On Bitcoin as a public randomness source

Joseph Bonneau[1], Jeremy Clark[2], and Steven Goldfeder[3]

[1] Stanford University
[2] Concordia University
[3] Princeton University

Abstract. We formalize the use of Bitcoin as a source of publicly-verifiable randomness. As a side-effect of Bitcoin's proof-of-work-based consensus system, random values are broadcast every time new blocks are mined. We can derive strong lower bounds on the computational min-entropy in each block: currently, at least 68 bits of min-entropy are produced every 10 minutes, from which one can derive over 32 near-uniform bits using standard extractor techniques. We show that any attack on this beacon would form an attack on Bitcoin itself and hence have a monetary cost that we can bound, unlike any other construction for a public randomness beacon in the literature. In our simplest construction, we show that a lottery producing a single unbiased bit is manipulation-resistant against an attacker with a stake of less than 50 bitcoins in the output, or about US$12,000 today. Finally, we propose making the beacon output available to smart contracts and demonstrate that this simple tool enables a number of interesting applications.

1 Introduction

The ability to generate randomness is the foundation of many security protocols. Often random numbers are used to keep information secret and are kept private. However, a number of interesting applications require random values that cannot be predicted prior to being generated, but are made public after generation.

Such public randomness can be used to build random protocols with accountability. Historically, many organizations have attempted to publicize the physical generation of randomness to establish credibility in a process which is claimed to be random. For example, the National Basketball Association holds an annual draft lottery to award teams exclusive rights to sign top amateur players. Though the league commissioner draws ping-pong balls on camera from a hopper to establish the draft order, fans have claimed for years that the process is rigged and it has proved difficult to dispel that notion. A more somber example is the 1969 US conscription lottery, in which several politicians mixed capsules containing days of the year in a rotating drum and then drew them randomly by hand. Statisticians have since determined the process was botched, with an improbably high number of late-year dates selected due to insufficient mixing.

Generating public randomness is a common problem, supporting the desire to build a dedicated service which can be more efficient and (potentially) offer higher security against manipulation. This was first formalized by Rabin [32] who proposed a trusted service called a *beacon* that broadcasts fresh random numbers at regular intervals which any party can sample. Beacons have many

potential applications outside of cryptographic protocols, including verifiably selecting lottery winners or choosing precincts to audit after an election.

The use-cases are compelling enough that NIST has developed and hosts a beacon service which currently generates and publishes 512-bits of randomness every minute [29]. Commercial beacons also exist, such as Random.org, which have been audited and approved for use by gambling regulators. While these services undergo auditing and NIST has published extensive documentation of their apparatus for extracting randomness from quantum-mechanical effects, ultimately these services are trusted third-parties with good reasons to avoid. Random.org openly states that they "buffer" their randomness before publishing it and hence could be in a position to profit from advance knowledge of the beacon output. NIST's reputation has been significantly tarnished by its role in the publication of the Dual Elliptic Curve PRNG standard [4], which it has now admitted contained a backdoor [19, 34]. Proprietary beacons may also be unreliable: NIST's beacon was offline during the 2013 US Government shutdown.

For these reasons we would strongly like to construct a decentralized beacon with *no trusted parties*. There are several proposals for building a beacon from a publicly observable natural process such as meteorological events or cosmic background radiation. Stock market prices are easy to agree on and have been shown to contain a high amount of entropy [11]. They have even been used to determine random auditing in real elections, including a municipal election in Takoma Park, WA, USA [9]. However, it remains unclear to what extent financial exchanges are in fact trusted parties which could manipulate the beacon output by subtly altering which prices they announce. Financial data also has limited availability as most exchanges are closed far more often than not.

In this paper, we propose and analyze the construction of a beacon from Bitcoin's central data structure, the block chain [26]. Several applications have emerged which extract random numbers from Bitcoin, including at least one website, BitcoinMegaLottery [1] which attempts to implement a verifiable lottery using Bitcoin blocks.[4] Some Bitcoin-based protocols [8, 16, 25] already rely on the block chain for randomness, although without any precise security model.

Our central contribution is to formalize the use of Bitcoin as a randomness source and provide precise security guarantees. A Bitcoin-based beacon has a number of advantages due to the decentralization of Bitcoin and the large amount of computational work expended on maintaining the Bitcoin block chain. Bitcoin's cryptographic structure lends itself to a concrete analysis of its entropy and its function as a currency system allows us to directly compute the financial cost of attempting to manipulate the beacon output.

Our work shows that Bitcoin is indeed suitable as a public randomness beacon for applications beyond the Bitcoin-focused protocols in which it has so far been considered. Our primary contribution is defining and analyzing a Bitcoin-based beacon is the first in the literature which requires no trusted parties. We also for-

[4] BitcoinMegaLottery hashes block data as well as IDs assigned to tweets by Twitter. This does not appear secure as there are no guarantees the tweet IDs are generated before the block chain data.

mally define a powerful attacker model, capable of *bribing* all miners in existence and provide precise manipulation costs in this model. Finally, we demonstrate some practical examples for incorporating a blockchain-based beacon directly into smart contracts.

2 Preliminaries

2.1 Extractors

An extractor is a function $y = \mathsf{Ext}_k(x)$ that takes an n-bit input x of "sufficient" entropy and an ℓ-bit key k and returns an m-bit output y of "high" entropy where $m < n$. Here sufficient entropy means the min-entropy $H_\infty(x)$ is at least m bits and high entropy means that the statistical distance between the distribution on y and an m-bit uniform distribution is a negligible. The key k is not a secret, but is used to randomly select from a family of extractors.

While much of the literature focuses on combinatorial extractors [27, 33, 35, 36]; in practice constructions for extractors based on HMACs and block ciphers in CBC-MAC mode are most commonly used. These cryptographic extractors have been proven secure when the input has at least $2m$ bits of min-entropy [13].

2.2 Beacon

A beacon is a function $r = \mathsf{Beacon}(t)$ which returns an m-bit near-uniformly random value r at each time interval t. Typically the beacon samples from some source of randomness $\mathcal{D}_t$ upon which we make no assumptions beyond a lower-bound on min-entropy. Given a sample $\mathcal{D}_t$ the uniformly random output r is computed using an extractor:

$$r \leftarrow \mathsf{Beacon}(t) = \mathsf{Ext}_k(\mathcal{D}_t)$$

Beacons should satisfy a few security properties:

- *unpredictable*: any adversary's ability to predict any information about r prior to time t is negligible.
- *unbiased*: r is statistically close to an m-bit uniformly random string.
- *universally sampleable:* after time t any party can efficiently compute $\mathsf{Beacon}(t)$.
- *universally verifiable:* the sample $\mathcal{D}_t$ can be verified to be unknown to any party prior to time t.

Note that there is no formal property stating the beacon cannot be manipulated. This property is covered by unpredictability: if a beacon can be manipulated, then it can be predicted with non-negligible accuracy.

2.3 Bitcoin consensus

Bitcoin [26] is an extremely complex system (for a more detailed description see [6]). Here we only provide a brief overview of the consensus protocol, sometimes referred to as *Nakamoto consensus*. Bitcoin miners maintain the *block chain*, a public data structure serving as a global ledger of all transactions in the history of the system. New batches of transactions are published in a *block* approximately every 10 minutes. Any party can work as a Bitcoin *miner* attempting to publish the next valid block of transactions, but doing so is computationally difficult due to the proof-of-work system in Bitcoin. A valid block B must have a hash value starting with d consecutive zeroes. The difficulty parameter d is continually adjusted to maintain the average mining rate of 1 block per 10 minutes. At the time of this writing $d \approx 66.4$. Each block contains a hash of the previous valid block as determined by the block's miner. The set of all purported blocks thus forms a tree, with the longest chain deemed as the one valid block chain defining current ownership of all bitcoins.

Miners are incentivized to publish only valid blocks (consistent with the rules of the Bitcoin protocol) and to always extend the longest chain as they earn newly minted coins if they successfully publish a valid block. A block's validity is established by decentralized consensus; if a quorum of miners with the majority of computational power accept a given block as valid by mining on top of it then eventually it will stay embedded in the longest chain and the miner will get to keep their mining reward.

For our purposes, the only critical properties are that finding blocks is computationally difficult and that miners receive a substantial reward for finding them. We note that our beacon construction is equally applicable to many Bitcoin-derived currencies (e.g. LiteCoin) which utilize a similar block chain structure.

3 Beacon construction

Our basic construction applies an extractor to the the header of one or more blocks in the Bitcoin block chain. Each block header is a 640-bit data structure consisting of the fields listed in Table 1, as well as the block hash. The amount of nominal entropy in the Merkle tree of transactions alone is likely thousands of bits, as it incorporates hundreds of Bitcoin transactions, nearly all containing ECDSA signatures which rely on strong randomness for security. However, for the purposes of a beacon which is unpredictable we care only about entropy conditional on all public information just prior to the moment the block is first announced. This eliminates most of the entropy from transactions, as they are nearly all published in transaction pools prior to being included in a block. Instead, we rely only on unpredictability inherent to the mining process itself.

