Designing Proof of Human-work Puzzles for Cryptocurrency and Beyond

Jeremiah Blocki[*]
Microsoft Research New England
jblocki@microsoft.com

Hong-Sheng Zhou
Virginia Commonwealth University
hszhou@vcu.edu

February 16, 2016

Abstract

We introduce the novel notion of a Proof of Human-work (PoH) and present the first distributed consensus protocol from hard Artificial Intelligence problems. As the name suggests, a PoH is a proof that a *human* invested a moderate amount of effort to solve some challenge. A PoH puzzle should be moderately hard for a human to solve. However, a PoH puzzle must be hard for a computer to solve, including the computer that generated the puzzle, without sufficient assistance from a human. By contrast, CAPTCHAs are only difficult for other computers to solve — not for the computer that generated the puzzle. We also require that a PoH be publicly verifiable by a computer without any human assistance and without ever interacting with the agent who generated the proof of human-work. We show how to construct PoH puzzles from indistinguishability obfuscation and from CAPTCHAs. We motivate our ideas with two applications: HumanCoin and passwords. We use PoH puzzles to construct HumanCoin, the first cryptocurrency system with human miners. Second, we use proofs of human work to develop a password authentication scheme which provably protects users against offline attacks.

1 Introduction

The emergence of decentralized cryptocurrencies like Bitcoin [49] has the potential to significantly reshape the future of distributed interaction. These recent cryptocurrencies offer several advantages over traditional currencies, which rely on a centralized authority. At the heart of Bitcoin-like cryptocurrencies is an efficient distributed consensus protocol that allows for all users to agree on the same public ledger. When combined with other cryptographic tools like digital signatures the distributed consensus protocol prevents users from engaging in dishonest behavior like "double spending" their money or spending another user's money. Fundamentally, the applications of a tamper-proof blockchain like the one in Bitcoin are not limited to cryptocurrency. For example, a tamper proof blockchain could help us develop secure and fair multiparty computation protocols [1, 7, 40, 41, 36, 38], build distributed notary/timestamp systems [44], develop smart contracts [55, 38], and build distributed autonomous agents, to name a few applications. In this paper we propose a fundamentally new technique, Proofs of Human-work (PoH), for constructing a secure blockchain, and we show that our techniques have several other valuable applications like password protection and non-interactive bot detection.

At its core, Bitcoin's distributed consensus protocol is based on moderately hard Proofs of Work (PoW) [23]. In Bitcoin the Hashcash [3] PoW puzzles are used to extend the blockchain, a cryptographic data-structure in which the public ledger is recorded. A PoW puzzle should be moderately hard for a

[*]This work was completed in part while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

computer to solve, but the PoW solution should be easy for a computer to verify. Cryptocurrencies like Bitcoin require that this hardness parameter of PoW puzzles be tunable. An adversary would need to control 51% of the computational power in the Bitcoin network to be able to alter the blockchain and prevent users from reaching the correct consensus[1]. While Bitcoin cleverly avoids the Sybil attack by using PoW puzzles, there are still many undesirable features of this distributed consensus protocol. For example, constructing the proofs of work is energy intensive making the mining process in this distributed consensus protocol environmentally unfriendly. Furthermore, the mining process is dominated by a smaller number of professional miners with customized hardware making it unprofitable for others to join — this raises the natural concern that a few professional miners might collude to alter the public ledger [50]. Indeed, the mining pool GHash.io[2] recently exceeded 50% of the computational power in Bitcoin. While other techniques like Proofs of Space [51, 25] or Proofs of Stake [8, 42] have been proposed to build the blockchain in a distributed consensus protocol each of these techniques has its own drawbacks. It is clearly desirable to find new techniques for reaching a stable distributed consensus. In this paper we ask the following question:

> *Is it possible to design proof of human-work puzzles that are suitable for a decentralized cryptocurrency?*

We believe that a cryptocurrency based on Proof of Human-work might offer many advantages over other approaches. First, the mining process would be eco-friendly. Second, it might be possible to base the proofs of human work on activities that are fun [34], educational [33] or even beneficial to society [58, 35]. Third, proofs of human work are fair by nature. Professional or rich miners would not have an advantage over regular users. Thus, this cryptocurrency could provide employment opportunities. Finally, we believe that the cryptocurrency would be less-vulnerable to 51% attacks by nation states or by a few professional miners. See the appendix for additional discussion.

1.1 Cryptocurrencies meet AI: Proof of Human-work Puzzles

In this work we introduce the novel notion of Proofs of Human-work (PoH) which would be suitable for cryptocurrencies. Proofs of Human-work are fundamentally different from standard Proofs of Work. Informally, a PoH puzzle should be moderately hard for a human to solve meaning that it should require modest effort for a human to produce a valid proof of human work — again we require that this hardness parameter should be tunable. Furthermore, the puzzles should be easy for a computer to generate, but they need to be difficult for a computer to solve without sufficient human assistance — even for the computer that generated the puzzle. Finally, the puzzles need to be publicly verifiable meaning that it should be easy for a computer to verify the solution to the puzzle without any human assistance — even if the computer did not generate the puzzle. We stress that there is no interaction during the puzzle generation or during the puzzle verification process, and there is no trusted server in our distributed setting. Thus, a computer will need to validate proofs of human-work that were generated and solved by agents with whom it has never interacted.

Our description of a PoH puzzle might remind the reader of a CAPTCHA (Completely Automated Public Turing-Test to tell Computers and Humans Apart) [57]. CAPTCHAs have been widely deployed on the Internet to fight spam and protect against sybil attacks. Informally, a CAPTCHA is a puzzle that is easy for a human to solve, but difficult for a computer. CAPTCHAs are based on the assumption that some underlying artificial intelligence (AI) problem is hard for computers, but easy for humans (e.g., reading distorted letters).

While we do use CAPTCHAs to construct proofs of human work, we stress that a CAPTCHA itself *cannot* achieve our notion of proofs of human-work. Let (Z, σ) be a CAPTCHA puzzle-solution pair.

[1] Technically byzantine agreement is only possible when the adversary has less than 50% of the hashing power and the network has high synchronicity — otherwise we need to ensure that the adversary has at most 33.3% of the hashing power [29].

[2] See `http://arstechnica.com/security/2014/06/bitcoin-security-guarantee-shattered-by-anonymous-miner-with-51-network-power/`.

Verifiers who receive the pair (Z,σ) would not necessarily be able to check that σ is the correct solution without interacting with a human. More importantly, the computer that generates the puzzle Z could produce the solution σ *without any human effort* because CAPTCHA generation algorithms start by randomly selecting a target solution σ and then outputting a randomly generated puzzle Z with the solution σ. Thus, a pair (Z,σ) does not constitute a proof of human work. The PoH verifier would need to ensure, without interacting with any other human agent or any other computer agent, that the challenge generator did not already have the answer σ to the puzzle Z.

We believe that our Proof of Human-work puzzles could also have applications in many other contexts. For example, to limit spam or prevent phishing attacks it might useful to verify that some human effort went into producing a message. When a human user is busy it would be convenient if the computer could validate this proof of human effort automatically without needing to interact with the sender who may no longer be available when the message is received. Similarly, proofs of human-work might be a useful tool for honest preference elicitation — a challenging problem in mechanism design. A human could demonstrate that a particular issue or outcome is truly important to him by producing a proof of human-work.

1.2 AI meets Obfuscation: Constructing Proof of Human-work Puzzles

It is not immediately clear how to construct PoH puzzles. CAPTCHAs allow a computer to generate puzzles that other computers cannot solve, but how could a computer generate a puzzle that is meaningful to a human without learning the answer itself? Even if this were possible how could a puzzle verifier be convinced that the puzzle(s) was generated honestly (e.g., in a way that does not reveal the answer) without any interaction? How could the verifier be convinced that the answer is correct without help from a human? Building PoH puzzles is a challenging problem.

To address these issues, we need to have a way to generate CAPTCHAs *obliviously* in the sense that a computer is able to generate a well-formed puzzle instance Z without learning the corresponding solution σ. This is feasible by leveraging recent breakthroughs in indistinguishability obfuscation [30]. At an intuitive level, we can have a CAPTCHA puzzle Z generated inside an obfuscator, and now the corresponding answer σ remains hidden inside the obfuscated program. We note that the puzzle solution verification can also take place inside an obfuscated program, even without having human effort involved.

Once we have the idea of generating a CAPTCHA puzzle oblivously as mentioned above, we then can mimic the steps of constructing Proof of Work puzzle in Bitcoin to get a PoH scheme. In PoW, a prover/miner is given an puzzle instance x. The prover will compute the cryptographic hash $H(x,s)$ for many distinct witness s until the value $H(x,s)$ is smaller than a target value. In PoH, the miner uses (x,s) as the input for an obfuscated program, and inside the obfuscated program, a pseudorandom string r is generated from the input (x,s), and this r will be used for generating the solution σ and the puzzle instance Z. The miner obtains Z but has no access to the internal state r and σ.

A human miner is now able to obtain the solution σ from the puzzle Z. As in PoW, the miner will repeat this process until he finds a witness s so that $H(x,s,\sigma,Z)$ is smaller than a target value. We note that, once a successful miner publishes a valid tuple (x,s,σ,Z), any verifier is able to verify it without interaction with human: The verifier can reproduce Z inside the obfuscated program along with a verification tag, *tag*. While the verification tag allows the verifier to check whether a given solution σ is correct this value will not expose the solution σ (e.g., *tag* might be an obfuscated point function which outputs 1 on input $x=\sigma$ and 0 on all other inputs).

Our PoH scheme maintains many of the same desirable properties as a PoW. For example, we can tune the hardness of our PoH puzzle generator by having the verifier reject a valid triple (x,s,σ,Z) with probability $1-2^{\omega}$ so that a human would need to generate and solve 2^{ω} on average to produce a valid proof of human-work. Thus, the hardness of the PoH puzzles could be tuned by adjusting ω.

While the conceptual understanding of our PoH construction is quite simple, the security analysis is a bit tricky. In the PoW, we sample from a uniform distribution via random oracle, here we need to sample from a more sophisticated distribution. We rely on a newly developed tool *universal samplers* by Hofheinz et al. [32], which is based on the existence of indistinguishability obfuscation and one-way functions in the random oracle model. As discussed in [32], we stress that the random oracle is only used outside of obfuscated programs.

There has been tidal wave of new cryptographic constructions using indistinguishability obfuscation since the groundbreaking results of Garg et al. [30]. However, to the best of our knowledge we are the first rigorous paper to explore the connection between AI and program obfuscation[3]. We believe that obfuscation is a powerful new tool that has the potential to fundamentally shape the nature of human-computer interaction. Could program obfuscation allow for a human to interact with a computer in fundamentally new ways? We view our work as a first step towards answering this question.

Remark. We view our Proof of Human-work construction as a novel proof of concept that is not yet practical due to the use of indistinguishability obfuscation. Since the work of Garg et al. [30] several other candidate indistinguishability obfuscation schemes have been proposed, but a practical obfuscation scheme would still be a major breakthrough. We note that PoH puzzles do not necessarily require general purpose indistinguishability obfuscation. It would be sufficient to obfuscate a few very simple programs (e.g., a CAPTCHA puzzle generator). Constructing PoH puzzles without obfuscation (or without general purpose obfuscation) is an interesting open problem.

Other Applications. The applications of our techniques are not limited to cryptocurrency. In Section 5 we use our ideas to build a password authentication scheme that provably resists offline attacks even if the adversary breaches the authentication server. The basic idea is to require a proof of human-work during the authentication process so that it is not economically feasible for the adversary to check millions of password guesses. We also show how to develop a non-interactive bot detection protocol which allows Alice to send a message m to Bob along with a proof of human-work. Bob is able to verify that human-effort was used in the production/transmission of the message m without ever interacting with Alice.

1.3 Related Work

While there are many variations of CAPTCHAs [57], they are all based on the fundamental assumption that some underlying AI problem is hard (e.g., reading garbled text [58], voice recognition with distorted audio [53],image recognition [26] or even motion recognition). While several CAPTCHAs have been broken (e.g., [47, 56, 16]) there is still a clear gap between human intelligence and artificial intelligence. We conjecture that in the foreseeable future we will continue to have viable CAPTCHA candidates suitable for proofs of human work. CAPTCHAs have many applications in security: fighting spam [57], mitigating Sybil attacks [20], preventing denial of service attacks [61] and even preventing fully automated man-in-the-middle attackers [24]. As we noted earlier CAPTCHAs alone are not suitable as PoH puzzles. Kumarasubramanian et al. [39] introduced the notion of human-extractable CAPTCHAs, and used them to construct concurrent non-malleable zero-knowledge protocols.

Canneti et al. [18] proposed a slight modification of notion of CAPTCHAs that they called HOSPs (Human Only Solvable Puzzles) as a defense against offline attacks on passwords. HOSPs are similar to PoHs in that the puzzles must be difficult even for the computer that generates them, but HOSP puzzles are not publicly verifiable by a computer and their construction assumes the existence of a large centralized storage server filled with unsolved CAPTCHA challenges. This makes their protocol vulnerable to pre-

[3] Several existing altcoins (e.g., Bytecent, CaptchaCoin) do involve CAPTCHAs, but they rely on a trusted third party to generate the CAPTCHAs. There has also been informal discussion on the Bitcoin research chatroom about basing cryptocurrency on proofs of human labor. For example, see https://download.wpsoftware.net/bitcoin/wizards/2014-05-29.html or http://vitalik.ca/files/problems.pdf.

computation attacks[4]. By contrast, in Section 5 we present a protocol for password storage that provably protects users against offline attacks, does not require a large centralized storage server and is not vulnerable to pre-computation attacks. Blocki et al. [9] introduced GOTCHAs (Generating panOptic Turing Tests to Tell Computers and Humans Apart) as a defense against offline dictionary attacks on passwords. GOTCHAs are not suitable for cryptocurrency because the puzzle generation protocol requires interaction with a human. Furthermore, GOTCHAs are not publicly verifiable by a computer. We refer an interested reader to the appendix for more details about CAPTCHAs and HOSPs.

