Nonoutsourceable Scratch-Off Puzzles to Discourage Bitcoin Mining Coalitions

Andrew Miller Ahmed Kosba Jonathan Katz
University of Maryland
{amiller,akosba,jkatz}@cs.umd.edu

Elaine Shi
Cornell University
runting@gmail.com

***Abstract*—An implicit goal of Bitcoin's reward structure is to diffuse network influence over a diverse, decentralized population of individual participants. Indeed, Bitcoin's security claims rely on no single entity wielding a sufficiently large portion of the network's overall computational power. Unfortunately, rather than participating independently, most Bitcoin miners join coalitions called *mining pools* in which a central pool administrator largely directs the pool's activity, leading to a consolidation of power. Recently, the largest mining pool has accounted for more than half of network's total mining capacity. Relatedly, "hosted mining" service providers offer their clients the benefit of economies-of-scale, tempting them away from independent participation. We argue that the prevalence of mining coalitions is due to a limitation of the Bitcoin proof-of-work puzzle – specifically, that it affords an effective mechanism for enforcing cooperation in a coalition. We present several definitions and constructions for "nonoutsourceable" puzzles that thwart such enforcement mechanisms, thereby deterring coalitions. We also provide an implementation and benchmark results for our schemes to show they are practical.**

I. Introduction

Bitcoin [36] and subsequent decentralized cryptocurrencies have gained rapid popularity in recent years, and are often quoted as "a peek into the future financial and payment infrastructure". Security of such cryptocurrencies is critical, and to ensure security the most fundamental assumption made by decentralized cryptocurrencies is that *no single entity or adminstration wields a large fraction of the computational resources in the network*. Violation of this assumption can lead to severe attacks such as history revision and double spending which essentially nullify all purported security properties that are widely believed today.

However, two recent trends in mining – namely, *mining pools* and *hosted mining* – have led to the concentration of mining power, and have cast serious doubt on the well-foundedness of these fundamental assumptions that underly the security of Bit-coin-like cryptocurrencies. Specifically, mining pools exist because solo miners wish to hedge mining risks and obtain rewards at a more stable, steady rate. At several times over the past two years, the largest handful of mining pools have accounted for well over a third of the network's overall computing effort [11]. For example, recently the largest mining pool, GHash.IO, has even exceeded 50% of the total mining capacity.[1] Currently, Hosted mining, on the other hand, allows individuals to outsource their mining effort to one or a few large service providers. Hosted mining services have already emerged, such as Alydian [10], whose "launch day pricing was $65,000 per Terahash, and mining hosting contracts are available in 5 and 10 Th/sec blocks" [10]. Hosted mining is appealing because it can potentially reduce miners' cost due to economies of scale. Henceforth we will refer to both mining pools and hosted mining as mining coalitions.

Such large mining coalitions present a potential lurking threat to the security of Bitcoin-like cryptocurrencies. To exacerbate the matter, several recent works [21], [28] showed that it may be incentive compatible for a mining coalition to deviate from the honest protocol – in particular, Eyal and Sirer [21] showed that a mining concentration of about $1/3$ of the network's mining power can obtain disproportionately large rewards by exhibiting certain "selfish mining" behavior.

While alternatives to centralized mining pools are well-known and have been deployed for several years, (such as *P2Pool*, [49] a decentralized mining pool architecture), these have unfortunatley seen extremely low user adoption (at the time of writing, they account for less than 2% of the network). Fundamentally, the problem is that Bitcoin's reward mechanism provides no particular incentive for users to use these decentralized alternatives.

Increasing understanding of these problems has prodded extensive and continual discussions in the broad cryptocurrency community, regarding how to deter such coalitions from forming and retain the decentralized nature of Bitcoin-like cryptocurrencies [31]. The community demands a technical solution to this problem.

A. Our Results and Contributions

Our work provides a timely response to this community-wide concern [31], providing the *first formally founded solution* to combat Bitcoin mining centralization. Our key observation is the following: an enabling factor in the growth of mining pools is a simple yet effective enforcement mechanism; members of a mining pool do not inherently trust one another, but instead submit cryptographic proofs (called "shares") to the other pool members (or to the pool operator), in order to demonstrate they are contributing work that can only benefit the pool (e.g., work that is tied to the pool operator's public key).

Strongly nonoutsourceable puzzles. Our idea, therefore, is to disable such enforcement mechanisms in a cryptographically strong manner. To this end, we are the first to propose *strongly nonoutsourceable puzzles*, a new form of proof-of-work puzzles which additionally guarantee the following:

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

If a pool operator can effectively outsource mining work to a worker, then the worker can steal the reward without producing any evidence that can potentially implicate itself.

Intuitively, if we can enforce the above, then any pool operator wishing to outsource mining work to an untrusted worker runs the risk of losing its entitled mining reward, thus effectively creating a disincentive to outsource mining work (either in the form of mining pools or hosted mining). Our nonoutsourceable puzzle is broadly powerful in that it renders unenforceable even *external contractual agreements* between the pool operator and the worker. In particular, no matter whether the pool operator outsources work to the worker through a cryptocurrency smart contract or through an out-of-the-band legal contract, we guarantee that the worker can steal the reward without leaving behind evidence of cheating.

Technical insights. At a technical level, our puzzle achieves the aforementioned guarantees through two main insights:

- **P1:** We craft our puzzle such that if a worker is doing a large part of the mining computation, it must possess a sufficiently large part of a "signing key" such that it can later sign over the reward to its own public key – effectively stealing the award from the pool operator;
- **P2:** We offer a zero-knowledge spending option, such that a worker can spend the stolen reward in a way that reveals no information (including potential evidence that can be used to implicate itself).

As a technical stepping stone, we formulate a weaker notion of our puzzle referred to as a weakly nonoutsourceable puzzle. A weakly nonoutsourceable puzzle essentially guarantees property P1 above, but does not ensure property P2. As a quick roadmap, our plan is to first construct a weakly nonoutsourceable puzzle, and from there we devise a generic zero-knowledge transformation to compile a weakly nonoutsourceable puzzle into a strongly nonoutsourceable one. It turns out that a weakly nonoutsourceable puzzle is the implicit security notion adopted by the recent work of Permacoin [33] but without being formalized there. In Section VI, we argue that weakly nonoutsourceable puzzles alone are inadequate to defeat mining coalitions, and in particular hosted mining.

Implementation and practical performance. We show implementation and evaluation results to demonstrate the practical performance of our puzzles. Based on an instantiation using the succinct zero-knowledge option of Libsnark [6], we show that it would take a cheating worker only **14 seconds** (using approximately a thousand parallel cores) to successfully steal a block reward. Further, stealing a block's reward in zero knowledge consumes only **$10** worth of Amazon AWS compute-time, which is very small in comparison with the block's reward – roughly **$8,750**, based on Bitcoin's current market price. Clearly this provides a sufficiently strong deterrent against mining coalitions. Note also that this zero-knowledge spending option is not normally incurred, since honest miners can simply adopt a cheap plaintext spending option whose cost is insignificant (see Section VI). For both the zero-knowledge and the cheap plaintext spending options, the block verification overhead is insignificant (at most 1.7 seconds) in comparison with the present Bitcoin epoch length (roughly 10 minutes).

Deployment considerations. For our nonoutsourceable puzzles to be practically deployed, it is also important to address several additional challenges, such as how to still allow miners to reduce mining uncertainty (i.e., the positive effects of mining pools), and how to simultaneously address various other design goals such as ASIC resistance, and lightweight clients. We give detailed explanations to address these practical issues related to deployment (Section VII-C). Notably, inspired by the design of state lottery games, we propose a new, multi-tier reward system that allows us to achieve the best of both worlds: ensure non-outsourceability of puzzles, and meanwhile allow smaller players to reduce payoff variance.

Community demand and importance of formal security. The community's demand for a nonoutsourceable puzzle is also seen in the emergence of new altcoins [35], [46] that (plan to) adopt their own home-baked versions of nonoutsourceable puzzles. Their solutions, however, offer only weak nonoutsourceability, and do not provide any formal guarantees. The existence of these custom constructions further motivates our efforts, and demonstrates that it is non-trivial to both formalize the security notions as well as design constructions with provable security. To date, our work provides *the only formally-founded solution*, as well as the *first strongly nonoutsourceable puzzle construction*.

II. Bitcoin Background

We define puzzles and nonoutsourceable puzzles as an independent concept, abstracting away the less relevant details about the Bitcoin protocol itself. Later, however, we will discuss how the puzzles we introduce can be integrated into a Bitcoin-like distributed digital currency. For this reason, as well as to understand the motivation behind our formal definitions, we first present some additional background on Bitcoin and its use of computational puzzles. For a more thorough explanation of the Bitcoin protocol, we refer the readers to [4], [8], [36].

Puzzles, rewards, and epochs. In Bitcoin, new money is printed at a predictable rate, through a distributed coin-minting process. At the time of writing, roughly speaking, 25 bitcoins are minted every 10 minutes (referred to as an epoch) on average. When an epoch begins, a public puzzle instance is generated by computing an up-to-date hash of the global transaction log (called the "blockchain"). Then, Bitcoin nodes race to solve this epoch's puzzle. Whoever first finds an eligible solution to the puzzle can claim the newly minted coins corresponding to this epoch.

In slightly more detail, miners start with the puzzle instance puz, and construct a payload m which contains (a tree hash over) the miners public key and a new set of transaction to commit to the log during this epoch. He then searches for a nonce r such that $\mathcal{H}(\mathsf{puz}\|m\|r) < 2^{\lambda-d}$, where $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ is a hash function and d is a difficulty parameter. The difficulty parameter is adjusted according to the total amount of computational resources devoted to mining to ensure that each epoch lasts 10 minutes on average.

In Section IV, we formally define a generalization of Bitcoin's puzzle called *scratch-off puzzles*. More detail about the original Bitcoin puzzle construction can found in the Appendix.

Consensus mechanism. Bitcoin nodes reach consensus on the history of transactions by having nodes accept the blockchain with the largest total difficulty. Roughly speaking, this defeats history revision attacks, since to revise history would involve computing a blockchain that is more difficult than the known good chain. An adversary must therefore possess a significant fraction of the total computational resources to successfully race against the rest of the network in extending the chain.

Bitcoin is novel in its use of computational puzzles as part of a consensus protocol for anonymous networks without any pre-established PKI. A related approach was earlier proposed by Aspnes et al. [2], although their network model nonetheless retained a strong assumption about pre-established point-to-point channels.

III. Scratch-off Puzzles

As introduced earlier, the Bitcoin protocol is built around a moderately hard computational puzzle. Bitcoin miners compete to solve these puzzles, and whoever solves a puzzle first in each epoch receives a reward. As there is no shortcut to solving this puzzle, for an attacker to dominate the network would require the attacker to expend more computational resources than the rest of the honest participants combined. Although the Bitcoin puzzle is commonly referred to as a *proof-of-work* puzzle, the requirements of the puzzle are somewhat different than existing definitions for proof-of-work puzzles [15], [19], [24], [47].

Before proceeding with our main contribution of nonoutsourceable puzzles, we first provide a formal definition of the basic requirements of the Bitcoin puzzle, which we call a *scratch-off puzzle*.[2] In particular, while a traditional proof-of-work puzzle [24] need only be solvable by a single sequential computation, a scratch-off puzzle must be solvable by several concurrent non-communi-cating entities.

In what follows, we let λ denote a security parameter. A scratch-off puzzle is parameterized by parameters $(\underline{t}, \mu, d, t_0)$ where, informally speaking, $\underline{t}$ denotes the amount of work needed to attempt a single puzzle solution, μ refers to the maximum amount by which an adversary can speed up the process of finding solutions, d affects the average number of attempts to find a solution, and t_0 denotes the initialiazation overhead of the algorithm. We typically assume that $t_0 \ll 2^d \underline{t}$, where $2^d \underline{t}$ is the expected time required to solve a puzzle.

Definition 1. *A scratch-off puzzle is parameterized by parameters $(\underline{t}, \mu, d, t_0)$, and consists of the following algorithms (satisfying properties explained shortly):*

1) $\mathcal{G}(1^\lambda) \to \mathsf{puz}$*: generates a puzzle instance.*

2) $\mathsf{Work}(\mathsf{puz}, m, t) \to \mathsf{ticket}$*: The* Work *algorithm takes a puzzle instance* puz*, some payload m, and time parameter t. It makes t unit scratch attempts, using $t \cdot \underline{t} + t_0$ time steps in total. Here $\underline{t} = \mathsf{poly}(\lambda)$ is the* unit scratch time*, and t_0 can be thought of as the initialization and finalization cost of* Work*.*

3) $\mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket}) \to \{0, 1\}$*: checks if a ticket is valid for a specific instance* puz*, and payload m. If a* ticket *passes this check, we refer to it as a* winning ticket *for* (puz, m)*.*

Intuitively, the honest Work algorithm makes t unit scratch attempts, and each attempt has probability 2^{-d} of finding a winning ticket, where d is called the puzzle's difficulty parameter. For simplicity, we will henceforth use the notation

$$\zeta(t, d) := 1 - (1 - 2^{-d})^t$$

to refer to the probability of finding a winning ticket using t scratch attempts. For technical reasons that will become apparent later, we additionally define the shorthand $\zeta^+(t, d) := \zeta(t+1, d)$. For the remainder of the paper, we assume that the puzzle's difficulty parameter d is fixed, hence we omit the d and write $\zeta(t)$ and $\zeta^+(t)$ for simplicity. We also define the algorithm $\mathsf{WorkTillSuccess}(\mathsf{puz}, m)$ as $\mathsf{Work}(\mathsf{puz}, m, \infty)$; i.e., this algorithm runs until it finds a winning ticket for the given instance and payload.