3.1 Min-entropy in normal operation

If Bitcoin miners are disinterested, then there exists a strong lower bound of d bits of min-entropy in a valid block header due to Bitcoin's proof-of-work puzzles.

field	*size (bits)*	*min-entropy*
version number	32	0
previous block hash	256	0
Merkle hash of transactions	256	$> d - \mu$
timestamp (seconds)	32	≈ 10
current difficulty level	32	0
nonce	32	μ

Table 1. Fields in a Bitcoin block header and lower bounds of their min-entropy. Currently $d > 68$. Technically we might have $0 < \mu < 32$ but in practice $\mu \approx 32$.

Anybody can create a block with a random nonce which will have a probability 2^{-d} of being valid. It is widely believed that there is no faster way to check a block's validity (even probabilistically) than computing the hash. Similarly, it is believed that there is no feasible way of choosing a block (including a nonce) such that it has a probability of validity $> 2^{-d}$ prior to computing the hash. If there were an efficient algorithm for producing such blocks, this would lead to arbitrage as this would be a more efficient way to mine Bitcoins. Indeed, even if such an algorithm were ever discovered, d would increase due to the increased rate of blocks being found and the system would revert to the same security level assuming miners continue to dedicate the same amount of computational power to mining. Thus, it is strongly believed that no party is capable of predicting the next published block header with probability $> 2^{-d}$. This gives the distribution of blocks a *computational min-entropy* of at least d.

In practice that this entropy is split between the nonce value (which contains $\mu \leq 32$ bits of entropy) and the Merkle root of transactions which contains the rest, as noted in Table 1. Because nonces are limited to 32 bits, most attempted transaction Merkle roots to have no possible nonce which yields a valid block, meaning miners must vary both the nonce and the transaction tree while searching for a valid block. The transaction tree also includes an explicit `extraNonce` parameter specifically for this purpose. In theory, miners could ignore the nonce field and only try modifications of the transaction set, though in practice efficient miners will always exhaust the nonce-space before modifying the transaction tree because this requires only computing the final block hash and not the more expensive recomputation of the transaction set's Merkle root. In any case, for our construction we hash both the nonce and the transaction tree to ensure we extract all available entropy from the mining process.

3.2 Malicious miners

If miners are malicious and attempting to influence the value output by the beacon, they might attempt to only mine blocks which would produce a certain beacon output. However, we can make this financially expensive by including both the block header and its hash in our extractor:

$$\mathsf{Beacon}(t) = \mathsf{Ext}_k(B_t || H(B_t))$$

This ensures that evaluating any property of the beacon output requires computing the hash of the block header and learning whether or not the block is valid. Hence there can be no better malicious mining strategy for finding blocks leading to a desired beacon output than normal mining followed by computing the extractor function and potentially withholding an otherwise-valid block by neglecting to publish it. However this is very costly as it means forgoing the block reward, as we now analyze.

4 Manipulation-resistant lotteries (MRLs)

Because miners are capable of withholding valid blocks to manipulate the beacon, we introduce the concept of a *manipulation-resistant lottery* built on top of a Bitcoin-based beacon. This need not be a lottery in the sense of a random cash sweepstakes, but any random event in which an attacker may have a significant interest in the outcome. The interesting property of this game which sets it apart from previously studied concepts is the explicit attacker cost for manipulating a beacon output by paying the cost of withheld blocks (defined in Section 4.4). While a Bitcoin beacon-based lottery cannot provide unconditional security, we can argue precisely about when it would not be economically rational to manipulate the lottery based on what financial stake any party has in its result. Our goal is to design a lottery which allows participants with as high of stakes as possible to participate without incentivizing any party to manipulate the lottery.

4.1 Formal definition

Formally, we define a manipulation-resistant lottery scheme as a finite Markov process which samples from a beacon to compute its state transitions. Many different formulations would allow equivalent analysis, such as a probabilistic finite automaton, but this one is probably the simplest:

Definition 1. *A lottery scheme $\mathcal{L}$ is a tuple $\{S, P, s_0\}$:*

- *S is a set of states*
- *P is an $|S|\times|S|$ transition matrix where $P_{i,j}$ is the probability of transitioning from state S_i to state S_j. The values in each column vector P_i sum to 1.*
- *$s_0 \in S$ is the start state*

In practice, we can restrict our attention to lotteries which are *Markov trees*. Formally this means that each state has a *level* denoted $\ell(s)$ and a non-zero transition probability $P_{i,j} > 0 \implies \ell(i) + 1 = \ell(j)$. The start state has level $\ell(s_0) = 0$ and the maximum level of any state is the *depth* of the lottery $d(\mathcal{L})$. We will also call this the number of *stages*. Using the tree restriction we can define p_s for all states $s \in S$ in the tree as the probability that s will be reached from the start state using the transition matrix P.

A lottery participant is a principal who receives a reward function based on the final output of the lottery:

Definition 2. *A participant in a lottery $\mathcal{L}$ is a function $A : S \to \mathbb{Z}$, where $A(s)$ is the reward received if the lottery terminates in state s.*

The participant's expected outcome in the lottery is:

$$\mathbb{E}_{\mathcal{L}}^{A} = \sum_{s \in S | \ell(s) = d(\mathcal{L})} p_s \cdot A(s)$$

4.2 Block withholding attacks

Assuming the beacon construction is secure, the only computationally feasible method for biasing the lottery is to discard otherwise-valid blocks. This has a direct financial cost because valid blocks earn a substantial reward we will denote B. The current[5] value of B is at least[6] Ƀ25 reward worth roughly US$6,000 at today's prices. For simplicity, we can fix $B = 1$ and simply denote our value in units of the block reward.

We model a strong *bribing attacker* who is able to pay any miner exactly B to suppress a valid block whenever the attacker desires. In practice, this might mean the adversary arranges with miners to pay them $B + \epsilon$ in exchange for not publishing a block. There doesn't appear to be a means within Bitcoin to enforce such a contract, but we assume our adversary has overcome the "honor among thieves" problem.

This attacker model is strictly stronger than an attacker who is a miner themselves and can only withhold blocks (but not bribe other miners). Assuming transaction fees are significantly lower than fixed block rewards (see Appendix E for further discussion), such an attacker still faces an opportunity cost of B to withhold blocks (in the form of lost rewards), but may no longer be able to withhold a block if it is found by another miner. Likewise, even an attacker controlling a mining pool is strictly weaker than an attacker able to bribe all miners. If the mining pool contains 100% of the network's hashing power, their ability to manipulate the beacon would be equivalent to our bribing attacker.

This strong attack model also encapsulates forking attacks, where an attacker introduces a fork in the block chain in the event of an undesirable beacon output based on multiple mined blocks. We assume the attacker can announce a desired fork, pay miners in exchange for their now-orphaned blocks, and the mining community will fork at a point of the attacker's choosing.

4.3 Denial-of-service attacks

Attackers may also attempt to control the beacon result by attacking Bitcoin's P2P network, specifically by trying to prevent a block from propagating if it produces an undesirable beacon output. Bitcoin's P2P network is a largely uncontrolled and poorly-studied aspect of the network but is subject to numerous

[5] The block reward halves every 4 years, next scheduled for 2017.
[6] The full reward is slightly higher because miners also collect transaction fees.

potential attacks [7, 12, 14]. In a worst case scenario we could imagine a Dolev-Yao attacker who controls the *entire* network and propagation is only possible through this attacker. Such an attacker, if it existed, appears able to fix the beacon results by merely refusing to propagate blocks it doesn't like.

However, even this attacker would theoretically face the same opportunity cost as a block-withholding attacker described above: such a powerful attacker would be able to extract regular bribes from all miners to publish any blocks at all. In the limit this would be an ultimatum game [17] which suggests the network controller could extract a bribe of $B - \epsilon$ for any $\epsilon > 0$. Thus, refusing to propagate blocks which produced a bad network outcome would *still* cost this attacker $\sim B$ in lost rent. This analysis equally applies to any attacker standing between some miners and the network; this attacker would be able to extract large bribes for propagating blocks but could only do so from some participants.

A second attack scenario is when a "natural" collision occurs as two miners find blocks near-simultaneously (this happens in just less than 1% of all blocks). An attacker controlling many network nodes could choose to propagate blocks based on the resulting beacon output. However, this attacker would still face an opportunity cost because they could theoretically extract a bribe from the miners who had found blocks proportional to the percentage of mining power controlled by the attacker.