The problem of designing distributed consensus protocols that work in the presence of a adversarial (Byzantine) parties has been around for decades [43, 22, 2]. Typically distributed consensus require that $2/3$ of the parties are honest [43]. On the internet this assumption is typically not valid because it is often possible for a malicious user to register for multiple fake accounts — a Sybil attack [20]. Bitcoin's distributed consensus protocol is based on the observation that that an adversary cannot gain computation power by registering for duplicate accounts. It was originally claimed that distributed consensus protocol in Bitcoin requires that an adversary controls at most 49% of the computational power [49], but Garay et al. [29] found a flaw in the argument. Garay et al. [29] provide a more careful analysis demonstrating that byzantine agreement is possible when the adversary has less than 50% of the hashing power if we assume that the network has high synchronicity — otherwise we need to ensure that the adversary has at most 33.3% of the hashing power [29]. Bitcoin was originally described in a pseudoanonymous publication by 'Nakamoto' [49]. At its core Bitcoin is based on an elegant distributed consensus protocol which in turn is based on Proof of Work puzzles [23] to allow users to agree on a common blockchain. Bitcoin uses the Hashcash Proof of Work algorithm due to Back [3]. Other cryptocurrencies like SpaceCoin [51] use proofs of space [25] under the assumption that no malicious user possesses most of the storage space on the network. Eyal and Sirer [27] pointed out that Bitcoin's consensus protocol is not incentive compatible, but that the protocol could be modified so that it is incentive compatible as long as all mining pools command at most 25% of the hashing power. Thus, it is concerning that the Bitcoin mining pool GHash.io currently has more than 50% of the computational power. Miller et al. [46] introduced the notion of non-outsourceable puzzles to discourage the formation of large mining coalitions.

Since the breakthrough result of Garg et al. [30], demonstrating the first candidate of indistinguishability obfuscation for all circuits, a myriad of uses for indistinguishability obfuscation in cryptography have been found. Among these results, the puncturing methodology by Sahai and Waters [52] has been found very useful. Hofheinz et al explored the puncturing technique further introducing and constructing universal samplers in the random oracle model [32]. Their universal sampler is one of the key building blocks in our construction of proof of human-work puzzles. We remark that our work is distinct from previous applications in that we are using obfuscation to develop a new way for *humans* to interact with computers. Finally, we point out that we here will not give a full list of recent results about obfuscations. We refer readers to [60, 59] for new constructions, applications, and limitations of obfuscations.

2 Preliminaries

We adopt the following notational conventions: Given a randomized algorithm $\mathcal{A}$ we use $y \leftarrow \mathcal{A}(x)$ to denote a random sample from the distribution induced by an input x. If we fix the random bits r then we will use $y := \mathcal{A}(x;r)$ to denote the deterministic result.

We will consider two types of users: machine-only users and human-machine users. A machine-only user is a probabilistic polynomial time (PPT) algorithm who does not interact with a human. In general, when we say "human" user we mean a "human user equipped with a PPT machine." Accordingly, we also consider two types of adversaries: a machine-only adversary $\mathcal{A}$, and a human-machine adversary $\mathcal{B}^{\mathcal{H}}$.

[4] In particular, the adversary might pay to solve every CAPTCHA challenge on the server. While expensive, this one-time cost would amortize over the number of users being attacked.

The machine-only adversary is a PPT algorithm that does not get to query a human. The human-machine adversary $\mathcal{B}^{\mathcal{H}}$ is a PPT algorithm that gets to interact with a human oracle $\mathcal{H}$ which could, for example, solve CAPTCHA puzzles. We typically restrict the total number of queries that human-machine adversary can make to the human oracle.

2.1 CAPTCHAs

CAPTCHAs are a fundamental building block in our construction of Proof of Human-work puzzles. Traditionally, a CAPTCHA generator $\mathtt{G}$ is defined as a randomized PPT algorithm that outputs a puzzle Z and a solution σ. In every CAPTCHA generator that we are aware of the program $\mathtt{G}$ first generates a random target solution σ and then produces a random puzzle Z with solution σ (e.g., by distorting the string σ). Given public parameters PP for CAPTCHA puzzle generation we adopt the syntax $(Z, tag) \leftarrow \mathtt{G}(\mathrm{PP}, \sigma)$ to emphasize that the target puzzle Z is generated with complete knowledge of the CAPTCHA solution. In traditional CAPTCHA applications it is desirable for the agent who generates a puzzle Z to have knowledge of the corresponding answer σ so that he can verify another agent's response to the challenge Z. However, in our setting this property is problematic since the agent who generates the puzzle Z is trying to produce a convincing proof of human-work. Thus, we will need additional tools to obtain proof of human-work puzzles from CATPCHAs. Formally, a CAPTCHA puzzle-system is defined as follows.

Definition 2.1 (CAPTCHA). *A CAPTCHA puzzle system consists of a tuple of algorithms* $(\mathtt{Setup}, \mathtt{W}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{Verify})$*, where*

- $\mathtt{Setup}$ *is a randomized system setup algorithm that takes as input* 1^λ *(*λ *is the security parameter), and outputs a system public parameter* $\mathrm{PP} \leftarrow \mathtt{Setup}(1^\lambda)$*, which includes a puzzle size parameter* $\ell = \mathrm{poly}(\lambda)$*;*
- $\mathtt{W}$ *is a randomized sampling algorithm that takes as input the public parameter* PP *and outputs a target solution* $\sigma \leftarrow \mathtt{W}(\mathrm{PP})$ *(e.g., a witness) of length* ℓ*;*
- $\mathtt{G}$ *is a randomized puzzle generation algorithm that takes as input the public parameter* PP *and a solution* σ*, and outputs* $(Z, tag) \leftarrow \mathtt{G}(\mathrm{PP}, \sigma)$ *where* Z *is a CAPTCHA puzzle and tag is a string that may be used to help verify a solution to* Z*;*
- $\mathtt{Verify}$ *is a verification algorithm that takes as input the public parameters* PP*, a puzzle* Z *along with the associated tag and a proposed solution* σ' *outputs a bit* $b := \mathtt{Verify}(\mathrm{PP}, Z, tag, \sigma')$*. We require that* $b = 1$ *whenever* $(Z, tag) \leftarrow \mathtt{G}(\mathrm{PP}, \sigma)$ *and* $\sigma' = \sigma$*;*
- $\mathtt{C}^{\mathcal{H}}$ *is a solution finding algorithm (i.e., human-machine solver) that takes as input the public parameter* PP *and a puzzle* Z*, and outputs a value* $a \leftarrow \mathtt{C}^{\mathcal{H}(\cdot)}(\mathrm{PP}, Z)$ *as the solution to the puzzle* Z*. Here,* $\mathcal{H}(\cdot)$ *denotes the human oracle which takes intermediate human-efficient objects (such as images) as inputs, and returns machine-efficient values as outputs.*

We typically require that $\mathtt{Setup}$*,* $\mathtt{W}$*,* $\mathtt{G}$ *are probabilistic polynomial time algorithms, and* $\mathtt{Verify}$ *a deterministic polynomial time algorithm.* $\mathtt{C}$ *should be a probabilistic polynomial time oracle machine.*

For example, if we are defining a text based CAPTCHA puzzle-system the public parameters PP might specify the set of characters Σ, the set of fonts and a set of font sizes/colors. The public parameters PP would also describe the length $\ell = |\sigma|$ of the target solution (e.g., the number of characters in the CAPTCHA). In general, larger security parameters λ would imply longer puzzles. $\mathtt{W}$ is a randomized algorithm that outputs a random string $\sigma \in \Sigma^*$ (the target solution), and $\mathtt{G}$ is the randomized algorithm that produces a puzzle Z along with a *tag* which may be used for public verification of a potential solution σ'. We view the solution function $\mathtt{C}^{\mathcal{H}}$ as a a human equipped with a PPT computer. Typically the computer would just be used to display the challenge to the user, but it could also apply a more sophisticated algorithm to post-process the user's answer.

Fixing the security parameter λ we define one human work unit to be the amount of time/energy that it takes a human to solve one honestly generated CAPTCHA puzzle $Z \leftarrow \mathtt{G}(\mathrm{PP}, \sigma)$. Any CAPTCHA puzzle-system should be human usable, meaning that a typical human can consistently solve randomly generated CAPTCHA puzzles. While we recognize that solving a CAPTCHA puzzle may require more effort for some people than for others we will use the term human-work unit to denote the amount of human effort necessary to solve one CAPTCHA puzzle with security parameter λ.[5]

Definition 2.2 (Honest Human Solvability)**.** *We say that a human-machine solver* $\mathtt{C}^{\mathcal{H}}$ *controls m human-work units if the machine* $\mathtt{C}$ *can query the human oracle* $\mathcal{H}(\cdot)$ *at least m times. We say a CAPTCHA puzzle-system* $(\mathtt{Setup}, \mathtt{W}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{Verify})$ *is* honest human solvable *if for every polynomial* $m = m(\lambda)$ *and for any human* $\mathtt{C}^{\mathcal{H}}$ *who controls m human-work units, it holds that*

$$\Pr\left[\begin{array}{c} \mathrm{PP} \leftarrow \mathtt{Setup}(1^\lambda); \forall i \in [m]\big(\sigma_i^* \leftarrow \mathtt{W}(\mathrm{PP})\big); \\ \forall i \in [m]\big((Z_i^*, tag_i^*) \leftarrow \mathtt{G}(\mathrm{PP}, \sigma_i^*)\big); \\ (\sigma_1^*, \ldots, \sigma_m^*) \leftarrow \mathtt{C}^{\mathcal{H}(\cdot)}(\mathrm{PP}, Z_1^*, \ldots, Z_m^*) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda)$$

Finally, we require that CAPTCHAs are hard for computers to invert. More concretely, no PPT adversarial machine should be able to find the solutions to $m+1$ honestly-generated puzzles given only m-human work units. We introduce two similar notions of computer uncrackable CAPTCHAs. The first version states that an adversary with m human-work units cannot find the solution to $m+1$ CAPTCHAs with non-negligible probability when he is only given the puzzles $Z_1^*, \ldots, Z_n^*$ $(n > m)$.

Definition 2.3 (Computer Uncrackable v1)**.** *We say that a CAPTCHA puzzle-system* $(\mathtt{Setup}, \mathtt{W}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{Verify})$ *is* computer uncrackable *if for any* PPT *adversary* $\mathcal{A}$ *who has at most m human-work units and any* $n = poly(\lambda)$

$$\Pr\left[\begin{array}{l} \mathrm{PP} \leftarrow \mathtt{Setup}(1^\lambda);\ \forall i \in [n]\big(\sigma_i^* \leftarrow \mathtt{W}(\mathrm{PP})\big); \\ \forall i \in [n]\big((Z_i^*, tag_i^*) \leftarrow \mathtt{G}(\mathrm{PP}, \sigma_i^*)\big); \\ S \leftarrow \mathcal{A}^{\mathcal{H}(\cdot)}(\mathrm{PP}, Z_1^*, \ldots, Z_n^*); \\ \forall i \in [n]\big(b_i \leftarrow \max_{\sigma \in S} \mathtt{Verify}(\mathrm{PP}, Z_i^*, tag_i^*, \sigma)\big); \\ \qquad \sum_{i \in [n]} b_i \geq m+1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Our second formulation of computer uncrackable CAPTCHAs is slightly non-standard due to the fact that the adversary is given a tag tag_i along with each challenge Z_i. In particular, the value tag_i allows the adversary to run $\mathtt{Verify}(\mathrm{PP}, Z_i, tag_i, \sigma_i')$ to test different candidate CAPTCHA solutions. While this formulation is non-standard we argue that we would expect that any CAPTCHA that is secure under definition 2.3 can be transformed into a CAPTCHA that is secure under definition 2.3. For example, tag_i might be the cryptographic hash of the solution σ_i or we might set $tag_i = i\mathcal{O}\big(I_{Z_i \sigma_i}\big)$ to be the indistinguishability obfuscation of a point function $I_{Z_i,\sigma_i}(x) = 1$ if $x = \big(Z_i, \sigma_i\big)$; otherwise $I_{Z_i,\sigma_i}(x) = 0$[6]. In the later case $\mathtt{Verify}\big(\mathrm{PP}, Z_i, tag_i, \sigma'\big)$ would simply output $tag_i\big(Z_i, \sigma'\big)$.

Definition 2.4 (Computer Uncrackable v2)**.** *We say that a CAPTCHA puzzle-system* $(\mathtt{Setup}, \mathtt{W}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{Verify})$ *is* computer uncrackable *if for any* PPT *adversary* $\mathcal{A}$ *who has at most m human-work units and any*

[5] In the same way some computers (ASICs) are much faster at evaluating the SHA256 hash function than others. However, we expect this difference to be less extreme for human users.

[6] Indistinguishability obfuscation provides 'best case' obfuscation [31] so it would be highly surprising if an adversary could use tag_i to extract σ_i as this would immediately imply that a host of alternative cryptographic techniques (e.g., one way functions, collision resistance hash functions) fail to hide σ_i. A recent result of Barak et al [4] provides evidence that evasive circuit families (e.g., point functions) can be obfuscated.

$n = poly(\lambda)$

$$\Pr\left[\begin{array}{l} \text{PP} \leftarrow \mathtt{Setup}(1^{\lambda});\ \forall i \in [n]\big(\sigma_i^* \leftarrow \mathtt{W}(\text{PP})\big); \\ \forall i \in [n]\big((Z_i^*, tag_i^*) \leftarrow \mathtt{G}(\text{PP}, \sigma_i^*)\big); \\ S \leftarrow \mathcal{A}^{\mathcal{H}(\cdot)}\left(\text{PP}, (Z_1^*, tag_1^*), \ldots, (Z_n^*, tag_n^*)\right); \\ \forall i \in [n]\big(b_i \leftarrow \max_{\sigma \in S} \mathtt{Verify}(\text{PP}, Z_i^*, tag_i^*, \sigma)\big); \\ \qquad \sum_{i \in [n]} b_i \geq m+1 \end{array}\right] \leq \mathsf{negl}(\lambda)$$

We will require λ to be large enough that a computer cannot reasonably find a solution by brute force. As Von Ahn et al. [57] observed we can always increase λ by composing CAPTCHA puzzles. Of course this will increase the amount of time that it would take to solve a puzzle. Bursztein et al. [17] conducted a large scale experiment on Amazon's Mechanical Turk to evaluate human performance on a variety of different CAPTCHAs. Based on these results we estimate that it is plausible to obtain security parameter $\lambda = 100$ if we define one human work unit to be about two minutes of human effort. For traditional CAPTCHA applications like bot detection this would make the solution impracticable due to the high usability costs. However, for our applications such a delay can be acceptable (e.g., in Bitcoin the parameters are tuned so that a new block is mined every 10 minutes) .