A scratch-off puzzle must satisfy three requirements:

1) **Correctness.** For any (puz, m, t), if $\mathsf{Work}(\mathsf{puz}, m, t)$ outputs $\mathsf{ticket} \neq \perp$, then $\mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket}) = 1$.

2) **Feasibility and parallelizability.** Solving a scratch-off puzzle is feasible, and can be parallelized. More formally, for any $\ell = \mathsf{poly}(\lambda)$, for any $t_1, t_2, \ldots, t_\ell = \mathsf{poly}(\lambda)$, let $t := \sum_{i \in [\ell]} t_i$.

$$\Pr\left[\begin{array}{l} \mathsf{puz} \leftarrow \mathcal{G}(1^\lambda), \\ m \leftarrow \{0,1\}^\lambda, \\ \forall i \in [\ell] : \mathsf{ticket}_i \leftarrow \mathsf{Work}(\mathsf{puz}, m, t_i) : \\ \exists i \in [\ell] : \mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket}_i) \end{array}\right] \geq \zeta(t) - \mathsf{negl}(\lambda).$$

Intuitively, each unit scratch attempt, taking time $\underline{t}$, has probability 2^{-d} of finding a winning ticket. Therefore, if ℓ potentially parallel processes each makes t_1, t_2, ..., t_ℓ attempts, the probability of finding one winning ticket overall is $\zeta(t) \pm \mathsf{negl}(\lambda)$ where $t = \sum_{i \in [\ell]} t_i$.

3) **μ-Incompressibility.** Roughly speaking, the work for solving a puzzle must be incompressible in the sense that even the best adversary can speed up the finding of a puzzle solution by at most a factor of μ. More formally, a scratch-off puzzle is μ-incompressible (where $\mu \geq 1$) if for any probabilistic poly-nomial-time adversary $\mathcal{A}$ taking at most $t \cdot \underline{t}$ steps,

$$\Pr\left[\begin{array}{l} \mathsf{puz} \leftarrow \mathcal{G}(1^\lambda), \\ (m, \mathsf{ticket}) \leftarrow \mathcal{A}(\mathsf{puz}) : \\ \mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket}) = 1 \end{array}\right] \leq \zeta^+(\mu t) \pm \mathsf{negl}(\lambda).$$

Note that $\zeta^+(t) = 1 - (1 - 2^{-d})^{t+1}$ is roughly the probability of outputting a winning ticket after t unit scratch attempts, though we additionally allow the adversary to make a final guess at the end (as in [47]), and hence the $t+1$ in the exponent instead of just t. Ideally, we would like the compressibility factor μ to be as close to 1 as possible.

[2] The terms "scratch-off puzzle" and "winning ticket" are motivated by the observation that Bitcoin's coin minting process resembles a scratch-off lottery, wherein a participant expends a unit of effort to learn if he holds a winning ticket.

When $\mu = 1$, the honest Work algorithm is the optimal way to solve a puzzle.

This definition implies, in particular, that solutions to previous puzzles do not help in solving a freshly generated puzzle unseen ahead of time.

A. Non-Transferability

For a practical scheme we could integrate into Bitcoin, we should require that the payload of a ticket is non-transferable, in the following sense: if an honest party publishes a ticket attributed to a payload m (e.g., containing a public key belonging to the party to whom the reward must be paid), the adversary should not gain any advantage in obtaining a puzzle solution attributed to some different payload m^* for the same puz. This is because in Bitcoin, each epoch is defined by a globally known, unique puzzle instance puz; at most one winning ticket for puz and a payload message is accepted into the blockchain; and a user who solves a puzzle only receives the reward if their message is the one that is attributed. If an adversary can easily modify a victim's winning ticket to be attributed to a different payload of its choice, then the adversary can listen for when the victim's ticket is first announced in the network, and then immediately start propagating the modified ticket (e.g., containing its own public key for the reward payment) and attempt to outrace the victim. It is possible that the network will now deem the adversary as the winner of this epoch—this is especially true if the adversary has better network connectivity than the victim (as described in [21]). For simplicity in developing our constructions and nonoutsourceable definition, we define this non-transferability requirement separately below. Intuitively, non-transferability means that seeing a puzzle solution output by an honest party does not help noticeably in producing a solution attributed to a different payload m^*.

Definition 2. *Let δ be a nonnegative function of ℓ. A scratch-off puzzle is* δ-non-transferable *if it additionally satisfies the following property:*

For any $\ell = \text{poly}(\lambda)$, and for any adversary $\mathcal{A}$ taking $t \cdot \underline{t}$ steps,

$$\Pr\left[\begin{array}{l} \mathsf{puz} \leftarrow \mathcal{G}(1^\lambda) \\ m_1, m_2, \ldots, m_\ell \leftarrow \mathcal{A}(1^\lambda) \\ \forall i \in [\ell] : \mathsf{ticket}_i \leftarrow \mathsf{WorkTillSuccess}(\mathsf{puz}, m_i), \\ (\mathsf{puz}, m^*, \mathsf{ticket}^*) \leftarrow \mathcal{A}(\mathsf{puz}, \{m_i, \mathsf{ticket}_i\}_{i=1}^{\ell}) : \\ \quad \mathsf{Verify}(\mathsf{puz}, \mathsf{ticket}^*, m^*) \ \wedge \ (\forall i \in [\ell] : m^* \neq m_i) \end{array}\right] \leq \zeta^+((\mu+\delta)t) + \mathsf{negl}(\lambda)$$

IV. Outsourced Mining and Weakly Nonoutsourceable Puzzles

The Bitcoin scratch-off puzzle described in the previous section is amenable to secure outsourcing, in the sense that it is possible for one party (the *worker*) to perform mining work for the benefit of another (the *pool operator*) and to *prove* to the pool operator that the work done can only benefit the pool operator.

To give a specific example, let m be the public key of the pool operator; if the worker performs $2^{d'}$ scratch attempts, on average it will have found at least one value r such that $\mathcal{H}(\mathsf{puz}\|m\|r) < 2^{\lambda - d'}$. The value r can be presented to the pool operator as a "share" (since it represents a portion of the expected work needed to find a solution); intuitively, any such work associated with m cannot be reused for any other $m^* \neq m$. This scheme is an essential component of nearly every Bitcoin mining pool to date [43]; the mining pool operator chooses the payload m, and mining participants are required to present shares associated with m in order to receive participation credit. The rise of large, centralized mining pools is due in large part to the effectiveness of this mechanism.

We now formalize a generalization of this outsourcing protocol, and then proceed to construct puzzles that are *not* amenable to outsourcing (i.e., for which no effective outsourcing protocol exists).

A. Notation and Terminology

Pool operator and Worker. We use the terminology *pool operator* and *worker* referring respectively to the party outsourcing the mining computation and the party performing the mining computation. While this terminology is natural for describing mining pools, we stress that our results are intended to simultaneously discourage both mining pools and hosted mining services. In the case of hosted mining, the roles are roughly swapped; the cloud server performs the mining work, and the individuals who hire the service receive the benefit and must be convinced the work is performed correctly. We use this notation since mining pools are more well-known and widely used today, and therefore we expect the mining-pool oriented terminology to be more familiar and accessible.

Protocol executions. A protocol is defined by two algorithms $\mathcal{S}$ and $\mathcal{C}$, where $\mathcal{S}$ denotes the (honest) worker, and $\mathcal{C}$ the (honest) pool operator. We use the notation $(o_{\mathcal{S}}; o_{\mathcal{C}}) \leftarrow (\mathcal{S}, \mathcal{C})$ to mean that a pair of interactive Turing Machines $\mathcal{S}$ and $\mathcal{C}$ are executed, with $o_{\mathcal{S}}$ the output of $\mathcal{S}$, and $o_{\mathcal{C}}$ the output of $\mathcal{C}$.

In this paper we assume the pool operator executes the protocol program $\mathcal{C}$ correctly, but the worker may deviate arbitrarily.[3] We use the notation $(\mathcal{A}, \mathcal{C})$ to denote an execution between a malicious worker $\mathcal{A}$ and an honest pool operator $\mathcal{C}$. Note that protocol definition always uses the honest algorithms, i.e., $(\mathcal{S}, \mathcal{C})$ denotes a protocol or an honest execution; whereas $(\mathcal{A}, \mathcal{C})$ represents an execution.

B. Definitions

Outsourcing protocol. We now define a generalization of outsourced mining protocols, encompassing both mining pools and hosted mining services. Our definition of outsourcing protocol is broad – it captures any form of protocol where the pool operator and worker may communicate as interactive Turing Machines, and at the end, the pool operator may obtain a winning ticket with some probability. The protocol is parametrized by three parameters $t_{\mathcal{C}}, t_{\mathcal{S}}$, and t_e, which roughly models the pool operator's work, honest worker's work, and the "effective" amount of work during the protocol.

[3] This is without loss of generality, and does not mean that we assume the mining pool operator is honest, since the protocol $(\mathcal{S}, \mathcal{C})$ may deviate from "honest" Bitcoin mining.

Definition 3. *A* $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e)$-outsourcing protocol *for scratch-off puzzle* $(\mathcal{G}, \mathsf{Work}, \mathsf{Verify})$*, where* $t_e < t_S + t_C$ *and* $t_c < t_e$*, is a two-party protocol,* $(\mathcal{S}, \mathcal{C})$*, such that*

- *The pool operator's input is* puz*, and the worker's input is* $\perp$*.*
- *The pool operator* $\mathcal{C}$ *runs in at most* $t_{\mathcal{C}} \cdot \underline{t}$ *time, and the worker* $\mathcal{S}$ *in at most* $t_{\mathcal{S}} \cdot \underline{t}$ *time.*
- $\mathcal{C}$ *outputs a tuple* (ticket, m) *at the end, where* ticket *is either a winning ticket for payload* m *or* $\mathsf{ticket} = \perp$*. Further, when interacting with an honest* $\mathcal{S}$*,* $\mathcal{C}$ *outputs a* $\mathsf{ticket} \neq \perp$ *with probability at least* $\zeta(t_e) - \mathsf{negl}(\lambda)$*.*

Formally,

$$\Pr\left[\begin{array}{l}\mathsf{puz} \leftarrow \mathcal{G}(1^\lambda)\\ (\cdot; \mathsf{ticket}, m) \leftarrow (\mathcal{S}, \mathcal{C}(\mathsf{puz})) :\\ \mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket})\end{array}\right] \geq \zeta(t_e) - \mathsf{negl}(\lambda).$$

The parameter t_e is referred to as the *effective billable work*, because the protocol $(\mathcal{S}, \mathcal{C})$ has the success probability of performing t_e unit scratch attempts. Note that it must be the case that $t_e < \mu(t_{\mathcal{S}} + t_{\mathcal{C}})$. Intuitively, an outsourcing protocol allows effective outsourcing of work by the pool operator if $t_e \gg t_{\mathcal{C}}$.

Note that this definition does not specify how the payload m is chosen. In typical Bitcoin mining pools, the pool operator chooses m so that it contains the pool operator's public key. However, our definition also includes schemes where m is jointly computed during interaction between $\mathcal{S}$ and $\mathcal{C}$, for example.

Weak nonoutsourceability. So far, we have formally defined what an outsourcing protocol is. Roughly speaking, an outsourcing protocol generally captures any possible form of contractual agreement between the pool operator and the worker. The outsource protocol defines exactly what the worker has promised to do for the pool operator, i.e., the "honest" worker behavior. If a worker is malicious, it need not follow this honest prescribed behavior. The notion of weak non-outsourceability requires that no matter what the prescribed contractual agreement is between the pool operator and the worker– as long as this agreement "effectively" outsources work to the worker– there exists an adversarial worker that can always steal the pool operator's ticket should the pool operator find a winning ticket during the protocol. Effectiveness is intuitively captured by how much effective work the worker performs vs. the work performed by the pool operator in the honest protocol. Note that there always exists a trivial, ineffective outsourcing protocol, where the pool operator always performs all the work by itself – in this case, a malicious worker will not be able to steal the ticket. Therefore, the weak non-outsourceability definition is parametrized by the effectiveness of the honest outsourcing protocol.

More specifically, the definition says that the adversarial worker can generate a winning ticket associated with a payload of its own choice, over which the pool operator has no influence. In a Bitcoin-like application, a natural choice is for an adversarial worker to replace the payload with a public key it owns (potentially a pseudo-nym), such that it can later spend the stolen awards. Based on this intuition, we now formally define the notion of a *weakly nonoutsourceable* scratch-off puzzle.