While we omit a formal proof, there is a theoretical equivalence between a denial-of-service attacker and a block-withholding attacker in that both face opportunity costs approaching the block reward for keeping a block from being broadcast to the network. Our *transition pruning* attack model, introduced in the next section, captures both equally.

In practice, denial-of-service attacks may be easier to execute, as they do not require money to change hands while bribing a miner to withhold blocks would. However, this form of attack would constitute an attack on Bitcoin's viability as a currency and hence likely cause re-structuring of the network. In practice, most large miners have already ceased relying solely on the public Bitcoin network and instead communicate on a shadow network of known, trusted peers [7], making the risk of denial of service attacks considerably lower.

4.4 Transition pruning

The critical capability for an attacker is the ability to *prune* any transitions from the lottery's transition matrix through block-withholding attacks. This attack comes at a cost, of course, which increases as probability of the pruned transition increases. Formally, an attacker may choose a transition $P_{i,j}$ to prune, at which point $P_{i,j}$ is set to zero and the column vector P_i is re-normalized by raising all other transition probabilities from state s_i.

This definition requires transitions to be pruned completely or not at all and does not allow the attacker to merely reduce the probability of a transition, which would correspond to an attacker being willing to suppress blocks leading to a transition with some probability $\alpha < 1$. This may appear limiting, but we prove in Appendix B.1 that it is always in the attackers' interest to prune completely

or not at all.[7] The intuition is that pruning becomes cheaper as a transition's probability increases, so if an attacker is ever incentivized to suppress a block, they will always do so from that state.

When pruning a transition which has probability p, each potential Bitcoin beacon output produced by miners is a Bernoulli trial with a probability $1-p$ of success for the attacker and a cost of $B = 1$ (the block reward) for each failure. The number of blocks the attacker will have to pay off will be geometrically distributed, with an expected value:

$$c\,(1-p) = \frac{1-(1-p)}{(1-p)} = \frac{p}{1-p} \quad \text{(pruning cost)} \tag{1}$$

4.5 Lottery manipulation

Finally, we can define a *lottery manipulation* attack and the security property of manipulation-resistance.

Definition 3. *A lottery manipulation algorithm is a function $\mathcal{A}$ which takes a lottery $\mathcal{L} = \{S, P, s_0\}$ as input and outputs a lottery $\mathcal{L}' = \{S, P', s_0\}$ with modified state transition P' after pruning some transitions.*

We can denote as $P_i^{\varnothing}$ the set of all transitions pruned from state s_i by the attacker, that is, the set of all states s_j such that $P_{i,j} > 0$ and $P'_{i,j} = 0$. Defining $\left|P_i^{\varnothing}\right|$ as the sum of the probabilities of each of these pruned transitions from state s_i, we can define the attack algorithm's expected cost:

Definition 4. *The cost $C(\mathcal{A})$ of a lottery manipulation algorithm is:*

$$C(\mathcal{A}) = \sum_{i \in S} p'_i \cdot c\left(1 - \left|P_i^{\varnothing}\right|\right)$$

where p'_i is the probability of reaching state s_i from the start state under the modified transition matrix P' and $|P_i^{\varnothing}|$ is the cumulative probability of all pruned transitions from s_i.

Finally, we can define the security property we seek to ensure for a manipulation-resistant lottery:

Definition 5. *For a lottery $\mathcal{L}$ and a participant A, we say that $\mathcal{L}$ is manipulation-resistant with respect to A if there is no attack algorithm such that:*

$$\mathbb{E}^A_{\mathcal{A}(\mathcal{L})} - C(\mathcal{A}) > \mathbb{E}^A_{\mathcal{L}}$$

Note that manipulation-resistance is defined with respect to a specific adversary. No finite lottery scheme in our model can be manipulation-resistant against

[7] A real attacker may not be able to prune completely if it cannot guarantee miners will accept its payment to withhold blocks, but we are modeling a strong attacker who is always able to bribe as desired.

an adversary with unbounded reward functions for reachable states, as they can pay an unbounded amount in block-withholding costs in order to increase their chances of reaching those states. We will also see in Section A.2 that very simple lotteries may be highly manipulation-resistant against a specific attacker but not against an attacker with opposing goals.

5 Single-stage lotteries

We apply our model here to the case of a lottery with a single level of state transitions; we analyze more complex multi-stage lotteries in Appendix A. A single-stage lottery produces output directly based on a single beacon output. We consider any participant with a binary reward function:

$$A(s) = \begin{cases} W & : s \in S_* \\ 0 & : s \notin S_* \end{cases}$$

Denoting $p = |S_*|$, equivalent to the participant's chance of winning in an unmanipulated lottery, we have:

$$\mathbb{E}^A_{\mathcal{L}} = p \cdot W \tag{2}$$

The obvious manipulation strategy is to prune all transitions to states $s \notin S_*$, which has expected cost:

$$C(\mathcal{A}) = c\,(p) = \frac{1-p}{p} \tag{3}$$

The expected reward is increased to $\mathbb{E}^A_{\mathcal{A}(\mathcal{L})} = W$ as this strategy ensures a win every time.

5.1 Security

Combining Equations 2 and 3, the attack algorithm is advantageous whenever:

$$\begin{aligned} \mathbb{E}^A_{\mathcal{A}(\mathcal{L})} - C(\mathcal{A}) &> \mathbb{E}^A_{\mathcal{L}} \\ W - \frac{1-p}{p} &> p \cdot W \\ W(1-p) &> \frac{1-p}{p} \\ W &> \frac{1}{p} \end{aligned} \tag{4}$$

Thus, a single stage lottery is manipulation-resistant against any binary attacker who has a stake less than $\frac{1}{p}$ in an event with probability p. Recall that this is expressed in units of the block reward. Thus, a single unbiased bit generated from one block is manipulation-resistant against an adversary who has less than $2B$ at stake, or over ₿50.

6 Enhanced security

6.1 Delayed output via slow hash functions

Another desirable countermeasure is to make computing the beacon output intentionally very slow. This could be achieved, for example, by an iterated hash [22,31], a memory-hard function [15,30]. There are also proposals for asymmetric slow functions [2, 20, 23] which are inherently serial (and therefore high-latency) but can be easily checked once complete. Now, a malicious miner who finds a valid block cannot quickly decide whether or not to withhold it. During this time, any other miner may find and publish a valid block before them, potentially causing the miner to lose the financial reward as well as any influence on the beacon. Additionally, the function takes on the order of tens of minutes to compute, the attack cost is increased significantly as changing the block at that point requires a deeper fork of the network.

For a simple iterated hash function, this construction may offer a poor security trade-off if it pits honest parties wishing to sample from a secure beacon, who may not control any significant financial resources, against professional miners. For example, a 2^{40} computation may be sufficiently costly to prevent legitimate parties from using the beacon, whereas many miners can compute this in a matter of seconds.[8] Thus we believe an asymmetric construction is necessary [2, 20, 23], which remains an active area of research.

6.2 Random inputs from trusted delegates

For a beacon run by one or more semi-trusted delegates, such as election administrators the beacon output can be supplemented by requesting a random input from each of the delegates $D_0, D_1 \ldots$. Each delegate must commit to their random nonce when the beacon's parameters are published. That is, each D_i published $\mathsf{commit}s(x_i)$ for a random nonce x_i. The beacon result is then:

$$\mathsf{Beacon}(t) = \mathsf{Ext}_k\left(B_t||x_0||x_1||\ldots\right)$$

This means that the result cannot be computed until all delegates share their nonce. If at least one delegate keeps their nonce private until well after block B_t has been committed to the block chain, it will then be too late for miners to attempt to manipulate the beacon result. In an election scenario, delegates might include the election authorities and representatives from each candidate.

Of course, this requires that all delegates faithfully reveal their commitment-failure to do so will consist of a denial of service attack on the beacon as the result will not be computable. This may be feasible with a single delegate (the beacon authority itself) or a small number whose reputation will be irreparably harmed if they fail to reveal their nonce. Alternately, it is possible for them to post funds in escrow in Bitcoin which will be lost if they don't reveal their nonce in a timely manner [3] as we will discuss further in Section 8.1, but this requires posting a large amount of capital which we would like to avoid.

[8] This may not hold true for all miners, as this construction requires a sequential hash whereas miners optimize to compute parallel hashes.

7 Practical considerations

The number of potential applications of beacons is huge, as we alluded to in the introduction. Beacons are are also useful for defining public coin client puzzles for DoS and spam mitigation [18, 37] and for enabling efficient Byzantine agreement protocols in large-scale distributed systems [28], among many diverse applications. A canonical application is election auditing, where a beacon is used to choose specific precincts and/or ballots for auditing. A secure beacon is required to ensure that election officials do not attempt to steer the audit away from known discrepancies in the tally (which happened in 2004 in Cuyahoga County, Ohio [21]). In this section we will discuss some practical considerations based on the experience of deploying beacons for election protocols.

