. P. Karvelas and A. Kiayias. Efficient proofs of secure erasure. In Security and Cryptography for Networks - 9th International Conference, SCN 2014, Amalfi, Italy, September 3-5, 2014. Proceedings, pages 520–537, 2014.
[19] S. King and S. Nadal. Ppcoin: Peer-to-peer cryptocurrency with Proof-Of-Stake.
[20] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In Workshop on the Economics of Information Security, June 2013.
[21] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. In 2014 IEEE Symposium on Security and Privacy, pages 475–490. IEEE Computer Society Press, May 2014.
[22] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. http://bitcoin.org/bitcoin.pdf.
[23] J. F. Nash. Equilibrium points in n-person games. Proceedings of the National Academy of Sciences, 36(1):48–49, 1950.
[24] W. J. Paul, R. E. Tarjan, and J. R. Celoni. capacity bounds for a game on graphs. Mathematical systems theory, 10(1):239–251, 1976–1977.
[25] D. Perito and G. Tsudik. Secure code update for embedded devices via proofs of secure erasure. In D. Gritzalis, B. Preneel, and M. Theoharidou, editors, ESORICS 2010, volume 6345 of LNCS, pages 643–662. Springer, Sept. 2010.
[26] S. Valfells and J. H. Egilsson. Minting money with megawatts: How to mine bitcoin profitably, September 2015. http://www.researchgate.net/publication/278027487 Minting Money With Megawatts - How to Mine Bitcoin Profitably.

APPENDIX