While some spammers have paid human workers to solve CAPTCHAs in bulk [48] we do not consider this an attack on our definition because human effort was involved to find the solution. A HumanCoin miner could pay users to solve CAPTCHAs for him, but he would have to pay the miners a fair wage because they could simply mine HumanCoins on their own. Some CAPTCHA candidates have been broken by PPT algorithms (e.g. [56, 47, 54, 16]). For example, [16] was able to solve reCAPTCHA with accuracy 33.34%. There are solid guidelines about generating CAPTCHAs that are harder for a computer to crack (e.g., [62]). Furthermore, we stress that even apparently "broken" CAPTCHAs may still be useful in our proof of human work context because it is acceptable to use CAPTCHA puzzles that take a long time (e.g., 2 minutes) for a human to solve. By contrast, most deployed CAPTCHAs (e.g., reCAPTCHA) are meant to be solvable in a few seconds. As long as there is some gap between human intelligence and artificial intelligence we can use standard hardness amplification techniques (e.g., parallel repetition) to obtain stronger CAPTCHAs [57].

Ultimately, any candidate CAPTCHA construction must be based on a hard problem in artificial intelligence. Many AI researchers believe that any CAPTCHAs will eventually be broken as AI eventually advances to the point that computer is able to complete any task that a human can complete. AI is indeed advancing. However, we believe that we are still very far away from this point.

2.2 Universal Samplers

In [32], Hofheinz et al introduce the notion of universal samplers. The essential property of a universal sampler scheme is that given the sampler parameters U, and given any program d that generates samples from randomness, it should be possible for any party to use the sampler parameters U and the description of d to obtain induced samples that look like the samples that d would have generated given uniform and independent randomness.

Definition 2.5. *A universal sampler scheme consists of algorithms* $(\mathtt{Setup}, \mathtt{Sample})$ *where*

- $U \leftarrow \mathtt{Setup}(1^{\lambda})$ *is a randomized algorithm which takes as input a security parameter* 1^{λ} *and outputs sampler parameters* U.
- $p_d \leftarrow \mathtt{Sample}(U, d)$ *takes as input sampler parameters* U *and a circuit* d *of size at most* $\ell = \mathrm{poly}(\lambda)$, *and outputs induced samples* p_d.

In our construction in the next section, we will use a slightly extended version of universal sampler scheme which allows an additional input. Note that in the basic version of universal sampler scheme in Definition 2.5 above, the algorithm $\mathtt{Sample}(U, d)$ receives as input a program d which specifies certain distribution. In our application the program d will be fixed ahead of time, and $\mathtt{Sample}$ takes an additional input β where β is an index for specifying randomness for the program to generate a CAPTCHA puzzle Z with tag. Thus, for the slightly extended version of universal sampler scheme with an additional input, we will use the notation $\texttt{Sample}(U,d,\beta)$ instead of $\texttt{Sample}(U,d)$. This allows us to provide alternative and flexible description for a circuit d without changing its functionality. We note that this slightly extended version has been explored in [32], and it is straightforward to extend $\texttt{Sample}$, without requiring a new construction or security analysis.

We next recall the definition of adaptive security for the slightly extended universal samplers with additional inputs. We intend to formulate the notion that guarantees that induced samples are indistinguishable from honestly generated samples to an arbitrary interactive system of adversarial and honest parties. In a universal sampler with additional inputs, the program d is fixed, and when additional input β are provided, the induced samples can be computed $p_\beta \leftarrow \texttt{Sample}(U,d,\beta)$.

We first consider an "ideal world," where a trusted party with a fixed program description d, on input β, simply outputs $d(r_\beta)$ where r_β is independently chosen true randomness, chosen once and for all for each given β. In other words, if F is a truly random function, then the trusted party outputs $d(F(\beta))$. In this way, if any party asks for samples corresponding to a specific value of β, they are all provided with the same honestly generated value.

In the real world, however, all parties would only have access to the trusted sampler parameters. Parties would use the sampler parameters to derive induced samples $d(r_\beta)$ for any specific inputs β. Now r_β is a pseudo random value corresponding to the randomness index β. We will require that for every real-world adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that can provide simulated sampler parameters U to the adversary such that these simulated sampler parameters U actually induce the completely honestly generated samples $d(F(\beta))$ created by the trusted party: in other words, that $\texttt{Sample}(U,d,\beta) = d(F(\beta))$. Note that since honest parties are instructed to simply honestly compute induced samples, this ensures that honest parties in the ideal world would obtain these completely honestly generated samples $d(F(\beta))$.

Definition 2.6. *Consider efficient algorithms* $(\texttt{Setup}, \texttt{Sample})$ *where* $U \leftarrow \texttt{Setup}^{\texttt{RO}}(1^\lambda)$, *$d$ is the fixed program supporting additional input, and* $p_\beta \leftarrow \texttt{Sample}^{\texttt{RO}}(U,d,\beta)$. *We say* $(\texttt{Setup}, \texttt{Sample})$ *is an adaptively-secure universal sampler scheme for a circuit d, if there exist efficient interactive Turing Machines* $\texttt{SimSetup}$, $\texttt{SimRO}$ *such that for every efficient admissible adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}()$ *such that the following two conditions hold:*

$$\Pr[\mathbf{Real}(1^\lambda) = 1] - \Pr[\mathbf{Ideal}(1^\lambda) = 1] = \mathsf{negl}() \textit{ and } \Pr[\mathbf{Ideal}(1^\lambda) = \textit{aborts}] < \mathsf{negl}()$$

where admissible adversaries, the experiments **Real** *and* **Ideal** *and the notion of the* **Ideal** *experiment aborting, are described below*

- *An admissible adversary $\mathcal{A}$ is an efficient interactive Turing Machine that outputs one bit, with the following input/output behavior:*
    - *$\mathcal{A}$ initially takes input security parameter 1^λ and sampler parameters U, as well as the program d.*
    - *$\mathcal{A}$ can send a message* (RO, x) *corresponding to a random oracle query. In response, $\mathcal{A}$ expects to receive the output of the random oracle on input x.*
    - *$\mathcal{A}$ can send a message* $(\texttt{sample}, \beta)$. *The adversary does not expect any response to this message. Instead, upon sending this message, $\mathcal{A}$ is required to honestly compute* $p_\beta = \texttt{Sample}(U,d,\beta)$, *making use of any additional RO queries, and $\mathcal{A}$ appends* (β, p_β) *to an auxiliary tape.*

      Remark. *Intuitively,* $(\texttt{sample}, \beta)$ *messages correspond to an honest party seeking a sample generated by the fixed program d on input β. Recall that $\mathcal{A}$ is meant to internalize the behavior of honest parties.*
- *The experiment* $\mathbf{Real}(1^\lambda)$ *is as follows:*

- *Throughout this experiment, a random oracle* $\mathtt{RO}$ *is implemented by assigning random outputs to each unique query made to* $\mathtt{RO}$*.*
- $U \leftarrow \mathtt{Setup}^{\mathtt{RO}}(1^\lambda)$.
- $\mathcal{A}(1^\lambda, U, d)$ *is executed; when* $\mathcal{A}$ *sends every message of the form* (RO, x)*, it receives the response* $\mathtt{RO}(x)$*.*
- *The output of the experiment is the final output of the execution of* $\mathcal{A}$ *(which is a bit* $b \in \{0,1\}$*).*

– *The experiment* $\mathbf{Ideal}(1^\lambda)$ *is as follows:*

- *Throughout this experiment, a Samples Oracle* $\mathcal{O}$ *is implemented as follows: On input* β*,* $\mathcal{O}$ *outputs* $d(F(\beta))$*, where* F *is a truly random function.*
- $(U, \tau) \leftarrow \mathtt{SimSetup}(1^\lambda)$*. Here,* $\mathtt{SimSetup}$ *can make arbitrary queries to the Samples Oracle* $\mathcal{O}$*.*
- $\mathcal{A}(1^\lambda, U, d)$ *and* $\mathtt{SimRO}(\tau)$ *begin simultaneous execution. Messages for* $\mathcal{A}$ *or* $\mathtt{SimRO}$ *are handled as:*
  1. *Whenever* $\mathcal{A}$ *sends a message of the form* (RO, x)*, this is forwarded to* $\mathtt{SimRO}$*, which produces a response to be sent back to* $\mathcal{A}$*.*
  2. $\mathtt{SimRO}$ *can make any number of queries to the Samples Oracle* $\mathcal{O}$*.*
  3. *In addition, after* $\mathcal{A}$ *sends messages of the form* $(\mathtt{sample}, \beta)$*, the auxiliary tape of* $\mathcal{A}$ *is examined until* $\mathcal{A}$ *adds entries of the form* (β, p_β) *to it. At this point, if* $p_\beta \neq d(F(\beta))$*, the experiment aborts and we say that an "Honest Sample Violation" has occurred. Note that this is the only way that the experiment* **Ideal** *can abort. In this case, if the adversary itself "aborts", we consider this to be an output of zero by the adversary, not an abort of the experiment itself.*
- *The output of the experiment is the final output of the execution of* $\mathcal{A}$ *(which is a bit* $b \in \{0,1\}$*).*

3 Proof of Human-work Puzzles

In this section, we first define the syntax and security for proof of human-work puzzles; then we demonstrate a construction using universal samplers and CAPTCHAs.

3.1 Definitions

In a proof of work (PoW) puzzle, a party (i.e., prover) is allowed to prove to a bunch of verifiers that he completed some amount of computation/work. In general, those parities are machines. A typical PoW puzzle scheme consists of several algorithms: setup algorithm $\mathtt{Setup}()$ for generating the global system parameters and policies, puzzle instance generation algorithm $\mathtt{G}()$, puzzle solution finding algorithm $\mathtt{C}()$, and solution verification algorithm $\mathtt{V}()$. To enable a consensus protocol, the PoW puzzle has to meet the following requirements: (i) it has to be moderately hard to compute (for machines), and no prover can create a proof of work in no time; (ii) it has to be easy to verify (for machines), and all verifiers can efficiently check if a proof is valid; (iii) the difficulty needed in order to solve the proof has to be adjustable in a linear way; and (iv) it has to be possible to ensure that proofs of work cannot be reused multiple times, and the proofs of work should be linked to some public information, e.g., the hash of the block header in a consensus protocol.

Proof of human-work puzzles are very similar to PoW puzzles, except that we intend to have the human in the loop for finding the solution. The key difference is that the prover (problem solver) should not be machine-only. In the above listed requirements, we therefore expect the PoH puzzle to be *moderately hard* *to compute for humans, and infeasible to compute for machines*. On the other hand, as in PoW, we expect the verification to be easy for machines[7]. The syntax is as follows:

Definition 3.1 (Proof of Human-work Puzzle)**.** *A proof of human-work puzzle-system consists of a tuple of algorithms* $(\mathsf{Setup}, \mathsf{G}, \mathsf{C}^{\mathcal{H}}, \mathsf{V})$*, where*

- Setup *is a randomized system setup algorithm that takes as input* 1^λ *(*λ *is the security parameter) and* 1^ω *(*ω *is the difficulty parameter), and outputs a system public parameter* $\mathrm{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^\omega)$*;*
- G *is a randomized puzzle generation algorithm that takes as input the public parameter* PP*, and outputs puzzle* $x \leftarrow \mathsf{G}(\mathrm{PP})$*;*
- $\mathsf{C}^{\mathcal{H}}$ *is a solution finding algorithm (i.e., human-machine solver) that takes as input the public parameter* PP *and a puzzle x, and outputs value* $a \leftarrow \mathsf{C}^{\mathcal{H}(\cdot)}(\mathrm{PP}, x)$ *as the solution to the puzzle x. Here,* $\mathcal{H}(\cdot)$ *denotes the human oracle which takes intermediate human-efficient objects (such as images) as inputs, and returns machine-efficient values as outputs.*
- V *is a deterministic puzzle-solution verification algorithm that takes as input the public parameter* PP *and a puzzle-solution pair* (x, a)*, and outputs bit* $b := \mathsf{V}(\mathrm{PP}, x, a)$ *where* $b = 1$ *if a is a valid solution to the puzzle x, and* $b = 0$ *otherwise.*

Following notation of Miller et al. [46] we will let $\zeta(m, \omega) \doteq 1 - (1 - 2^{-\omega})^m$. Intuitively, $\zeta(m, \omega)$ denotes the probability of finding a valid solution with m queries to the human-oracle.

Definition 3.2 (Honest Human Solvability)**.** *A PoH puzzle system* $(\mathsf{Setup}, \mathsf{G}, \mathsf{C}^{\mathcal{H}}, \mathsf{V})$ *is* honest human solvable *if for every polynomial* $m = m(\lambda)$*, and for any honest human-machine solver* $\mathsf{C}^{\mathcal{H}(\cdot)}$ *who controls m human-work units, it holds that*

$$\Pr\left[\begin{array}{l} \mathrm{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^\omega); \\ x^* \leftarrow \mathsf{G}(\mathrm{PP}); \\ a^* \leftarrow \mathsf{C}^{\mathcal{H}(\cdot)}(\mathrm{PP}, x^*); \\ \quad \mathsf{V}(\mathrm{PP}, x^*, a^*) = 1 \end{array}\right] \geq \zeta(m, \omega) - \mathsf{negl}(\lambda)$$

Definition 3.3 (Adversarial Human Unsolvability)**.** *A PoH puzzle system* $(\mathsf{Setup}, \mathsf{G}, \mathsf{C}^{\mathcal{H}}, \mathsf{V})$ *is* adversarial human unsolvable *if for every polynomial* $m = m(\lambda)$ *and for any human-machine adversary* $\mathcal{B}^{\mathcal{H}(\cdot)}$ *who controls at most m human-work units, it holds that*

$$\Pr\left[\begin{array}{l} \mathrm{PP} \leftarrow \mathsf{Setup}(1^\lambda, 1^\omega); \\ x^* \leftarrow \mathsf{G}(\mathrm{PP}); \\ a^* \leftarrow \mathcal{B}^{\mathcal{H}(\cdot)}(\mathrm{PP}, x^*); \\ \quad \mathsf{V}(\mathrm{PP}, x^*, a^*) = 1 \end{array}\right] \leq \zeta(m+1, \omega) + \mathsf{negl}(\lambda)$$

Remark 3.4. *We remark that the above definition can be strengthened by providing the adversarial* $\mathcal{B}$ *additional access to polynomial number of* (x_i, a_i) *pairs, where* $x_i \leftarrow \mathsf{G}(\mathrm{PP})$ *and* $\mathsf{V}(\mathrm{PP}, x_i, a_i) = 1$*. The definition can be strengthened further by providing the adversarial* $\mathcal{B}$ *multiple puzzle instances* $x_1^*, \ldots, x_k^*$*, and asking* $\mathcal{B}$ *to output a valid* a_j^* *where* $j \in [k]$*. Our construction in next section can achieve these strengthened notions. For simplicity, we focus on the above simplified notion in this paper.*

3.2 Construction

In this subsection, we show how to construct PoH puzzles for cryptocurrency. In Bitcoin each PoW puzzle instance is specified by the public ledger x. A motivated miner (i.e., the PoW prover) will produce a PoW

[7] We remark that, it might also be interesting to consider the variant in which verification is easy for human but not for machine-verifiers.

by repeatedly querying a random oracle $\mathtt{RO}$ (e.g., the SHA256 hash function) to sample uniformly random elements in an attempt to produce a "small" output. More concretely, the miner computes random elements $y_i = \mathtt{RO}(x, s_i)$ for different strings s_i's. If there exist i so that $y_i < T_\omega$, then the corresponding s_i can be viewed as the PoW solution. Given a random oracle $\mathtt{RO} : \{0,1\}^* \to \{0,1\}^n$ we will use the notation $T_\omega \doteq 2^{n-\omega}$. Intuitively, this ensures that $\mathtt{RO}(x, s_i) < T_\omega$ with probability $2^{-\omega}$.