7.1 Min-entropy requirements

While no current jurisdiction directly specifies a required amount of min-entropy required from a beacon used for auditing, many jurisdictions do mandate that a random selection of ballots are recounted post-election to statistically confirm the published tally. For example, the State of California requires an audit of 1% of precincts, while Colorado requires a risk-limiting audit that sustains the recount until statistical confidence is established. In practice, even 32 bits of min-entropy as provided by the Bitcoin beacon, would be vastly more than is needed to ensure that all precincts have a close to 1% chance of being audited.

7.2 Public confidence

Any beacon construction must be able to garner public belief in its fairness. While this is difficult to judge and using Bitcoin may seem outlandish due to the protocol's novelty, there is actually a precedent for using Bitcoin in cryptographically verifiable elections. In the 2011 municipal election at Takoma Park, MA, the block chain was used to provide unforgeable proof that pre-election commitments were actually made prior to the election even when the verifier only sees the values after the election is complete [10].

7.3 Time uncertainty

Bitcoin blocks are not published at fixed time intervals, but are randomly found in a Poisson process. For scenarios such as election auditing, the authorities would have to commit to beacon parameters in advance of the election with confidence that the beacon results would not be available until voting is closed.

The time until the next block is found is exponentially distributed while the amount of time to find n blocks follows an Erlang distribution (a special case of the Gamma distribution). Thus if a beacon output is needed in the real world after time t, a suitable Bitcoin block index i to sample the beacon from must be chosen such that block i won't be found until after time t with high confidence.

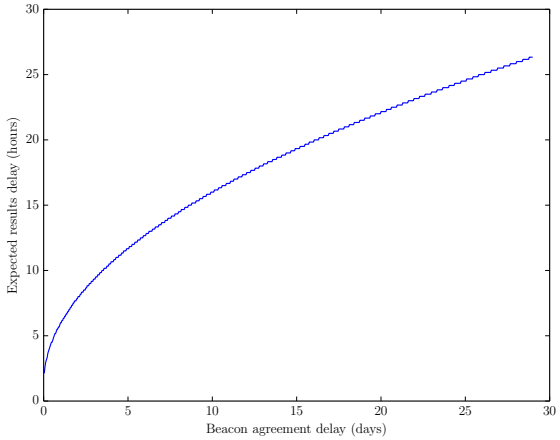

Fig. 1. Required time delay as a function of how far in advance the beacon parameters are published. We assume 99% confidence is required that the beacon results will not be available early and a standard 6 block waiting time to confirm the results is required.

We plot the expected latency added for $p = 99\%$ confidence in Figure 1. Compared to other decentralized constructions, such as the stock-market data which imposes an entire trading day's delay, this is relatively modest.

For an election, if a beacon parameters must be published by the authorities 24 hours before the beacon results, then to maintain 99% confidence that the beacon result will not be available early requires adding an additional ≈ 3 hour delay to the mean time the block will be found. Given the auditing will probably commence the day after the election, we consider this acceptable.

8 Integration into smart contracts

In addition to using the Beacon as a source of randomness for arbitrary applications, there are also applications within the Bitcoin protocol itself which could benefit from the availability of a beacon. Bitcoin transactions are implemented using a simple scripting language which specifies the conditions under which funds can be moved. Currently, there is no opcode in the Bitcoin scripting language which allows for any type of random execution, as this would require miners to provide randomness themselves when evaluating the script and this would unverifiable. This is even true for more cryptocurrencies with more powerful scripting languages, such as Ethereum [38]. As a result, many applications require a multi-stage randomness protocol with participants committing to nonces and revealing them, requiring bonds to ensure completion of the protocol.

We propose instead using our blockchain-based randomness protocol to make public randomness available through a simple call. In Bitcoin, this could take the form of a new script opcode which we propose as `OP_BEACON`.The technical details of our proposal and an example script are included in Appendix C. We

stress that the creation of `OP_BEACON` is only necessary for the creation of smart contracts. Our core Beacon construction can be computed immediately without any modification to the block chain.

We highlight several applications of an `OP_BEACON` instruction. In each case, the mechanics of Bitcoin script allow us to build transactions which serve as smart contracts in the sense that the security properties are self-enforcing.

8.1 Multi-player lotteries

Andrychowicz et al. [3] demonstrated a method to create a multiparty lottery in Bitcoin without using trusted authorities. In their solution, all parties publish a commitment to a random number and a *bond* consisting of funds which are frozen in escrow until that random number is revealed. These escrow bonds must be greater than the prize money which is also held in escrow. Their protocol also requires publishing multiple complex transactions in multiple stages.

Using `OP_BEACON`, we can perform a secure lottery with a much simpler construction and without putting any additional funds in escrow. In the two-party case, Alice proposes to Bob a wager transaction which will (eventually) be claimable by one of them at random based on a beacon output. The redemption script for the wager redeems funds from each of Alice and Bob, samples from the beacon and, based on the beacon output, will allow either a signature from Alice or Bob to redeem the transaction and claim the entire amount wagered. The key which is selected to be able to redeem the transaction is effectively the winner. If the wager transaction is mutually agreeable, Alice and Bob may each sign it at which point their funds will be effectively held in escrow until the beacon has been sampled. The odds of the wager can be set by comparing the beacon output to any arbitrary threshold and the protocol is easily extensible to multiple parties. We provide an example implementation of our protocol in Bitcoin script with our proposed new opcode in Appendix C.1.

Our protocol is efficient, requiring only one short transaction, and requires no money in escrow beyond that being wagered. While the techniques of Andrychowicz et al. are more powerful and support general multi-party-computations, for multi-party lotteries our protocol utilizing an explicit beacon instruction is significantly simpler and more efficient.

8.2 Non-interactive cut-and-choose

Another protocol enabled by `OP_BEACON` is secure, non-interactive cut-and-choose. Suppose Peggy wants to prove that she knows a specific random hash preimage. Peggy can publish a special Bitcoin transaction containing bond money and k values $\{H(x_1), \ldots, H(x_k)\}$ which, using randomness obtained through `OP_BEACON`, requires a random subset of $k-1$ of the x_i values to be revealed in order to be redeemed. After the beacon output, if Peggy wants to reclaim her bond money she must publish the randomly-chosen x_i values in her transaction reclaiming her bond. For sufficiently large k (perhaps using multiple hierarchical

rounds), this provides high confidence that Peggy knows x_i for the value not required to be revealed by the cut-and-choose protocol.

It has already been argued that the Fiat-Shamir heuristic is generally not suitable to making cut-and-choose protocols non-interactive due to the relatively small space of outcomes [11], making a beacon the only known construction for non-interactive cut-and-choose. Using Bitcoin we can also make this protocol self-enforcing in that an arbitrary penalty can be paid if the prover aborts the protocol. We believe this is the first proposal for a cut-and-choose scheme that is both self-enforcing and non-interactive.

8.3 Randomized mixing fees

The use of Bitcoin blocks for randomness was proposed in Mixcoin [8], a protocol for building an accountable mix which uses randomized all-or-nothing mixing fees by hashing future values of the block chain. Essentially, this is a beacon construction. Mixcoin would not benefit from `OP_BEACON` directly, since Mixcoin contracts are not implemented in Bitcoin script, but we can analyze the scale of randomized mixing fees which could be securely collected using a one-stage lottery. Mixcoin suggests mixing fee rates on the order of $\tau = 1\%$, which would be manipulation-resistant (from Equation 4) whenever the chunk size $c > \frac{1}{1-\tau} \approx 1.01B$, or over Ƀ25. This is orders of magnitude higher than what is typically used for mixing, implying that a Bitcoin-based beacon can be easily made manipulation-resistant for randomized mixing fees. Mixes should of course use a different extractor key (nonce) for each outstanding contract to ensure they can't efficiently manipulate the outcome for multiple clients simultaneously.

Using `OP_BEACON`, we can also extend this idea to incorporate randomized mixing fees in CoinSwap [24], a "trustless" mixing protocol which mixes funds through a third-party using multiple transactions to eliminate the possibility of theft. A simple modification to the CoinSwap protocol would add a call to `OP_BEACON` which, with probability τ, would allow the mix to retain the user's funds. This would encourage for-profit, high availability mixing services to operate while retaining the anonymity benefits of consistent transaction sizes, solving an important problem with CoinSwap.

9 Concluding remarks

Bitcoin is facilitating of a remarkable number of interesting security protocols, such as secure timestamping [10] and multi-party computation [3]. To this we add a public randomness beacon, for which Bitcoin provides an unprecedented opportunity to build a highly available beacon which has a convincing cryptographic argument of security with no trusted third parties.