Definition 4. *A scratch-off-puzzle is* $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e, \alpha, p_s)$-weakly nonoutsourceable *if for every* $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e)$*-outsourcing protocol* $(\mathcal{S}, \mathcal{C})$*, there exists an adversary* $\mathcal{A}$ *that runs in time at most* $t_{\mathcal{S}} \cdot \underline{t} + \alpha$*, such that:*

- *Let* $m^* \xleftarrow{\$} \{0,1\}^\lambda$*. Then, at the end of an execution* $(\mathcal{A}(\mathsf{puz}, m^*), \mathcal{C}(\mathsf{puz}))$*, the probability that* $\mathcal{A}$ *outputs a winning ticket for payload* m^* *is at least* $p_s\zeta(t_e)$*. Formally,*

$$\Pr\left[\begin{array}{l}\mathsf{puz} \leftarrow \mathcal{G}(1^\lambda); m^* \xleftarrow{\$} \{0,1\}^\lambda\\ (\mathsf{ticket}^*; \mathsf{ticket}, m) \leftarrow (\mathcal{A}(\mathsf{puz}, m^*), \mathcal{C}(\mathsf{puz})) :\\ \mathsf{Verify}(\mathsf{puz}, \mathsf{ticket}^*, m^*)\end{array}\right] \geq p_s\zeta(t_e).$$

- *Let* view_h *denote the pool operator's view in an execution with the honest worker* $(\mathcal{S}, \mathcal{C}(\mathsf{puz}))$*, and let* view^* *denote the pool operator's view in an execution with the adversary* $(\mathcal{A}(\mathsf{puz}, m^*), \mathcal{C}(\mathsf{puz}))$*. Then,*

$$\mathsf{view}^* \stackrel{c}{\equiv} \mathsf{view}_h.$$

When $\mathcal{C}$ *interacts with* $\mathcal{A}$*, the view of the pool operator* view^* *is computationally indistinguishable from when interacting with an honest* $\mathcal{S}$*.*

Later, when proving that puzzles are weakly nonoutsourceable, we typically construct an adversary $\mathcal{A}$ that runs the honest protocol $\mathcal{S}$ until it finds a ticket for m, and then transforms the ticket into one for m^* with probability p_s. For this reason, we refer to the adversary $\mathcal{A}$ in the above definition as a *stealing adversary* for protocol $(\mathcal{S}, \mathcal{C})$. In practice, we would like α to be small, and $p_s \leq 1$ to be large, i.e., $\mathcal{A}$'s run-time is not much different from that of the honest worker, but $\mathcal{A}$ can steal a ticket with high probability.

If the pool operator outputs a valid ticket for m *and* the worker outputs a valid ticket for m^*, then there is a race to determine which ticket is accepted by the Bitcoin network and earns a reward. Since the μ-incompressibility of the scratch-off puzzle guarantees the probability of generating a winning ticket associated with either m or m^* is bounded above by $\zeta^+(\mu(t_{\mathcal{S}} + t_{\mathcal{C}}))$, the probability of the pool operator outputting a ticket — but not the worker — is bounded above by $\zeta^+(\mu(t_{\mathcal{S}} + t_{\mathcal{C}})) - p_s\zeta(t_e)$.

Note that weak nonoutsourceability does not imply that the puzzle is transferable. In other words, a puzzle can be simultaneously non-transferable and weakly nonoutsourceable. This is so because the stealing adversary $\mathcal{A}$ may rely on its view of the entire outsourcing protocol when stealing the ticket for its own payload m^*, whereas the adversary for the non-transferability game is only given winning tickets as input (but no protocol views).

As we mentioned in the beginning of this section, the prevalence of Bitcoin mining pools can be attributed in part to the effective outsourcing protocol used to coordinate untrusted pool members - in other words, the Bitcoin puzzle is not nonoutsourceable. We state and prove a theorem to this effect in Appendix B.

V. A Weakly Nonoutsourceable Puzzle

In this section, we describe a weakly nonoutsourceable construction based on a Merkle-hash tree construction. We prove that our construction satisfies weak nonoutsourceability (for a reasonable choice of parameters) in the random oracle model. Informally, our construction achieves the following:

> For *any* outsourcing protocol that can effectively outsource a fixed constant fraction of the effective work, an adversarial worker will be able to steal the puzzle with at least constant probability.

Our construction is inspired by the Floating Preimage Signature (FPS) scheme used in Permacoin [33], which is a puzzle integrated with a proof-of-retrievablity. However, Permacoin [33] only described the issue of nonoutsourceability informally, and made no attempt to formalize the definition nor to discuss nonoutsourceability beyond the context of archival storage. Our construction is formally defined in Figure 4, but here we provide an informal explanation of the intuition behind it.

Intuition. To solve a puzzle, a node first builds a Merkle tree with random values at the leaves; denote the root by digest. Then the node repeatedly samples a random value r, computes $h = \mathcal{H}(\mathsf{puz}||r||\mathsf{digest})$, and uses h to select q leaves of the Merkle tree and their corresponding branches (i.e., the corresponding Merkle proofs). It then hashes those branches (along with puz and r) and checks to see if the result is less than $2^{\lambda-d}$.

Once successful, the node has a value r what was "difficult" to find, but is not yet bound to the payload message m. To effect such binding, a "signing step" is performed in which $h' = \mathcal{H}(\mathsf{puz}||m||\mathsf{digest})$ is used to select a set of $4q'$ leaf nodes (i.e., using h' a seed to a pseudorandom number generator). Any q' of these leaves, along with their corresponding branches, constitute a signature for m and complete a winning ticket.

Intuitively, this puzzle is weakly nonoutsourceable because in order for the worker to perform scratch attempts, it must

- either know a large fraction of the leaves and branches of the Merkle tree, in which case it will be able to sign an arbitrary payload m^* with high probability – by revealing q' out of the $4q'$ leaves (and their corresponding branches) selected by m^*,
- or incur a large amount of overhead, due to aborting scratch attempts for which it does not know the necessary leaves and branches,
- or interact with the pool operator frequently, in which case the pool operator performs a significant fraction of the total number of random oracle queries.

To formally prove this construction is weakly nonoutsourceable, we assume that the cost of the Work algorithm is dominated by calls made to random oracles. Thus, for simplicity, in the following theorems we equate the running time with the number of calls to the random oracle. However, the theorem can be easily generalized (i.e., relaxing by a constant factor) as long as the cost of the rest of the computation is only a constant fraction of the random-oracle calls.

Theorem 1. *The construction in Figure 4 is a scratch-off puzzle.*

We defer this proof to the Appendix.

Theorem 2. *Let $q, q' = O(\lambda)$. Let the number of leaves $L \geq q + 8q'$. Suppose $d > 10$ and $t_e \cdot 2^{-d} < 1/2$. Under the aforementioned cost model, the above construction is a $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e, \alpha, p_s)$ weakly nonoutsourceable puzzle, for any $0 < \gamma < 1$ s.t. $t_{\mathcal{C}} < \gamma t_e$, $p_s > \frac{1}{2}(1-\gamma) - \mathsf{negl}(\lambda)$, and $\alpha = O(\lambda^2)$; and is 0-non-transferable.*

In other words, if the pool operator's work $t_{\mathcal{C}}$ is a not a significant fraction of t_e, i.e., work is effectively outsourced, then an adversarial worker will be able to steal the pool operator's ticket with a reasonably big probability, and without too much additional work than the honest worker.

The proof that this puzzle is weakly nonoutsourceable can be found in the Appendix, but we sketch the main idea here. Informally, to "effectively" outsource work to the worker, the worker must know more than a constant fraction (say, $1/3$) of the leaves before calling the random oracle to determine whether an attempt is successful. However, if the worker knows more than $1/3$ fraction of the leaves, due to a simple Chernoff bound, it will be able to easily steal the solution should one be found. To make this argument formally is more intricate. The proof that this puzzle is non-transferable is deferred to the appendix.

VI. Strongly Nonoutsourceable Puzzles

In the previous section, we formally defined and constructed a scheme for weakly nonoutsourceable puzzles, which ensure that for any "effective" outsourcing protocol, there exists an adversarial worker that can steal the pool operator's winning ticket with significant probability, should a winning ticket be found. This can help deter outsourcing when individuals are expected to behave selfishly.

One critical drawback of the weakly nonoutsourceable scheme (and, indeed, of Permacoin [33]) is that a stealing adversary may be detected when he spends his stolen reward, and thus might be held accountable through some external means, such as legal prosecution or a tainted public reputation.

For example, a simple detection mechanism would be for the pool operator and worker to agree on a $\lambda/2$-bit prefix of the nonce space to serve as a watermark. The worker can mine by randomly choosing the remaining $\lambda/2$-bit suffix, but the pool operator only *accepts* evidence of mining work bearing this watermark. If the worker publishes a stolen puzzle solution, the watermark would be easily detectable.

Ideally, we should enable the stealing adversary to evade detection and leave no incriminating trail of evidence. Therefore, in this section, we define a "strongly nonoutsourceable" puzzle, which has the additional requirement that a stolen ticket cannot be distinguished from a ticket produced through independent effort.

Let NIZK be a non-interactive zero-knowledge proof system. Also assume that $\mathcal{E} = (\mathsf{Key}, \mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure public-key encryption scheme.

Let $(\mathcal{G}', \mathsf{Work}', \mathsf{Verify}')$ be a weakly nonoutsourceable scratch-off puzzle scheme. We now construct a strongly nonoutsourceable puzzle scheme as below.

- $\mathcal{G}(1^\lambda)$: Run the puzzle generation of the underlying scheme $\mathsf{puz}' \leftarrow \mathcal{G}'(1^\lambda)$. Let $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$; and let $(\mathsf{sk}_\mathcal{E}, \mathsf{pk}_\mathcal{E}) \leftarrow \mathcal{E}.\mathsf{Key}(1^\lambda)$. Output $\mathsf{puz} \leftarrow (\mathsf{puz}', \mathsf{crs}, \mathsf{pk}_\mathcal{E})$
- $\mathsf{Work}(\mathsf{puz}, m, t)$:
  Parse $\mathsf{puz} := (\mathsf{puz}', \mathsf{crs}, \mathsf{pk}_\mathcal{E})$.
  $\mathsf{ticket}' \leftarrow \mathsf{Work}'(\mathsf{puz}', m, t)$,
  Encrypt $c \leftarrow \mathsf{Enc}(\mathsf{pk}_\mathcal{E}; \mathsf{ticket}'; \mathsf{r})$.
  Set $\pi \leftarrow \mathsf{NIZK.Prove}(\mathsf{crs}, (c, m, \mathsf{pk}_\mathcal{E}, \mathsf{puz}'), (\mathsf{ticket}', r))$
  for the following NP statement:
  $\quad \mathsf{Verify}'(\mathsf{puz}', m, \mathsf{ticket}') \wedge c = \mathsf{Enc}(\mathsf{pk}_\mathcal{E}; \mathsf{ticket}'; \mathsf{r})$
  Return $\mathsf{ticket} := (c, \pi)$.
- $\mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket})$;
  Parse $\mathsf{puz} := (\mathsf{puz}', \mathsf{crs}, \mathsf{pk}_\mathcal{E})$, and parse ticket as (c, π).
  Check that $\mathsf{Verify}(\mathsf{crs}, (c, m, \mathsf{pk}_\mathcal{E}, \mathsf{puz}'), \pi) = 1$.

Fig. 1: A generic transformation from any *weakly nonoutsourceable* scratch-off puzzle to a *strongly nonoutsourceable* puzzle.

Definition 5. *A puzzle is* $(t_\mathcal{S}, t_\mathcal{C}, t_e, \alpha, p_s)$-strongly nonoutsourceable *if it is* $(t_\mathcal{S}, t_\mathcal{C}, t_e, \alpha, p_s)$*-weakly nonoutsourceable, and additionally the following holds:*

For any $(t_\mathcal{S}, t_\mathcal{C}, t_e)$*-outsourcing protocol* $(\mathcal{S}, \mathcal{C})$*, there exists an adversary* $\mathcal{A}$ *for the protocol such that the stolen ticket output by* $\mathcal{A}$ *for payload* m^* *is computationally indistinguishable from a honestly computed ticket for* m^**, even given the pool operator's view in the execution* $(\mathcal{A}, \mathcal{C})$*. Formally, let* $\mathsf{puz} \leftarrow \mathcal{G}(1^\lambda)$*, let* $m^* \xleftarrow{\$} \{0,1\}^\lambda$*. Consider a protocol execution* $(\mathcal{A}(\mathsf{puz}, m^*), \mathcal{C}(\mathsf{puz}))$*: let* view^* *denote the pool operator* $\mathcal{C}$*'s view and* ticket^* *the stolen ticket output by* $\mathcal{A}$ *in the execution. Let* ticket_h *denote an honestly generated ticket for* m^**, (*$\mathsf{ticket}_h := \mathsf{WorkTillSuccess}(\mathsf{puz}, m^*)$*), and let* view_h *denote the pool operator's view in the execution* $(\mathcal{S}, \mathcal{C}(\mathsf{puz}))$*. Then,*

$$(\mathsf{view}^*, \mathsf{ticket}^*) \stackrel{c}{\equiv} (\mathsf{view}_h, \mathsf{ticket}_h)$$

Recall that in Bitcoin, the message payload m typically contains a Merkle root hash representing a set of new transactions to commit to the blockchain in this round, including the public key to which the reward is assigned. Thus to take advantage of the strongly nonoutsourceable puzzle, the stealing worker should bind its substituted payload m^* to a *freshly generated* public key for which it knows the corresponding private key. It can then spend its stolen reward anonymously, for example by laundering the coins through a mixer [9].

In Figure 1, we present a generic transformation that turns any weakly nonoutsourceable puzzle into a strongly nonoutsourceable puzzle. The strengthened puzzle is essentially a *zero-knowledge* extension of the original – a ticket for the strong puzzle is effectively a proof of the statement "I know a solution to the underlying puzzle."

Theorem 3. *If* $(\mathsf{GenKey}', \mathsf{Work}', \mathsf{Verify}')$ *is a* $(t_\mathcal{S}, t_\mathcal{C}, t_e, \alpha, p_s)$ *weakly nonoutsourceable puzzle, then the puzzle described in Figure 1 is a* $(t_\mathcal{S}, t_\mathcal{C}, t_e, \alpha + t_{enc} + t_{NIZK}, p_s - \mathsf{negl}(\lambda))$ *strongly nonoutsourceable puzzle, where* $t_{enc} + t_{NIZK}$ *is the maximum time required to compute the encryption and NIZK in the honest* Work *algorithm.*

We next state a theorem that this generic transformation essentially preserves the non-transferability of the underlying puzzle.

