uMine: a Blockchain based on Human Miners*

(Full Version)

Henning Kopp[1], Frank Kargl[1], Christoph Bösch[1], and Andreas Peter[2]

[1] Institute of Distributed Systems, Ulm University, Germany
{henning.kopp, frank.kargl, christoph.boesch}@uni-ulm.de
[2] Services, Cybersecurity and Safety Group, University of Twente, The Netherlands
a.peter@utwente.nl

Abstract. Blockchain technology like Bitcoin is a rapidly growing field of research which has found a wide array of applications. However, the power consumption of the mining process in the Bitcoin blockchain alone is estimated to be at least as high as the electricity consumption of Ireland which constitutes a serious liability to the widespread adoption of blockchain technology. We propose a novel instantiation of a proof of human-work which is a cryptographic proof that an amount of human work has been exercised, and show its use in the mining process of a blockchain. Next to our instantiation there is only one other instantiation known which relies on indistinguishability obfuscation, a cryptographic primitive whose existence is only conjectured. In contrast, our construction is based on the cryptographic principle of multiparty computation (which we use in a black box manner) and thus is the first known feasible proof of human-work scheme. Our blockchain mining algorithm called uMine, can be regarded as an alternative energy-efficient approach to mining.

Keywords: Blockchain, Applied Cryptography, Peer-to-Peer, Proof of Work

1 Introduction

The last few years have seen a rising interest in the use of blockchain technology. Originally, blockchain architectures emerged from the design of the cryptographic cash system Bitcoin [26] to construct alternative cryptocurrencies. Nowadays, there are applications besides cryptocurrencies, like secure and fair multiparty computations [2, 7, 19] or smart contracts [23, 35, 22], though decentralized cryptocurrencies are still the main driving force behind the blockchain trend. In a nutshell, blockchains provide an immutable distributed ledger and thus can potentially be used to record various forms of asset ownership in different domains.

* This article is the full version of the paper appearing in the proceedings of ICICS 2018. The final authenticated version is available online at Springer.

One of the major drawbacks of blockchain technology is its huge energy consumption. According to de Vries [34] Bitcoin alone consumes at least 2.55 gigawatts of energy making it comparable to countries such as Ireland's electricity consumption (3.1 gigawatts). We identify this problem to be one of the main challenges of scaling blockchains and allowing for their widespread adoption.

In this article we tackle the issue of the huge energy consumption of blockchains by introducing uMine, a mining algorithm based on a novel proof of human-work construction. Proofs of human-work are cryptographic mechanisms where a prover can convince a verifier that it has spent some amount of human work. In particular, proof of human-work puzzles can only be solved by humans and not by computers under the hardness assumption of some underlying AI problem. This allows us to lower the energy consumption of the blockchain by exchanging the costly proof of work mining algorithm by a proof of human-work which can only be provided by humans.

Proofs of human-work were originally developed by Blocki and Zhou [8] but their construction relies on indistinguishability obfuscation, a theoretical cryptographic primitive where no realization is known. Our new construction in contrast is based on multiparty computation where multiple feasible instantiations exist [5, 27, 10, 9, 17].

Our contributions can be summarized as follows:

- We provide a novel instantiation of a proof of human-work which does not rely on indistinguishability obfuscation but instead uses multiparty computation as a black box.
- We prove the security of our proof of human-work given a secure captcha.
- We use our proof of human-work to construct uMine, a novel energy efficient mining algorithm where the mining is performed by human miners creating proofs of human-work.

This paper is the full version of our publication at ICICS 2018. In particular, this version includes a proof of security of our proof of human-work construction, and a protocol for the dynamic choice of captcha generators. Further, we included some remarks on implementing our scheme.

Roadmap: The next section introduces notation and the basic primitives used in the remainder of our exposition. Section 3 provides a thorough explanation of our constructions. In Section 4 we analyze the security properties of our solution. Section 5 describes considerations when implementing our proof of human-work scheme. Related work is discussed in Section 6. Finally, Section 7 concludes our work.

2 Building Blocks

Notation: We write $a \leftarrow \mathcal{A}(x)$ to assign to a the output of running the randomized algorithm $\mathcal{A}$ on the input x. We denote with $a \leftarrow \mathcal{A}(x; r)$ the deterministic result of running $\mathcal{A}$ on input x with the fixed randomness r. We say that an algorithm $\mathcal{A}$ is ppt if it runs in probabilistic polynomial time.

2.1 Blockchain

A blockchain is a distributed append-only database together with a consensus algorithm where nodes decide which data is persisted. Usually blockchains are frequently used in the design of cryptographic currencies, to agree on the order of transactions and provide a single immutable log where all transactions are recorded. The most prominent example is Bitcoin [26], which was the first to introduce the idea of a blockchain. The participants in the consensus protocol bundle transactions into blocks and try to append them to the blockchain by partially inverting a hash function, a process which is called proof of work [12, 11]. Since each node is granted a financial reward in the underlying cryptocurrency for finding a new correct block, the so called mining reward, proof of work achieves alignment of incentives. If these nodes, called miners, solve a proof of work to include wrong transactions in the chain, their financial reward is annihilated, since the other nodes reject wrong blocks. Thus it is rational for miners to only persist valid information in the blockchain.

The proof of work mining algorithm includes a difficulty parameter which in Bitcoin is adjusted every 2016 blocks (approximately two weeks) such that one block is expected to be found every ten minutes assuming no changes in the hash rate. This mechanism allows the global hash rate to change while preserving the block creation rate. While the optimal adjustment of the difficulty parameter is not well understood it is clear that a difficulty parameter needs to be supported when designing alternative mining algorithms.

For further reading regarding blockchains we refer the reader to the survey by Tschorsch et al. [33] or the book by Antonopoulos [3].

While the original vision of blockchains [26] was that each processor has the same chance to mine a block, nowadays the mining industry is dominated by few corporations with specialized mining hardware. Due to this commercialized mining arms race the power consumption of the whole Bitcoin network has increased significantly. For the scalability and the further development of blockchain technology this clearly constitutes a problem.

2.2 Slow Hash Functions

Slow hash functions are a special kind of hash function. While usual hash functions $\mathcal{H}$ are designed to be easy to compute, the evaluation of a slow hash function $\overline{\mathcal{H}}$ in contrast is computationally costly. Normally the evaluation of a slow hash function like bcrypt [31] or scrypt [30] is on the order of several hundred milliseconds, thus slowing down brute-force attacks significantly. The intuition behind slow hash functions is that an authorized user needs to evaluate them only once, and thus the overhead is negligible.

2.3 Captchas

Captcha is an acronym for a **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part. They are challenge response tests to determine if the user is a human or a program. One major application is to prevent automated registrations of accounts in web services. The most common form of a captcha puzzle consists of a set of warped letters, where the user is requested to recognize the letters, a task which is supposedly hard for computers and easy for humans. There are also other forms like audio-based captchas where the user is challenged to recognize speech data. To enable automatic verification of a given solution without human assistance the service provider has usually stored a secret set of puzzle-solutions pairs. These pairs are generated by computing a puzzle from a known solution. For verification, access to these puzzle-solution pairs is needed and hence captchas are in general not publicly verifiable.

Since captchas are based on the assumption that some fundamental AI problem is hard to solve, the need to model the human solver as an entity distinct from an algorithm arises. Sometimes this is done in the form of a (yet) unknown algorithm. Since we prefer giving a clearer exposition to giving a philosophically correct one we simply model the human as an oracle that can provide the solutions to a captcha puzzle along the lines of Blocki and Zhou [8].