To have human in the loop, we need to first sample CAPTCHA instances for human solvers. Those instances are not in uniform distribution, and it is unclear if we can use a random oracle $\mathtt{RO}$ to generate such instances. We here use a cryptographic tool called "universal sampler" recently developed by Hofheinz et al. [32] to generate such CAPTCHA instances. Universal sampler can be viewed as an extended version of RO, which can generate elements in any efficiently samplable distributions. More concretely, we fix d to be a circuit for computing the CAPTCHA generation function $\textsc{capt}.\mathtt{G}$. Thus, $d(r)$ generates a CAPTCHA puzzle Z_r and a tag tag_r from randomness r. Now, the miner begins by computing $(Z_i, tag_i) = \mathtt{Sample}(U, d, \beta = (x, s_i))$; then the miner solves Z_i via human effort to get the corresponding CAPTCHA solution σ_i; at this moment, we can adapt the strategy in the original PoW by computing $y_i = \mathtt{RO}(x, s_i, \sigma_i)$ and if $y_i < T_\omega$ and if the CAPTCHA solution σ_i is correct, then the corresponding pair (s_i, σ_i) can be viewed as the PoH solution. We can verify that the solution is correct by re-sampling $(Z_i, tag_i) \leftarrow \mathtt{Sample}(U, d, \beta = (x, s_i))$ and checking that $\mathtt{Verify}(Z_i, tag_i, \sigma_i) = 1$ and that $\mathtt{RO}(x, s_i, \sigma_i) < T_\omega$.

Construction Details In our proof of human-work puzzle construction, we use a universal sampler scheme $\textsc{uni} = \textsc{uni}.\{\mathtt{Setup}, \mathtt{Sample}\}$, a CAPTCHA scheme $\textsc{capt} = \textsc{capt}.\{\mathtt{Setup}, \mathtt{W}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{Verify}\}$, and a hash function $\mathbf{G}$. We will treat $\mathbf{G}$ as a random oracle in our analysis. The constructed PoH puzzle scheme consists of algorithms $\textsc{poh}.\{\mathtt{Setup}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{V}\}$. Note that $\mathcal{H}$ denotes a human oracle.

- The setup algorithm $\textsc{pp} \leftarrow \textsc{poh}.\mathtt{Setup}(1^\lambda, 1^\omega)$: Compute $\tilde{\textsc{pp}} \leftarrow \textsc{capt}.\mathtt{Setup}(1^\lambda)$; Compute $U \leftarrow \textsc{uni}.\mathtt{Setup}(1^\lambda)$; Define a program d as follows: On input randomness $r = (r_1, r_2)$, compute $\sigma := \textsc{capt}.\mathtt{W}(\tilde{\textsc{pp}}; r_1)$, $(Z, tag) := \textsc{capt}.\mathtt{G}(\tilde{\textsc{pp}}, \sigma; r_2)$, and output (Z, tag). Set $\textsc{pp} := \big(U, d, \tilde{\textsc{pp}}, T = T_\omega, \textsc{param}\big)$ where $\textsc{param}$ denotes the instructions of using the system.
- The puzzle generation algorithm $x \leftarrow \textsc{poh}.\mathtt{G}(\textsc{pp})$: Parse $\textsc{pp}$ into $(U, d, \tilde{\textsc{pp}}, T, \textsc{param})$; Based on the description of $\textsc{param}$, sample x.
- The solution function $a \leftarrow \textsc{poh}.\mathtt{C}^{\mathcal{H}}(\textsc{pp}, x)$: Upon receiving puzzle instance x, parse $\textsc{pp}$ into $(U, d, \tilde{\textsc{pp}}, T, \textsc{param})$; Randomly choose $s \leftarrow \{0,1\}^\lambda$; Compute CAPTCHA puzzle instance $(Z, tag) \leftarrow \textsc{uni}.\mathtt{Sample}(U, d, \beta = (x, s))$; Use the human oracle $\mathcal{H}$ to find a solution to CAPTCHA puzzle instance Z, i.e., $\sigma \leftarrow \textsc{capt}.\mathtt{S}^{\mathcal{H}}(\tilde{\textsc{pp}}, Z)$; If $\mathbf{G}(x, s, \sigma) < T$, then set $a := (s, \sigma)$. Otherwise set $a := \perp$.
- The puzzle verification algorithm $b := \textsc{poh}.\mathtt{V}(\textsc{pp}, x, a)$: Parse a into (s, σ); Parse $\textsc{pp}$ into $(U, d, \tilde{\textsc{pp}}, T, \textsc{param})$; Compute $(Z, tag) \leftarrow \textsc{uni}.\mathtt{Sample}(U, d, \beta = (x, s))$; If $\textsc{capt}.\mathtt{Verify}(\tilde{\textsc{pp}}, Z, tag, \sigma) = 1$ and $\mathbf{G}(x, s, \sigma) < T$, then set $b := 1$. Otherwise set $b := 0$.

It is easy to verify that the PoH scheme is honest human solvable if the underlying universal sampler is correct and the CAPTCHA scheme is honest-human solvable. Next we state a theorem for the security of our PoH scheme.

Theorem 3.5. *If* $\textsc{uni}$ *is an adaptively secure universal sampler, and* $\textsc{capt}$ *is a computer uncrackable CAPTCHA (definition 2.4), then the above proof of human-work scheme* $\textsc{poh}$ *is adversarial human unsolvable in the random oracle model.*

Proof idea. The security of our PoH relies on the security of underlying building blocks, the universal sampler scheme $\textsc{uni}$, and the CAPTCHA scheme $\textsc{capt}$. We start from the real security game. Based on the security of the universal sampler scheme $\textsc{uni}$, we can modify the real security game into a hybrid world where CAPTCHA puzzle instances are generated independently and based on uniform randomness. Then we can use the security of $\textsc{capt}$ to argue about the security of PoH. That is, we can construct a $\textsc{capt}$ attacker

$\mathcal{A}_{\text{CAPT}}$ based on an PoH attacker $\mathcal{A}_{\text{POH}}$. The CAPT attacker $\mathcal{A}_{\text{CAPT}}$ can simulate an internal copy of $\mathcal{A}_{\text{POH}}$, and embed his challenge into a simulated hybrid for $\mathcal{A}_{\text{POH}}$. If $\mathcal{A}_{\text{POH}}$ wins with more than specified probability (i.e., $\zeta(m+1,\omega)$) plus non-negligible probability, then $\mathcal{A}_{\text{CAPT}}$ can also win the computer-unbreakable game with non-negligible probability. □

Proof. Proof of Theorem 3.5 We prove the main theorem via a sequence of hybrids.

Hybrid$_0$: This is the real experiment. The challenger computes $\text{PP} \leftarrow \text{POH}.\mathtt{Setup}(1^\lambda, 1^\omega)$ as in the construction where $\text{PP} := \big(U, d, \tilde{\text{PP}}, T, \text{PARAM}\big)$, $U \leftarrow \text{UNI}.\mathtt{Setup}(1^\lambda)$. Then the challenger computes $x^* \leftarrow \mathtt{G}(\text{PP})$, and provides (PP, x^*) to the adversary. The adversary may send a message (RO, x) and the challenger will respond with $\mathtt{RO}(x)$. The game ends when the adversary outputs a pair (x, a), and the adversary wins if and only if the following hold: $x = x^*$, $a = (s, \sigma)$ where $\mathbf{G}(x, s, \sigma) < T$ and $\text{CAPT}.\mathtt{Verify}\big(\tilde{\text{PP}}, Z, tag, \sigma\big) = 1$ where $(Z, tag) \leftarrow \text{UNI}.\mathtt{Sample}(U, d, \beta = (x, s))$. Note that, for any string s_i the adversary can sample $(Z_i, tag_i) \leftarrow \text{UNI}.\mathtt{Sample}(U, d, \beta_i = (x^*, s_i))$ by querying for $\mathtt{RO}(\beta_i)$. For each Z_i, the adversary could spend a human unit to obtain the underlying puzzle solution σ_i; more concretely, the adversary uses the solution function in the CAPTCHA system to compute the solution to CAPTCHA puzzle Z_i, i.e., $\sigma_i \leftarrow \text{CAPT}.\mathsf{C}^{\mathcal{H}}(\tilde{\text{PP}}, Z_i)$. Then the adversary could check that $\text{CAPT}.\mathtt{Verify}\big(\tilde{\text{PP}}, Z_i, tag_i, \sigma_i\big) = 1$ to verify that the solution σ_i is correct. Now if it holds that $\mathbf{G}(x^*, s_i, \sigma_i) < T$, then the adversary can win by setting $a^* := (s_i, \sigma_i)$ and outputting (x^*, a^*).

Hybrid$_1$: This hybrid corresponds to the **Ideal** experiment from Definition 2.6. In this experiment, the challenger will use the samples oracle $\mathcal{O}$, which on input β returns $d(F(\beta))$ where F is a truly random function. This hybrid is the same as **Hybrid**$_0$ except the following: (1) the challenger generates $\text{PP} \leftarrow \text{POH}.\mathtt{Setup}(1^\lambda, 1^\omega)$, $\text{PP} := \big(U, d, \tilde{\text{PP}}, T, \text{PARAM}\big)$, and now U is generated via the simulated setup algorithm, i.e., $U \leftarrow \text{UNI}.\mathtt{SimSetup}(1^\lambda)$. (2) Whenever the adversary sends a message of the form (RO, x), the challenger uses $\mathtt{SimRO}(x)$ to produce a response.

Claim 3.6. Hybrid$_0$ *and* **Hybrid**$_1$ *are computationally indistinguishable.*

Proof. Note that in **Hybrid**$_0$, the sampler parameters U are generated exactly as in the Real-experiment from Definition 2.6, while in **Hybrid**$_1$, the sampler parameters are generated as in the Ideal-experiment. Thus, by the security of universal sampler scheme, we have
$|\Pr[\mathbf{Hybrid}_0 = 1] - \Pr[\mathbf{Hybrid}_1 = 1]| \le \mathsf{negl}(\lambda)$. □

Claim 3.7. *The adversary wins in* **Hybrid**$_1$ *with at most probability* $\zeta(m+1,\omega) + \mathsf{negl}(\lambda)$.

Proof. As mentioned in the proof idea above, here we need to construct a CAPT attacker $\mathcal{A}_{\text{CAPT}}$ by using a POH attacker $\mathcal{A}_{\text{POH}}$.

$\mathcal{A}_{\text{CAPT}}$ simulates a copy of the POH attacker $\mathcal{A}_{\text{POH}}$ and also simulates **Hybrid**$_1$ for the POH attacker. For simplicity, we assume without loss of generality that (1) $\mathcal{A}_{\text{POH}}$ never makes the same query twice to samples oracle $\mathcal{O}$; (2) if $\mathcal{A}_{\text{POH}}$ outputs (s, σ) for some x, then it had previously queried $\mathcal{O}$ on (x, s) with response (Z, tag); (3) if $\mathcal{A}_{\text{POH}}$ outputs (s, σ) for x, then it had previously queried $\mathbf{G}(x, s, \sigma)$.

Let $q = q(\lambda)$ be a (polynomial) upper-bound on the number of queries made by $\mathcal{A}_{\text{POH}}$ to $\mathcal{O}$. Let $\big((Z_1, tag_1), \ldots, (Z_q, tag_q)\big)$ denote the corresponding CAPTCHA puzzle-tag pairs. Consider the following algorithm $\mathcal{A}_{\text{CAPT}}$:

- Initialize $S \leftarrow \emptyset$. S is a set of candidate CAPTCHA solutions.
- When $\mathcal{A}_{\text{POH}}$ sends i-th message $(\mathtt{sample}, \beta_i = (x_i, s_i))$, return (Z_i, tag_i) as the response.
- When $\mathcal{A}_{\text{POH}}$ requests a solution on puzzle instance Z, forward the query to the human oracle to obtain $\sigma \leftarrow \text{CAPT}.\mathsf{S}^{\mathcal{H}}(\tilde{\text{PP}}, Z)$. Note: The adversary $\mathcal{A}_{\text{POH}}$ may or may not choose to query the human oracle on one of the puzzles Z_i generated by the samples oracle $\mathcal{O}$. However, even if the adversary tries to modify

the puzzles to obtain solutions σ_i without querying the human oracle on Z_i, the adversary will still need to expose σ_i to query the random oracle $\mathbf{G}$.

- When $\mathcal{A}_{\text{POH}}$ requests the value $\mathbf{G}(x,s,\sigma)$, add σ to our candidate solution set $S \leftarrow S \cup \{\sigma\}$.
- When $\mathcal{A}_{\text{POH}}$ terminates with output a, $\mathcal{A}_{\text{CAPT}}$ outputs S.