Theorem 4. *If the underlying puzzle* $(\mathcal{G}', \mathsf{Work}', \mathsf{Verify}')$ *is* δ'*-non-transferable, then the derived puzzle through the generic transformation is* δ *non-transferable for*

$$\mu + \delta' \leq \frac{(\mu + \delta)t}{t \cdot \underline{t} + (t_{enc} + t_{nizk})\ell}$$

where t_{enc} *and* t_{nizk} *are the time for performing each encryption and NIZK proof respectively.*

Again, the proof of this theorem is deferred to the appendix.

Cheap plaintext option. Although we have shown it is plausible for a stealing worker (with parallel resources) to compute the zero-knowledge proofs, this would place an undue burden on honest independent miners. However, it is possible to modify our generic transformation so that there are *two* ways to claim a ticket: the first is with a zero-knowledge proof as described, while the second is simply by revealing a plaintext winning ticket for the underlying weakly nonoutsourceable puzzle.

VII. Implementation and Micro-benchmarks

In order to demonstrate the practicality of our schemes, we implemented both our weakly nonoutsourceable and strongly nonoutsourceable puzzle schemes and provide benchmark results below.

Metrics. We are concerned with two main performance criteria. First, the size of a ticket and cost of verifying a ticket should be minimal, since each participant on the network is expected to verify every ticket independently. Second, in order for our scheme to be an effective deterrent, the cost and *latency* required to "steal" a ticket should be low enough that it is at least *plausible* for an outsourcing worker to compute a stolen ticket *and* propagate it throughout the network before any other solution is found.

When comparing the verification cost of our schemes to that of the current Bitcoin protocol, we include both the cost due to the puzzle itself, as well as the total cost of validating a block including transactions. At present, there are over 400 transactions per block on average; [4] we assume each transaction carries at least 1 ECDSA signature that must be verified. In general, the computational cost of validating blocks in Bitcoin is largely dominated by verifying the ECDSA signatures in transactions rather than verifying puzzle solutions. We measured that the time to verify an ECDSA signature on a 2.4GHZ Intel CPU is 1.7ms.[5] On average, at the time of

[4] Average number of transactions per block: https://blockchain.info/charts/n-transactions-per-block

[5] Unless otherwise noted, we conducted our measurements over at least 1000 trials, and omit the error statistic if the standard deviation is within $\pm 1\%$.

TABLE I: Estimated puzzle and block verification costs for various schemes

	Puzzle only				Transactions Included			
Scheme	Verif.	Ratio	Size	Ratio	Verif.	**Ratio**	Size	**Ratio**
Bitcoin	11.7μs	1	80B	1	0.68s	**1**	350KB	**1**
Weak	15.1ms	1.3e3	6.6KB	83	0.70s	**1.02**	357KB	**1.02**
Strong ($C = 4$)	0.48s	4.1e+04	10.8KB	135	1.16s	1.71	361KB	**1.03**
Strong ($C = 3$)	0.62s	5.3e+04	17.0KB	213	1.30s	1.91	367KB	**1.05**
Strong ($C = 2$)	0.93s	8e+04	23.2KB	290	1.61s	2.37	373KB	**1.07**
Strong ($C = 1$)	1.68s	1.4e+05	29.4KB	368	2.36s	3.47	379KB	**1.08**

writing, a block contains about 350 kilobytes of data and 600 transactions, each with an average of two signatures. [6]

A. Our Weakly Nonoutsourceable Puzzle

The weakly nonoutsourceable puzzle is straightforward to implement, and its overhead relative to the Bitcoin puzzle consists only of $\lambda \log \lambda$ additional hashes; we implemented this in unoptimized Python and discuss its performance later on. In contrast, the strongly nonoutsourceable puzzle requires much more care in implementation due to the NIZK proof, which we will describe shortly.

We used the SHA-1 hash function throughout our implementation, since this has a relatively efficient implementation as an arithmetic circuit [40]. We restricted our focus to the following puzzle parameters: the signature tree consists of $2^h = 2^{10}$ leaves, and the number of leaves revealed during a scratch attempt and a claim is $q = q' = 10$. This provides roughly 50 bits of security for the non-transferability property.

Performance results. In Table I (first and second rows), we show that if we replace Bitcoin's puzzle with our weakly nonoutsourceable puzzle, the slowdown for the block verification operation will be only **2%**. More specifically, while our puzzle verification itself is over a thousand times more expensive than the Bitcoin puzzle, puzzle verification only accounts for a very small percentage of the overall verifier time. Therefore, the overall performance slowdown is insignificant for practical purposes. Likewise, while the size of the ticket in our scheme is almost a hundred times larger than that of Bitcoin, the ticket is a small fraction of the total size of a block when transactions are included.

An adversarial worker can steal a ticket in a marginal amount of time (only one additional hash in expectation, for example, assuming the worker knows at least a third of the Merkle tree branches used during scratch attempts). This cost is insignificant compared to the expected time for solving a puzzle.

B. Our Strongly Nonoutsourceable Puzzle

We next describe more details of our instantiation and implementation of our strongly nonoutsourceable puzzle, followed by evaluation.

We implemented our puzzle twice, each time using a different NIZK libraries: Pinocchio [40] and Libsnark [6]. Both are implementations of a generic [23] NIZK scheme. [7] Pinocchio includes a compiler that generates an arithmetic circuit from high-level C code, while Libsnark [6] provides a library C++ for composing systems of equations. We used a combination of hand-tuned and generated-from-C-code arithmetic circuits, and developed an adapter for Libsnark to use Pinocchio's arithmetic circuit files.

In Appendix E we discuss a concrete parameterization of our scheme. We implemented an optimization to improve the parallel running time of the prover. Essentially, we break the overall statement into many substatements, all of which can be proven concurrently; the overall proof consists of a proof for one "Type II" statement, and proofs for some number of "Type I" statements. The number of Type I statements is determined by a parameter C (smaller C means a larger number of smaller circuits). We discuss this in more detail in the Appendix.

Performance results. The prover and verifier costs for our strongly nonoutsourceable implementation are presented in Tables Iand II. Each of the bottom four rows of Table I and top four rows of Table II corresponds to a different setting of the parameter C, the number of 160-bit blocks (of the underlying ticket) checked by each substatement (smaller values of C indicate higher degrees of parallelism). The total number of substatements required (#) is reported along with computing time *per circuit* for the prover and verifier. We also report the total verification time over all the statements, as well as the total proof size. Note that our benchmarks are for a sequential verifier, although verification could also be parallelized. The bottom row is for the second type of statement, which does not depend on C. Due to the longer time required to compute these proofs, the quantities reported are averaged over only three trials.

The reader may immediately notice the vast improvement in prover performance using Libsnark rather than Pinocchio for our implementation; in particular the speedup is much greater than previous reports (i.e, several orders of magnitude vs one order of magnitude) [6]. This is readily explained with reference to the highly sequential nature of our statements, which yields deep and highly-connected circuits. Profiling reveals that the cost of generating our proofs in Pinocchio is dominated by the polynomial interpolation step, which greatly exceeds that of simpler circuits with comparable number of gates [40].

[6] Average block size: https://blockchain.info/charts/

[7] Libsnark [6] implements several optimizations over the original GGPR [23] scheme. The version we used includes an optimization that turns out to be unsound. [39] Libsnark has since been patched to restore soundness; the patch is reported only to incur an overhead of 0.007% on typical circuits, hence we report our original figures.

Keeping in mind our goal is to prove it is *plausible* for a worker to produce stolen ticket proof with low latency, we believe it is reasonable to assume that such a worker has access to parallel computing resources. Using Libsnark, the combination of our statement-level parallelism and the parallel SNARK implementation leads to proof times in **under 15 seconds** at the $C = 2$ setting. Since the average time between puzzle solutions in the Bitcoin network is 10 minutes, this can be a wholly plausible deterrent. At this setting, verification of an entire proof takes under one second. Since approximately 144 Bitcoin puzzle solutions are produced each day, it would take approximately two minutes for a single-threaded verifier to validate a day's worth of puzzle solutions.

Assuming computational power can be rented at $1.68 per hour (based on Amazon EC2 prices for the c3.8xlarge used in our trials, which provides 32 cores), it would cost an attacker less than $10 in total to produce a stolen ticket proof within 20 seconds. This is vastly less than reward for a puzzle solution, which at the current time is approximately $8,750.

C. Cryptocurrency Integration

We now discuss several practical aspects of integrating nonoutsourceables puzzles within existing cryptocurrency designs.

Integrating the puzzle with Bitcoin-like cryptocurrencies. In our definitions, we indicate that $\mathsf{GenPuz}(1^\lambda)$ must be a random function that generates a puzzle instance, and in all of our schemes $\mathsf{GenPuz}(1^\lambda)$ simply returns a uniform random string. However, in the actual Bitcoin protocol, the next puzzle instance is generated by applying a hash function to the solution of the previous puzzle. Our approach is likewise to determine each next puzzle instance from the hash of the previous solution and message, $\mathsf{puz}' := \mathcal{H}(\mathsf{puz} \| m \| \mathsf{ticket})$.

Further Integration Issues. In the Appendix, we further discuss how our nonoutsourceable puzzles can be combined with other proposals for complementary properties, such as faster blocks [30], [45], support for lightweight mobile clients [36], and either ASIC-resistance [48] or backward-compatibility with existing mining equipment [20].

VIII. Multi-Tier Block Rewards

We want to arrive at a cryptocurrency design that simultaneously discourages centralized mining pools and hosted mining services, yet encourages participation from individual miners and provides similar overall functionality and security as Bitcoin today. To achieve this, there are two major remaining challenges.

Challenge 1: Lower variance rewards for individual miners. Individual miners should not have to wait an unreasonable amount of time to earn a Bitcoin reward. Intuitively, we can achieve this by decreasing the average time between blocks, so that rewards are given out much more frequently. We are constrained, however, by the latency of network propagation, and the time it takes to compute the zero knowledge proofs used in the strongly nonoutsourceable puzzle.

Challenge 2: Discourage statistical enforcement over time. Our definition of nonoutsourceable puzzles essentially describes a one-shot game, and ensures that the worker can steal a single puzzle solution from the pool operator and evade detection. However, this definition does not immediately eliminate statistical enforcement techniques over time. For example, pool operators could monitor the output of a hosted service provider and punish it (e.g., through legal prosecution) if it underperforms significantly. Intuitively, we should address this by giving out larger rewards much less frquently, so that the worker can steal solutions over some reasonable timeframe and plausibly claim it was just unlucky.

Conflicting requirements. What we need is a reward structure that simultaneously answers the above challenges. Challenge 1 desires paying out small rewards rapidly, whereas Challenge 2 clearly favors paying out large rewards less frequently. Further, to satisfy Challenge 1, if we reduce the inter-block time to the order of seconds, we phase another challenge: since it takes at least 14 seconds to generate the zero-knowledge proof, it would be infeasible for a miner to steal a block this way. The miner could choose to steal the reward using the plaintext option, but since the reward at stake is low, the mining pool could require a small collateral deposit to discourage such blatant stealing.

A. Proposed Multi-Tier Reward Structure

We propose to satisfy both of these properties by designing a reward structure with multiple possible prizes. Our multi-tier design is inspired by the payoff structure of state lottery games, which often have several consolation prizes as well as large, less frequent jackpots [38], [42]. The effectiveness of such lotteries at encouraging wide participation has long been proven in practice. Our implicit assumption is that miners will tolerate a high-variance payoff overall, as long as they earn *some* reward fairly frequently.

In Table III, we provide a concrete example of such a multi-tier reward schedule, the rationale for which we discuss below. Each attempt at solving a puzzle yields some chance of winning each of three possible prizes (in contrast with Bitcoin today, in which every block earns the same reward). The prizes are not only associated with different reward values, but also count with varying weight towards the blockchain "difficulty" scoring function. The first two columns indicate the average time between rewards of a given type, along with their relative frequency (adding up to 1). The middle two columns indicate the prize value (in btc), along with the realtive contribution to the total expected reward value (adding up to 1). Note that the overall expected payout rate is the same as in Bitcoin today (25btc every 10 minutes, on average). The final pair of columns indicates the blockchain difficulty weight associated with each reward type along with their expected relative contributions towards the overall difficulty of a blockchain (again, adding up to 1).

Low-value consolation prize: provides low-variance rewards to solo-miners. The consolation prize is awarded the most frequently (e.g., once every three seconds). The prize is small, less than a tenth of a Bitcoin (≈$21 at the time of writing), but the small prizes contribute overall to 70% of the total expected value.

TABLE II: Proof and verification micro-benchmarks for strongly nonoutsourceable puzzles

Type I Statements										
			Pinocchio [40]			Libsnark [6]				
C	#	Gates	Prove	Verify	Total	Prove (Single-core)	Prove (Multi-core)	Verify	Total	Size
1	220	213k	268.2s	11ms	2.42s	16.33s	9.84s	7.6ms	1.672s	29.4KB
2	120	280k	578.4s	11ms	1.32s	20.29s	**13.90s**	7.7ms	**0.924s**	23.2KB
3	80	392k	1002.9s	11ms	0.80s	26.92s	17.18s	7.7ms	0.616s	17.0KB
4	60	467k	1242.1s	11ms	0.66s	32.88s	20.71s	7.8ms	0.468s	10.8KB
Type II Statements										
	#	Gates	Prove	Verify	Total	Prove (Single-core)	Prove (Multi-core)	Verify	Total	Size
	1	282K	508.5s	10ms	0.01s	19.42s	**13.34s**	7.8ms	0.008s	<1KB

Medium-value main prize: ensures block confirmations arrive regularly. The medium-value prizes are necessary to ensure that the log of transactions approximately as quickly and securely as in Bitcoin. They are given out at the same rate as ordinary Bitcoin blocks; they carry a larger reward than the low-value prize, but contribute much less to the overall expected payout. However, the medium-value blocks account for nearly 75% of the total difficulty, and a miner who finds one of these blocks has an average of 7.5 minutes to propagate her solution before it would become stale. The 14 seconds it takes to compute a zero-knowledge proof is relatively small in comparison.