Definition 1. *Captcha ([8]): A Captcha* C *is a quintuple of algorithms* $(\mathsf{Setup}, \mathsf{W}, \mathsf{G}, \Sigma^{human}, \mathsf{Verify})$ *with the following properties:*

- $PP \leftarrow \mathsf{C.Setup}(1^\lambda)$ *is the generation of the public parameters* PP *given a security parameter* λ.
- $\sigma \leftarrow \mathsf{C.W}(PP)$ *is a randomized algorithm sampling a solution* σ *given the public parameters.*
- $Z \leftarrow \mathsf{C.G}(PP, \sigma)$ *generates a captcha-puzzle* Z *with solution* σ. *We write* $\mathsf{C.G}(PP, \sigma; r)$ *if we fix the randomness* r, *i.e., if we consider* $\mathsf{C.G}$ *as a deterministic function.*
- $\sigma \leftarrow \mathsf{C}.\Sigma^{human}(PP, Z)$ *is a solution finding algorithm that takes as input the public parameters and a puzzle* Z *and outputs a solution* σ. *It has internal access to a human oracle.*
- $b := \mathsf{C.Verify}(PP, Z, \sigma)$ *outputs a single bit which is* 1 *whenever there is a random* r, *such that* $\mathsf{C.G}(PP, \sigma; r) = Z$.

The original definition of a captcha by Blocki and Zhou [8] additionally uses a tag which is generated together with the puzzle and needed for the verification of a solution σ. We stress that our construction later also works with the definition of Blocki and Zhou, where the tag is set as undefined. However, the tags are not necessary in our construction and thus we decided to present our work using a simpler definition to aid in the understanding.

If the randomness r which was used to generate the captcha in the algorithm $\mathsf{C.G}$ is known it may be possible to invert $\mathsf{C.G}$. In the case of image based captchas r determines the chosen transformations, e.g., rotation, addition of noise, and their parameters applied on the solution to yield a puzzle [1]. Knowledge of these may allow an attacker to invert the used transformations and thus recover the solution σ from a puzzle Z without the use of human work. Consequently the security of a captcha puzzle $Z = \mathsf{C.G}(PP, \sigma; r)$ is usually based on the secrecy of the random value r, which was used to generate the puzzle [1, Section Who knows What?].

Additional Requirements for our Construction: In contrast to the original definition by Blocki and Zhou [8] we require the generation of the puzzles $\mathsf{C.G}(PP, \sigma; r)$ to be collision-free, i.e., injective, in its randomness r and in its solutions σ. Regarding the solutions σ it is natural to assume that there can be no two different solutions to the same puzzle. Regular image based captchas do have this property. Regarding injectivity in the randomness r we can assume that it serves as an enumeration of the puzzle space for a given captcha solution. Consider the case of image based captchas where the randomness determines the type of transformations. Different transformations with different parameters yield different puzzles and thus collision freeness can be assumed.

Use in our Instantiation: In our construction of a proof of human-work the randomness r used in the puzzle generation $\mathsf{C.G}$ is set to a deterministic value containing a hash of the solution $\overline{\mathcal{H}}(\sigma)$, which was computed by a slow hash function. This way it is easy to verify a solution publicly, given a puzzle, since one only needs to regenerate the puzzle from the solution σ using the same randomness and check if the given puzzle equals the computed one. The use of a slow hash function is necessary to prevent bruteforcing of the solution using the verification algorithm.

Security Properties: We require that any captcha should be solvable by a human. We use the term human-work unit to denote the effort needed to solve a single instance of a captcha. Although the time needed to solve a captcha may depend on the human and his abilities, we expect these differences to be small and similar to the differences in performance of different computer hardware.

Definition 2. *Honest Human Solvability ([8]): We say that a human-machine solver* $\mathsf{C}.\Sigma^{human}$ *controls m human-work units if it can query its human oracle at least m times. We say that a captcha system* $\mathsf{C} = (\mathsf{Setup}, \mathsf{W}, \mathsf{G}, \Sigma^{human}, \mathsf{Verify})$ *is honest human solvable if for every polynomial $m = m(\lambda)$ and for any human*

$\mathsf{C}.\Sigma^{human}$ *controlling m human-work units, it holds that*

$$P\left[\begin{array}{c}\forall PP \leftarrow \mathsf{C.Setup}(1^\lambda);\\ \forall i \in [m]\big(\sigma_i^* \leftarrow \mathsf{C.W}(PP)\big);\\ \forall i \in [m]\big(Z_i^* \leftarrow \mathsf{C.G}(PP, \sigma_i^*)\big):\\ (\sigma_1^*, \ldots, \sigma_m^*) \leftarrow \mathsf{C}.\Sigma^{human}(PP, Z_1^*, \ldots, Z_m^*)\end{array}\right] \geq 1 - negl(\lambda)$$

Finally we require that captchas are hard for computers to solve without access to a human oracle.

Definition 3. *Captcha Break ([8]): We say that a ppt adversary $\mathcal{A}$ who has at most m human-work units breaks security of a captcha system* $\mathsf{C} = (\mathsf{Setup}, \mathsf{W}, \mathsf{G}, \Sigma^{human}, \mathsf{Verify})$ *if there exist polynomials $m = m(\lambda), n = poly(\lambda)$ and $\mu(\lambda)$ such that if $\mathcal{A}$ controls at most m human-work units it holds that*

$$P\left[\begin{array}{l}\forall PP \leftarrow \mathsf{C.Setup}(1^\lambda);\\ \forall i \in [n]\big(\sigma_i^* \leftarrow \mathsf{C.W}(PP)\big);\\ \forall i \in [n]\big(Z_i^* \leftarrow \mathsf{C.G}(PP, \sigma_i^*)\big);\\ S \leftarrow \mathcal{A}(PP, Z_1^*, \ldots, Z_n^*);\\ \forall i \in [n]\big(b_i \leftarrow \max_{\sigma \in S} \mathsf{C.Verify}(PP, Z_i^*, \sigma)\big):\\ \qquad\qquad \sum_{i \in [n]} b_i \geq m+1\end{array}\right] \geq \frac{1}{\mu(\lambda)}$$

It is debatable, whether in AI research the concept of a security parameter applies [1]. AI research does not deal with asymptotics and thus it can be argued that problem classes are either solvable or unsolvable, independent of the concrete problem in the problem class. This is in contrast to classical cryptography where it may be feasible to solve certain "small" instances of problems without solving all, e.g., factorization of small integers may be possible, without being able to factorize all integers. Thus, if captchas are either solvable or unsolvable in the real world our definitions can be made even stronger by setting the negligible term in the definition of honest human solvability to zero. Without a tunable security parameter, a captcha is called broken if the attacker has a success probability of 1 of finding solutions without access to a human oracle.

2.4 Proof of Human-work Puzzles

Proof of human-work puzzles (PoH) were first introduced by Blocki and Zhou [8]. Their goal was to construct a publicly verifiable proof that some amount of human work has been exercised. Their construction relies on indistinguishability obfuscation [14] and thus is currently infeasible.

A PoH in contrast to a captcha has a tunable difficulty parameter and is publicly verifiable. That means that no secret knowledge is needed neither to generate nor to verify a PoH. Especially, the solution does not need to be known beforehand to generate the puzzle as is the case with captchas. The difficulty parameter is necessary to enable its use as a mining algorithm in a blockchain as explained above.

Definition 4. *Proof of Human-work Puzzle ([8]): A proof of human-work puzzle system* POH *consists of four algorithms* $(\mathsf{Setup}, \mathsf{G}, \Sigma^{human}, \mathsf{V})$ *where:*

- $PP \leftarrow \mathsf{POH.Setup}(1^\lambda, 1^\omega)$ *is a randomized system setup algorithm that takes as input a security parameter* λ *and a difficulty parameter* ω *and outputs public parameters of the system* PP*.*
- $x \leftarrow \mathsf{POH.G}(PP)$ *is a randomized algorithm that takes as input the public parameters* PP *and outputs a puzzle* x*.*
- $a \leftarrow \mathsf{POH}.\Sigma^{human}(PP, x)$ *is a solution finding algorithm that has access to a human oracle. It takes as input the public parameters and a puzzle* x *and outputs a solution* σ *to the puzzle.*
- $b := \mathsf{POH.V}(PP, x, a)$ *is a deterministic verification algorithm that takes as input the public parameters* PP*, together with a puzzle* x *and a solution* a *and outputs a bit* b *where* $b = 1$ *if and only if* a *is a valid solution to the puzzle* x*.*

Similar to a captcha we require from PoHs that they are solvable by a human, with a success probability depending on the difficulty parameter. Following the notation of Blocki and Zhou [8] and Miller et al. [25] we define $\zeta(m, \omega) := 1 - (1 - 2^{-\omega})^m$. This describes the probability of finding a valid solution using m queries to the human oracle.