We note that the probability that our algorithm $\mathcal{A}_{\text{CAPT}}$ outputs S containing $m+1$ valid solutions σ must be negligible $\mathsf{negl}(\lambda)$ by definition of a computer uncrackable CAPTCHA because the CAPTCHA puzzles are generated independently at random in $\mathbf{Hybrid}_1$ (the ideal world). Conditioning on the event that S contains at most m valid CAPTCHA solutions the probability that the adversary finds a valid PoH solution in any of these m solutions is at most $\zeta(m,\omega)$. Thus, the adversary's success rate is at most $\mathsf{negl}(\lambda)+\zeta(m,\omega)$ in this case. □

Based on the claims above, we can prove the POH scheme is secure as in Definition 3.3. □

4 Application 1: HumanCoin

In this section we outline how a new cryptocurrency called *HumanCoin* could be built using Proofs of Human-work. At a high level HumanCoin closely follows the Bitcoin protocol, except that we use PoH puzzles to extend the blockchain instead of PoW puzzles. We will not attempt to describe HumanCoin in complete detail. Instead we will focus on the key modifications that would need to be made to an existing cryptocurrency like Bitcoin to use Proof of Human work puzzles. In our discussion we will use lowercase bitcoin (resp. humancoin) to denote the base unit of currency in the Bitcoin (resp. HumanCoin) protocol.

4.1 Bitcoin Background

We begin by highlighting several of the key features of Bitcoin. Our overview follows the systemization of knowledge paper by Bonneau et al. [13]. However, our discussion of Bitcoin is overly simplified and this choice is intentional. For example, we will completely ignore the use of Merkle Trees [45] in Bitcoin to compress the blockchain even though it is quite useful in practice. We make this choice so that we can focus on the key differences of HumanCoin (the use of Merkle Trees [45] in HumanCoin and Bitcoin would be identical). We do include additional discussion of Bitcoin in the appendix, but even this discussion is not intended to be complete. We refer interested readers to the excellent lectures by Narayanan et al. [50] for more details about Bitcoin or the original paper published under the pseudonym Nakamoto [49].

Blockchain. In Bitcoin all transactions (e.g., "Alice sends Bob 50 bitcoins") are published on a public ledger. This public ledger is stored on a cryptographic data structure called a blockchain $b = B_0,\ldots,B_t$. A blockchain b is valid if and only if all of the blocks B_i ($i \leq t$) are valid and an individual block $B_i = (tx_i, s_i, h_{i-1})$ is valid if and only if three key conditions are satisfied. First, all of the transactions recorded in the transcript tx_i must be valid (e.g., each transaction is signed by the sender and the spender has sufficient funds). Second, the block B_i must contain the cryptographic hash $h_{i-1} = hash(B_{i-1})$ of the previous block B_{i-1}[8]. Finally, the block B_i should contain a nonce s_i which ensures that cryptographic hash $hash(B_i)$ begins with at least ω leading zeros, where ω is a hardness parameter that we will discuss later. Finding such a nonce s constitutes a proof of work in the Hashcash [3] puzzle system. The first property ensures that users cannot spend money they don't have and that they cannot spend someone else's money. The second property ensures that it is impossible to tamper with blocks B_i in the middle of the blockchain without creating an entirely new blockchain $b' = B_0,\ldots,B_{i-1},B'_i,B'_{i+1},\ldots,B'_t$. Finally, the third property ensures that it is moderately difficult to add new blocks to a blockchain. To incentivize miners to help validate transactions (i.e. extend

[8] Bitcoin uses the cryptographic hash function $hash$ =SHA256. The function $hash$ is typically treated as a random oracle in security analysis of Bitcoin.

the blockchain by finding a valid nonce s) the miner is allowed to add a special transaction (e.g., "I create 25 new bitcoins and give them to myself") to the new block as a reward .

Distributed Consensus Protocol. Bitcoin's distributed consensus protocol is simple, yet elegant. An agent should accept a transaction if and only if it is recorded on a block B_i of a valid blockchain $b = B_0, \ldots, B_t$ and b is the longest valid that the agent has seen and $i \leq t-6$. Unless a miner controls at least 25% of the hash power in the network the rational mining strategy is always to extend the longest blockchain because nobody will accept the Bitcoins they try to mine in a shorter blockchain (e.g., the special transaction in which a miner claims 25 bitcoins' would only be recorded on a shorter blockchain which nobody accepts) [27]. Assuming that the network has high synchronicity [29] and that a malicious user controls at most 49% of the computational mining power he will never be able to tamper with any of the transactions in a block B_i from the middle of the blockchain because he would need to eventually produce a new blockchain $b' = B_0, \ldots,$ $B_{i-1}, B'_i, B'_{i+1}, \ldots, B'_t$ that is at least as long as the true blockchain b and he will fail to accomplish this goal with high probability [49].

4.2 HumanCoin

Similar to Bitcoin all HumanCoin transactions (e.g., "Alice sends Bob 50 humancoins") are recorded inside a blockchain $b = B_0, \ldots, B_t$, where each block $B_i = (tx_i, a_i, h_{i-1})$ contains three components: a list of transactions tx_i, a hash $h_{i-1} = hash(B_{i-1})$ of the previous block, and a Proof of Human-work which is encoded by a_i. As before all of the transactions in tx_i must be valid and the block must contain the hash $h_{i-1} = hash(B_{i-1})$ of the previous block. We additionally require that the PoH verifier accepts the Proof of Human-Work solution a_i. More formally, suppose that we are given a PoH puzzle system $(\mathtt{Setup}, \mathtt{G}, \mathtt{C}^{\mathcal{H}}, \mathtt{V})$ and that we have already run $\mathtt{Setup}\left(1^{\lambda}, 1^{\omega}\right)$ to obtain public parameters PP which are available to every miner. A valid block B_i must contain a value a_i such that the public verifier $\mathtt{V}(\mathrm{PP}, x_i, a_i)$ outputs 1, where $x_i = \mathtt{G}(\mathrm{PP}; r = hash(tx_i, h_{i-1}))$. Given a valid blockchain $b = B_0, \ldots, B_t$ a miner can earn HumanCoins by finding a valid block $B_{t+1} = (tx_{t+1}, a_{t+1}, x_{t+1}, h_t)$ extending b. To find such a block the human-computer miner would first set $r = hash(tx_{t+1}, h_t)$ and then sample $x \leftarrow \mathtt{G}(\mathrm{PP}; r)$. Finally, the human-computer miner can run $\mathtt{C}^{\mathcal{H}}(\mathrm{PP}, x)$ to obtain a potential solution a. If $a = \perp$ then the miner will need to try again. Otherwise, the miner has found a valid proof of human-work and he can produce a valid new block $B_{t+1} = (tx_{t+1}, a, h_t)$ by adding inserting the PoH solution a into the block B_{t+1}. As before the miner is allowed to insert a special transaction into the new block (e.g., "I create 25 humancoins and give them to myself") as a reward for extending the blockchain.

Parameter Selection. In Bitcoin ω is a public parameter is tuned to ensure that, on average, miners will add one new block to the blockchain every 10 minutes [50] — on average we need 2^{ω} hash evaluations to create one new block. The Bitcoin protocol would most likely work just fine with a shorter delay (e.g., 5 minutes) or a slightly longer delay (e.g., 20 minutes) between consecutive blocks — there is nothing magical about the specific target value of 10 minutes. However, it is clear that there needs to be some delay to promote stability. If multiple miners find a new block at the same time then we could end up with competing blockchains resulting in temporary confusion. Note that if the value of ω remains fixed then the average time to create one new block would begin to decrease as more miners join Bitcoin, or as existing miners upgrade their computational resources. Thus, the value of ω must be adjusted periodically. In Bitcoin the value of ω is adjusted every $2,016$ blocks, which works out to two weeks on average (2 weeks = 2016×10 minutes), using the formula $\omega = \omega_{old} - \log\left(\frac{t_{elapsed}}{2016 \times 10 \text{ min}}\right)$, where $t_{elapsed}$ denotes the time span that it actually took to generate the last $2,016$ blocks [50] .

In HumanCoin we adjust ω in exactly the same way. Note that the PoH hardness parameter $T_{\omega} = 2^{n-\omega}$ in our PoH construction is a public parameter PP and can easily be modified as it is not embeded into any of the obfuscated programs. In HumanCoin we will need to select an initial value of ω that is *much* smaller than in Bitcoin if we want ensure that new block are discovered every 10 minutes. This is because computers can evaluate a hash function *hash* much faster than a human can solve a long CAPTCHA puzzle. However, we could still use the same basic formula to tune the hardness parameter ω of our proof of work puzzles in the event that many miners join/leave.

5 Application 2: Password Protection

An adversary who breaches an authentication server is able to mount an automated brute-force attack by comparing the cryptographic hash of each user's password with the cryptographic hashes of likely password guesses. These offline attacks have become increasingly prevalent and dangerous as password cracking resources has improved. In particular, the cost of computing a hash function H like SHA256 or MD5 on an Application Specific Integrated Circuit (ASIC) is orders of magnitude smaller than the cost of computing H on traditional hardware [21, 50]. Similarly, data from previous breaches allow adversaries to improve their guessing strategies. Recent security breaches (e.g., Ashley Madison, LastPass, RockYou, LinkedIn and eBay to name a few [9]), which have affected millions of users, highlight the importance of this problem.

Canneti et al. [18] had a clever idea to deter an offline attacker that they called Human Only Solvable Puzzles. They proposed filling a hard drive with a dataset of unsolved CAPTCHA puzzles. When a user authenticates he will be challenged with a pseudorandom CAPTCHA puzzle from the dataset, and the server will append the solution to the user's password before computing the hash value. The choice of the pseudorandom CAPTCHA puzzle becomes deterministic once the user's password and username are fixed. Thus, if the user types in the same password he will receive the exact same CAPTCHA puzzle as a challenge. If the underlying CAPTCHA system is human usable, then the user will always be able to authenticate successfully provided that he can remember his password. If an offline advesary wants to verify a password guess he will need to find and solve the corresponding CAPTCHA puzzle. The key point is that each time the adversary tries a new guess he will need to solve a different CAPTCHA challenge.

Unfortunately, the Human Only Solvable Puzzles solution of [18] has one critical drawback. There are a finite number of CAPTCHAs on the hard drive, and the defense will break down once the adversary manages to solve all (or most) of them. Blocki et al. [9] estimated that it would cost about $\$10^6$ to solve all of the CAPTCHAs on an 8 TB hard drive. While this is certainly an expensive start-up cost it may not be sufficient to deter the adversary because these costs would amortize over all user accounts. Many password breaches affect millions of users, and each cracked password has significant value on the black market (e.g., \$4–\$30). Blocki et al. [9] introduced their own scheme called GOTCHA based on inkblot images, but their protocol had higher usability costs and was based on newer untested AI assumptions.

In this section we introduce a provably secure password authentication scheme in the Random Oracle model using CAPTCHAs and program obfuscation. Unlike Blocki et al. [9] our solution can be based on standard CAPTCHA assumptions. Unlike Canneti et al. [18] our solution is not vulnerable to pre-computation attacks[10].

5.1 Password Authentication Scheme

We first formalize the notion of a password authentication scheme. Definition 5.1 formalizes the account creation and authentication algorithms from the perspective of an authentication server. We note that the server is allowed to interact with the human user $\mathcal{H}$ during the account creation and authentication protocols.

Definition 5.1. *A password authentication scheme consists of a tuple of algorithms* $(\mathtt{Setup}, \mathtt{CreateAccount}^{\mathcal{H}}, \mathtt{Authenticate}^{\mathcal{H}})$ *and a random oracle* $\mathbf{G}$*, where*

[9] See http://www.privacyrights.org/data-breach/ (Retrieved 9/1/2015).

[10] Of course the main downside to our approach is the dependence on indistinguishability obfuscation, which does not have practical solutions at this time.

- $\mathtt{Setup}$ *is a randomized system setup algorithm that takes as input* 1^λ *(λ is the security parameter) and outputs a system public parameter* $\mathtt{PP} \leftarrow \mathtt{Setup}(1^\lambda)$*;*
- $\mathtt{CreateAccount}^{\mathcal{H}}$ *is an account creation algorithm that takes as input the public parameter* $\mathtt{PP}$*, a username u and a password pwd and outputs a tuple* (h,s)*. Here, s is typically a random bit string (salt) and h is a hash value produced by the random oracle. We note that* $\mathtt{CreateAccount}^{\mathcal{H}}$ *is a human-machine algorithm and thus the hash value h may include the solution to CAPTCHAs that the human solves as well as the password pwd and salt s;*
- $\mathtt{Authenticate}^{\mathcal{H}}$ *is the algorithm that is invoked when a user wants to authenticate. The algorithm takes as input the public parameter* $\mathtt{PP}$*, a username u, a password pwd, a hash h and a salt value s and outputs a bit* $b \in \{0,1\}$ *indicating whether or not the authentication attempt was successful. We note that* $\mathtt{Authenticate}^{\mathcal{H}}$ *is a human-machine algorithm and thus the human* $\mathcal{H}$ *may be asked to solve CAPTCHAs as part of the authentication procedure.*

Our next definition says what it means for a password authentication scheme to be costly to crack. The game mimics an offline advesary who has breached the authentication server and stolen the record (u,h,s) indicating that user u has an account with salt value s and the salted hash of the user's password needs to match h. In our definition we let $\mathcal{P}$ denote a distribution over the passwords $\{pwd_1,\ldots,pwd_n\}$ that the user might select and let $p_i = \Pr_{\mathcal{P}}[pwd_i]$ denote the probability that the user selects password pwd_i. We assume that p_i and pwd_i are known to the adversary for all i and for convenience we assume that the passwords are ordered such that $p_1 \geq p_2 \geq \ldots \geq p_n$. Informally, our definition states that an adversary with B units of human-work will succeed in cracking the user's password with probability at most $p_1 + \ldots + p_B + \mathsf{negl}(\lambda)$.

Definition 5.2 (Costly to Crack)**.** *We say that a password authentication scheme* $\{\mathtt{Setup}, \mathtt{CreateAccount}^{\mathcal{H}}, \mathtt{Authenticate}^{\mathcal{H}}, \mathbf{G}\}$ *is costly to crack if for any* PPT *adversary* $\mathcal{A}^{\mathcal{H}(\cdot)}$ *with B human-work units an every user u it holds that*

$$\Pr\left[\begin{array}{l} \mathtt{PP} \leftarrow \mathtt{Setup}(1^\lambda);\ pwd \leftarrow \mathcal{P} \\ (h,s) \leftarrow \mathtt{CreateAccount}^{\mathcal{H}}\big(\mathtt{PP},u,pwd\big); \\ x \leftarrow \mathcal{A}^{\mathcal{H}(\cdot)}(\mathtt{PP},h,s); \\ \qquad \mathbf{G}(x) = h \end{array}\right] \leq p_1 + \ldots + p_B + \mathsf{negl}(\lambda)$$

We remark that we do not require adversary's success probability to be negligibly small. Indeed, if the user selects passwords from a distribution with low entropy (and many users do [12]) then the adversary may have a good success rate. Indeed, this problem is unavoidable as long a users are allowed to select low-entropy passwords[11]. We do not focus on helping users to select strong passwords [14, 10]. While this is an important direction of research our work addresses an orthogonal issue. Our goal is to provide the best possible protection for the passwords that users actually select.