We hope our work will renew interest in cryptographic beacons, for which there are a vast number of applications. In the long run, we consider it an important concept to promote to the general public, given that it has considerable potential to increase transparency and accountability for a number of processes

that today rely on difficult-to-audit physical generation of randomness or make no attempt to establish accountability at all. The social utility of public randomness also may be an interesting counterpoint to arguments against the "wasteful" hashing underlying Nakamoto consensus.

References

1. Bitcoin mega lottery. `https://bitcoinmegalottery.com/faq`, 2014.
2. Giulia Alberini, Tal Moran, and Alon Rosen. Public verification of private effort. Cryptology ePrint Archive, Report 2014/983, 2014. `http://eprint.iacr.org/`.
3. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. Secure Multiparty Computations on BitCoin. *IEEE Symposium on Security and Privacy*, 2014.
4. Elaine Barker, John Kelsey, et al. Special Publication 800-90A: Recommendation for Random Number Generation Using Deterministic Random Bit Generators. *NIST*.
5. Michael Ben-Or, Oded Goldreich, Silvio Micali, and Ronald L Rivest. A fair protocol for signing contracts. *IEEE Transactions on Information Theory*, 1990.
6. Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Research Perspectives and Challenges for Bitcoin and Cryptocurrencies (Extended Version). Cryptology ePrint Archive, Report 2015/261, 2015.
7. Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. SoK: Bitcoin and second-generation cryptocurrencies. In *2015 IEEE Symposium on Security and Privacy*, May 2015.
8. Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll, and Edward W. Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. *Financial Cryptography and Data Security (FC)*, 2014.
9. Richard Carback, David Chaum, Jeremy Clark, John Conway, Aleksander Essex, Paul S. Hernnson, Travis Mayberry, Stefan Popoveniuc, Ronald L. Rivest, Emily Shen, Alan T. Sherman, and Poorvi L. Vora. Scantegrity II municipal election at Takoma Park: The first E2E binding governmental election with ballot privacy. *Usenix Security Symposium*, 2010.
10. Jeremy Clark and Aleksander Essex. Commitcoin: carbon dating commitments with bitcoin. *Financial Cryptography and Data Security (FC)*, 2012.
11. Jeremy Clark and Urs Hengartner. On the Use of Financial Data as a Random Beacon. *Usenix EVT/WOTE*, 2010.
12. Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *IEEE P2P*, pages 1–10. IEEE, 2013.
13. Yevgeniy Dodis, Rosario Gennaro, J Hastad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the CBC, Cascade and HMAC modes. *Advances in Cryptology (CRYPTO)*, 2004.
14. Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartı. The Bitcoin P2P network. In *BITCOIN'14: The First Workshop on Bitcoin Research*, January 2014.
15. Cynthia Dwork, Andrew Goldberg, and Moni Naor. On memory-bound functions for fighting spam. In *Advances in Cryptology-Crypto 2003*, pages 426–444. Springer, 2003.
16. Christina Garman, Matthew Green, Ian Miers, and Aviel D Rubin. Rational Zero: Economic Security for Zerocoin with Everlasting Anonymity. In *BITCOIN*, 2014.
17. Werner Güth, Rolf Schmittberger, and Bernd Schwarze. An experimental analysis of ultimatum bargaining. *Journal of economic behavior & organization*, 3(4):367–388, 1982.
18. J Alex Halderman and Brent Waters. Harvesting verifiable challenges from oblivious online sources. *ACM Conference on Computer and Communications Security (CCS)*, 2007.

19. Jennifer Huergo. Nist removes cryptography algorithm from random number generator recommendations. *NIST announcement*, April 2007.
20. Yves Igor Jerschow and Martin Mauve. Modular square root puzzles: Design of non-parallelizable and non-interactive client puzzles. *Computers & Security*, 35:25–36, 2013.
21. Doug Jones and Barbara Simons. *Broken Ballots: Will Your Vote Count?* Center for the Study of Language and Information, 2012.
22. John Kelsey, Bruce Schneier, Chris Hall, and David Wagner. Secure applications of low-entropy keys. In *Information Security*, pages 121–134. Springer, 1998.
23. Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015. `http://eprint.iacr.org/`.
24. Gregory Maxwell. CoinSwap: Transaction graph disjoint trustless trading, October 2013.
25. Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. *IEEE Symposium on Security and Privacy*, 2013.
26. Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009.
27. Noam Nisan and Amnon Ta-Shma. Extracting randomness: A survey and new constructions. *Journal of Computer and System Sciences*, 58(1), 1999.
28. Olumuyiwa Oluwasanmi and Jared Saia. Scalable byzantine agreement with a random beacon. *Stabilization, Safety, and Security of Distributed Systems*, 2012.
29. Rene Peralta et al. NIST Randomness Beacon. 2011.
30. Colin Percival and Simon Josefsson. The scrypt password-based key derivation function. 2015.
31. Niels Provos and David Mazieres. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, pages 81–91, 1999.
32. Michael O Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.
33. Ronen Shaltiel. An introduction to randomness extractors. In *Automata, languages and programming*, pages 21–41. Springer, 2011.
34. Dan Shumow and Niels Ferguson. On the possibility of a back door in the NIST SP800-90 Dual EC PRNG. *Microsoft Lecture*, 2007.
35. Luca Trevisan and Salil Vadhan. Extracting randomness from samplable distributions. In *41st Annual Symposium on Foundations of Computer Science.*, pages 32–42. IEEE, 2000.
36. Salil P Vadhan. Randomness extractors and their many guises. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, page 9. IEEE, 2002.
37. Brent Waters, Ari Juels, J. Alex Halderman, and Edward W. Felten. Dos resistance: New client puzzle outsourcing techniques for dos. *ACM Conference on Computer and Communications Security (CCS)*, 2004.
38. Gavin Wood. Ethereum: A secure decentralized transaction ledger. `http://gavwood.com/paper.pdf`, 2014.

A Multi-stage lotteries

A.1 Symmetry

The single-stage construction introduced in Section 5 that it is completely *symmetric*, that is, it is equivalently manipulation-resistant against an attacker in-

terested in any outcome with probability p. In general, this will not be true for multi-stage lotteries as we will show. We can formalize this as:

Definition 6. *Consider a lottery $\mathcal{L}$ with two players A, A' having binary stakes W, W' on outcomes with probabilities p, p'. We call $\mathcal{L}$ **symmetric** with respect to A, A' if, whenever $W \cdot p = W' \cdot p'$, $\mathcal{L}$ is manipulation-resistant against A if and only if it is manipulation-resistant against A'.*

Intuitively, this shows that if two attackers have the same expected value in an honest lottery, there should be equivalent security against both. The single-stage lottery is in fact symmetric against any two possible adversaries, which we can define as a strong property:

Definition 7. *We call $\mathcal{L}$ **completely symmetric** if is symmetric with respect to any pair of players with binary outcomes on events $o, o' \in O$, the space of possible labeled outputs of the lottery.*

A.2 Asymmetric lotteries

In designing a multi-stage lottery, we first consider the case of designing for manipulation-resistance only against a single possible binary attacker with a stake in some outcome with probability p. We are explicitly not designing against any other attacker, enabling an asymmetric design. In Appendix D, we provide a natural example of where an asymmetric lottery is useful, namely Ben-Or et al.'s fair-contract signing protocol [5].

The key insight is to minimize the chance of outputting a winning outcome for the attacker from any any individual state, since this would be the easiest state to prune transitions from. To achieve this, building a lottery with depth N, we will guarantee a winning outcome for the attacker (and effectively terminate early) after state s_i with probability q_i (drawn based on the randomness from that block), otherwise we will move to the next state s_j. Only from state s_N, with probability $(1 - q_N)$, will we output a losing outcome. This enables each state to have as large of a transition as possible that the attacker would prefer to prune, making manipulation as expensive as possible. While we generally speak of a tree of states, here there are exactly N states in a linear sequence, each corresponding to a single beacon output.

Transition probabilities The second important insight is to balance the lottery. That is, at any moment the attacker should be exactly ambivalent between attacking during the current state or waiting until the next state. If this were not the case, this would mean one of the two states is more profitable for the attacker and hence the construction would not be optimal. Denoting the probability of outputting a winning outcome from state s_i as q_i (with a $(1 - q_i)$ probability of

transitioning to state s_{i+1}, we can derive the constraint that:

$$\begin{aligned} c\left(q_i\right) &= (1-q_i)c\left(q_{i+1}\right) \\ \frac{1-q_i}{q_i} &= (1-q_i)\frac{1-q_{i+1}}{q_{i+1}} \\ \frac{1}{q_i} &= \frac{1-q_{i+1}}{q_{i+1}} \\ q_i &= \frac{q_{i+1}}{1-q_{i+1}} \end{aligned} \tag{5}$$

We can also re-arrange this to obtain:

$$\begin{aligned} \frac{1}{q_i} &= \frac{1-q_{i+1}}{q_{i+1}} \\ \frac{1+q_i}{q_i} &= \frac{1}{q_{i+1}} \\ q_{i+1} &= \frac{q_i}{1+q_i} \end{aligned} \tag{6}$$

In Lemma 1 we obtain a simple closed-form for every q_i in terms of q_1.