Definition 5. *Honest Human Solvability ([8]): We say that a PoH system* $\mathsf{POH} = (\mathsf{Setup}, \mathsf{G}, \Sigma^{human}, \mathsf{V})$ *is honest human solvable if for every polynomial* $m = m(\lambda)$*, and for any honest human-machine solver* $\mathsf{POH}.\Sigma^{human}$ *who controls* m *human-work units, it holds that*

$$P\left[\begin{array}{l} \forall PP \leftarrow \mathsf{POH.Setup}(1^\lambda, 1^\omega); \\ x^* \leftarrow \mathsf{POH.G}(PP); \\ a^* \leftarrow \mathsf{POH}.\Sigma^{human}(PP, x^*) : \\ \quad \mathsf{POH.V}(PP, x^*, a^*) = 1 \end{array}\right] \geq \zeta(m, \omega) - negl(\lambda)$$

Further, we require that any adversary that controls too few human-work units succeeds in solving a PoH only with negligible probability.

Definition 6. *Adversarial Human Unsolvability ([8]): We say that a ppt algorithm* $\mathcal{A}$ *breaks security of the PoH system* $\mathsf{POH} = (\mathsf{Setup}, \mathsf{G}, \Sigma^{human}, \mathsf{V})$ *if for some polynomials* $m = m(\lambda)$ *and* $\mu(\lambda)$ *when* $\mathcal{A}$ *controls at most* m *human-work units, it holds that*

$$P\left[\begin{array}{l} \forall PP \leftarrow \mathsf{POH.Setup}(1^\lambda, 1^\omega); \\ x^* \leftarrow \mathsf{POH.G}(PP); \\ a^* \leftarrow \mathcal{A}(PP, x^*) : \\ \quad \mathsf{POH.V}(PP, x^*, a^*) = 1 \end{array}\right] \geq \zeta(m+1, \omega) + \frac{1}{\mu(\lambda)}$$

2.5 Multiparty Computation Protocol

Multiparty computation protocols (MPC) are cryptographic protocols that allow a set of mutually distrusting parties to collaboratively compute a function with private input values. For example the parties may decide to evaluate some $f(y_1, \ldots, y_n)$, where the input y_i is only known to party i. The participants in the protocol do not learn anything beyond their own inputs and the solution $f(y_1, \ldots, y_n)$.

While traditional schemes suffered from severe performance issues, over the last few years, multiple practical solutions that can deal with arbitrary computable functions f have emerged [5, 27, 10, 9, 17]. In our case we require a secure multiparty protocol with k different parties, where $k-1$ participants can be controlled by an *active* attacker. An active (malicious) attacker can arbitrarily deviate from any protocol execution in an attempt to cheat. This is in contrast to passive (semi-honest) attackers who try to gather as much information about the underlying inputs and (intermediate) outputs but honestly follow the prescribed steps in the given protocol.

We use MPC as a black box in this article, having secret sharing based MPC protocols in mind (such as SPDZ [9]). For this we define an MPC protocol MPC as a triple of ppt algorithms $(\mathsf{Setup}, \mathsf{Share}, \mathsf{Reveal})$ where:

- $PP \leftarrow \mathsf{MPC.Setup}(1^\lambda)$ is a randomized algorithm that takes a security parameter λ as input and sets up the protocol by distributing the keys and parameters. It outputs the public parameters for the system.
- $\langle y \rangle \leftarrow \mathsf{MPC.Share}(PP, y)$ shares the value y among the k participants using a secret sharing scheme such that each of the k participants receives one share. We use $\langle y \rangle$ to denote the vector of secret shares of y. Note that the Share algorithm can be executed by one of the participants or any other external party with access to the parameters PP.
- $y \leftarrow \mathsf{MPC.Reveal}(PP, \langle y \rangle)$ reconstructs the value y from its secret shares $\langle y \rangle$. The MPC participants send their secret shares $\langle y \rangle$ to an external party who can then execute $\mathsf{MPC.Reveal}$ and learn the value y; consequently being the only party knowing y in the clear.

By abuse of notation, we apply computable functions on secret shares to denote the computation of the secret shares of the result of the function applied to the clear values, i.e., we denote $\langle f(y_1, \ldots, y_n) \rangle$ by $f(\langle y_1 \rangle, \ldots, \langle y_n \rangle)$. The clear values are not revealed by this operation. Note that knowledge of the public parameters may be needed for this computation, but is left out to simplify our notation.

One possible MPC framework for our use is SPDZ [9], which consists of a preprocessing and an online phase. The preprocessing phase is independent of the function to be computed as well as of the inputs. In the online phase the actual function is evaluated. The online phase has a total computational and communication complexity linear in the number of participants k. The work done by each participant in SPDZ is only a small constant factor larger than what would be required to compute the function in the clear. Thus, SPDZ provides an efficient framework which satisfies our requirements.

3 Our Construction

3.1 Overview

On a high level we are interested in exchanging the proof of work by a PoH. The parties involved in our system are human miners, i.e., miners who control some human-work units, who try to solve the PoH puzzles in order to gain the block rewards, as well as a consortium of k puzzle generators. To mine a new block, each human miner requests a puzzle for a proof of human-work from the puzzle generators. The puzzle is linked to the transactions the human miner wants to persist, as well as to the current block in the blockchain. Throughout the generation of the puzzle, the solution is unknown to any single party, in contrast to regular captchas. If the human miner does not succeed in solving the puzzle it can request a new puzzle from the captcha generators. If the human miner succeeds however, it can publish the new block containing the captcha puzzle, its solution, and the transactions. A node which receives a new block can check the transactions and the proof of human-work for validity. It accepts the block if all of these are correct and mining continues on top of the new block.

3.2 Our Proof of Human-Work

We give a new instantiation of a PoH puzzle which does not rely on indistinguishability obfuscation [14] like the work of Blocki and Zhou [8], but instead on MPC.

Our construction shown in Figure 1 is the first PoH which is feasible and does not involve a trusted third party. In contrast to the work of Blocki and Zhou [8], computing the algorithm $\mathsf{POH.G}$ in our construction needs interaction with a set of captcha generators $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$. This set can be a fixed consortium of k parties. However, in Subsection 3.4 we elaborate on alternatives for choosing the captcha generators. The assumption of interaction poses no problem for our use case since for mining on a blockchain the miners are required to be online anyway to receive the latest blocks and transactions.

The intuition behind our construction is that the captcha generators collaboratively compute a captcha puzzle using multiparty computation. This computation is done in such a way that each captcha generator has access to only a secret share of the solution, but not to the solution itself. Consequently, the solution is unknown to any single party.

```
$PP \leftarrow \mathsf{POH.Setup}(1^\lambda, 1^\omega)$
    1. Return the parameters $PP$ containing $\lambda$, $\omega$, and the public keys of the captcha generators $\mathcal{C}_1, \dots, \mathcal{C}_k$.
$x \leftarrow \mathsf{POH.G}(PP)$
    1. Parse $\lambda$ and $\omega$ from $PP$.
    2. Compute the public parameters for the captcha as $\mathsf{C}.PP \leftarrow \mathsf{C.Setup}(1^\lambda)$.
    3. Each captcha generator $\mathcal{C}_j$ chooses a secret random value $y_j$ and distributes the secret shares $\mathsf{MPC.Share}(\mathsf{MPC}.PP, y_j)$.
    4. The captcha generators compute $\langle\sigma\rangle = \mathsf{C.W}(\mathsf{C}.PP, \langle y_1\rangle \oplus \cdots \oplus \langle y_k\rangle)$ using MPC.
    5. The captcha generators compute the puzzle $\langle Z\rangle \leftarrow \mathsf{C.G}\left(\mathsf{C}.PP, \langle\sigma\rangle; \overline{\mathcal{H}}(\langle\sigma\rangle)\right)$ and $\tau$ as a signature on their share $\langle Z\rangle$.
    6. Reveal the captcha puzzle $Z$ by executing $Z = \mathsf{MPC.Reveal}(\mathsf{MPC}.PP, \langle Z\rangle)$.
    7. Return the PoH puzzle $x = (\mathsf{C}.PP, Z, \tau)$.
$a \leftarrow \mathsf{POH}.\Sigma^{human}(PP, x)$
    1. Parse $\lambda$ and $\omega$ from $PP$.
    2. Parse the puzzle $x$ as $(\mathsf{C}.PP, Z, \tau) = x$.
    3. Execute $\sigma \leftarrow \mathsf{C}.\Sigma^{human}(\mathsf{C}.PP, Z)$.
    4. If $\mathcal{H}(\sigma) < T_\omega$ return the solution $a = \sigma$.
    5. Otherwise return $a = \perp$.
$b := \mathsf{POH.V}(PP, x, a)$
    1. Parse $\sigma = a$
    2. If $\mathcal{H}(\sigma) \geq T_\omega$ return $b = 0$.
    3. Parse $(\mathsf{C}.PP, Z, \tau) = x$ and check the signatures and the final shares of the puzzle $\tau$ for correctness
    4. If $\tau$ is invalid return $b = 0$.
    5. If $\mathsf{C.G}\left(\mathsf{C}.PP, \sigma; \overline{\mathcal{H}}(\sigma)\right) = Z$ return $b = 1$.
    6. Return $b = 0$.
```