High-value jackpot: defends against statistical detection of cheating hosted mining services. The jackpot prize is very rare, and accounts for a small, yet *disproportionately large* fraction of the total expected value. The role of this reward is to engender distrust of hosted mining providers; they would profit greatly by stealing these rewards, but it would be hard to obtain statistical evidence that they have done so.

Implementing Multi-Tier Rewards. It is straightforward to implement multi-tier rewards on top of any known scratch-off puzzle constructions (i.e., Bitcoin or our nonoutsourceable constructions). Recall that in these constructions, the critical step of a mnining attempt is to compare a hash value to a threshold, $\mathcal{H}(\mathsf{puz}\|\mathsf{ticket}) \stackrel{?}{<} T$, where the threshold $T = 2^{\lambda - d}$ is parameterized by the difficulty d. To implement three reward tiers, we introduce two additional thresholds, $T_{\mathtt{medium}} < T_{\mathtt{high}} < T$. If the hash value lies between $T_{\mathtt{high}}$ and T, then this attempt earns a high-value reward; if it is between $T_{\mathtt{medium}}$ and $T_{\mathtt{high}}$ it earns a medium-value reward; and otherwise it earns a low-value reward. These thresholds must be set according to the desired frequency of each reward type.

B. Economic Analysis of Multi-Tier Rewards

We argue that our proposed reward structure would simultaneously satisfy the necessary properties.

First, our scheme offers small payoff variance. At the time of writing, the overall Bitcoin hashpower is over $3.5 \cdot 10^{17}$ hashes per second. The most cost effective entry-level Bitcoin ASIC we know of is the $8.0 \cdot 10^{14}$ ASICMiner BE Tube, which costs $320. Using this device to solo-mine, the expected time to find a block would be over 8.3 years. However, under our proposed scheme and typical parameters, over a 60 day period, the mining rig mentioned earlier has a better than 98% chance of winning at least one of these prizes.

TABLE III: **Reward schedules for Bitcoin & our scheme.** We give a typical parametrization for the multi-tier reward structure. Parameters can be tuned based on different scenarios.

	Time	Freq	Prize (btc)	(rel)	Weight	(rel)
Bitcoin	10m	1	25	1	1	1
Low	3s	.995	8.8E-2	0.7	1	0.2499
Med	10m	.005	5	0.2	600	0.7497
High	3mo	3.8E-7	3.3E4	0.1	1800	3.4E-4

Next we argue that the payoff structure is also effective at preventing temporal statistical detection. Suppose a large hosted mining provider controls 25% of the network hashpower. Under the original Bitcoin reward structure, it should expect to mine 6574 blocks during a six-month period, and the chance of it mining fewer than 6429 blocks ($\approx 98\%$) is less than one in a thousand. Hence, it could expect steal at most 145 puzzle solutions (worth $900,000, at today's price) over this time period before being implicated with high confidence – and even less before generating considerable suspicion. On the other hand, under our proposed scheme, even if the service provider is honest, it has a better than 60% chance of failing to find *any* jackpot during the same time period. Thus if it *does* steal one, it would arouse no suspicion, yet the expected value of this strategy is over $4.1 million USD.

Finally, we explain that as in the strawman scheme, it is plausible that if miners joined pools requiring small collateral deposits that they would prefer not to defect when they find low-value blocks. However, since the low-value blocks account for only 25% of the blockchain's total difficulty, even if a coercer influences all the transactions in these blocks, this would be insufficient to enforce a blacklist policy, for example. The difficulty weight of the main prize is high enough to provide ample time to steal the puzzle solution (7.5 minutes) and adequate incentive to do so (thousands of dollars worth). Since these blocks account for 75% of the overall difficulty, pool members would be encouraged at least to steal these blocks. Note that the jackpot blocks count more towards the blockchain weight than the other blocks (so that a miner who finds a jackpot block has a long time window (45 minutes on average) before it becomes stale), but contributes very little to the total difficulty of a chain. This prevents an attacker from revising a large span of history by finding a single jackpot block.

IX. Discussion

We have proposed a technical countermeasure against the consolidation of mining power that threatens the decentral-

ization of Bitcoin and other cryptocurrencies. Although we have presented a formal definition that captures the security guarantees of our construction and described how it can be practically integrated into a cryptocurrency, due to the difficulty involved in modifying an in-use cryptocurrency (i.e., via a "hard fork" upgrade [8]) and the high stakes involved in cryptocurrencies generally, the bar for adopting a new design is set very high. Our work provides a significant step in this direction by providing a sound and practical approach to discouraging centralization. However, in order for our solution to be deployed we must provide a thorough and compelling argument that this solution is *fully effective, preferable to all alternatives*, and *does not conflict with other aspects of the system*. Towards this end, we address several typical objections we have encountered in the past, from academic reviewers and the Bitcoin community alike:

"Mining pools are good because they lower the variance for solo miners. Therefore, nonoutsourceable puzzles are not well-motivated." In Section I we describe the severe consequences that can occur due to the concentration of mining power — basically all purported security properties of decentralized cryptocurrencies can be broken if mining coalitions with significant mining power misbehave (and in some cases it may be in their best interest to misbehave [21]). Further, our multi-tier reward system design (see Section VIII-B) achieves the best of both worlds, (i.e., ensuring low variance for solo mining as well as discouraging mining coalitions).

"Can miners still use smart contracts or legal contracts to enforce mining coalitions in spite of the nonoutsourceable puzzles?" Our definition of nonoutsourceable puzzles prevents the enforcement of contractual mechanisms *including* smart contracts or legal contracts. An enforcement mechanism, such as seizing collateral deposits or legal prosecution, is only effective if it can be applied with few false positives. The worker can steal the puzzle solutions without being held accountable, since the zero-knowledge spending option ensures the worker can spend stolen coins without revealing any evidence that can later be used to implicate it.

"What about collecting statistical evidence cheating workers?" Suppose a pool operator monitors the puzzle production rate of a worker over time, to detect if the worker is potentially cheating. One enforcement mechanism might be for the pool to require that a worker submit a deposit to join, such that in case the worker is not producing solutions at the expected rate, the deposit can be confiscated and redistributed.

As mentioned earlier in Section VIII-B, our puzzle definition is by nature a one-shot game. Although our nonoutsourceable puzzle alone does not prevent the collection of statistical evidence, in Section VIII-B, we argued that *by combining our puzzle with a multi-tier reward system, we effectively make it highly costly or unreliable to accumulate statistical evidence over time.* In particular, a worker can opt to steal only the "jackpot prize" (which happens only infrequently but offers a large reward), while behaving honestly when it finds a "consolation prize" which is of much smaller amount but paid off at a frequent interval. Such an attack cannot be reliably detected within a reasonable of timeframe (e.g., several years).

"Can coalitions be prevented by other, simpler solutions that do not require zero knowledge proofs?" The Bitcoin community has put forth two main alternative approaches to ours. First, we could promote the use of P2Ppool and other forms of "responsible" mining, so that users can join pools without ceding full control of their resources to a central authority. This has been unsuccessful so far. At the time of writing, P2Pool accounts for less than 2% of the total hashpower; and while some pools support a protocol (called "getblocktemplate") that allows pool members to see the contents of the blocks they are asigned to work on (and, hence, could leave if they detect the pool is applying some disagreeable policy), the top six pools (which account for more than two thirds of the total hashpower) do not. A second approach is to monitor large pools and apply social pressure to limit their size. However, pools have been accused of *hiding* their bandwidth to avoid backlash. In any case, we make an analogy to coercion-resistance in electronic voting (see Section X): although social deterrents to undesired beahvior may in some cases be effective, greater confidence can be derived from a technical and economic deterrent.

"It's too late to change Bitcoin; and regardless, large miners wouldn't support this change." While we have described our design as a proposed modification to Bitcoin, this is primarily for ease of presentation; our design is also applicable as the basis for a new cryptocurrency, or as a modification to any of the hundreds of Bitcoin-like "altcoins" [8] which compete with Bitcoin (though, at the time of writing, Bitcoin remains far-and-away the most popular). Indeed altcoins have already begun to experiment with (weak) nonoutsourceable puzzles.

It seems unlikely our proposed design will soon be adopted by Bitcoin. Due to the coordination involved and the risk of splintering the network, there is (understandably) considerable political resistance within the Bitcoin community to adopting "hard fork" protocol changes, except in extreme cases [8]. However, such changes have occurred in the past, and could occcur again. Though miners are influential and it would be unwise to adopt a new policy that causes them to leave, they aren't unilaterally responsible for Bitcoin governance [8]; instead stakeholders include payment processors and services, operators of "full nodes" that may not mine, and developers of popular clients. Additionally, as we mentioned in Section VII-C and explain in detail in the Appendix, our nonoutsourceable puzzle constructions can be made backward compatible with existing Bitcoin mining equipment, lessening the impact on established miners [20]. Finally, even if our design is not adopted, the mere public knowledge of a viable coalition-resistant design alternative that the community *could* adopt — if necessary — may already serve as a deterrent against large coalitions.

X. Related Work

Computational puzzles. Moderately hard computational puzzles, often referred to as "proofs of work," were originally proposed for the purpose of combating email spam [19] (though this application is nowadays generally considered impractical [29]). Most work on computational puzzles has focused instead on "client puzzles," which can be used to prevent

denial-of-service attacks [27]. Recently, several attempts have been made to provide formal security definitions for client puzzles [15], [24], [47].

Theoretical and economic understanding of Bitcoin. Although a purely digital currency has been long sought by researchers [13], [14], [16], Bitcoin's key insight is to frame the problem as a consensus protocol and to provide an incentive for users to participate. Although Bitcoin's security has initially been proven (informally) in the "honest majority" model [22], [34], [36], this assumption is unsatisfying since it says nothing about whether the incentive scheme indeed leads to an honest majority. An economic analysis of Bitcoin by Kroll et al. [28] showed that honest participation in Bitcoin may be incentive compatible under assumptions such as a homogeneous population of miners and a limited strategy space. More recently, Eyal and Sirer [21] showed that with a more realistic strategy space, when a single player (or coalition) comprises more than a third of the network's overall strength, the protocol is not incentive compatible (and in fact the threshold is typically much less than one-third, depending on other factors involving network topology). This result underscores the importance of discouraging the formation of Bitcoin mining coalitions.

Decentralized Mining Pools. While most mining pools (including the largest) are operated by a central administrator, P2Pool [49] is a successful protocol for *decentralized* mining pools that achieve the desired effect (lower payout variance for participants) that does not require an administrator. It is possible that engineering efforts to improve P2Pool's performance and usability and public awareness campaigns may steer more users to P2Pool rather than centralized mining pools (at the time of writing[8], P2Pool accounts for only 1% of the total mining capacity while the two largest pools together account for 49%). However, as P2Pool inherently requires more overhead than a centralized pool, we believe it is wiser to directly discourage coalitions through the built-in reward mechanism.

Altcoins. Numerous attempts have been made to tweak the incentive structure by modifying Bitcoin's underlying puzzle. The most popular alternative, Litecoin[9] uses an scrypt-based [41] puzzle intended to promote the use of general purpose equipment (especially CPUs or GPUs) rather than specialized equipment (e.g., Bitcoin mining ASICs). Another oft-cited goal is to make the puzzle-solving computation have an intrinsically useful side effect (this is discussed, for example, in [28]). To our knowledge, we are the first to suggest deterring mining coalitions as a design goal.

Zerocoin [32], Zerocash [5], and PinocchioCoin [17] focus on making Bitcoin transactions anonymous by introducing a public cryptographic accumulator for mixing coins. Spending a coin involves producing a zero-knowledge proof that a coin has not yet been spent. Although our zero-knowledge proof construction may bear superficial resemblance to this approach, our work addresses a completely different problem.

[8]According to https://blockchain.info/pools retrieved on August 1, 2014

[9]https://litecoin.org/

Coercion-resistance in Electronic Voting. The approach we take is inspired by notions of *coercion-resistance* in electronic voting. Vote buying (as well as other forms of coercion) is illegal in all US state and federal elections [26]. While the threat of legal prosecution already poses a deterrent against such behavior, electronic voting schemes have been designed to provide technical countermeasures as well [7], [37]. In short, such schemes ensure that voters are unable to obtain any *receipt* which could demonstrate how they voted to a coercive attacker. We draw an analogy between vote buying and what we call *outsourcing schemes*; analogous to receipt-freeness, the (strongly) nonoutsourceable property prevents a worker from proving to a pool operator how its hashpower is used.

Most Closely Related Work. In Permacoin [33], Miller et al. proposed a Bitcoin-like system that achieves decentralized data storage as a useful side effect of mining. As part of their development, they implicitly developed a weakly nonoutsourceable puzzle that deters consolidation of storage capacity. Our Merkle-tree-based weakly nonoutsourceable puzzle construction is directly inspired by the construction in Permacoin. However, Permacoin does not make any attempt to formalize the notion of (weakly) nonoutsourceable puzzles, nor to consider the goal of deterring outsourcing outside the context of archival storage. Our paper provides the first formal treatment of nonoutsourceable puzzles. Additionally, we introduce a new notion of *strongly* nonoutsourceable puzzles, which *repairs a critical flaw in Permacoin* (namely, that weakly nonoutsourceable puzzles provide no deterrence against hosted mining providers with valuable reputations, or against pools who collect collateral deposits from their members).

In independent work, Eyal and Sirer [20] developed a technique for combining a weakly nonoutsourceable puzzle with an arbitrary scratch-off puzzle, resulting in a puzzle that retains the best properties of the constituents; this can be used, for example, to create a nonoutsourceable puzzle that is backward-compatible with existing Bitcoin mining equipment. We discuss applications of this technique in the Appendix.

