Proofs of Space

Matteo Campanelli
Protocol Labs

binarywhales.org  matteo@protocol.ai

ASCRrypto, Oct 3rd 2023
Yesterday’s Talk  
(by Arantxa)

Witness w for a relation $R(x,w)$

**Today:** proofs yes; but proofs of ”resources”

Can we construct a proof which persuades we are *spending a certain (amount of) resource X*?

Resource=Space

Resource=Time
Fighting Junk Mail (Spam)
Fighting Junk Mail Through \textit{PoW} (Proofs of Work)

[DworkNaor (Crypto92)]

Proof of "work" (or, CPU cycles)

I will not consider the email "received" unless the PoW checks

\textbf{Rationale:}
- Spammers’ model: "send cheaply a vast volume of mail"
- PoW now requires them to spend some amount per email (e.g. 0.0000005 cents)
- Not convenient any more in bulk
- \textbf{NB}: this is still OK for the honest (occasional) sender
A Simple Example of Proofs of Work

• Hash-based:
  • Puzzle: Sample random x (this is our challenge)
  • Solution: find r such that we find a certain amount of trailing zeroes
    \[ H(x \| r) = 0x(\ldots\text{anything}\ldots)000\ldots000 \]

• Intuition: If there are M trailing zeroes this requires roughly \(2^M\) on average
• To check the solution as a verifier: receive r; check trailing zeroes in \(H(x \| r)\)
  • (verifying is way faster than searching for the solution)
The single thing I want you to learn from this talk if you really really want to mentally leave right now

• PoW can save cat lives from spammers
  • (non-joke version: PoW is useful against spam)
• You can make a PoW by using some hash thing-y
  (assuming the hash output looks random enough)
Next on proofs of work  (I lied: I would you like to take away a little bit more out of this talk)

• **Definition**: What does a definition for PoW look like?

• **Applications**: What are other applications for PoW?

• **Caveats**: what are limitations of applying Proofs of Work?
Defining PoW: Syntax

Proof of Work

At a high level, a *Proof of Work* involves three algorithms:

- **Gen**(1^n) is a randomized algorithm that produces a *challenge* c.
- **Solve**(c) is an algorithm that solves the challenge c, producing a solution s.
- **Verify**(c, s) is a (possibly randomized) algorithm that verifies the solution s to c.

From Ball et al. 2017 (“Proofs of Useful Work”) https://ia.cr/2017/203
Defining PoW: What about Security?

• This standard definition template won’t work:
  • “For all \textup{PPT} * \textup{Adv} ... then Adv cannot win a certain game”

\begin{align*}
\text{Gen}(\ldots) & \rightarrow c \\
\text{c} & \\
\text{"bad" solution} & \\
\text{Verify}(c, \text{bad\_solution}) & = 1
\end{align*}

* \text{PPT}: Probabilistic Polynomial Time
Defining PoW: Intuition for the right def

• Denote by $T^* := \text{Time(Eval}(c))$ (honest solver’s time)
• For all adversaries running in time $\ll T^*$
• This happens with very high probability:

$\text{Gen(...)} \rightarrow c$

\[c\rightarrow^* \text{"bad" solution}\]

\[\text{Verify}(c, \text{bad_solution}) = 0 \text{ (we do not accept)}\]

Are we done?
One more thing: 
\textit{amortization should be impossible.}

If I ask you 100 challenges you should spend roughly 100 $T^*$ work.
Applications of Proofs of Work

• Fighting Spam/Denial-of-Service
• Bulletin Boards
  • (next block in a chain)
The **second** thing I want you to make sure you take from this talk in case you want to start snoozing now

**When defining PoW:**

- We do not use PPTs (as usual in cryptography); we specify the power of the adversary.
- We need to be careful about *amortization*.

**Applications:** denial of service in general; bulletin boards (e.g. Bitcoin)

Questions?
The third thing I want you to take from this talk before you start fantasizing about lunch

PoW have limitations.
Limitations/Caveats of PoW

• Waste
• The problem of ASICs*:
  • Or the ”Honest/Malicious gap”

*ASICs: Application-Specific Integrated Circuits
Issue: Waste of Energy in PoW
How to mitigate the problem of energy waste?

• Option 1:
  • Shifting to other resources (e.g. space)

• Option 2:
  • Actually making them “useful”: using that grinding for “natural problems”
Mitigating Waste: Make PoW Useful

• The classic hash-based PoW does “nothing useful”
• Can we obtain a PoW that is “useful”?  
• **Intuition**: “the grinding we are proving can be used for something else”
• **Example**: PrimeCoin (2013)  
  • Introduces PoW based on search for prime numbers
A Syntax for Proofs of Useful Work

Standard Proof of Work

At a high level, a Proof of Work involves three algorithms:

- Gen($1^n$) is a randomized algorithm that produces a challenge $c$.
- Solve($c$) is an algorithm that solves the challenge $c$, producing a solution $s$.
- Verify($c, s$) is a (possibly randomized) algorithm that verifies the solution $s$ to $c$.