Fig. 1. Our novel instantiation of a PoH.

Nevertheless, since it is a captcha the solution can be found by querying the human oracle. If the hash of the solution σ of the captcha puzzle is above a difficulty parameter the solution is deemed invalid for the PoH. Thus, for creating a valid PoH one may need to solve multiple captchas depending on the difficulty, until a captcha solution with a small hash is found. This captcha solution then constitutes a PoH a.

In order to achieve public verifiability, remember that the generation of a captcha puzzle is a probabilistic algorithm using the solution. If the randomness used in the captcha generation is known it is possible to regenerate the captcha puzzle from the solution. This allows public verification of the solution since the recomputed puzzle can be compared to the given puzzle. In our construction the randomness in the captcha generation is derived from the captcha solution itself.

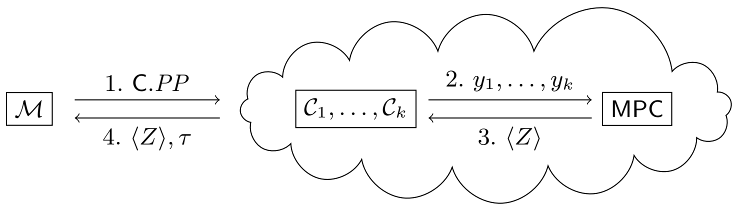

Fig. 2. Simplified Overview of our Proof of Human-work Construction

A standard workflow is shown in Figure 2. As a first step the captcha generators $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ are initialized by executing $\mathsf{MPC.Setup}$ with an appropriate security parameter. Now, suppose a human $\mathcal{M}$ wants to compute a proof of human-work.

First, the human sets up the public parameters PP for the proof of human-work by executing $\mathsf{POH.Setup}$. The public parameters consist of the public keys of the captcha generators, a difficulty parameter ω, and a security parameter λ.

Next, the human queries the captcha generators to obtain a captcha by executing $\mathsf{POH.G}$. To this end, he computes public parameters $\mathsf{C}.PP$ for the captcha by running the setup algorithm of the captcha with security parameter λ. The public parameters $\mathsf{C}.PP$ are distributed to the captcha generators $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$ (Step 1 in Figure 2). Each captcha generator $\mathcal{C}_j$ chooses a random value y_j and shares it among the other captcha generators, according to the multiparty computation protocol (Step 2 in Figure 2). Together, the captcha generators sample a solution σ of the captcha by using the sum of their chosen randomness $y_1 \oplus \cdots \oplus y_k$ in the sampling algorithm $\mathsf{C.W}$ together with the public parameters of the captcha $\mathsf{C}.PP$. This solution σ is not revealed, but rather stays secret shared between the captcha generators. Thus, no captcha generator knows the solution. From the shared solution $\langle\sigma\rangle$ the shares of the captcha puzzle $\langle Z\rangle$ are computed by the captcha generators using multiparty computation as $\langle Z\rangle \leftarrow \mathsf{C.G}\left(\mathsf{C}.PP, \langle\sigma\rangle; \overline{\mathcal{H}}(\langle\sigma\rangle)\right)$. Here, $\overline{\mathcal{H}}$ denotes a slow hash function. Each captcha generator signs its share of the puzzle and sends it to the human $\mathcal{M}$ (Step 4 in Figure 2). We call these signed shares τ. The human then reveals the puzzle Z by executing the $\mathsf{MPC.Reveal}$ algorithm of the multiparty computation protocol. $\mathcal{M}$ is now the only person knowing the captcha puzzle Z, and the solution σ is unknown to any single party. The puzzle to the PoH is $x = (\mathsf{C}.PP, Z, \tau)$.

In order to create the PoH the human solves the captcha puzzle Z by executing its captcha solving algorithm. This yields a solution σ to the captcha. If $\mathcal{H}(\sigma) < T_\omega$ then this constitutes a valid proof of human-work. Here, $\mathcal{H}$ is a hash function and $T_\omega = 2^{n-\omega}$ analogous to Blocki and Zhou [8], where ω is the difficulty parameter and n is the bit size of the output of $\mathcal{H}$. If this is not the case, i.e., if the hash of the solution is not small enough, the human has to start again by querying the captcha generators for a new puzzle until he succeeds in solving a captcha with a small solution. The PoH consists of the solution to the captcha puzzle $a = \sigma$.

To verify the PoH, i.e., to execute $\mathsf{POH.V}$, the public parameters PP, the puzzle x, and its solution a are needed. The verifier first needs to check if the puzzle has been computed in a correct way, i.e., by the captcha generators. This can be done by checking τ, the signatures on the shares of the solution which are included in the puzzle x. Next, the verifier checks that the hash of the solution is small enough, i.e., if $\mathcal{H}(\sigma) < T_\omega$. As a final step, the verifier checks that the solution is a correct solution to the captcha. This can be done by simply regenerating a puzzle from the solution and checking equality between the recomputed puzzle and the original puzzle. I.e., it needs to be checked if $\mathsf{C.G}\left(\mathsf{C}.PP, \sigma; \overline{\mathcal{H}}(\sigma)\right) = Z$. If any of these three steps fails, the PoH is rejected. Otherwise it is considered valid.

Construction 1. *Let* C *be a secure human solvable captcha and* MPC *be a secure MPC scheme initialized with public parameters* $\mathsf{MPC}.PP \leftarrow \mathsf{MPC.Setup}(1^\lambda)$*. Let* $\mathcal{H} : \{0,1\}^\bullet \rightarrow \{0,1\}^n$ *be a hash function. We define* $T_\omega = 2^{n-\omega}$ *analogous to Blocki and Zhou [8]. We use* T_ω *to scale our difficulty parameter* ω*, since* $P(\mathcal{H}(r) < T_\omega) = 2^{-\omega}$ *for a random* r*. We now construct a PoH by defining the following operations.*

- $PP \leftarrow \mathsf{POH.Setup}(1^\lambda, 1^\omega)$ *outputs the parameters* PP *containing* λ, ω, *and the public keys of the captcha generators* $\mathcal{C}_1, \ldots, \mathcal{C}_k$.
- $x \leftarrow \mathsf{POH.G}(PP)$ *is computed by interacting with the set of captcha generators* $\{\mathcal{C}_1, \ldots, \mathcal{C}_k\}$*. First we parse* λ *and* ω *from* PP *locally and compute the public parameters for the captcha as* $\mathsf{C}.PP \leftarrow \mathsf{C.Setup}(1^\lambda)$*. These are then given to the captcha generators. Each captcha generator* $\mathcal{C}_j$ *chooses a secret random value* y_j *and uses* $\mathsf{MPC.Share}(\mathsf{MPC}.PP, y_j)$ *to distribute shares of its value* y_j *among the* k *captcha generators. In a next step the captcha generators compute* $\langle\sigma\rangle = \mathsf{C.W}(\mathsf{C}.PP, \langle y_1\rangle \oplus \cdots \oplus \langle y_k\rangle)$ *using MPC such that each of the captcha generators now possesses a secret share of the solution* σ *to the captcha. The solution* σ *is not revealed but stays in the secret shared domain. Next, the captcha generators compute the puzzle* $\langle Z\rangle \leftarrow \mathsf{C.G}\left(\mathsf{C}.PP, \langle\sigma\rangle; \overline{\mathcal{H}}(\langle\sigma\rangle)\right)$*. The captcha generators each sign their shares* $\langle Z\rangle$ *of the puzzle as* τ *which later guarantees that each of the captcha generators* $\mathcal{C}_j$ *has participated in the protocol.*