XI. Conclusion

The prevalence of Bitcoin mining coalitions (including both mining pools and hosted mining services), which lead to consolidation of power and increased systemic risk to the network, are a result of a built-in design limitation of the Bitcoin puzzle itself – specifically, that it admits an effective coalition enforcement mechanism. To address this, we have proposed formal definitions of *nonoutsourceable* puzzles for which no such enforcement mechanism exists. We have contributed two constructions: a weak nonoutsourceable puzzle provable in the random oracle model, and a generic transformation from any weak nonoutsourceable puzzle to a strong one. The former may already be a sufficient deterrent against mining pools, while the latter thwarts both hosted mining and mining pools. We have implemented both of our techniques and provide performance evaluation results showing these add only a tolerable overhead to the cost of Bitcoin blockchain validation. Overall, we are optimistic that our approach, combined with suitable modifications to the reward structure, could be used to guarantee that participation as an independent individual is the most effective mining strategy.

Acknowledgments We thank the readers and reviewers of earlier drafts of this paper. This work was supported in part by NSF awards #0964541, #1223623, and #1518765.

REFERENCES

[1] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. Dhaes: An encryption scheme based on the diffie-hellman problem. Submission to IEEE P1363, 1999.

[2] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged byzantine impostors. *Department of Computer Science, Yale University, New Haven, CT, Tech. Rep*, 2005.

[3] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains, 2014.

[4] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better – how to make bitcoin a better currency. In *Financial Cryptography and Data Security*, pages 399–414. Springer, 2012.

[5] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *Security and Privacy (SP), 2014 IEEE Symposium on. IEEE*. IEEE, 2014.

[6] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 781–796, San Diego, CA, August 2014. USENIX Association.

[7] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 544–553. ACM, 1994.

[8] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua Kroll, and Edward W. Felten. Research perspectives on bitcoin and second-generation digital currencies. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015.

[9] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A Kroll, and Edward W Felten. Mixcoin: Anonymity for bitcoin with accountable mixes. In *Financial Cryptography and Data Security*, pages 486–504. Springer, 2014.

[10] Danny Bradbury. Alydian targets big ticket miners with terahash hosting. http://www.coindesk.com/alydian-targets-big-ticket-miners-with-terahash-hosting/, August 2013.

[11] Vitalik Buterin. Bitcoin network shaken by blockchain fork. http://bitcoinmagazine.com/3668/bitcoin-network-shaken-by-blockchain-fork/, 2013.

[12] Vitalik Buterin. Towards a 12 twelve second block time. https://blog.ethereum.org/2014/07/11/toward-a-12-second-block-time/, July 2014.

[13] David Chaum. Blind signatures for untraceable payments. In *Crypto*, volume 82, pages 199–203, 1982.

[14] David Chaum, Amos Fiat, and Moni Naor. Untraceable electronic cash. In *Advances in Cryptology—CRYPTO'88*, pages 319–327. Springer, 1990.

[15] Liqun Chen, Paul Morrissey, Nigel P Smart, and Bogdan Warinschi. Security notions and generic constructions for client puzzles. In *Advances in Cryptology–ASIACRYPT 2009*, pages 505–523. Springer, 2009.

[16] Wei Dai. b-money. http://www.weidai.com/bmoney.txt, 1998.

[17] George Danezis, Cedric Fournet, Markulf Kohlweiss, and Bryan Parno. Pinocchio coin: building zerocoin from a succinct pairing-based proof system. In *Proceedings of the First ACM workshop on Language support for privacy-enhancing technologies*, pages 27–30. ACM, 2013.

[18] Orr Dunkelman, Nathan Keller, and Jongsung Kim. Related-key rectangle attack on the full shacal-1. In *Selected Areas in Cryptography*, pages 28–44. Springer, 2007.

[19] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1993.

[20] Ittay Eyal and Emin Gün Sirer. How to disincentivize large bitcoin mining pools. Blog post: http://hackingdistributed.com/2014/06/18/how-to-disincentivize-large-bitcoin-mining-pools/, June 2014.

[21] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[22] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Advances in Cryptology-EUROCRYPT 2015*, pages 281–310. Springer, 2015.

[23] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology–EUROCRYPT 2013*, pages 626–645. Springer, 2013.

[24] Bogdan Groza and Bogdan Warinschi. Cryptographic puzzles and dos resilience, revisited. *Designs, Codes and Cryptography*, pages 1–31, 2013.

[25] Helena Handschuh, H Helena, and David Naccache. Shacal. In *First Open NESSIE Workshop*, November 2000.

[26] Richard L Hasen. Vote buying. *California Law Review*, pages 1323–1371, 2000.

[27] Ari Juels and John G Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *NDSS*, volume 99, pages 151–165, 1999.

[28] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of bitcoin mining or, bitcoin in the presence of adversaries. *WEIS*, 2013.

[29] Ben Laurie and Richard Clayton. Proof-of-work proves not to work; version 0.2. In *Workshop on Economics and Information, Security*, 2004.

[30] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. Inclusive block chain protocols. *Financial Cryptography and Data Security. Springer*, 2015.

[31] Jon Matonis. The bitcoin mining arms race: Ghash.io and the 51% issue. http://www.coindesk.com/bitcoin-mining-detente-ghash-io-51-issue/, July 2014.

[32] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[33] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for long-term data preservation. In *IEEE Symposium on Security and Privacy*, 2014.

[34] Andrew Miller and Joseph J. LaViola, Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. UCF Tech Report. CS-TR-14-01.

[35] myVBO. ziftrcoin : A cryptocurrency to enable commerce. Whitepaper, October 2014.

[36] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcon.org/bitcoin.pdf, 2008.

[37] Valtteri Niemi and Ari Renvall. How to prevent buying of votes in computer elections. In *Advances in CryptologyASIACRYPT'94*, pages 164–170. Springer, 1995.

[38] Emily Oster. Are all lotteries regressive? evidence from the powerball. *National Tax Journal, June*, 2004.

[39] Bryan Parno. A note on the unsoundness of vntinyram's snark. Cryptology ePrint Archive, Report 2015/437, 2015. http://eprint.iacr.org/.

[40] Bryan Parno, Craig Gentry, Jon Howell, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Symposium on Security and Privacy*, pages 238–252, 2013.

[41] Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. 2012.

[42] John Quiggin. On the optimal design of lotteries. *Economica*, 58(229):1–16, 1991.

[43] Meni Rosenfeld. Analysis of bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980*, 2011.

[44] Srinath Setty, Benjamin Braun, Victor Vu, Andrew J Blumberg, Bryan Parno, and Michael Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84. ACM, 2013.

[45] Yonatan Sompolinsky and Aviv Zohar. Secure high-rate transaction processing in bitcoin. *Financial Cryptography and Data Security. Springer*, 2015.

[46] Spreadcoin. http://spreadcoin.net/files/SpreadCoin-WhitePaper.pdf, October 2014.

[47] Douglas Stebila, Lakshmi Kuppusamy, Jothi Rangasamy, Colin Boyd, and Juan Gonzalez Nieto. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols. In *Topics in Cryptology–CT-RSA 2011*, pages 284–301. Springer, 2011.

[48] John Tromp. Cuckoo cycle: a new memory-hard proof-of-work system. *Bitcoin Research Workshop*, 2015.

[49] Forrest Voight. p2pool: Decentralized, dos-resistant, hop-proof pool. https://bitcointalk.org/index.php?topic=18313, June 2011.

APPENDIX

A. Preliminaries

A Non-Interactive Zero-knowledge Proof system (NIZK) is a collection of three algorithms $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$:

- $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda)$: Takes in the security parameter λ, and generates a common reference string crs.
- $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w)$: Takes in crs, a statement stmt, and a witness w such that $(\mathsf{stmt}, w) \in L$, outputs a proof π.
- $b \leftarrow \mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi)$: Takes in the crs, a statement stmt, and a proof π, and outputs 0 or 1, denoting rejection or acceptance. stmt, and a witness w,

Completeness. A NIZK system is said to be complete, if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $(\mathsf{stmt}, w) \in R$, we have

$$\Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda),\\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w) :\\ \mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1\end{array}\right] = 1 - \mathsf{negl}(\lambda)$$

Soundness. A NIZK system is said to be sound if it is infeasible for any polynomial-time adversary $\mathcal{A}$ to prove a false statement. More formally,

$$\Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda),\ (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}):\\ (\mathsf{stmt} \notin L)\ \wedge\ (\mathsf{Verify}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1)\end{array}\right] = \mathsf{negl}(\lambda)$$

Zero-knowledge. Informally, a NIZK system is computationally zero-knowledge, if the proof does not reveal any information about the witness to any polynomial-time adversary. More formally, a NIZK system is said to be computationally zero-knowledge, if there exists a simulator $S = (\mathsf{SimSetup}, \mathsf{SimProve})$, such that for all non-uniform polynomial-time adversary $\mathcal{A}$, for any stmt, w such that $(\mathsf{stmt}, w) \in R$, it holds that

$$\left|\begin{array}{l}\Pr\left[\begin{array}{l}\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda),\\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{stmt}, w) :\\ \mathcal{A}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1\end{array}\right] -\\ \Pr\left[\begin{array}{l}(\widetilde{\mathsf{crs}}, \tau) \leftarrow \mathsf{SimSetup}(1^\lambda, \mathsf{stmt}),\\ \widetilde{\pi} \leftarrow \mathsf{SimProve}(\mathsf{crs}, \mathsf{stmt}, \tau) :\\ \mathcal{A}(\widetilde{\mathsf{crs}}, \mathsf{stmt}, \widetilde{\pi}) = 1\end{array}\right]\end{array}\right| = \mathsf{negl}(\lambda)$$

Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^\lambda$ be a hash function modeled as a random oracle.

- $\mathcal{G}(1^\lambda)$: Draw a puzzle randomly, $\mathsf{puz} \xleftarrow{\$} \{0,1\}^\lambda$.
- $\mathsf{Work}(\mathsf{puz}, m, t)$:
  For $i \in [t]$: *//repeat up to t unit scratch attempts*
  Draw a random nonce, $r \xleftarrow{\$} \{0,1\}^\lambda$.
  If $\mathcal{H}(\mathsf{puz}\|m\|r) < 2^{\lambda-d}$ then return $\mathsf{ticket} := r$.
  Return $\perp$.
- $\mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket})$.
  Check that $\mathcal{H}(\mathsf{puz}\|m\|\mathsf{ticket}) < 2^{\lambda-d}$.

Fig. 2: An abstraction of the Bitcoin scratch-off puzzle.

B. The Bitcoin Scratch-Off Puzzle

An abstraction of Bitcoin's scratch-off puzzle is shown in Figure 2. We assume that each random-oracle call takes time t_{RO}, and all other work in each iteration of Work takes t_{other} time. We then have the following:

Theorem 5. *The construction in Figure 2 is a $(d, \underline{t}, t_0, \mu)$-scratch-off puzzle, where $\underline{t} = O(\lambda) \cdot t_{RO}$, $t_0 = 0$, and $\mu = t_{RO}/(t_{RO} + t_{other})$.*

Proof: The correctness proof is trivial, as is the proof of feasibility and parallelizability. For μ-incompressibility, observe that for any adversary that makes t random oracle calls, its probability of successfully finding a winning ticket is at most $\zeta^+(t)$. Since the honest Work algorithm takes $(t_{\text{RO}}+t_{\text{other}})\cdot t$ time, this scratch-off puzzle is $t_{\text{RO}}/(t_{\text{RO}}+t_{\text{other}})$-incompressible. ∎

Bitcoin is Outsourceable. We now describe an outsourcing protocol for the Bitcoin puzzle that resembles the scheme currently used in practice by Bitcoin mining pools to coordinate untrusted pool members. Let $d' < d$ be a parameter called the "share difficulty". Intuitively, the pool operator chooses a payload m (which, in practice, would contain a reward payment to the pool operator's public key, and other new Bitcoin transactions at the pool operator's discretion), and the worker performs $2^{d'}$ scratch-off attempts. On average, the worker finds at least one value r (called a "share") such that $\mathcal{H}(\mathsf{puz}\|m\|r) < 2^{\lambda-d'}$; the worker sends this value to the pool operator, who uses it to distinguish between an honest or dishonest worker (in practice, the worker is only paid when it submits a timely share: see [43] for a detailed explanation of payment policies in Bitcoin mining pools).

Let $\mu = t_{\text{RO}}/(t_{\text{RO}} + t_{\text{other}})$ be the maximum potential speedup an adversary can gain over Work. Let $t_g = 2^{d'} - O(1)$ be a lower bound on the expected number of random oracle queries an adversary must take in order to produce a share with statistically similar probability as the honest worker.