Proof of Useful Work

- Gen($x$) is a randomized algorithm that takes an instance $x$ and produces a challenge $c_x$.
- Solve($c_x$) is an algorithm that solves the challenge $c_x$, producing a solution sketch $s$.
- Verify($c_x, s$) is a randomized algorithm that verifies the solution sketch $s$ to the challenge $c_x$.
- Recon($c_x, s$) is an algorithm that given a valid $s$ for $c_x$ reconstructs $f(x)$.

From Ball et al. 2017 ("Proofs of Useful Work") [https://ia.cr/2017/203](https://ia.cr/2017/203)
Examples of Proofs of Useful Work

**ORTHOGONAL VECTORS**

Is there \( u \in U, v \in V \):

\[ \langle u, v \rangle = 0? \]

(\( u, v \) represent disjoint subsets of \([n]\))

Orthogonal Vectors Conjecture

For \( d = \omega(\log n) \), \( \forall \epsilon > 0 \),

OV cannot be solved in time \( O(n^{2-\epsilon}) \)

[Williams' 05]

Other natural examples: 3SUM, etc...

Credits to Marshal Ball for pics
Another Limitation: the “ASIC” problem

Incentive: making them pay 0.0000005 cents

They will not pay the same anymore!
Last slides

• Proof of Work
  • Limitations:
    • can be wasteful, ASICs can raise the bar for honest parties
  • Mitigating waste
    • Useful Work

• What we will look at next:
  • Proof of Space
    • Same application realms
    • Our hope is to remove the limitations we saw:
      • Reduce the energy waste (we will talk about usefulness first)
      • Removing the "ASIC problem"
Proofs of Space
(an intuition through our old friend: spam)

I will not consider the email "received" unless the PoS checks

Proof of "space"

The hoped guarantee: the adversary is must have used Y amount of space if proof checks

Why would this address PoW’s limitations?
- **Energy**: CPU vs RAM (or disk)
- "**ASIC**": no equivalent for memory that can provide savings of orders of magnitude like ASICs did for CPU
Applications: Making Proofs of Space Useful

• Applications of PoS (Proofs of Space)
  • Could be the same as before:
    • Fighting denial of service, etc.
    • Consensus for next block (Chia)
  • Or more (e.g., through usefulness)
    • By analogy, usefulness in PoW was: “I am using grinding for computation f(x)”
      (where f is some natural function)
    • “Natural usefulness in PoS”: let’s use PoS to guarantee storage of useful files
      • (E.g., data sets, Wikipedia, the web in general)
    • => System where we can obtain cryptographic incentives of somebody using
      a certain amount of space AND using it that space for storing a specific file
Earlier during this talk:

Then

Now

**Proof of Useful Space:**
System where we you can obtain cryptographic incentives of somebody using a certain amount of space AND using it that space for storing a specific file

**Next:** I want to give a flavor of how one can define this
A syntax for Proofs of Useful Space

Security intuition: “If prf checks then Prover is storing a certain amount of storage related to F”
A non-solution

Prover

Verifier

Init(id, F):

digest <- hash(F)

Check F against digest by hashing

Security intuition: “If prf checks then Prover is storing a certain amount of storage related to F”
A non-solution—Issue 1: succinctness

Security intuition: “If prf checks then Prover is storing a certain amount of storage related to F”
A non-solution—Issue 2: space requirement

Prover

Verifier

Init(id, F):
digest <- hash(F)

Security intuition: "If prf checks then Prover is storing a certain amount of storage related to F"

Which requirements does this construction satisfy?
A non-solution—Issue 2: space requirement

**Security intuition:** “If \( \text{prf} \) checks then Prover is storing a certain amount of storage related to \( F \)”

The file is **compressible**
=> The prover does not need to store \( F \) to provide it.

**F**

\( F = “1111...111111” \)
(one repeated one trillion times)

**NB:** Not all “problematic” files are obviously so.
Example: \( F = (\text{PRF}_k(1), \text{PRF}_k(2),...) \)

What we want instead:
Space requirement should hold for any file, even compressible ones.

What is **usefulness** requirement:
Check \( F \) against digest by hashing.

**Space requirement**

\[
\text{prf} = \text{clg}
\]
Thea naïve solution does not work.
So how do we solve these problems??
More resources and wrap up

• https://proofofspace.org/references
• Filecoin.io

For any other questions:
matteo@protocol.ai

binarywhales.org

Thanks!