The next definition quantifies human usability. Informally, the password authentication scheme is usable if an honest human user will always be able to authenticate if he remembers his password. We stress that our definition does not say anything about how easy it will be to remember the password. While this is certainly an important consideration it is orthogonal to our work. We are not focused on how to get users to choose stronger passwords, but rather how to more effectively protect the passwords that users actually choose. Our definition merely says that an honest user won't be locked out of his account as long as he remembers his password (e.g., because he cannot solve the CAPTCHAs).

Definition 5.3 (Human Usable)**.** *We say that a password authentication scheme* $\{\mathtt{Setup}, \mathtt{CreateAccount}^{\mathcal{H}}, \mathtt{Authenticate}^{\mathcal{H}}, \mathbf{G}\}$ *is human usable if for every human user* $\mathcal{H}$ *who controls* 1 *human-work unit during*

[11] In addition to their high usability costs [28], policies aimed at forcing users to chose stronger passwords (e.g., requiring numbers and capital letters) can have the opposite affect on password strength [37, 11].

authentication and 1 *human work unit during account creation, it holds that*

$$\Pr\left[\begin{array}{c} \mathrm{PP} \leftarrow \mathtt{Setup}(1^\lambda);\ pwd \leftarrow \mathcal{P} \\ (h,s) \leftarrow \mathtt{CreateAccount}^{\mathcal{H}}\big(\mathrm{PP},u,pwd\big); \\ \mathtt{Authenticate}^{\mathcal{H}}\big(\mathrm{PP},u,pwd,h,s\big) = 1 \end{array}\right] \geq 1-\mathsf{negl}(\lambda)$$

5.2 Construction

Construction Details In our construction we use a universal sampler scheme $\mathrm{UNI} = \mathrm{UNI}.\{\mathtt{Setup},\mathtt{Sample}\}$, a CAPTCHA scheme $\mathrm{CAPT} = \mathrm{CAPT}.\{\mathtt{Setup},\mathtt{W},\mathtt{G},\mathtt{C}^{\mathcal{H}},\mathtt{Verify}\}$, and a hash function **G**. We will treat **G** as a random oracle in our analysis. The constructed password authentication scheme consists of algorithms $\mathtt{Password}.\{\mathtt{Setup},\mathtt{CreateAccount}^{\mathcal{H}},\mathtt{Authenticate}\}$. Note that $\mathcal{H}$ denotes a human oracle.

- The setup algorithm $\mathrm{PP} \leftarrow \mathtt{Password.Setup}(1^\lambda,1^\omega)$:
  Compute $\tilde{\mathrm{PP}} \leftarrow \mathrm{CAPT}.\mathtt{Setup}(1^\lambda)$; Compute $U \leftarrow \mathrm{UNI}.\mathtt{Setup}(1^\lambda)$; Define a program d as follows: On input randomness $r=(r_1,r_2)$, compute $\sigma := \mathrm{CAPT}.\mathtt{W}(\tilde{\mathrm{PP}};r_1)$, $(Z,tag) := \mathrm{CAPT}.\mathtt{G}(\tilde{\mathrm{PP}},\ \sigma;r_2)$, and output Z. Set $\mathrm{PP} := \big(U,d,\tilde{\mathrm{PP}},\mathrm{PARAM}\big)$ where PARAM denotes the instructions of using the system.
- The account creation algorithm $(h,s) \leftarrow \mathtt{Password.CreateAccount}^{\mathcal{H}}(\mathrm{PP},u,pwd)$:
  Parse PP into $(U,d,\tilde{\mathrm{PP}},\mathrm{PARAM})$; randomly choose $s \leftarrow \{0,1\}^\lambda$. Set $\beta = (u,pwd,s)$ and compute CAPTCHA puzzle instance $Z \leftarrow \mathrm{UNI}.\mathtt{Sample}(U,d,\beta=(x,s))$; Use the human oracle $\mathcal{H}$ to find a solution to CAPTCHA puzzle instance Z, i.e., $\sigma \leftarrow \mathrm{CAPT}.\mathtt{S}^{\mathcal{H}}(\tilde{\mathrm{PP}},Z)$; Compute $h \leftarrow \mathbf{G}(pwd|\sigma|s)$ and output (h,s).
- The authentication algorithm $b \leftarrow \mathtt{Password.Authenticate}^{\mathcal{H}}(\mathrm{PP},u,pwd,h,s)$:
  Parse PP into $(U,d,\tilde{\mathrm{PP}},\mathrm{PARAM})$, set $\beta = (u,pwd,s)$ and compute CAPTCHA puzzle instance $Z_\beta \leftarrow \mathrm{UNI}.\mathtt{Sample}(U,d,\beta)$; Use the human oracle $\mathcal{H}$ to find a solution to CAPTCHA puzzle instance Z, i.e., $\sigma \leftarrow \mathrm{CAPT}.\mathtt{S}^{\mathcal{H}}(\tilde{\mathrm{PP}},Z)$; Compute $h' \leftarrow \mathbf{G}(pwd|\sigma|s)$. If $h'=h$ then output $b=1$; otherwise output 0.

It is easy to verify that Password is human usable if the underlying CAPTCHA scheme CAPT is honest human solvable. Theorem 5.4, our main technical result in this section, states that the above password authentication scheme is also costly to crack. We stress that we only need to assume that the underling CAPTCHA scheme is computer uncrackable in the more traditional sense of Definition 2.3 (e.g., the adversary is only given the puzzles $Z_1,\ldots,Z_n$ and not the associated verification tags).

Theorem 5.4. *If* UNI *is an adaptively secure universal sampler, and* CAPT *is a computer uncrackable CAPTCHA (Definition 2.3), then the above password authentication scheme* $\mathtt{Password}.\{\mathtt{Setup},\mathtt{CreateAccount}^{\mathcal{H}},\mathtt{Authenticate}^{\mathcal{H}}\}$ *is costly to crack (Definition 5.2).*

Proof idea. At a high level we show that we can construct an adversary that breaks CAPTCHAs (under Definition 2.3) from an adversary that breaks the password authentication scheme. To do this we embed challenge CAPTCHAs $Z_1,\ldots,Z_n$ inside the UniversalSampler UNI (we can do this by the security of the Universal Sampler scheme). Intuitively, in order to check that a password guess pwd_i is correct will need to query the random oracle **G** with the value $\beta_i = (pwd_i|\sigma_i|u)$, where σ_i is the correct solution to CAPTCHA Z_i. If the adversary queries **G** with $B+1$ unique solutions then we can win the CAPTCHA challenge (Definition 2.3) by simply outputting these $B+1$ solutions. If the adversary queries **G** with at most B unique solutions then we can show that his success rate is at most $p_1+\ldots+p_B+\mathsf{negl}(\lambda)$. □

Proof. Suppose that the adversary succeeds with probability $p_1+\ldots+p_B+1/(\mathrm{poly}(\lambda))$. Then we can construct an adversary that breaks CAPTCHAs (under Definition 2.3). Let $Z_1,\ldots,Z_n$ denote randomly generated CAPTCHAs (e.g., $(Z_i,tag_i) \leftarrow \mathrm{CAPT}.\mathtt{G}(\tilde{\mathrm{PP}},\ \sigma_i)$ for unknown $\sigma_i \leftarrow \mathrm{CAPT}.\mathtt{W}(\tilde{\mathrm{PP}})$ and tag_i). We prove security by a sequence of two hybrids $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$.

$\mathbf{Hybrid}_0$ is the real experiment. The challenger computes $\textsc{pp} \leftarrow \texttt{Password.Setup}(1^\lambda)$. The user u selects a password $pwd \leftarrow \mathcal{P}$ and then runs the account creation algorithm to obtain $(h,s) \leftarrow \texttt{Password.CreateAccount}^{\mathcal{H}}(\textsc{pp},u,pwd)$ and provides $(\textsc{pp},h,s,u)$ to the adversary. At any time the adversary may send a message (RO,x) and the challenger will respond with $\texttt{RO}(x)$, where $\texttt{RO}$ is the random oracle used in the universal sample $\textsc{uni}$. At any time the adversary may also send a message $(\mathbf{G},x)$ and the challenger will respond with $\mathbf{G}(x)$. The game ends when the adversary outputs a value x, the adversary wins if and only if $\mathbf{G}(x) = h$. Note that, for any string β_i the adversary can sample $Z_i \leftarrow \textsc{uni}.\texttt{Sample}(U,d,\beta_i)$ by querying for $\texttt{RO}(\beta_i)$.

$\mathbf{Hybrid}_1$: This hybrid corresponds to the **Ideal** experiment from Definition 2.6. In this experiment, the challenger will use the samples oracle $\mathcal{O}$, which on input β_i returns $d(F(\beta_i))$ where F is a truly random function. This hybrid is the same as $\mathbf{Hybrid}_0$ except the following: (1) the challenger generates $\textsc{pp} \leftarrow \textsc{poh}.\texttt{Setup}(1^\lambda, 1^\omega)$, $\textsc{pp} := \left(U, d, \tilde{\textsc{pp}}, T, \textsc{param}\right)$, and now U is generated via the simulated setup algorithm, i.e., $U \leftarrow \textsc{uni}.\texttt{SimSetup}(1^\lambda)$. (2) Whenever the adversary sends a message of the form (RO,x), the challenger uses $\texttt{SimRO}(x)$ to produce a response.

$\mathbf{Hybrid}_2$: This hybrid is the same as $\mathbf{Hybrid}_0$ except that we fix the responses of $\mathcal{O}$ to be our challenge CAPTCHAs: on input β_i $\mathcal{O}$ returns Z_i. $\mathbf{Hybrid}_2$ is actually equivalent to $\mathbf{Hybrid}_1$ because the CAPTCHAs Z_i are also generated randomly.

Claim 5.5. *$\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are computationally indistinguishable.*

Proof. Note that in $\mathbf{Hybrid}_0$, the sampler parameters U are generated exactly as in the Real-experiment from Definition 2.6, while in $\mathbf{Hybrid}_1$, the sampler parameters are generated as in the Ideal-experiment. Thus, by the security of universal sampler scheme, we have
$|\Pr[\mathbf{Hybrid}_0 = 1] - \Pr[\mathbf{Hybrid}_1 = 1]| \leq \mathsf{negl}(\lambda)$. □

We construct our CAPTCHA solver in $\mathbf{Hybrid}_2$ by adding σ_i' to our set S anytime the adversary queries $\mathbf{G}$ with a string $\beta = (pwd_i, \sigma_i', s)$ — we using our B-units of human work to solve any (up to B) queries made by adversary to the human oracle $\mathcal{H}$. If the adversary queries $\mathbf{G}$ with at least $B+1$ unique/valid CAPTCHA solutions then we will win the CAPTCHA game by simply outputting S. Thus, it suffices to argue that an adversary cannot break passwords in $\mathbf{Hybrid}_2$ with probability $p_1 + \ldots + p_B + 1/(\mathrm{poly}(\lambda))$ unless (with non-negligible probability) the adversary queries $\mathbf{G}$ on at least $B+1$ unique/valid CAPTCHA solutions. Our theorem now follows immediately from the next lemma.

Lemma 5.6. *Suppose that the adversary succeeds with probability $p_1 + ... + p_B + 1/(g(\lambda))$ in $\mathbf{Hybrid}_2$ for some polynomial $g(\lambda)$. The probability that the adversary queries the random oracle $\mathbf{G}$ with at least $B+1$ unique/valid CAPTCHA solutions is at least $1/g(\lambda) - \mathsf{negl}(\lambda)$.*

Proof. Set $t = 0$ and increment t every time the adversary queries $\mathbf{G}$. Let $C^t \subseteq [n]$ denote the set of indices for which, by time t, the adversary has queried $\mathbf{G}(pwd, \sigma_i, s)$ with a valid solution σ_i to CAPTCHA puzzle Z_i (i.e., $\textsc{capt}.\texttt{Verify}(\tilde{\textsc{pp}}, Z_i, tag_i, \sigma_i) = 1$). Let $View^t$ denote the view of the adversary and time t and let $p_{C^t} = \sum_{i \in C^t} p_i$. Now we condition on the event E^t that $pwd = pwd_{j*}$ for $j* \notin C^t$. For every password pwd_j with $j \notin C^t$ we have

$$\Pr_{\mathcal{P}}\left[pwd_j | E^t, View^t\right] = \frac{p_j}{1 - \sum_{i \in C^t} p_i} + \mathsf{negl}(\lambda) \ .$$

Thus, if the adversary has failed to find the password thus far and if the adversary queries the human oracle on at most 1 more CAPTCHA then the adversary's conditional success rate is at most

$$\max_{j \notin C^t} \frac{p_j}{1 - \sum_{i \in C^t} p_i} + \mathsf{negl}(\lambda) \ .$$

Now we work inductively. Initially, ($t = 0$) we have $\max_{j \notin C^0} = \frac{p_j}{1-\sum_{i \in C^t} p_i} = p_1$. If the adversary only queries the human oracle once then the adversary's odds are at best $p_1 + \mathsf{negl}(\lambda)$ [12], Suppose inductively that after the adversary's makes B' valid queries ($C^{B'} = \{i_1, ..., i_{B'}\}$) to the random oracle (i.e., intuitively the adversary uses B' units of human work) that his chance of cracking the password is at most $pwd_{i_1} + ... + pwd_{i'_B} + \mathsf{negl}(\lambda)$. Now suppose that adversary's makes one extra query to $\mathbf{G}$ with a valid CAPTCHA solution $C^{B'+1} = \{i_1, ..., i_{B'}\} \cup \{i_{B'+1}\}$. The adversary's success rate is now at most

$$\begin{aligned} & \sum_{j=1}^{B'} p_{i_j} + \mathsf{negl}(\lambda) + \left(1 - \sum_{j=1}^{B'} p_{i_j} - \mathsf{negl}(\lambda)\right) \times \Pr_{\mathcal{P}}\left[pwd_{i_{B'+1}} \middle| E^{B'}, View^{B'}\right] \\ \leq \quad & \sum_{j=1}^{B'} p_{i_j} + \mathsf{negl}(\lambda) + \left(1 - \sum_{j=1}^{B'} p_{i_j} - \mathsf{negl}(\lambda)\right) \times \left[\frac{p_{i_{B'+1}}}{1 - \sum_{j=1}^{B'} p_{i_j}} + \mathsf{negl}(\lambda)\right] \\ \leq \quad & p_{i_1} + \ldots + p_{i_{B'+1}} + \mathsf{negl}(\lambda) \\ \leq \quad & p_1 + ... + p_{B'+1} + \mathsf{negl}(\lambda) \ . \end{aligned}$$

□

□

Discussion. We believe that the construction of our secure password authentication scheme might lead to many other useful applications. For example, the scheme might allow us to use human memorable (i.e., lower entropy) secrets to secure highly confidential data like secret keys. Let pwd_i be the user's password and let σ_i denote the solution to the corresponding CAPTCHA challenge. The random oracle value $R_i = \mathbf{G}\big(pwd_i, \sigma_i, s, 1|i\big)$ is completely uncorrelated with any information that the adversary can obtain without discovering the user's password. The random values $R_1, R_2, \ldots$ could be used as a one-time pad to efficiently encrypt/decrypt information on a hard drive or to (re)derive private keys for a signature scheme.