Theorem 6. *The Bitcoin puzzle defined in Figure 2 is not $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e, \alpha, p_s)$-weakly nonoutsourceable, when $t_{\mathcal{C}} = O(\lambda)$, $t_{\mathcal{S}} \le 2^{d'}$, $t_e \ge 2^{d'}$, and $p_s > \zeta^+(\mu(t_e + \alpha/\underline{t}) - t_g)/\zeta(t_e)$.*

Proof: Consider the $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e)$-outsourcing protocol $(\mathcal{S}, \mathcal{C})$ as described above, and assume in particular that $m \xleftarrow{\$} \{0,1\}^\lambda$ is chosen randomly; with high probability, $m \neq m^*$. Suppose for contradiction that an adversary $\mathcal{A}$ runs in time $t_{\mathcal{S}}\underline{t} + \alpha$, that the view of the pool operator

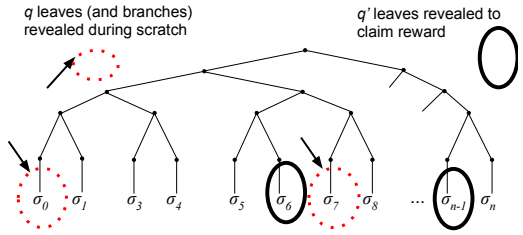

Fig. 3: Illustration of the weak nonoutsourceable puzzle (from [33])

is indistinguishable between executions $(\mathcal{A},\mathcal{C})$ and $(\mathcal{S},\mathcal{C})$, and that the worker outputs a ticket for m^* with probability $p_s\zeta(t_e)$. This means that the adversary, in expectation, must make more than $\mu(t_e + \alpha/\underline{t}) - t_g$ oracle calls of the form $\mathcal{H}(\mathsf{puz}\|m^*\|...)$, and t_g oracle calls of the form $\mathcal{H}(\mathsf{puz}\|m\|...)$. This requires more than $t_{\mathcal{S}}\underline{t} + \alpha$ total steps on average, which contradicts the worst-case running time assumption of $\mathcal{S}$. □

C. Weak nonoutsourceable puzzles

Our construction for a weak nonoutsourceable puzzle (based on [33]) is formally defined in Figure 4, and illustrated in Figure 3. We now restate and prove theorems (from Section V) about its correctness:

Theorem 1. *The construction in Figure 4 is a scratch-off puzzle.*

Proof: Correctness, feasibility/parallelizability proofs are trivial. We now prove incompressibility.

For any adversary $\mathcal{A}$ to obtain a winning ticket with $\zeta^+(t_e)$ probability, the adversary must have made at least t_e good scratch attempts. A good scratch attempt consists of at least two random oracle queries, $h := \mathcal{H}(\mathsf{puz}||r||\mathsf{digest})$ and $\mathcal{H}(\mathsf{puz}||r||\sigma_h)$ such that the branches σ_h are consistent with h and digest. For each good scratch attempt the adversary must know at least a constant fraction (for any constant < 1) of the branches, and have made random oracle calls to generate the branches such that they are consistent with the digest. Otherwise, when the adversary calls the random oracle for selecting the leaves (notice that the digest is an input to this call for selecting leaves), except with negligible probability, no polynomial-time adversary can output the selected branches (which are then fed into another random oracle to determine the difficulty). Also, for each good scratch attempt (that contributes to the $\zeta^+(t_e)$ success probability), the adversary must also make the random oracle call to select the branches, and the call to determine the difficulty. Therefore, the adversary must make an arbitrarily large constant fraction (for any constant < 1) of the random oracle calls for each good scratch attempt. Therefore, the scratch-off puzzle is μ-incompressible for an appropriate choice of μ, assuming that the rest of the computation is only at most a constant fraction of the random oracle calls. ■

Theorem 2. *Let $q, q' = O(\lambda)$. Let the number of leaves $L \geq q + 8q'$. Suppose $d > 10$ and $t_e \cdot 2^{-d} < 1/2$. Under the aforementioned cost model, the above construction is a $(t_{\mathcal{S}}, t_{\mathcal{C}}, t_e, \alpha, p_s)$ weakly nonoutsourceable puzzle, for any $0 < \gamma < 1$ s.t. $t_{\mathcal{C}} < \gamma t_e$, $p_s > \frac{1}{2}(1-\gamma) - \mathsf{negl}(\lambda)$, and $\alpha = O(\lambda^2)$; and is 0-non-transferable.*

For simplicity of presentation, we prove this for the case when $\gamma = 1/2$. It is trivial to extend the proof for general $0 < \gamma < 1$. To summarize, we would like to prove the following: for a protocol $(\mathcal{S},\mathcal{C})$, if no adversary $\mathcal{A}$ (running in time not significantly more than the honest worker) is able to steal the winning ticket with more than $\frac{1}{2}\zeta(t_e)$ probability, then $t_{\mathcal{C}}$ must be a significant fraction of t_e, i.e., the pool operator must be doing a significant fraction of the effective work. This would deter outsourcing schemes by making them less effective.

If the ticket output at the end of the protocol execution contains a σ_h such that 1) the selected leaves corresponding to σ_h were not decided by a random oracle call during the execution; or 2) σ_h itself has not been supplied as an input to the random oracle during the execution, then this ticket is valid with probability at most 2^{-d}. For $(\mathcal{S},\mathcal{C})$ to be an outsourcing protocol with t_e effective billable work, the honest protocol must perform a number of "good scratch attempts" corresponding to t_e. Every good scratch attempt queries the random oracle twice, one time to select the leaves, and another time to hash the collected branches. We now define the notion of a "good scratch attempt".

Definition 6. *During the protocol $(\mathcal{S},\mathcal{C})$, if either the pool operator or worker makes the two random oracle calls $h := \mathcal{H}(\mathsf{puz}||r||\mathsf{digest})$ and $\mathcal{H}(\mathsf{puz}||r||\sigma_h)$ for a set of collected branches σ_h that is consistent with h and* digest*, this is referred to as a good scratch attempt. Each good scratch attempt requires calling the random oracle twice – referred to as scratch oracle calls.*

Without loss of generality, we assume that in the honest protocol, if a good scratch attempt finds a winning ticket, the pool operator will accept the ticket. This makes our proof simpler because all good scratch attempts contribute equally to the pool operator's winning probability. If this is not the case, the proof can be easily extended – basically, we just need to make a weighted version of the same argument. For each good scratch attempt, there are two types of random oracle calls. Type 1 calls select the leaves. Type 2 calls hash the collected branches. Assume in the extreme case that the worker makes all the Type 1 calls (which accounts for $1/2$ of all work associated with good scratch attempts). Now consider Type 2 work which constitutes the other half: for each good scratch attempt, if the worker knows $< 1/3$ fraction of leaves of the corresponding tree before the Type 1 random oracle call for selecting the leaves, then the pool operator must have done at least one unit of work earlier when creating the Merkle tree digest. This is because if the worker knows $< 1/3$ fraction of the leaves before leaves are selected, then the probability that the selected leaves all fall into the fraction known by the worker is negligible. Since this is a good scratch attempt, for the selected leaves that the worker does not know, the pool operator must then know the leaves and the corresponding Merkle branches. This means that the pool operator earlier called the random oracle on those leaves to construct the Merkle digest.

If we want the pool operator's total work to be within $1/2$ of the total effective work, then for at least $1/2$ of good scratch attempts: worker must know at least $1/3$ of leaf nodes before the Type 1 oracle is called to select the leaves.

Let $\mathcal{H}, \mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^\lambda$ denote random oracles.

- $\mathcal{G}(1^\lambda)$: generate $\mathsf{puz} \xleftarrow{\$} \{0,1\}^\lambda$
- $\mathsf{Work}(\mathsf{puz}, m, t)$:

  Construct Merkle tree. Sample L random strings $\mathsf{leaf}_1, \ldots, \mathsf{leaf}_L \xleftarrow{\$} \{0,1\}^\lambda$, and then construct a Merkle tree over the leaf_i's using $\mathcal{H}_2$ as the hash function. Let digest denote the root digest of this tree.

  Scratch. Repeat the following scratch attempt t times:

  - Draw a random nonce, $r \xleftarrow{\$} \{0,1\}^\lambda$.
  - Compute $h := \mathcal{H}(\mathsf{puz} \| r \| \mathsf{digest})$, and use the value h to select q distinct leaves from the tree.
  - Let $B_1, B_2, \ldots, B_q$ denote the branches corresponding to the selected leaves. In particular, for a given leaf node, its *branch* is defined as the Merkle proof for this leaf node, including leaf node itself, and the sibling for every node on the path from the leaf to the root.
  - Compute $\sigma_h := \{B_i\}_{i \in [q]}$ in sorted order of i.
  - If $\mathcal{H}(\mathsf{puz} \| r \| \sigma_h) < 2^{\lambda - d}$ then record the solution pair (r, σ_h) and goto "Sign payload".

  Sign payload. If no solution was found, return $\perp$. Otherwise, sign the payload m as follows:

  - Compute $h' := \mathcal{H}(\mathsf{puz} \| m \| \mathsf{digest})$, and use the value h' to select a set of $4q'$ distinct leaves from the tree such that these leaves are not contained in σ_h. From these, choose an arbitrary subset of q' distinct leaves. Collect the corresponding branches for these q' leaves, denoted $B_1, B_2, \ldots, B_{q'}$.
  - Let $\sigma'_h := \{B_i\}_{i \in [q']}$ in sorted order of i.
  - Return $\mathsf{ticket} := (\mathsf{digest}, r, \sigma_h, \sigma'_h)$.
- $\mathsf{Verify}(\mathsf{puz}, m, \mathsf{ticket})$:

  Parse $\mathsf{ticket} := (\mathsf{digest}, r, \sigma_h, \sigma'_h)$
  Compute $h := \mathcal{H}(\mathsf{puz} || r || r || \mathsf{digest})$ and $h' := \mathcal{H}(\mathsf{puz} || m || \mathsf{digest})$.
  Verify that σ_h and σ'_h contain leaves selected by h and h' respectively.
  Verify that σ_h and σ'_h contain valid Merkle paths with respect to digest.
  Verify that $\mathcal{H}(\mathsf{puz} || r || \sigma_h) < 2^{\lambda - d}$.

Fig. 4: A weakly nonoutsourceable and non-transferable scratch-off puzzle.

Suppose that $d > 10$ is reasonably large and that $t_e \cdot 2^{-d} < 1/2$. Basically, in this case, the probability that two or more tickets are found within t_e good attempts are a constant fraction smaller than the probability of one winning ticket being found. If for at least $1/2$ of the good scratch attempts, the worker knows at least $1/3$ fraction of leaves before leaves are selected, then an adversarial worker $\mathcal{A}$ would be able to steal the ticket with constant probability given that a winning ticket is found. To see this, first observe that the probability that a single winning ticket is found is a constant fraction of $\zeta^+(t_e)$. Conditioned on the fact that a single winning ticket is found, the probability that this belongs to an attempt that the worker knows $> 1/3$ leaves before leaves are selected is constant. Therefore, it suffices to observe the following fact.

Fact 1. *For a good scratch attempt, if a worker knows $> 1/3$ fraction of leaves before leaves are selected, then conditioned on the fact that this good scratch attempt finds a winning ticket, the worker can steal the ticket except with probability proportional to $\exp(-cq')$ for an appropriate positive constant c.*

Proof: By a simple Chernoff bound. The argument is standard. In expectation, among the selected $4q'$ leaves, the worker knows $1/3$ fraction of them. Further, the worker only needs to know $1/4$ fraction of them to steal the ticket. The probability that the worker knows less than q' out of $4q'$ leaves can be bounded using a standard Chernoff bound, and this probability is upper bounded by $\exp(-q'/27)$. ∎

Proof of Non-Transferability. For non-transferability, we need to show that for any polynomial time adversary $\mathcal{A}$, knowing polynomially many honestly generated tickets to puz for payload $m_1, m_2, \ldots, m_\ell$ does not help noticeably in computing a ticket for m^* to puz, where $m^* \neq m_i$ for any $i \in [\ell]$.

The adversary $\mathcal{A}$ may output two types of tickets for m^*: 1) m^* uses the same Merkle digest as one of the m_i's; and 2) m^* uses a different Merkle digest not seen among the m_i's.

In the latter case, it is not hard to see that the adversary $\mathcal{A}$ can only compress the computation at best as the best incompressibility adversary. Therefore, it suffices to prove that no polynomial time adversary can succeed with the first case except with negligible probability. Below we prove that.

Notice that the honest Work algorithm generates a fresh Merkle digest every time it is invoked. Therefore, with the honest algorithm, each Merkle digest will only be used sign a single payload except with negligible probability. Since the number of leaves $L \geq q + 8q'$, there are at least $8q'$ leaves to choose from in the signing stage. q' of those will be revealed for signing a message m. The probability that the revealed q' leaves are a valid ticket for message $m' \neq m$ is bounded by $\binom{8q'}{3q'} / \binom{8q'}{4q'} \propto \exp(-c_2 q')$. If the adversary has seen honestly generated tickets for ℓ different payloads, by union bound, the probability that there exists a ticket, such that its q' revealed leaves constitute a valid signature for a different message m^* is bounded by $\ell \cdot \exp(-c_2 q')$.

D. Strong nonoutsourceable puzzles

Theorem 4. *If the underlying puzzle $(\mathcal{G}', \mathsf{Work}', \mathsf{Verify}')$ is δ'-non-transferable, then the derived puzzle through the generic*

transformation is δ non-transferable for

$$\mu + \delta' \leq \frac{(\mu+\delta)t}{t \cdot \underline{t} + (t_{enc} + t_{nizk})\ell}$$

where t_{enc} and t_{nizk} are the time for performing each encryption and NIZK proof respectively.

Proof: We show that if an adversary $\mathcal{A}$ running in time t can win the non-transferability game of the derived puzzle, we can construct another adversary $\mathcal{A}'$ running in slightly more time than t that can win the non-transferability game of the underlying puzzle.