Lemma 1. *In a balanced asymmetric lottery, the probability of outputting a winning result in state s_i is $q_i = \frac{q_1}{1+(i-1)q_1}$ for all $i \geq 1$, where q_1 is the probability of outputting a winning result in the first block.*

Proof. We can prove this by weak induction on i. In the base case of $i = 1$, this is true by identity:

$$\begin{aligned} q_i &= \frac{q_1}{1+(i-1)q_1} \\ q_1 &= \frac{q_1}{1+(1-1)q_1} \\ q_1 &= q_1 \end{aligned}$$

In the inductive case, assuming the formula is correct for $(i-1)$ we can prove it is true for i using Equation 6:

$$\begin{aligned} q_i &= \frac{q_{i-1}}{1+q_{i-1}} \\ q_i &= \frac{\frac{q_1}{1+(i-2)q_1}}{1+\frac{q_1}{1+(i-2)q_1}} \quad \text{(using ind. hypothesis)} \\ q_i &= \frac{q_1}{1+(i-2)q_1+q_1} \\ q_i &= \frac{q_1}{1+(i-1)q_1} \end{aligned}$$

Therefore the result is true for all $i \geq 1$. □

Choosing q_1 Given the above formula, we only need to find the value of q_1 such that the overall probability of a winning output is p. This imposes the constraint that:

$$\begin{aligned}
p_{\text{total}} &= q_1 + (1-q_1)q_2 + (1-q_1)(1-q_2)q_3 + \dots \\
&= \sum_{i=1}^{N} \left(q_i \cdot \prod_{j=1}^{i-1} (1-q_1) \right) \\
&= \sum_{i=1}^{N} \left(q_i \cdot \prod_{j=1}^{i-1} \left(1 - \frac{q_1}{1+(j-1)q_1} \right) \right) \\
&= \sum_{i=1}^{N} \left(q_i \cdot \prod_{j=1}^{i-1} \left(\frac{1+(j-2)q_1}{1+(j-1)q_1} \right) \right) \\
&= \sum_{i=1}^{N} \left(q_i \cdot \frac{1-q_1}{1+(i-2)q_1} \right) \\
&= \sum_{i=1}^{N} \frac{q_1}{1+(i-1)q_1} \cdot \frac{1-q_1}{1+(i-2)q_1} \\
&= q_1(1-q_1) \sum_{i=1}^{N} \frac{1}{(1+(i-1)q_1) \cdot (1+(i-2)q_1)} \qquad (7)
\end{aligned}$$

At this point, the summation on the right side appears intractable, but it actually reduces to a simple closed-form:

Lemma 2. *For all integers $N \geq 3$, we have:*

$$\sum_{i=3}^{N} \frac{1}{(1+(i-1)x)\,(1+(i-2)x)} = \frac{N-2}{(1+x)\,(1+(N-1)x)}$$

Proof. We can prove this by weak induction on N. This is easy to show for the base case of $N = 3$:

$$\begin{aligned}
&\sum_{i=3}^{3} \frac{1}{(1+(i-1)x) \cdot (1+(i-2)x)} \\
=&\frac{1}{(1+(3-1)x) \cdot (1+(3-2)x)} \\
=&\frac{3-2}{(1+x) \cdot (1+(3-1)x)}
\end{aligned}$$

In the inductive case, assuming the property is true for $N-1$, we can show it is true for N:

$$\sum_{i=3}^{N} \frac{1}{(1+(i-1)x) \cdot (1+(i-2)x)}$$

$$
\begin{aligned}
&= \left(\sum_{i=3}^{N-1} \frac{1}{(1+(i-1)x) \cdot (1+(i-2)x)} \right) + \dots \\
&= \frac{N-3}{(1+x)\,(1+(N-2)x)} + \frac{1}{(1+(N-1)x)\,(1+(N-2)x)} \\
&= \frac{(N-3)\,(1+(N-1)x) + 1 + x}{(1+x) \cdot (1+(N-1) \cdot x) \cdot (1+(N-2) \cdot x)} \\
&= \frac{(N-2)\,(1+(N-1)x) - (1+(N-1)x) + 1 + x}{(1+x) \cdot (1+(N-1) \cdot x) \cdot (1+(N-2) \cdot x)} \\
&= \frac{(N-2)\,(1+(N-1)x) - Nx + 2x}{(1+x) \cdot (1+(N-1) \cdot x) \cdot (1+(N-2) \cdot x)} \\
&= \frac{(N-2)\,(1+(N-1)x) - (N-2)x}{(1+x) \cdot (1+(N-1) \cdot x) \cdot (1+(N-2) \cdot x)} \\
&= \frac{(N-2)\,(1+(N-2)x)}{(1+x) \cdot (1+(N-1) \cdot x) \cdot (1+(N-2) \cdot x)} \\
&= \frac{(N-2)}{(1+x) \cdot (1+(N-1) \cdot x)}
\end{aligned}
$$

Therefore the result is true for all $N \geq 3$. □

We can plug this formula into Equation 7 and reduce to:

$$
\begin{aligned}
p_{\text{total}} &= q_1(1-q_1) \sum_{i=1}^{N} \frac{1}{(1+(i-1)q_1) \cdot (1+(i-2)q_1)} \\
&= q_1 + \frac{q_1(1-q_1)}{1+q_1} + \sum_{i=3}^{N} \frac{1}{(1+(i-1)q_1)\,(1+(i-2)q_1)} \\
&= q_1 + \frac{q_1(1-q_1)}{1+q_1} + \frac{q_1(1-q_1)(N-2)}{(1+q_1) \cdot (1+(N-1) \cdot q_1)} \\
&= \frac{2q_1}{1+q_1} + \frac{q_1(1-q_1)(N-2)}{(1+q_1) \cdot (1+(N-1) \cdot q_1)} \\
&= \frac{2q_1 \cdot (1+(N-1) \cdot q_1) + q_1(1-q_1)(N-2)}{(1+q_1) \cdot (1+(N-1) \cdot q_1)} \\
&= \frac{2q_1 \cdot (1+(N-1) \cdot q_1) + q_1(1-q_1)(N-2)}{(1+q_1) \cdot (1+(N-1) \cdot q_1)} \\
&= \frac{2q_1 + 2Nq_1^2 - 2q_1^2 + Nq_1 - Nq_1^2 - 2q_1 + 2q_1^2}{(1+q_1) \cdot (1+(N-1) \cdot q_1)} \\
&= \frac{Nq_1^2 + Nq_1}{(1+q_1) \cdot (1+(N-1) \cdot q_1)} \\
&= \frac{Nq_1}{(1+(N-1) \cdot q_1)} \qquad (8)
\end{aligned}
$$

Given constraints $p_{\text{total}} = p$ and N, we can solve for q_1:

$$\begin{aligned} p &= \frac{Nq_1}{(1 + (N-1) \cdot q_1)} \\ p\,(1 + (N-1) \cdot q_1) &= Nq_1 \\ p + pNq_1 - pq_1 - Nq_1 &= 0 \\ q_1(pN - p - N) &= -p \\ q_1 &= \frac{p}{N + p - pN} \\ q_1 &= \frac{p}{(1-p)N + p} \end{aligned} \tag{9}$$

We can also derive a general equation for each q_i using our formula from Lemma 1:

$$\begin{aligned} q_i &= \frac{q_1}{1 + (i-1)q_1} && \text{(Lemma 1)} \\ q_i &= \frac{\frac{p}{(1-p)N+p}}{1 + (i-1)\frac{p}{(1-p)N+p}} && \text{(Eq. 9)} \\ q_i &= \frac{p}{(1-p)N + p + (i-1)p} \\ q_i &= \frac{p}{(1-p)N + ip} \end{aligned} \tag{10}$$

Security Given our formula for the optimal q_1 in Equation 9, we can now compute the cost of manipulation. Recall that our design ensured that pruning any transition is equivalently beneficial as pruning any other. Thus, we can analyze the cost of the simplest manipulation algorithm which forces a winning output from the very first state:

$$\begin{aligned} C(\mathcal{A}) &= c\,(q1) \\ &= \frac{1 - q1}{q1} \\ &= \frac{1 - \frac{p}{N+p-pN}}{\frac{p}{N+p-pN}} \\ &= \frac{N + p - pN - p}{p} \\ &= \frac{N - pN}{p} \\ &= N \cdot \frac{1-p}{p} \\ &= N \cdot c\,(p) \end{aligned} \tag{11}$$

This result shows that, regardless of p, we can get an exactly N-fold increase in manipulation-resistance by building a lottery with N stages. An intuitive

explanation for this is to consider a slightly stronger attacker model who can not only pay miners to withhold blocks, but can retroactively pay to eliminate published blocks. Such an attacker, for any N-stage construction, could simply allow N blocks to be mined as usual and then pay to retract all of them if they result in a losing lottery outcome. The cost of this manipulation would also be $N \cdot c\,(p)$, as each failed lottery run would now require paying the cost of all N blocks used. This suggests that $N \cdot c\,(p)$ is a natural limit and indicates that our asymmetric construction is optimal.