As another application we could use the same general framework as a way to detect bots *without* interaction! Suppose that we rename the algorithms $\mathtt{CreateAccount}^{\mathcal{H}}$ and $\mathtt{Authenticate}^{\mathcal{H}}$ to $\mathtt{GenerateVerifiedMessage}^{H}$ and $\mathtt{VerifyMessage}^{H}$. The algorithms have essentially the same functionality except for a few minor modifications: 1) the password field *pwd* is renamed to denote a message m that a user Alice wishes to send to Bob, 2) we replace the username u with a pair (u_1, u_2) where u_1 denotes the sender and u_2 denotes the intended receiver, and we fix the salt value $s = \mathbf{G}(u_1, u_2, m)$ for a given message m that a user u_1 wishes to send to u_2. To send the message m to Bob Alice would first execute $\mathtt{GenerateVerifiedMessage}^{H}(\mathtt{PP}, (Alice, Bob), m)$ and solve the corresponding CAPTCHA to obtain a tuple (h, s). Now Alice sends the tuple $(Alice, Bob, m, h, s)$ to Bob. At this point Alice is finished with the protocol. Bob runs $\mathtt{VerifyMessage}^{H}\big(\mathtt{PP}, (Alice, Bob), m, h, s\big)$ and solves the corresponding CAPTCHA to obtain a bit b. If $b = 1$ then Bob accepts that a human (possibly Alice) spent time and energy to send the him the message m[13]. If $b = 0$ then Bob may dismiss the message as potentially being produced by a bot.

6 Future Challenges

While we believe that Proofs of Human Work could have many benefits, we see three primary challenges for future research. First, because our construction of PoH puzzles is based on $i\mathcal{O}$ HumanCoin is not

[12] Suppose that the adversary has never queried $\mathbf{G}$ with a valid CAPTCHA solution and then he queries the random oracle with the solution σ_{i_1} to puzzle Z_{i_1}. This adversary learns whether or not the corresponding password pwd_{i_1} is correct or not. The probability that password pwd_{i_1} is correct is at most $p_{i_1} + \mathsf{negl}(\lambda)$.

[13] If Bob wanted to additionally verify that Alice was the human that sent the message Alice and Bob would need to use other cryptographic tools like digital signatures.

practical without a large breakthrough in the design of practical $i\mathcal{O}$ schemes. Could we design efficient targeted obfuscation schemes for specific programs like our PoH algorithms? Second, because our PoH puzzles rely on the assumption that some underlying AI problem is hard it is possible that a cryptocurrency like HumanCoin might have a shorter shelf life (e.g., if it takes 15 years for AI researchers to break the underlying CAPTCHA then HumanCoin would expire in at most 15 years). Would it possible for HumanCoin participants to reach a consensus to change the underlying CAPTCHA in the event of an AI breakthrough? Finally, our Proof of Human Work construction, and by extension HumanCoin, requires an initial trusted setup phase for the Proof of Human Work construction. If the Proof of Human Work system is generated by a malicious party then that party might be able to insert a trapdoor which would allow him to mine HumanCoins without any human effort. We note that this concern is not unique to HumanCoin. Other cryptocurrencies like Zerocash [5] also require an initial trusted setup phase[14]. Ben-Sasson et al. [6] proposed to run this trusted setup phase using secure multiparty computation. As long as at least one of the parties in this computation are honest it would be impossible for a malicious adversary to insert a backdoor. Similar techniques could also be used to minimize risks during the HumanCoin setup phase.

In addition to cryptocurrency we also showed that our PoH techniques could be applied to protect passwords and to detect bots without interaction. What other applications are possible?

Acknowledgement: The authors thank Andrew Miller for helpful discussions.

References

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014. 1

[2] J. Aspnes, C. Jackson, and A. Krishnamurthy. Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, July 2005. 5

[3] A. Back. Hashcash — A denial of service counter-measure. 2002. http://hashcash.org/papers/hashcash.pdf. 1, 5, 14, 25, 27

[4] B. Barak, N. Bitansky, R. Canetti, Y. T. Kalai, O. Paneth, and A. Sahai. Obfuscation for evasive functions. In Y. Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 26–51. Springer, Heidelberg, Feb. 2014. 7

[5] E. Ben-Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014. 21

[6] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *2015 IEEE Symposium on Security and Privacy*, pages 287–304. IEEE Computer Society Press, May 2015. 21

[7] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 421–439. Springer, Heidelberg, Aug. 2014. 1

[8] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld. Proof of activity: Extending bitcoin's proof of work via proof of stake. In *Proceedings of the ACM SIGMETRICS 2014 Workshop on Economics of Networked Systems, NetEcon*, 2014. 2

[9] J. Blocki, M. Blum, and A. Datta. Gotcha password hackers!, 2013. http://www.cs.cmu.edu/~jblocki/papers/aisec2013-fullversion.pdf. 5, 16, 28

[10] J. Blocki, S. Komanduri, L. F. Cranor, and A. Datta. Spaced repetition and mnemonics enable recall of multiple strong passwords. In *NDSS 2015*. The Internet Society, Feb. 2015. 17

[14] Arguably, even Bitcoin does require some trust assumptions during setup. For example, we need to trust that the cryptographic hash function $h =$SHA256, which is modeled as a random oracle in the Bitcoin protocol, does not have any secret backdoors. A malicious miner with a secret backdoor could easily reverse old transactions.

[11] J. Blocki, S. Komanduri, A. Procaccia, and O. Sheffet. Optimizing password composition policies. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 105–122. ACM, 2013. 17

[12] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552. IEEE Computer Society Press, May 2012. 17

[13] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten. SoK: Research perspectives and challenges for bitcoin and cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, pages 104–121. IEEE Computer Society Press, May 2015. 14, 24

[14] J. Bonneau and S. Schechter. ”toward reliable storage of 56-bit keys in human memory”. In *Proceedings of the 23rd USENIX Security Symposium*, August 2014. 17

[15] D. Bradbury. Feathercoin hit by massive attack, 2013. 26

[16] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell. The end is nigh: Generic solving of text-based captchas. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, San Diego, CA, Aug. 2014. USENIX Association. 4, 8

[17] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. How good are humans at solving captchas? a large scale evaluation. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 399–413. IEEE, 2010. 8

[18] R. Canetti, S. Halevi, and M. Steiner. Mitigating dictionary attacks on password-protected local storage. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 160–179. Springer, Heidelberg, Aug. 2006. 4, 16, 28

[19] W. Daher and R. Canetti. Posh: A generalized captcha with security applications. In *Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 1–10. ACM, 2008. 28

[20] J. R. Douceur. The sybil attack. In *Peer-to-peer Systems*, pages 251–260. Springer, 2002. 4, 5, 25

[21] C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 426–444. Springer, Heidelberg, Aug. 2003. 16

[22] C. Dwork, J. Y. Halpern, and O. Waarts. Performing work efficiently in the presence of faults. *SIAM Journal on Computing*, 27(5):1457–1491, 1998. 5

[23] 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, Heidelberg, Aug. 1993. 1, 5, 25, 27

[24] S. Dziembowski. How to pair with a human. In J. A. Garay and R. D. Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 200–218. Springer, Heidelberg, Sept. 2010. 4

[25] S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605. Springer, Heidelberg, Aug. 2015. 2, 5

[26] J. Elson, J. R. Douceur, J. Howell, and J. Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *Proc. of CCS*, pages 366–374. Citeseer, 2007. 4, 27

[27] I. Eyal and E. G. Sirer. Majority is not enough: Bitcoin mining is vulnerable. In N. Christin and R. Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 436–454. Springer, Heidelberg, Mar. 2014. 5, 15, 25, 26

[28] D. Florêncio and C. Herley. Where do security policies come from. In *Proc. of SOUPS*, page 10, 2010. 17

[29] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 281–310. Springer, Heidelberg, Apr. 2015. 2, 5, 15, 25

[30] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, Oct. 2013. 3, 4, 5

[31] S. Goldwasser and G. N. Rothblum. On best-possible obfuscation. In S. P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 194–213. Springer, Heidelberg, Feb. 2007. 7

[32] D. Hofheinz, T. Jager, D. Khurana, A. Sahai, B. Waters, and M. Zhandry. How to generate and use universal samplers. Cryptology ePrint Archive, Report 2014/507, 2014. http://eprint.iacr.org/2014/507. 4, 5, 8, 9, 12

[33] K.-F. Hwang, C.-C. Huang, and G.-N. You. A spelling based captcha system by using click. In *Biometrics and Security Technologies (ISBAST), 2012 International Symposium on*, pages 1–8, March 2012. 2, 26

[34] J. Kani and M. Nishigaki. Gamified captcha. In L. Marinos and I. Askoxylakis, editors, *Human Aspects of Information Security, Privacy, and Trust*, volume 8030 of *Lecture Notes in Computer Science*, pages 39–48. Springer Berlin Heidelberg, 2013. 2, 26

[35] R. A. Khot and K. Srinathan. icaptcha: Image tagging for free. In *the Proc. Conference on Usable Software and Interface Design*, 2009. 2, 26

[36] A. Kiayias, H.-S. Zhou, and V. Zikas. Fair and robust multi-party computation using a global transaction ledger. To appear in *EUROCRYPT 2016*, 2016. http://eprint.iacr.org/2015/574. 1

[37] S. Komanduri, R. Shay, P. Kelley, M. Mazurek, L. Bauer, N. Christin, L. Cranor, and S. Egelman. Of passwords and people: measuring the effect of password-composition policies. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, pages 2595–2604. ACM, 2011. 17

[38] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. To appear in *2016 IEEE Symposium on Security and Privacy*, 2016. http://eprint.iacr.org/. 1

[39] A. Kumarasubramanian, R. Ostrovsky, O. Pandey, and A. Wadia. Cryptography using captcha puzzles. In K. Kurosawa and G. Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 89–106. Springer, Heidelberg, Feb. / Mar. 2013. 4, 28

[40] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 14*, pages 30–41. ACM Press, Nov. 2014. 1

[41] R. Kumaresan, T. Moran, and I. Bentov. How to use bitcoin to play decentralized poker. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 195–206. ACM Press, Oct. 2015. 1

[42] J. Kwon. Tendermint: Consensus without mining, 2014. 2

[43] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982. 5, 24

[44] T. Melin and T. Vidhall. Namecoin as authentication for public-key cryptography. 2014. 1

[45] R. C. Merkle. A digital signature based on a conventional encryption function. In C. Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, pages 369–378. Springer, Heidelberg, Aug. 1988. 14

[46] A. Miller, A. E. Kosba, J. Katz, and E. Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In I. Ray, N. Li, and C. Kruegel:, editors, *ACM CCS 15*, pages 680–691. ACM Press, Oct. 2015. 5, 11

[47] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–134. IEEE, 2003. 4, 8

[48] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: Captchas-understanding captcha-solving services in an economic context. In *USENIX Security Symposium*, volume 10, page 3, 2010. 8

[49] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf. 1, 5, 14, 15, 24, 25

[50] A. Narayanan, J. Bonneau, E. Felten, and A. Miller. *Bitcoin and Cryptocurrency Technology (online course)*. 2015. https://piazza.com/princeton/spring2015/btctech/resources. 2, 14, 15, 16, 24

[51] S. Park, K. Pietrzak, A. Kwon, J. Alwen, G. Fuchsbauer, and P. Gaži. Spacemint: A cryptocurrency based on proofs of space. Cryptology ePrint Archive, Report 2015/528, 2015. http://eprint.iacr.org/2015/528. 2, 5, 26

[52] A. Sahai and B. Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In D. B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014. 5

[53] G. Sauer, H. Hochheiser, J. Feng, and J. Lazar. Towards a universally usable captcha. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, 2008. 4, 27

[54] P. Y. Simard. Using machine learning to break visual human interaction proofs (hips. *Advances in Neural Information Processing Systems*, 17:265–272, 2004. 8

[55] N. Szabo. Formalizing and securing relationships on public networks. In *First Monday*, 1997. http://firstmonday.org/ojs/index.php/fm/article/view/548/469. 1

[56] J. Tam, J. Simsa, S. Hyde, and L. Von Ahn. Breaking audio captchas. *Advances in Neural Information Processing Systems*, 1(4), 2008. 4, 8

[57] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Advances in Cryptology EUROCRYPT 2003*, pages 294–311. Springer, 2003. 2, 4, 8

[58] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. reCAPTCHA: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008. 2, 4, 26, 27

[59] B. Waters. CS 395T Special Topic: Obfuscation in Cryptography. 2014. http://www.cs.utexas.edu/~bwaters/classes/CS395T-Fall-14/outline.html. 5

[60] B. Waters. How to use indistinguishability obfuscation. In *Visions of Cryptography*, 2014. Talk slides available at http://www.cs.utexas.edu/~bwaters/presentations/files/how-to-use-IO.ppt. 5

[61] B. Waters, A. Juels, J. A. Halderman, and E. W. Felten. New client puzzle outsourcing techniques for DoS resistance. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 04*, pages 246–256. ACM Press, Oct. 2004. 4

[62] J. Wilkins. Strong captcha guidelines v1. 2. *Retrieved Nov*, 10:2010, 2009. 8

A Additional Background for Cryptocurrency

At a high level HumanCoin closely follows the Bitcoin protocol, except that we use Proofs of Human work puzzles to extend the blockchain instead of PoW puzzles. In this section we briefly overview the key features of Bitcoin [49]. Our overview follows the systemization of knowledge paper by Bonneau et al. [13], but we present an overly simplified version of Bitcoin. We refer interested readers to the excellent lectures by Narayanan et al. [50] or the original paper published under the pseudonym Nakamoto [49] for more details about Bitcoin.