  Finally the captcha puzzle Z *is revealed by executing* $Z = \mathsf{MPC.Reveal}(\mathsf{MPC}.PP, , \langle Z\rangle)$ *and the PoH puzzle* $x = (\mathsf{C}.PP, Z, \tau)$ *is output.*
- $a \leftarrow \mathsf{POH}.\Sigma^{human}(PP, x)$ *computes a solution* a *to a PoH as follows. First, the parameters* λ, ω*, are parsed from* PP*. The puzzle is parsed as* $(\mathsf{C}.PP, Z, \tau) = x$*. Then the captcha solving algorithm is queried* $\sigma \leftarrow \mathsf{C}.\Sigma^{human}(\mathsf{C}.PP, Z)$*. If* $\mathcal{H}(\sigma) < T_\omega$ *we return the solution* $a = \sigma$*. Otherwise* $a = \perp$ *is returned.*
- $b := \mathsf{POH.V}(PP, x, a)$ *first parses* $\sigma = a$ *and checks if* $\mathcal{H}(\sigma) < T_\omega$*. If that is not the case* $b = 0$ *is returned. Otherwise we parse* $(\mathsf{C}.PP, Z, \tau) = x$ *and check the signatures and the final shares of the puzzle* τ *to ensure that the puzzle has been generated in a correct way, i.e., by the captcha generators* $\mathcal{C}_j$*,*

and not by anyone else. This is possible, since the public keys of the captcha generators needed for the verification of the signatures are contained in PP. If τ is invalid, we return $b = 0$. As a third step we need to ensure that the solution σ is a valid solution to the captcha. This can be done by checking if $\mathsf{C.G}\left(\mathsf{C}.PP, \sigma; \overline{\mathcal{H}}(\sigma)\right) = Z$. If that is the case, return $b = 1$, otherwise return $b = 0$.

Theorem 1. *If our construction is instantiated with a secure and honest human solvable captcha, then the resulting PoH is honestly human solvable and adversarial human unsolvable under the assumption that at least one of the k captcha generators $\mathcal{C}_1, \ldots, \mathcal{C}_k$ is honest.*

Proof. The basic idea of the proof is to reduce the security of the PoH to the security of the underlying building blocks.

Honest human solvability of the PoH follows from our definition of T_ω and since the captcha is honest human solvable. Further the puzzle has been generated correctly, since due to our assumptions that at least one of the k captcha generators is honest our MPC protocol will compute the puzzle correctly. The signed shares of the puzzle τ guarantee that the captcha was computed by the correct parties.

To show adversarial human unsolvability, we need to construct an attacker $\mathcal{A}_{\mathsf{C}}$ on the captcha mechanism, given an attacker $\mathcal{A}_{\mathsf{POH}}$ on the PoH. We can assume that if at least one of the k captcha generators is honest, the solution to a captcha does not leak in the generation, due to the security properties of our MPC protocol. Whenever $\mathcal{A}_{\mathsf{C}}(\mathsf{C}.PP, Z_1, \ldots, Z_n)$ is called it can create a set of PoH puzzles whose solutions a correspond to solutions σ of the captcha, whenever $\mathcal{H}(\sigma) < T_\omega$. For this $\mathcal{A}_{\mathsf{C}}$ needs to set up the public parameters PP of the PoH and construct τ_i by secret sharing the puzzles Z_i, thereby simulating the captcha generators. This can be done, since the public keys of the captcha generators are contained in the public parameters PP which the attacker can set accordingly. Thus the generated τ_i and the Z_i are valid. The attacker $\mathcal{A}_{\mathsf{C}}$ can then execute $\mathcal{A}_{\mathsf{POH}}(PP, x_i)$ on the PoH, where $x_i = (\mathsf{C}.PP, Z_i, \tau_i)$.

Since for a random value r we know that $P(\mathcal{H}(r) < T_\omega) = 2^{-\omega}$ we can conclude that if the success probability of $\mathcal{A}_{\mathsf{POH}}$ is larger than $\zeta(m+1, \omega) + \frac{1}{\mu(\lambda)}$ then the success probability of $\mathcal{A}_{\mathsf{C}}$ is non-negligible.

3.3 Block Generation

In this section we describe a design of a blockchain which is based on proofs of human-work. We call the resulting mining process uMine. In order to mine a new block B_i, a human miner $\mathcal{M}$ needs access to the previous block B_{i-1}. Further it needs to have a set of transactions Tx_i, which it wants to persist in the new block B_i.

To mine a new block, first, the algorithm $\mathsf{POH.Setup}(1^\lambda, 1^\omega)$ is run in order to generate the public parameters for the PoH. The security parameter λ is globally fixed but the difficulty parameter ω needs to be adjusted dynamically to ensure a stable block creation rate. In Bitcoin the difficulty parameter is adjusted every 2016 blocks such that the expected block generation interval is 10 minutes, assuming no changes in the global mining power. Although it is unknown if these parameters are optimal, there is insufficient research covering the choice of parameters and thus we see no reason to deviate from them.

The captcha generators $\mathcal{C}_1, \dots, \mathcal{C}_k$ are initialized by computing the algorithm $\mathsf{MPC}.PP \leftarrow \mathsf{MPC.Setup}(1^\lambda)$. After generating the public parameters for the PoH the human miner $\mathcal{M}$ contacts the set of captcha generators to receive a PoH puzzle x_i for the new block B_i as we will explain in the following.

The human miner splits the hash of the transactions $\mathcal{H}(Tx_i)$, as well as the hash of the current block h_{i-1} into secret shares $\langle\mathcal{H}(Tx_i)\rangle$, $\langle h_{i-1}\rangle$ which are distributed to the captcha generators.[3] Each captcha generator computes the captcha parameters as $PP \leftarrow \mathsf{C.Setup}(1^\lambda)$. Together they compute a random captcha solution in the secret shared domain as follows. First, each captcha generator $\mathcal{C}_j$ chooses a secret input y_j uniformly at random. This secret randomness is shared among the captcha generators by computing $\langle y_j\rangle = \mathsf{MPC.Share}(\mathsf{MPC}.PP, , y_j)$. The shared randomness is used to compute the secret shared random captcha solution as $\langle\sigma_i\rangle = \mathsf{C.W}(PP, \langle y_1\rangle \oplus \cdots \oplus \langle y_k\rangle)$. This way, none of the captcha generators knows the solution σ_i.

To be able to use our PoH construction from above in a blockchain we need to include a reference to the previous block $h_{i-1} = \mathcal{H}(B_{i-1})$ and the new transactions Tx_i in the puzzle. Otherwise, if an already persisted transaction in the blockchain is modified the PoH is still valid, and thus integrity of persisted transactions cannot be guaranteed. In order to connect the hash of the previous block and the transactions with the puzzle the captcha generators compute their secret shares of the captcha puzzle Z_i given their shares of the solution $\langle\sigma_i\rangle$ as $\langle Z_i\rangle = \mathsf{C.G}\left(PP, \langle\sigma_i\rangle; \overline{\mathcal{H}}\left(h_{i-1}, \mathcal{H}(Tx_i), \langle\sigma_i\rangle\right)\right)$. I.e., the hash of the previous block h_{i-1} and the current transactions $\mathcal{H}(Tx_i)$ are included in the randomness of the puzzle generation At this stage, each captcha generator has a share of a captcha puzzle $\langle Z_i\rangle$ where the solution is effectively unknown to any single party.