Security against inverse attacker Unfortunately, this design only provides increased security against manipulation by one party. If we design for security against an attacker trying to force a 1-bit outcome of `0`, the construction provides no additional security against an attacker motivated to force an outcome of `1`. In fact the construction is slightly weaker than a single-stage lottery against such an *inverse attacker* A'. Though this attacker is paying to prune a transition from every state, these transitions are all low-probability and hence the total number of expected blocks to pay off is very low:

$$\begin{aligned}
C(\mathcal{A}') &= \sum_{i}^{N} c\,(1 - q_i) \\
&= \sum_{i}^{N} c\left(1 - \frac{p}{(1-p)N + ip}\right) \\
&= \sum_{i}^{N} c\left(\frac{(1-p)N + (i-1)p}{(1-p)N + ip}\right) \\
&= \sum_{i}^{N} \frac{1 - \frac{(1-p)N+(i-1)p}{(1-p)N+ip}}{\frac{(1-p)N+(i-1)p}{(1-p)N+ip}} \\
&= \sum_{i}^{N} \frac{(1-p)N + ip - ((1-p)N + (i-1)p)}{(1-p)N + (i-1)p} \\
&= \sum_{i}^{N} \frac{p}{(1-p)N + (i-1)p} \qquad (12)
\end{aligned}$$

We can show by inequality that this is worse than the expected cost of an attack against a 1-stage lottery:

$$\begin{aligned}
C(\mathcal{A}') &= \sum_{i}^{N} \frac{p}{(1-p)N + (i-1)p} \\
&= p \cdot \sum_{i}^{N} \frac{1}{(1-p)N + (i-1)p}
\end{aligned}$$

$$
\begin{aligned}
&< p \cdot \sum_{i}^{N} \frac{1}{(1-p)N} \\
&= \frac{p}{1-p} \cdot \sum_{i}^{N} \frac{1}{N} \\
&= \frac{1-(1-p)}{1-p} \\
&= c\,(1-p) \qquad (13)
\end{aligned}
$$

Combined with a lower bound from Equation 12 we have:

$$p < C(\mathcal{A}') < \frac{p}{1-p} = c\,(1-p) \qquad (14)$$

A.3 Extended symmetric lotteries

Next we explore optimal multi-stage lottery constructions that are symmetric. We will demonstrate a 2-stage lottery that is optimal and symmetric against a set of k attackers who each win a binary reward with a set of mutually exclusive outcomes of probability $\frac{1}{k}$.

To do this, we introduce a general theorem about when an optimal attack algorithm should prune (or not prune) transitions, which we prove in Appendix B.2.

Theorem 1. *Consider a lottery manipulation algorithm $\mathcal{A}$ which has expected value $\mathbb{E}_i$ from state s_i in a lottery. For a state transition $P_{i,j}$ which $\mathcal{A}$ doesn't prune, a modified algorithm $\mathcal{A}'$ which does prune $P_{i,j}$ will produce a higher expected value if and only if:*

$$\mathbb{E}_i + 1 > \mathbb{E}_j$$

By symmetry, $\mathcal{A}'$ will be superior if it doesn't prune a transition $P_{i,k}$ which $\mathcal{A}$ does prune if and only if:

$$\mathbb{E}_i + 1 < \mathbb{E}_k$$

Now we can design our two-stage, k-player lottery. The insight is to motivate any attacker to prune branches in both stages. By setting each of k transitions from the start state to have $P_{*,j} = \frac{1}{k}$, we can ensure the construction is at least as manipulation-resistant as a one-stage lottery.

Thus we want k transitions in from the start state, each to a state that is slightly more favorable to one player and less favorable to all others, with probability $q > \frac{1}{k}$ to output symbol o and $\frac{1-q}{k-1}$ to output each of $k-1$ other symbols. We want to set q as high as possible to require pruning in the first round, but no higher to keep pruning in the second round as expensive as possible.

Theorem 1 lets us enumerate this constraint precisely. If we call the expected value for player A in one of the $k-1$ stages not favorable to A as $\mathbb{E}_{-}^{A}$, we have:

$$\begin{aligned}
\mathbb{E}_{*}^{A} &= \mathbb{E}_{-}^{A} + 1 \\
W - c\left(\frac{1}{k}\right) - c\left(q\right) + 1 &= W - c\left(\frac{1-q}{k-1}\right) \\
\frac{1-\frac{1}{k}}{\frac{1}{k}} + \frac{1-q}{q} + 1 &= \frac{1-\frac{1-q}{k-1}}{\frac{1-q}{k-1}} \\
k - 1 + \frac{1}{q} &= \frac{k+q-2}{1-q} \\
k + \frac{1}{q} &= \frac{k-1}{1-q} \\
\frac{kq+1}{q} &= \frac{k-1}{1-q} \\
kq + 1 - kq^2 - q &= kq - q \\
kq^2 &= 1 \\
q &= \frac{1}{\sqrt{k}}
\end{aligned} \tag{15}$$

Security Using the optimal value of q derived in Equation 15, we can compute the expected attack cost, recalling that the optimal manipulation strategy for all attackers is to prune in both stages:

$$\begin{aligned}
C(\mathcal{A}) &= c\left(\frac{1}{k}\right) + c\left(\frac{1}{\sqrt{k}}\right) \\
&= \frac{1-\frac{1}{k}}{\frac{1}{k}} + \frac{1-\frac{1}{\sqrt{k}}}{\frac{1}{\sqrt{k}}} \\
&= k + \sqrt{k} - 2
\end{aligned} \tag{16}$$

For a single-stage lottery, the cost is simply $k-1$. Thus we can definitively increase security with additional stages even in a symmetric lottery, albeit we can't obtain a linear increase as was possible with an asymmetric construction.

Extension to additional stages We can use this design strategy to extend the construction to $N > 2$ stages, adding transitions from the second-stage states into states with higher and higher probabilities of outputting one outcome compared to all others. Unfortunately, there does not appear to be a simple closed-form solution for the transition probabilities beyond 2-stages, as Equation 15 is replaced with a quadratic equation with a non-square discriminant. We have implemented and solved this equation by numerical approximation and observed empirically that an N stage lottery can be implemented, albeit with a cost of manipulation $C(\mathcal{A}) \in O(\lg N)$.

Limits against an unknown attacker Given the above construction, it may seem wise to simply choose a very large k, such as $k = 2^{128}$, making it equally difficult to manipulate the lottery to produce any bit string. The goal would be to render any attacker, regardless of their goal, unable to manipulate the beacon cost-effectively.

Unfortunately this approach doesn't work and ends up producing a lottery that it is only as manipulation-resistant as a one-stage lottery. Specifically, it fails as $k \to \infty$ against any attacker who has a binary reward in any outcome with non-negligible probability p. We can see this by considering the cost to an attacker who simply waits until the last round of the above construction before attacking:

$$\begin{aligned}
&\lim_{k\to\infty} C(\mathcal{A}) = \\
&= p \cdot c\left(\frac{1}{\sqrt{k}} + (pk-1)\frac{1-\frac{1}{\sqrt{k}}}{k-1}\right) + (1-p)c\left((pk)\frac{1-\frac{1}{\sqrt{k}}}{k-1}\right) \\
&= p \cdot c\,(p) + (1-p)c\,(p) \\
&= c\,(p)
\end{aligned}$$

An intuitive explanation is that if we have no way of determining what the attacker's goal function is then surely they will almost always reach a penultimate state with a roughly p chance of outputting a winning result; therefore the attacker can simply wait and attack in the last block. This is an important limitation to multi-stage lotteries: we are only able to obtain greater security than a 1-stage lottery by ensuring the attacker always reaches a final state which has a probability higher or lower than p of outputting a winning result, which on expectation is costlier to manipulate than single-stage lottery. This becomes impossible as the number of potential attacker goals grows infinite.

This result highlights that while multi-stage lotteries are possible, they must be carefully designed for a specific threat model and are not a panacea.