$\mathcal{A}'$ will call $\mathcal{A}$ as a blackbox. $\mathcal{A}'$ first receives a challenge for the underlying puzzle, in the form of $\mathsf{puz}', m'_1, m'_2, \ldots, m'_\ell$, and winning tickets $\mathsf{ticket}'_1, \ldots, \mathsf{ticket}'_\ell$. Next, $\mathcal{A}'$ picks crs honestly, and picks $\mathsf{pk}_\mathcal{E}$ such that $\mathcal{A}'$ knows the corresponding $\mathsf{sk}_\mathcal{E}$. $\mathcal{A}'$ now gives to $\mathcal{A}$ the puzzle $\mathsf{puz} := (\mathsf{puz}', \mathsf{crs}, \mathsf{pk}_\mathcal{E})$. For $i \in [\ell]$, $\mathcal{A}'$ computes the zero-knowledge version $\mathsf{ticket}_i := (c_i, \pi_i)$. where c_i is an encryption of ticket'_i, and π_i is the NIZK as defined in Figure 1. $\mathcal{A}'$ gives $m'_1, \ldots, m'_\ell$ and $\mathsf{ticket}_1, \ldots, \mathsf{ticket}_\ell$ to $\mathcal{A}$ as well. Since $\mathcal{A}$ wins the non-transferability game, it can output a winning ticket (m^*, ticket^*) for puzzle puz with at least $\zeta^+((\mu+\delta)t)$ probability where $t \cdot \underline{t}$ is the runtime of $\mathcal{A}$; further $m^* \neq m'_i$ for any $i \in [\ell]$.

$\mathcal{A}'$ now parses $\mathsf{ticket}^* := (c, \pi)$. $\mathcal{A}'$ then uses $\mathsf{sk}_\mathcal{E}$ to decrypt c and obtain ticket' – if the NIZK is sound, then ticket' must be a winning solution for the underlying puzzle puz' and payload m^* except with negligible probability – since otherwise one can construct an adversary that breaks soundness of the NIZK. Now, $\mathcal{A}'$ outputs (m^*, ticket') to win the non-transferability game. $\mathcal{A}'$ runs in $t \cdot \underline{t} + (t_{\text{enc}} + t_{\text{nizk}})\ell$ time, but wins the non-transferability game with probability at least $\zeta^+((\mu+\delta)t)$. This contradicts the fact that the underlying puzzle is δ'-non-transferable. ■

Theorem 3. *If $(\mathsf{GenKey}', \mathsf{Work}', \mathsf{Verify}')$ is a $(t_\mathcal{S}, t_\mathcal{C}, t_e, \alpha, p_s)$ weakly nonoutsourceable puzzle, then the puzzle described in Figure 1 is a $(t_\mathcal{S}, t_\mathcal{C}, t_e, \alpha + t_{enc} + t_{NIZK}, p_s - \mathsf{negl}(\lambda))$ strongly nonoutsourceable puzzle, where $t_{enc} + t_{NIZK}$ is the maximum time required to compute the encryption and NIZK in the honest* Work *algorithm.*

Proof: We first prove that this derived puzzle preserves weak nonoutsourceability of the underlying puzzle. Suppose that $(\mathcal{S}, \mathcal{C})$ is a $(t_\mathcal{S}, t_\mathcal{C}, t_e)$ outsourcing protocol. We will construct a suitable stealing adversary $\mathcal{A}$. To do so, we begin by deriving an outsourcing protocol $(\mathcal{S}', \mathcal{C}')$ for the underlying puzzle; the stealable property of the underlying puzzle allows to introduce an adversary $\mathcal{A}'$, from which we will derive $\mathcal{A}$. Let the outsourcing protocol $(\mathcal{S}', \mathcal{C}')$ for the underlying puzzle be constructed as follows: Suppose that $\mathcal{C}'$ is given the input puz'.

1) $\mathcal{S}'$ executes $\mathcal{S}$ unchanged.
2) $\mathcal{C}'$ first generates a keypair according to the encryption scheme, $(\mathsf{pk}_\mathcal{E}, \mathsf{sk}_\mathcal{E}) \leftarrow \mathcal{E}.\mathsf{Key}(1^\lambda)$, runs the NIZK setup $\mathsf{crs} \leftarrow \mathsf{NIZK.Setup}(1^\lambda)$, and then runs $\mathcal{C}$ using puzzle $(\mathsf{puz}', \mathsf{crs}, \mathsf{pk}_\mathcal{E})$.
3) If $\mathcal{C}$ outputs a ticket (c, π), then $\mathcal{C}'$ decrypts c and outputs $\mathsf{ticket}' \leftarrow \mathsf{Dec}(\mathsf{sk}_\mathcal{E}, \mathsf{c})$.

When run interacting with $\mathcal{S}$, the original pool operator $\mathcal{C}$ outputs a valid ticket with probability at least $\zeta(t_e) - \mathsf{negl}(\lambda)$; therefore, the derived pool operator $\mathcal{C}'$ decrypts a valid ticket with probability $\zeta(t_e) - \mathsf{negl}(\lambda)$. Since the underlying puzzle scheme is assumed to be weakly nonoutsourceable, there exists a stealing adversary $\mathcal{A}'$ running in time $t'_a = t_\mathcal{S} + \alpha$. We can thus construct an $\mathcal{A}$ that runs $\mathcal{A}'$ until it outputs $\mathsf{ticket}'_\mathcal{A}$ for the underlying puzzle, and then generate an encryption and a zero-knowledge proof.

We also need to prove it satisfies the additional indistinguishability property required from a strongly nonoutsourceable puzzle. This follows in a straightforward manner from the CPA-security of the encryption scheme, and the fact that the proof system is zero-knowledge. □

E. Optimizations and Parameterization of Strongly Nonoutsourceable Puzzles

Diffie-Hellman + SHACAL Encryption. Pinocchio and Libsnark each support field operations in $\mathbb{F}_p^*$, where p is a 254-bit prime corresponding to a family of "pairing-friendly" elliptic curves (p is different for each system). Unfortunately, the order of this group, $p-1$, is not prime, nor was p constructed so that $p-1$ has a large prime factor. Therefore, taking inspiration from "PinocchioCoin" [17], we do our public key operations in a prime-order subgroup of the Galois extension field $\mathbb{F}_{p^\mu}$, where we have chosen $\mu = 4$ such that p^μ is over 1000 bits, and $\mathbb{F}_{p^\mu}$ has a subgroup of suitably-large prime-order $q \approx 398$ bits.

Our encryption scheme is a variant of DHAES [1], except we replace the symmetric-key encryption component AES with SHACAL [25] (i.e. SHA-1 used as a block cipher) since SHA-1 has a relatively efficient implementation as an arithmetic circuit. Note that the known attacks on SHACAL-1 [18] require an adaptive distinguisher to ask for multiple encryptions under a single key, whereas in our setting an encryption key is used one time only, and therefore is not vulnerable to this attack.

Statement-level parallelism using HMAC commitments. The GGPR scheme underlying Pinocchio and Libsnark is in fact stronger than a NIZK; it is also *succinct* (i.e., a SNARK), meaning that the cost of verifying a proof is independent of the size of the circuit. However, the prover overhead grows superlinearly in the size of the circuit. Additionally, although the GGPR scheme is largely parallelizable, [10] the current Pinocchio implementation does not support parallelization and Libsnark's parallel support yielded less than linear speedup. Therefore, we take the approach of decomposing the overall NP statement into several smaller substatements, for which all the proofs may be computed separately. In particular, we produce two types of statements. The first type of statement (Type I statements) processes a portion (C layers at a time) of one of the $q + q'$ Merkle tree branches, and checks that they are encrypted properly; $(q+q')\lceil \frac{h}{C} \rceil$ of these statements must be proven. The Type II statement (of which we require only one instance) performs the public-key operation, checks the winning condition of the puzzle solution, and checks that

[10] For instance, Zaatar [44] reports a near-linear parallel speedup for a similar scheme.

the q' chosen tree indices are a subset of the $4q'$ selected based on the attribution message. For every variable shared between substatements, we add to the substatements a SHA-based HMAC commitment. In total, the ticket thus consists of $(q+q')\lceil \frac{h}{C} \rceil + 2$ 160-bit HMAC tags, $(q+q')(h+1)$ 160-bit SHACAL ciphertexts, and one 1024-bit Diffie-Hellman group element.

F. Integration in Cryptocurrencies

Composition with GHOST. The recently proposed GHOST protocol [30], [45] offers a promising approach for greatly increasing the rate of blocks – and thereby reducing the reward variance for solo miners – without compromising security. In short, this rule creates a block*graph* rather than a block*chain*, and allows for even the stale blocks to earn a reward and contribute towards the total difficulty score of the blockchain. An analysis by Vitalik Buterin suggests [12] that even a three-second block time (as in our proposed reward structure) could be feasible using GHOST. This protocol is complementary to and ultimately compatible with our nonoutsourceable puzzles; however, several subtleties must be addressed when combining the two designs.

First we recap the GHOST design and its rationale. The main obstacle to reduce Bitcoin's hardcoded block arrival rate (10 minutes) is that it increases the rate of "stale" blocks (i.e., the frequency at which two honest users find redundant, mutually exclusive blocks at the same height). This happens occasionally due to network latency and chance. Stale blocks decreases the honest network's effective hashpower, and fast convergence is not guaranteed if they occur too frequently. To compensate for this, instead of including the hash of a parent block, GHOST blocks can include a parent as well as an arbitrary number of "uncles" that would otherwise have been discarded as stale blocks. Various policies can be used for assigning discounted rewards to uncle blocks and prioritizing conflicting transactions.

The following subtleties arise when applying our puzzles to GHOST. First of all, we must forbid having multiple zero-knowledge puzzle solutions at the same height, since our definitions and constructions allow a puzzle solver to easily derive multiple zero-knowledge solutions from the same underlying solution. This restriction is not much detriment, as we have already argued that zero-knowledge puzzle solutions are most effective for rare, high-value rewards. Second, we must prevent fast blocks from introducing new tracing techniques; with faster blocks, it becomes more plausible that each miner may be mining on a unique combination of parent and uncle blocks. To avoid this problem, we allow the zero-knowledge puzzle solution to reveal only the parent block and omit any uncles that were committed; plaintext solutions reveal both the parent and uncles.

ASIC-Resistance or Backward Compatibility. Even if hosted mining schemes are thwarted, one may be concerned that mining will still inevitably trend towards centralization due to economies of scale alone. There has been significant interest from the Bitcoin community in the development of ASIC-resistant puzzles for which general purpose computing hardware can compete more effectively against customized hardware. An ASIC-resistant puzzle could have the benefit of encouraging even more diverse participation from individuals with spare computing cycles, and reducing the potential profit from large industrial miners. The predominant approach to ASIC-resistance (e.g., [48]) is to require memory accesses rather than just computation, since the performance and cost of memory has changed much less over time than logic and arithmetic circuits.

Fortunately, ASIC-resistant puzzles can be combined with our approach. Concurrent to our work, Eyal and Sirer developed a complementary technique called "Two-Phase Proofs-of-Work" [20] that constructs a weakly nonoutsourceable puzzle puz from an arbitrary scratch-off puzzle (such as [48]) puz_1 and an existing weakly nonoutsourceable puzzle puz_2 (such as our construction in Figure 4). In essence, this scheme involves breaking the overall difficulty parameter d into two smaller difficulty parameters d_1 and d_2 such that $d = d_1 \cdot d_2$. Each attempt at solving puz_1 has a chance 2^{-d_1} of success; among the solutions to puz_1, each has a chance 2^{-d_2} of further being a solution to puz_2. The composed puzzle is weakly nonoutsourceable (though with looser parameters than the underlying nonoutsourceable puzzle) as long as d_1 is set low enough that it is implausible for the pool operator to perform the puz_2 work himself and receive puz_1 solutions from a worker. As the resulting puzzle is weakly nonoutsourceable, it can be upgraded to strongly nonoutsourceable using our generic transformation (Figure 1).

Interestingly, this technique can also be used to achieve the *opposite* effect, namely to build a nonoutsourceable puzzle that is *backward-compatible* with existing Bitcoin mining ASICs (indeed this was the original motivation [20]). To achieve this, we simply compose a nonoutsourceable puzzle with the original Bitcoin puzzle rather than an ASIC-resistant one. It remains unclear whether backward-compatibility or ASIC resistance is more desirable; to resolve this would require detailed economic analysis. Regardless, our constructions are complementary to either goal.

SPV Clients. Not all nodes in the Bitcoin network process the entire contents of every block. Instead, Bitcoin supports a mode for lightweight or mobile clients called Simplified Payment Verification (SPV) that inspects only the proofs-of-work (and payments to the client itself) rather than all the transactions in the network. Although our implementation results show that weak nonoutsourceable puzzles impose only a modest additional overhead, increasing the block rate would create substantial additional load on SPV clients. However, a recent proposal for "compressed SPV proofs" [3] would allow for SPV clients to process only a small sample of the proofs-of-work, mitigating this concern.

Additional issues. While it is more natural to imagine that a stealing worker would typically steal a puzzle solution for itself and *withhold* it from the pool operator, such a strategy would indeed be distinguishable from an honest worker. For example, if the honest worker has no private inputs, and if a pool operator notices that a stolen ticket has been published, it could *audit* the worker by executing the honest worker protocol from scratch. This is impractical, however, since it imposes a high cost on pool operators and can be triggered

by false alarms due to zero-knowledge solutions produced independently. Thus, even though such a stealing strategy would not suffice for our nonoutsourceable definition, it is a plausible deterrent nonetheless.

Second, our formal model does not account for latency. A stealing worker that satisfies our formal requirements and does *not* attempt to suppress the pool operator's ticket would find itself in a race, since both the worker's ticket and the pool operator's ticket would be published and the Bitcoin network would have to eventually accept one or the other. In this scenario, the latency required to compute a stolen solution is critical, since it could make the difference in a close race. Even this strategy is plausible, however, since a hosted mining provider would have access to plenty of computational resources and could benefit from additional parallelism even beyond what is achieved in our benchmarks.

Finally, if the stealing worker wins a close race between its own ticket and the pool operator's, the pool operator could consider the coincidence of a zero-knowledge ticket published at nearly the same time as its own as evidence that the worker cheated. While timing-based detection heuristics fall outside our model, they do not cause much concern since they would be subject to false positives; since zero-knowledge tickets can be generated independently, it would be easy to "frame" an honest worker for cheating.