Next, the captcha generators send their signed final shares of the captcha puzzle to the human miner $\mathcal{M}$ who assembles them as a PoH $x_i = (\mathsf{C}.PP, Z_i, \tau_i)$, where $Z_i = \mathsf{MPC.Reveal}(\mathsf{MPC}.PP, \langle Z_i\rangle)$ is the revealed captcha puzzle. Here, τ_i is the set of the final signed shares of the puzzle from the multiparty protocol run. It is used to prove that each captcha generator participated in the protocol and thus guarantees that the puzzle has been generated in a correct way.

[3] We explain our protocol using classical secret sharing based MPC, where a dealer distributes shares of the input and a set of nodes computes on these shares. We hope this makes our explanations more clear. In a practical implementation we suggest the use of SPDZ [9] which is a highly optimized variant thereof.

The human can now try to solve its PoH x_i. If the hash of the solution to the encapsulated captcha is too big, that is, when $\mathcal{H}(\sigma) \geq T_\omega$, the human requests another PoH puzzle from the captcha generators. If the human eventually succeeds to find a solution σ'_i to the PoH, it can locally verify its solution, by checking if $Z_i = \mathsf{C.G}\big(\mathsf{C}.PP, \sigma'_i; \overline{\mathcal{H}}(h_{i-1}, \mathcal{H}(Tx_i), \sigma'_i)\big)$. If that is the case, it can publish the new block B_i containing the captcha puzzle x_i, its solution $a_i = \sigma'_i$, as well as the transactions Tx_i and a reference to the previous block in form of a hash h_{i-1}.

Each receiving node verifies the solution to the captcha, by running $\mathsf{POH.V}(PP, x_i, , a_i)$. More specifically, the captcha puzzle Z_i is recomputed from its solution and it is examined if this leads to the same Z_i, i.e., if $Z_i = \mathsf{C.G}\big(\mathsf{C}.PP, \sigma_i; \overline{\mathcal{H}}(h_{i-1}, \mathcal{H}(Tx_i), \sigma_i)\big)$. Additionally the signed shares of the puzzle τ_i are checked for correctness of the signature to guarantee that the puzzle was created by the correct parties. Further it is examined if $\mathcal{H}(\sigma) < T_\omega$ holds.

Beyond these steps of verification of the PoH for usage in a blockchain, the difficulty parameter ω and the validity of the transactions in the new block is checked, as in Bitcoin.

If any of these checks fails, the new block is discarded and mining continues on top of the old block B_{i-1}. Otherwise the human miners can continue to generate blocks on top of B_i.

If one of the captcha generators is malicious it may abort the generation of the puzzle to the PoH, thus preventing that new blocks can be mined by a PoH. To remedy this situation we additionally allow blocks to be mined by proof of work as in Bitcoin. However, in order to keep the advantages of the mining with human work, we use a distinct difficulty parameter from the proof of human-work. The difficulty parameter of the proof of work is chosen in such a way that mining a block using proof of work is significantly harder than mining with proofs of human-work. This ensures that the mining process is dominated by PoH and proof of work is only used as a fallback mechanism.

3.4 Choosing the Captcha Generators

One important design consideration is the choice of the captcha generators. We provide two alternatives, namely a static consortium and a dynamic choice of the captcha generators based on the randomness contained in the blockchain.

Static Consortium In the most simple case we can assume a consortium of k fixed entities. If some of them are not online, no proof of work puzzles will be generated. In this case proof of work can be used as a fallback mechanism as explained above. Thus, if the captcha generators are not online, our system collapses to proof of work mining. Due to our use of SPDZ [9] we can tolerate up to $k-1$ cheaters. However, if all k parties collude, they may be able to generate captchas where they already know the solution and thus mine faster than any human miner, achieving a significant financial gain. We can remedy this situation by providing incentives for captcha generators to expose collusions and then punish the colluding parties and reward the traitor (see next paragraph). Intuitively this provides incentives for the traitor to reveal collusion, thus preventing the formation of collusions in the first place. For this to work the captcha generators additionally publish a signed commit on their shares of the solution σ. These can be included in the information used to verify that the puzzle was generated by the correct parties τ which consists of the signed shares of the puzzle.

The Traitor Reward Protocol: The traitor reward protocol has two rounds. In a first round any captcha generator can claim that the captcha generators colluded by publishing the particular shares of the puzzle solution of each captcha generator. If collusion influenced the creation of the current block, at least the miner of the block knows this information.[4]

Other parties are allowed to chime in with their claims of collusion by also publishing commits to the respective shares of the captcha generators. After a fixed timespan the first round ends and the second round starts.

In the second round the captcha generators have to reveal their commitments on their shares of the solution. If any of them does not comply within a fixed time period, collusion can be assumed. The claims of the supposed traitors are handled in the order of their arrival. Note that since we are in a distributed setting there is no global time. However, since we want to reward only some traitor to deter collusion and not necessarily the first traitor, this poses no problem. The claimed shares of the solutions are compared with the real shares of the captcha generators and if they coincide, collusion has occurred. In this case, the witness of collusion can be persisted in the blockchain as a regular transaction and the block reward of the fraudulent block is granted to the traitor. Note that there is no need to invalidate the block which has been mined fraudulently, since it contains only valid transactions and thus, is a valid block.

The time periods for the traitor reward protocol need to be chosen appropriately and the block reward needs to be locked for a fixed amount of time to prevent that it is already spent before collusion claims can be handled.

Consequently each captcha generator can choose to either collude or not collude and orthogonally to betray the other nodes or refrain from doing so, leading to the four strategies (collude, betray), (not collude, claim betrayal), (collude,

[4] It may be the case that the k colluding parties decide to reveal the solution to the PoH by MPC, such that no one knows the partial solutions of the other captcha generators. Even then $k - 1$ nodes can collude to reveal their particular shares, recompute the missing share of the last captcha generator, and claim betrayal. This increases their reward in contrast to not betraying the last captcha generator. For our cases it is irrelevant if a subset of captcha generators or only a single one claims betrayal. Though for the sake of simplicity we assume a single traitor.

not betray), and (not collude, not claim betrayal). The incentives need to be designed such that not colluding and not claiming betrayal has to be the strictly dominant strategy in a game-theoretic sense, because this is the behavior we want to support in the captcha generators. Colluding and not betraying the others needs to be a strictly dominated strategy, such that colluding nodes gain a profit from betraying the other conspirators. However, the profit needs to be smaller than if there would have been no collusion at all. Otherwise it may be rational to stage betrayal and share the reward with the other nodes. For the other two strategies there are no restrictions.

It is interesting to note that under the assumption of rational actors the traitor reward protocol will never be executed. Thus, collusions are prevented by the existence of the traitor reward protocol and not by its execution.

Dynamic Consortium The second alternative is to choose the k miners dynamically. We elect the k last miners who found a block to serve as captcha generators. Since this choice is essentially random they collude only with small probability. The captcha generators can be financially rewarded for staying online and generating captchas in the form of an additional block reward in the newly mined blocks.

A choice of k among the last n miners who found a block as captcha generators is also conceivable. This decreases the probability that they are offline, but increases the probability of collusion.

The two remaining problems we face in the scenario with a dynamic choice of captcha generators are that (i) some of the captcha generators are offline and hence no new PoH puzzles are generated, or that (ii) all k of them collude.

If some of the captcha generators are offline and no new puzzles can be generated we allow the mining of blocks with a proof of work using a high difficulty parameter as fallback option as explained before in our construction. The difficulty of finding a block with a proof of work should be significantly higher than solving a proof of human-work, such that it is only used as a fallback option. With this design choice we can guarantee liveness of the chain even if all captcha generators are offline.

Concerning the second problem, when all k captcha generators collude, they can share the solutions and quickly mine blocks without PoH, thereby splitting the block rewards among themselves. As in the case of a static consortium traitors are incentivized by a reward to expose the collusion and the colluding parties are punished. The traitor reward protocol is the same as in the case of a static consortium explained above. Again, the captcha generators publish a signed commit on their shares of the solution σ for the traitor reward protocol to allow for verification of the witness of collusion.