A.1 Bitcoin

Overcoming the "Double Spending" Challenge One of the key challenges in any decentralized cryptocurrency is preventing users from "double spending" their money. While digital signatures prevent malicious users from spending another user's bitcoins (e.g., the transaction "Alice sends Bob 50 bitcoins" would be need to be signed by Alice) they do not necessarily prevent a user from trying to spend the same bitcoin twice. Suppose for example that Alice has exactly 50 bitcoins and simultaneously sends the signed messages "Alice sends Bob 50 bitcoins" and "Alice sends 50 Bitcoins to Charlie" to Bob and Charlie respectively. If Bob (resp. Charlie) is unaware of the message that Alice sent to Charlie (resp. Bob) then he might accept the payment. Later it would unclear whether Bob or Charlie was the rightful owner of the 50 Bitcoins. To solve the double spending problem it is necessary to develop a distributed consensus protocol which would allow users to agree on which transactions have or have not taken place, and these protocols must work even if some of the users are malicious. Traditional protocols for reaching distributed consensus in the presence of adversarial (byzantine) users assume that most (e.g., $2/3$) of the parties are honest [43].

However, this is not a meaningful assumption on the internet because the adversary can easily create multiple fake identities to mount a Sybil attack [20].

Bitcoin has an elegant distributed consensus protocol based on Proof of Work puzzles [23] — Bitcoin uses the HashCash algorithm due to Adam Back [3]. Transactions in Bitcoin are recorded in a cryptographic data-structure called the blockchain $x = B_0, \ldots, B_t$. A block B_i is only valid if several conditions are satisfies several conditions: (1) all of the transactions inside the block are valid (e.g., each transaction is signed by the sender and the spender has sufficient funds), (2) the block B_i contains the cryptographic hash $h(B_{i-1})$ of the previous block B_{i-1}[15], and (3) the block B_i is padded with a nonce s which ensures that cryptographic hash of the block B_i is sufficiently small $h(B_i) < T$ — finding such a nonce s constitutes the proof of work. Here, T is a public parameter that we will discuss later. The blockchain x is valid if all of its individual blocks are valid. The consensus protocol is simple: if a participant receives two contradictory block chains x and x' accept the longer one. A merchant accepts that a transaction is valid if and only if that transaction is recorded in block B_{t-6} and $B_0, \ldots, B_t$ is a prefix of the longest blockchain that he has received. Because each block B_i in the blockchain contains a cryptographic hash $h(B_{i-1})$ of the previous block B_{i-1} it is not possible for a ppt adversary to insert/modify/delete a block in the middle of the chain. If the adversary wanted to modify a block B_i (e.g., to erase a particular transaction) then that adversary would need to recreate a longer blockchain $x' = B_0, \ldots, B_{i-1}, B'_i, B'_{i+1}, \ldots, B'_{t+1}$ starting from block B'_i. Unless a miner controls at least 25% of the hash power in the network the rational strategy is always to extend the longest blockchain because nobody will accept the Bitcoins they try to mine in a shorter blockchain (e.g., the special transaction in which a miner claims 25 bitcoins' would only be recorded on a shorter blockchain which nobody accepts) [27]. Assuming that the network has high synchronicity [29] and that a malicious user controls at most 49% of the computational mining power he will never be able to tamper with any of the transactions in a block B_i from the middle of the blockchain because he would need to eventually produce a new blockchain $b' = B_0, \ldots, B_{i-1}, B'_i, B'_{i+1}, \ldots, B'_t$ that is at least as long as the true blockchain b — he will fail to accomplish this goal with high probability [49]. If the network does not have high synchronicity then the protocol is secure as long a malicious user controls at most 33.33% of the computational hash power [29].

Mining To incentivize miners to help validate transactions (i.e., extend the blockchain by finding a valid nonce s) a successful miner is given a reward (e.g., 25 new Bitcoins) for his effort. Mining in Bitcoin is a bit like a computational lottery. Suppose that $x = B_0, \ldots, B_t$ is the current blockchain and that we want to add a new block B_{t+1}, containing the latest Bitcoin transactions, to the blockchain. To add B_{t+1} to the blockchain we need to find a nonce s such that $h(B_{t+1}) < T$ after the block B_{t+1} is padded with s. The nonce s might be viewed as a lottery ticket, and the miner needs to do work (e.g., compute h) to see if his ticket is a winning ticket. A miner can increase his odds of winning each lottery by increasing his computation power (e.g., allowing him to "buy" more lottery tickets by trying more nonces s). Current Bitcoin miners tend to use customized hardware (e.g., ASICs) — it is no longer profitable for an typical user to mine bitcoins.

Parameter Selection In Bitcoin T is a public parameter that is tuned to ensure that, on average, miners will add one new block to the blockchain every 10 minutes. The Bitcoin protocol would most likely work just fine with a shorter delay (e.g., 5 minutes) or a slightly longer delay (e.g., 20 minutes) between consecutive blocks — there is nothing magical about the specific target value of 10 minutes. However, it is clear that there needs to be some delay to promote stability. If multiple miners find a new block at the same time then we could end up with competing blockchains resulting in temporary confusion. Note that if the value of T remains fixed then the average time to create one new block would begin to decrease as more miners join Bitcoin, or as existing miners upgrade their computational resources. Thus, the value of T must be adjusted periodically. The value of T is therefore adjusted every 2,016 blocks, which works out to two weeks on

[15] Bitcoin uses the cryptographic hash function $h =$SHA256. The hash function h is typically treated as a random oracle in security analysis of Bitcoin.

average (2 weeks = 2016 × 10 minutes), using the formula

$$T = T_{old} \times \frac{t_{elapsed}}{2016 \times 10 \text{ min}} ,$$

where $t_{elapsed}$ denotes the time span that it actually took to generate the last $2,016$ blocks.

A.2 HumanCoin Advantages

We believe that HumanCoin offers several advantages over traditional cryptocurrencies like Bitcoin:

- (Useful Work) One criticism of cryptocurrencies like Bitcoin is that the consensus protocol 'wastes' valuable computational resources like energy or space [51]. While HumanCoin relies on valuable human resources (e.g., time, energy), we observe that this human effort could potentially be used to do work that has personal and/or societal benefit. Previous research on CAPTCHAs has explored the possibility of developing CAPTCHA challenges that are fun [34], educational [33] or even productive [58, 35].

- (Environment) HumanCoin does not require massive power consumption like Bitcoin, which has been called it a "environmental disaster" by critics. The Bitcoin mining process 'wastes' valuable computer cycles and consumes a lot of energy. Previous research on cryptocurrencies have explored alternative consensus protocols like proofs of space [51] that would be more environmentally friendly. It could be argued that HumanCoin wastes human effort instead of electric energy. However, we argue that the human effort would not be wasted if involved fun or educational tasks.

- (51% Attacks) We believe that HumanCoin would be less vulnerable to 51% attacks than alternative cryptocurrencies because it would not be dominated by existing PoW mining pools. Cryptocurrencies typically assume that an adversary is able to control at most 49% of the computational power of all miners using that particular cryptocurrency. An attacker who controls 51% of the computing power for all miners in a particular cryptocurrency reverse old transactions. This is not just a theoretical concern. For example, Feathercoin [15] was compromised when a large Bitcoin mining pool suddenly switched to mine FeatherCoins and decided to behave adversarially. The new (adversarial) mining pool, which controlled well over 50.1% of the computational power in the Feathercoin, was able to reverse many older transactions in the blockchain. While it may be practically impossible to attack a well-established cryptocurrency like Bitcoin, this is a fundamental concern for any computational PoW-based cryptocurrency in its infancy. We observe that existing mining pools will be useless for attacking Human Coin because we rely on proofs of *human* effort instead of proofs of *computational* work.

- (Covert Nation State Attacks) While Bitcoin is more established it could still be vulnerable to covert 51% attacks from a large nation state who can afford to invest in powerful computers. It would be harder for a large nation state to covertly attack HumanCoin because it would need to hire many human workers. This is expensive and it would be difficult to hide such a large human endeavor.

- (Fairness) HumanCoin is "fair" by nature. Rich people can buy equipments to get BitCoins, but the only way to earn HumanCoins would be to mine the coins yourself or buy them from another miner. We believe that this would help avoid incentive compatibility issues like the one in Bitcoin. The collective mining pool GHash.io briefly accounted for more than 50% of computation hash power. Furthermore, honest mining is only a provably rational strategy if an agent controls at most 25% of computational hash power [27].

- (Unemployment) We believe that HumanCoin could have other positive social implications. Unemployment is a significant problem in every modern economy, but the HumanCoin miner job would always be available to anyone. On the negative side it is possible that HumanCoin might encourage more malicious people to create "human-mining sweatshops." While we cannot definitely rule out this possibility, we observe that a human miner would be free — in countries with proper labor regulation — to mine independently or to switch mining pools.

A.3 Proof of Work puzzle

Intuitively, a Proof of Work (PoW) allows a party to prove that they completed some level of computational work. Proofs of work have been proposed as mechanism to prevent denial-of-service attacks and e-mail spam [23] by making the attack too costly. In the spam prevention scenario a user could require senders to prove that they spent some minimal base level of computational effort before sending the message. While a proof of work is not a proof that the sender is actually trustworthy, it can be a good indicator. The proof of work needs to meet the following four requirements:

- It has to be moderately hard to compute. Otherwise, the spammer could easily produce valid proofs of work for each of his recipients.
- The difficulty needed in order to solve the proof has to be adjustable in a linear way. Otherwise, the function could not adapt to a changing amount of computational power over time.
- It has to be easy to verify, so that everyone can efficiently check if a proof is valid.
- It should be possible to ensure that proofs of work cannot used multiple times. Hence, it has to be possible to link the proof of work to some public signal. In our spam setting this could be a combination of the e-mail message, the intended recipient and the date. It could also be a function of the hash of the block header in a distributed consensus protocol.

A popular proof-of-work system which is also used in Bitcoin is Hashcash [3]. It was invented by Adam Back in 1997. The heart of the Hashcash cost-function is some kind of hash function. Back suggested SHA1 and MD5 in his paper, but more modern hash functions may be used nowadays – for instance, Bitcoin uses SHA256. The hash function just has to be efficiently computable. The idea of Hashcash is that the sender has to compute a hash with specific properties. The hash has to start with a configurable, but fixed number of zeros. In order to compute such a hash, a random number x, often called nonce, is added to the string to be hashed. If the resulting hash does not have the claimed requirements, x is incremented and another hash is computed. The sender than sends the string as well as the computed number x to the receiver which just has to check whether the hash of the string concatenated with the number fulfills the requirements. Thus, the proof require some computational work to obtain, but the verification of the proof is rather easy.

Other proof-of-work systems mainly differ from their cost-functions; the functions which are actually responsible for the work. Cost-functions should be expensive to compute and efficient to verify. Ideally, the expensiveness of the computation is configurable via a parameter.

B Related Work: Puzzles in AI

The term CAPTCHA (Completely Automated Public Turing-Test to tell Computers and Humans Apart) was coined by Von Ahn et al.. Informally, a CAPTCHA is a program that generates a puzzle Z — which should be easy for a human to solve and difficult for a computer to solve — as well as a solution a. Many popular forms of CAPTCHAs generate garbled text [58], which can be read by a human but is difficult for a computer to decipher. Other versions of CAPTCHAs rely on the natural human capacity for audio [53] or image recognition [26]. CAPTCHA puzzles have been widely deployed on the internet (e.g., to fight spam). In the standard CAPTCHA setting a server generates a random pair (Z,a) and sends the puzzle Z to the user as a challenge, while keeping the solution a secret. Later the secret solution will be used to verify the user's answer. The implicit assumption is that the answer and the random bits used to generate the puzzle remain hidden — otherwise a spam bot could simply generate the puzzle and the answer himself. While this assumption may be reasonable in the spam bot setting, it does not make sense in our distributed setting.

Canneti et al. [18] proposed a slight modification of notion of CAPTCHAs that they called HOSPs (Human Only Solvable Puzzles). A HOSP is different from a CAPTCHA in several key ways: (1) The challenge must remain difficult for a computer to solve even if the random bits used to generate the puzzle are made public. (2) There is no single correct answer to a HOSP. It is okay if different people give different responses to a challenge as long as people can respond to the challenges easily, and each user can consistently answer the challenges. The only HOSP construction proposed in [18] involved stuffing a hard drive with unsolved CAPTCHAs[16]. This solution is unsuitable in our settings for several reasons: First, there is no trusted centralized server on which the unsolved CAPTCHA puzzles could be stored. Either each miner would need to store all of the unsolved CAPTCHAs locally, or we would need to adopt an interactive secure and verifiable multiparty distributed storage protocol. Second, an adversary who solves all of the puzzles on the hard drive (e.g., by hiring humans) would be able to break the scheme completely. Finally, there is no way for another computer to verify that a solution a to a puzzle Z correct without interacting with a human.

Blocki et al. [9] introduced GOTCHAs (Generating panOptic Turing Tests to Tell Computers and Humans Apart) as a defense against offline dictionary attacks. GOTCHAs are similar to HOSPs except that the puzzle generation process itself involves interaction between a computer and a human. In their construction a computer generated random inkblot images and the human is asked to label those images. Later the human is challenged with the inkblot images as well as the labels (in permuted order) and is asked to match the labels with the appropriate inkblot. This approach is not suitable for us because we do not have a way for a computer to verify that a puzzle was generated honestly — due to human interaction in the puzzle generation process.

Kumarasubramanian et al. [39] introduced the notion of human-extractable CAPTCHAs, and used them to construct concurrent non-malleable zero-knowledge protocols. Intuitively, a CAPTCHA is human-extractable if an adversary must leak some information about a puzzle Z (via queries to a human solver) to find the answer a.

[16] The problem of finding a HOSP construction that does not rely on a dataset of unsolved CAPTCHAs was left as an open problem [18]. Several other candidate HOSP constructions have been experimentally evaluated in subsequent work [19] (they are called POSHs in the second paper), but the usability results for every scheme that did not rely on a large dataset on unsolved CAPTCHAs were underwhelming.