4 Security

Additionally to the trust assumptions in usual blockchain systems, we require that at least one of the k captcha generators is honest due to our use of SPDZ [9]. For the security of our PoH scheme we require that it is instantiated with a secure captcha system. If this is not the case and the captcha can be solved without human work, our uMine construction will not lose its functionality but instead degrade to a form of proof of work. Other than the use of a secure captcha we do not impose any additional trust assumptions.

Since our main focus is to substitute the proof of work by an environmentally friendly alternative, some of the attacks in Bitcoin also affect our scheme. In particular, since we treat forks as in Bitcoin, our construction is vulnerable to 51% attacks and eclipse attacks [16]. Although, to successfully pull off a 51% attack an attacker needs to be in charge of more than 50% of the human work units in the system instead of more than 50% of the computational resources, as in Bitcoin. However, we do not introduce any new security vulnerabilities under our assumptions.

In the following we discuss the infeasibility of selected attacks.

History Rewriting: If old transactions are changed in the blockchain, the solution to the captcha is invalidated, since the transactions are also used in the generation of the puzzle Z_i from the solution σ_i as follows: $Z_i = \mathsf{C.G}\big(PP, \sigma_i; \overline{\mathcal{H}}(B_{i-1}, \mathcal{H}(Tx_i), \sigma_i)\big)$.

Finding two sets of transactions $Tx_i \neq Tx'_i$ which yield the same puzzle Z_i for the solution σ_i, implies that $\overline{\mathcal{H}}(B_{i-1}, \mathcal{H}(Tx_i), \sigma_i) = \overline{\mathcal{H}}(B_{i-1}, \mathcal{H}(Tx'_i), \sigma_i)$, since $\mathsf{C.G}$ is collision-free in its randomness by assumption. So, an attacker would have to find a collision in the slow hash function $\overline{\mathcal{H}}$ to successfully change the old transactions which is assumed to be infeasible. Note that changing the solutions σ_i also does not yield an attack, since they are referenced in the next block.

Thus, the only way left to change transactions already persisted in the blockchain would be to split the chain after this block and redo the human work. This is only possible if an attacker controls more than 50% of the human resources in the network.

Transaction Denial Attack: In a transaction denial attack, the attacker tries to prevent a transaction from being confirmed. If the attacker is a human miner, it can only succeed if his chain grows faster than the chain containing the transaction it wants to censor. This is exactly the case if it has more than 50% of the human power in the system. As soon as that is not the case anymore, the transaction will be included in the blockchain.

If the attacker is one of the captcha generators instead, it cannot prevent inclusion of the transaction in the chain, by not serving a captcha to the miners which want to include that specific transactions. That is due to the fact that the captcha generators do not see the transactions but only their hash. Identifying if a transaction is included in a set of transactions, given only the hash of the set is infeasible.

Thus, an attacker is unable to target specific transactions for denial.

Bruteforcing of Solutions: Since practical captchas normally do not have much entropy—image based captchas consists of up to 12 characters—an attacker $\mathcal{A}$ may have the idea to simply brute-force the solution σ_i to a puzzle Z_i. This would possibly allow $\mathcal{A}$ to mine a block without spending human labor on it.

While we can almost never fully prevent brute-forcing, our use of a slow hash function impedes the attempts of the attacker. To brute-force a solution, an attacker needs to guess a σ_i', compute $Z_i' = \mathsf{C.G}\big(PP, \sigma_i'; \overline{\mathcal{H}}(B_{i-1}, \mathcal{H}(Tx_i), \sigma_i')\big)$ and then check if $Z_i' = Z_i$. I.e., $\mathcal{A}$ needs to evaluate a slow hash function for each guess, which is expensive.

5 Notes on Implementing our Scheme

While in theory it is possible to evaluate any computable function in a multiparty fashion, in practice this is not as straightforward. Most multiparty computation frameworks allow the evaluation of gates of a circuit only. Consequently, we need to represent a secure captcha generation algorithm as a circuit in order to implement our proof of human-work scheme.

We suggest to use SPDZ due to its high performance. SPDZ allows to describe the function for the multiparty computation protocol in a language which is heavily inspired by Python. In particular, it supports the basic data types and syntax of Python. The SPDZ framework can compile this high level language to virtual machine code for the SPDZ virtual machine. The virtual machine code is further compiled to an arithmetic circuit before it is evaluated. Thus, in order to implement our proof of human-work scheme, instead of representing a secure captcha generation algorithm as a circuit it suffices to represent it in a subset of Python.

We tried to implement our proof of human work scheme using the Python package captcha version 0.2.4 which can be used to create visual captchas. However, we did not succeed in implementing our scheme, since the captcha package uses many Python language constructs which are not supported in SPDZ. In particular, the font parsing, as well as the image processing is not done in Python, but in a C library and thus is unsupported by SPDZ.

In the following, we describe how our scheme can be implemented. Our description of a secure captcha is based on the Python captcha package.

As described in Figure 1, in order to generate proof of human work, i.e., to execute $\mathsf{POH.G}(PP)$, the scheme needs to be set up first. This is straightforward. The public parameters may contain the length of the captcha, as well as the set of allowed characters, and the typeface used. Next, each captcha generator $\mathcal{C}_j$ chooses a secret random value y_j and distributes the secret shares $\mathsf{MPC.Share}(\mathsf{MPC}.PP, y_j)$, which is fairly trivial and can be executed directly with SPDZ. After the secret shares are distributed, the captcha generators sample a secret shared solution to a captcha as $\langle\sigma\rangle = \mathsf{C.W}(\mathsf{C}.PP; \langle y_1\rangle \oplus \cdots \oplus \langle y_k\rangle)$. The algorithm $\mathsf{C.W}$ returns as output a solution to a captcha, i.e., a string whose length depends on the length contained in the public parameters. The randomness for this sampling process depends on $\langle y_1\rangle \oplus \cdots \oplus \langle y_k\rangle$. In order to implement W, a secure pseudo random number generator is used whose seed is set to $\langle y_1\rangle \oplus \cdots \oplus \langle y_k\rangle$. This is hard to do in practice, since it is not clear how to represent a secure pseudo random number generator as an arithmetic circuit.

As a next step, a captcha puzzle Z with the solution string σ is computed as $\langle Z\rangle \leftarrow \mathsf{C.G}\left(\mathsf{C}.PP, \langle\sigma\rangle; \overline{\mathcal{H}}(\langle\sigma\rangle)\right)$. Representing the slow hash function $\overline{\mathcal{H}}$ as an arithmetic circuit should be easy. However, computing $\mathsf{C.G}$ is difficult. In practice $\mathsf{C.G}$ takes the string σ and represents it as an image in the typeface given in $\mathsf{C}.PP$. Next, various transformations are applied on the resulting image such as rotation, warping, adding noise curves and noise dots. These transformations and their parameters are determined by $\overline{\mathcal{H}}(\langle\sigma\rangle)$. While these steps are trivial, implementing them as an arithmetic circuit is not. SPDZ currently has support for neither parsing TrueType fonts nor image processing. In order for our proof of human-work scheme to be implemented, it is necessary to implement these features as an arithmetic circuit.

For convenience it is possible to use a lookup table for individual letters of a fixed font instead of determining the typeface through the public parameters of the captcha. Images can be represented as vectors of numbers. However, for their transformations a separate library is desirable, as multiparty computation on images constitutes a reusable building block for other protocols. As the parameters for the transformation are determined by a pseudo random number generator whose seed is set to $\overline{\mathcal{H}}(\langle\sigma\rangle)$, again we need a pseudo random number generator represented as arithmetic circuit.

The further steps in creating the proof of human-work, i.e., signing the shares and revealing the puzzle are trivial.

In summary the problems concerning the implementation of our proof of human-work scheme are restricted to the representation of the captcha generation algorithm as an arithmetic circuit. After tackling this step, the SPDZ protocol guarantees execution as a multiparty protocol with only a constant overhead as opposed to normal execution.

6 Related Work

There is a series of related work which suggests an alternative to Bitcoin's wasteful proof of work.

The most famous among these approaches is probably **proof of stake** [20, 6], where the scarcity used to power the blockchain is the underlying currency itself. In proof of stake the miner of the next block is chosen pseudorandomly among the set of all miners. The probability of a miner being chosen to create a new block is dependent on its wealth which can lead to an undesirable "rich get richer" scenario. A common problem in proposals for proof of stake is that in the case of a blockchain fork miners have nothing to lose by trying to mine on both chains, thus preventing the fork from resolving. Peercoin [20] solves this problem by including centralized checkpoints in the code, thus introducing a trusted third party. Other protocols such as Algorand [15] do not provide incentives for the participants. The first construction to provably solve the proof of stake problem is due to Kiayias et al. [18].

Other approaches to substitute proof of work are **proofs of storage** [21, 32, 24] i.e., proving possession of a specific file, or **proofs of space** [4, 13, 28, 29], where a miner only constructs a proof about the size of its memory resources. However, these approaches will also invariably degrade into a hardware arms race and thus do not solve the problem of the vast energy consumption of blockchain technology.

A spiritual predecessor of our work is **HumanCoin** [8]. However HumanCoin is based on a PoH based on indistinguishability obfuscation [14], a cryptographic principle where no construction is known yet. Thus, HumanCoin is currently infeasible.

Their construction of a PoH, and consequently HumanCoin requires a trusted setup phase for the generation of the obfuscated programs which are used to generate the captchas without revealing the solutions.

If the unobfuscated programs are known, miners can generate puzzle-solution pairs to the PoH without spending any human work by running the puzzle-generation in the clear. In contrast, in our work we are able to verify that the puzzles have been created in such a way that no-one knows the solution by publishing and verifying the signed final shares of the captcha generators τ.

In HumanCoin, collision-freeness of the captcha puzzle generation algorithm is not stated, but if this is not assumed their scheme is trivially insecure. Their PoHs are generated by $x_i = \mathsf{C.G}\big(PP; \mathcal{H}(Tx_i, h_{i-1})\big)$, where h_{i-1} is the hash of the previous block. If there is no collision resistance in the randomness, old transactions can be changed without changing the puzzle, thus invalidating the integrity of the transactions stored in blockchain.

HumanCoin does not implement any countermeasures against brute-forcing the solution to the PoH from its verification function in contrast to our use of a slow hash function.

In contrast to HumanCoin our PoH puzzle generation phase is online and requires k captcha generators. However, this poses no problem, since to mine new blocks a miner has to receive new transactions and blocks and thus is required to be online anyway.

7 Conclusion

We have introduced uMine, an energy-efficient alternative to proof of work mining which utilizes human workers. Our construction is based on a novel instantiation of proofs of human-work which relies on MPC, thereby answering an open question of Blocki and Zhou [8] whether proofs of human work without indistinguishability obfuscation are possible. Our proof of human-work scheme is introduced as a separate building block and thus may find applications beyond cryptocurrencies.

Our uMine system may share similarities with early manual accounting systems, whose bookkeepers were financially compensated. Additionally, our system decentralizes the ledger and provides anyone with the opportunity to be an accountant, provided it accepts the remuneration. We leave the social and economical implications of our work as an open question.

Acknowledgements

A major part of this work was conducted during a research stay at the University of Twente. This work was partially funded by the Baden-Württemberg Stiftung and by the Netherlands Organisation for Scientific Research (NWO) in the context of the CRIPTIM project.

References

1. von Ahn, L., Blum, M., Hopper, N.J., Langford, J.: Captcha: Using hard ai problems for security. In: EUROCRYPT 2003. pp. 294–311. Springer (2003)
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: S&P 2014. pp. 443–458. IEEE (2014)
3. Antonopoulos, A.M.: Mastering Bitcoin, Unlocking Digital Cryptocurrencies. O'Reilly Media (December 2014), `https://github.com/aantonop/bitcoinbook`
4. Ateniese, G., Bonacina, I., Faonio, A., Galesi, N.: Proofs of space: When space is of the essence. In: SCN'14. pp. 538–557. Springer (2014)
5. Bendlin, R., Damgård, I., Orlandi, C., Zakarias, S.: Semi-homomorphic encryption and multiparty computation. In: EUROCRYPT 2011. pp. 169–188. Springer (2011)
6. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Financial Cryptography and Data Security 2016. pp. 142–157. Springer (2016)
7. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: CRYPTO 2014. pp. 421–439. Springer (2014)
8. Blocki, J., Zhou, H.S.: Designing proof of human-work puzzles for cryptocurrency and beyond. In: TCC 2016. pp. 517–546. Springer (2016)

9. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: CRYPTO 2012, pp. 643–662. Springer (2012)
10. Damgård, I., Zakarias, S.: Constant-overhead secure computation of boolean circuits using preprocessing. In: TCC 2013, pp. 621–641. Springer (2013)
11. Dwork, C., Goldberg, A., Naor, M.: On memory-bound functions for fighting spam. In: CRYPTO 2003. pp. 426–444. Springer (2003)
12. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: CRYPTO 1992. pp. 139–147. Springer (1992)
13. Dziembowski, S., Faust, S., Kolmogorov, V., Pietrzak, K.: Proofs of space. In: CRYPTO 2015. pp. 585–605. Springer (2015)
14. Garg, S., Gentry, C., Halevi, S., Raykova, M., Sahai, A., Waters, B.: Candidate indistinguishability obfuscation and functional encryption for all circuits. SIAM Journal on Computing **45**(3), 882–929 (2016)
15. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: SOSP'17. pp. 51–68. ACM (2017)
16. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: USENIX 2015. pp. 129–144 (2015)
17. Keller, M., Orsini, E., Scholl, P.: Mascot: faster malicious arithmetic secure computation with oblivious transfer. In: SIGSAC 2016. pp. 830–842. ACM (2016)
18. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO 2017. pp. 357–388. Springer (2017)
19. Kiayias, A., Zhou, H.S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: EUROCRYPT 2016. pp. 705–734. Springer (2016)
20. King, S., Nadal, S.: PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake (2012), `https://peercoin.net/whitepaper`
21. Kopp, H., Bösch, C., Kargl, F.: Koppercoin - a distributed file storage with financial incentives. In: ISPEC 2016. pp. 79–93. Springer (11 2016)
22. Kopp, H., Mödinger, D., Hauck, F.J., Kargl, F., Bösch, C.: Design of a privacy-preserving decentralized file storage with financial incentives. In: IEEE EuroS&P Workshops. pp. 14–22. IEEE (2017)
23. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: S&P 2016. pp. 839–858. IEEE (2016)
24. Miller, A., Juels, A., Shi, E., Parno, B., Katz, J.: Permacoin: Repurposing Bitcoin Work for Data Preservation. In: S&P 2014. pp. 475–490. IEEE (2014)
25. Miller, A., Kosba, A., Katz, J., Shi, E.: Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In: SIGSAC 2015. pp. 680–691. ACM (2015)
26. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2009), `https://bitcoin.org/bitcoin.pdf`
27. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: CRYPTO 2012. vol. 7417, pp. 681–700. Springer (2012)
28. Park, S., Pietrzak, K., Alwen, J., Fuchsbauer, G., Gazi, P.: Spacecoin: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528 (2015)
29. Park, S., Pietrzak, K., Kwon, A., Alwen, J., Fuchsbauer, G., Gaži, P.: Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528 (2015)

30. Percival, C.: Stronger key derivation via sequential memory-hard functions. Self-published (2009), `http://www.tarsnap.com/scrypt/scrypt.pdf`
31. Provos, N., Mazieres, D.: A future-adaptable password scheme. In: USENIX 1999. pp. 81–91 (1999)
32. Sengupta, B., Bag, S., Ruj, S., Sakurai, K.: Retricoin: Bitcoin based on compact proofs of retrievability. In: ICDCN'16. pp. 14:1–14:10. ACM (2016)
33. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys & Tutorials **18**(3), 2084–2123 (2016)
34. de Vries, A.: Bitcoin's growing energy problem. Joule **2**(5), 801–805 (2018)
35. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger (2014), `http://gavwood.com/paper.pdf`