B Additional proofs

B.1 Proof that partial pruning is sub-optimal

In Section 4.5 we claimed the following theorem:

Theorem 2. *An optimal lottery manipulation algorithm will always either reduce the probability of a transition $P_{i,j}$ to 0 or not reduce it at all.*

Proof. Denote the algorithm's expected value upon reaching state s_j as $\mathbb{E}_j$ and the expected value upon taking any other transition from state s_i as $\mathbb{E}'_j$. Now assume that an optimal attack algorithm were to reduce the transition probability

$P_{i,j}$ to $\alpha \cdot P_{i,j}$ for $\alpha > 0$. This would imply that:

$$\begin{aligned}
\mathbb{E}_{\text{prune}} &< \mathbb{E}_{\text{partial}} \\
\mathbb{E}'_j - c\,(1 - P_{i,j}) &< (1 - \alpha \cdot P_{i,j})\,\mathbb{E}'_j + \alpha \cdot P_{i,j} \cdot \mathbb{E}_j - c\,(1 - \alpha P_{i,j}) \\
-c\,(1 - P_{i,j}) &< \alpha \cdot P_{i,j}\left(\mathbb{E}_j - \mathbb{E}'_j\right) - c\,(1 - \alpha \cdot P_{i,j}) \\
-\frac{1 - (1 - P_{i,j})}{(1 - P_{i,j})} &< \alpha \cdot P_{i,j}\left(\mathbb{E}_j - \mathbb{E}'_j\right) - \frac{1 - (1 - \alpha \cdot P_{i,j})}{(1 - \alpha \cdot P_{i,j})} \\
-\frac{1}{(1 - P_{i,j})} &< \alpha\left(\mathbb{E}_j - \mathbb{E}'_j\right) - \frac{\alpha}{(1 - \alpha \cdot P_{i,j})} \\
-\frac{1}{(1 - \alpha P_{i,j})} &< \alpha\left(\mathbb{E}_j - \mathbb{E}'_j\right) - \frac{\alpha}{(1 - \alpha \cdot P_{i,j})} \\
0 &< \alpha\left(\mathbb{E}_j - \mathbb{E}'_j\right) - \frac{1 - \alpha}{(1 - \alpha \cdot P_{i,j})}
\end{aligned}$$

However, we must have $\mathbb{E}_j < \mathbb{E}'_j$, or else the attacker would not be pruning $P_{i,j}$ at all, and therefore $\mathbb{E}_j - \mathbb{E}'_j < 0$ The term on the right, $\frac{1-\alpha}{(1-\alpha \cdot P_{i,j})}$, must be positive because $P_{i,j}$ and α are both in the range $(0, 1)$. Therefore this term is positive and the inequality cannot possibly hold, meaning it is impossible for partial pruning of a transition to be more beneficial than complete pruning. □

B.2 Proof of Theorem 1

In Section A.3, we introduced Theorem 1 stating that for a lottery manipulation algorithm $\mathcal{A}$ which has expected value $\mathbb{E}_i$ from state s_i, pruning transition $P_{i,j}$ will lead to an improved algorithm $\mathcal{A}'$ if $\mathbb{E}_i + 1 > \mathbb{E}_j$.

Proof. To prove this, we'll assume consider the conditions in which the two algorithms are exactly equivalent and pruning makes no difference:

$$\begin{aligned}
\mathbb{E}_i^{\mathcal{A}} &= \mathbb{E}_i^{\mathcal{A}'} \\
\mathbb{E}_i^{\mathcal{A}} &= c\,(P_i'^{\star}) + \frac{\sum_{k \in S_i'^{\star}} P'_{i,k} \cdot \mathbb{E}_k^{\mathcal{A}'}}{P_i'^{\star}} \\
\mathbb{E}_i^{\mathcal{A}} &= \frac{1 - P_i'^{\star} + P_{i,j}}{P_i'^{\star} - P_{i,j}} + \frac{\sum_{k \in S_i'^{\star}} P_{i,k} \cdot \mathbb{E}_k^{\mathcal{A}'}}{P_i'^{\star} - P_{i,j}} \\
\mathbb{E}_i^{\mathcal{A}} &= \frac{P_i'^{\star}}{P_i'^{\star} - P_{i,j}} \cdot \frac{1 - P_i'^{\star} + P_{i,j} + \sum_{k \in S_i^{\star}} P_{i,k} \cdot \mathbb{E}_k^{\mathcal{A}} - P_{i,j}\mathbb{E}_j^{\mathcal{A}}}{P_i'^{\star}} \\
\mathbb{E}_i^{\mathcal{A}} &= \frac{P_i'^{\star}}{P_i'^{\star} - P_{i,j}}\left(\mathbb{E}_i^{\mathcal{A}} + \frac{P_{i,j}}{P_i'^{\star}} - P_{i,j}\mathbb{E}_j^{\mathcal{A}}\right) \\
\mathbb{E}_i^{\mathcal{A}} &= \frac{P_i'^{\star}}{P_i'^{\star} - P_{i,j}}\mathbb{E}_i^{\mathcal{A}} + \frac{P_{i,j}}{P_i'^{\star} - P_{i,j}}(1 - \mathbb{E}_j^{\mathcal{A}}) \\
0 &= \frac{P_{i,j}}{P_i'^{\star} - P_{i,j}}\mathbb{E}_i^{\mathcal{A}} + \frac{P_{i,j}}{P_i'^{\star} - P_{i,j}}(1 - \mathbb{E}_j^{\mathcal{A}})
\end{aligned}$$

$$0 = \mathbb{E}_i^{\mathcal{A}} + (1 - \mathbb{E}_j^{\mathcal{A}})$$
$$\mathbb{E}_i^{\mathcal{A}} = \mathbb{E}_j^{\mathcal{A}} - 1$$

As $\mathbb{E}_j^{\mathcal{A}}$, pruning the transition $P_{i,j}$ only becomes less beneficial, thus proving the theorem. □

This theorem also implies as a corollary that an algorithm $\mathcal{A}'$ not pruning a transition $\mathbb{E}_j^{\mathcal{A}}$ satisfying the same inequality will be superior to an algorithm $\mathcal{A}$ that does prune it.

C Technical details of beacon opcode

The `OP_BEACON` opcode will read several beacon parameters from the stack and pushes a 32-bit beacon-derived random value onto the stack.[9] If more random data is needed, `OP_BEACON` could be called multiple times with different nonces. The parameters passed to `OP_BEACON` (technically popped off of the stack) are:

1. A 64-bit nonce that will be included in the hash during the beacon computation. This will allow distinct beacon outputs to be obtained by different parties.
2. A 32-bit number specifying the number of blocks to be included in the beacon computation. Note that we are not proposing a multi-stage lottery, only a single beacon computation that may hash multiple blocks.
3. An offset from the current block to begin gathering blocks for the Bitcoin computation. The opcode will result in an exception if executed in a block prior to the block index in which the transaction is included, plus this offset, plus the number of blocks needed, so this effectively functions as a time-lock on the transaction.

Overall, this is a relatively small change to Bitcoin which is suitable for adoption as a "soft-fork."[10] The beacon computation, consisting solely of hashing, is very cheap compared to the signature verification which miners already perform, and the only data referenced is block headers which must already be cached to perform block chain verification. The new op code should have no effect on transaction validation caches, as once a transaction can be validated the beacon value will be fixed.[11]

The use of a relative offset is perhaps the most unusual aspect of our proposal compared to other Bitcoin opcodes, but it serves an important security function over the alternative of specifying a fixed index from which to calculate

[9] Currently arithmetic opcodes in Bitcoin only operate on signed 32-bit integers, so we work within this limit because modifying it would be a far larger change to Bitcoin.

[10] The soft-fork approach has been used, for example, to add the popular "pay-to-script-hash" feature.

[11] Of course, in the event of a deep block chain fork the beacon value might change. However, this case requires invaliding the transaction cache for all types of transactions.

the beacon, which we will describe in Appendix C.2. A similar feature which is already implemented is that coinbase transactions cannot be redeemed until 100 blocks after they are included in the block chain, suggesting that this logic is not prohibitively complicated to implement.

C.1 Example lottery script

The following Bitcoin transaction script demonstrates how to make use of our proposed `OP_BEACON` opcode to implement a two-player lottery, as described in Section 8.1:

```
// Beacon parameters:
0x41e053d8  // nonce
1           // number of blocks to include
1           // number of blocks to wait
            // before computing beacon
OP_BEACON
// Threshold to determine winner.
// Since integers are signed, an even
// two-party lottery has a threshold of 0
0
OP_LESSTHANOREQUAL
// Check for Alice's signature, if Alice won
OP_IF
   OP_DUP
   OP_HASH160
   <alicePubKeyHash>
   OP_EQUALVERIFY
   OP_CHECKSIG
// Check for Bob's signature, if Bob won
OP_ELSE
   OP_DUP
   OP_HASH160
   <bobPubKeyHash>
   OP_EQUALVERIFY
   OP_CHECKSIG
OP_ENDIF
```

C.2 The advantage of relative offsets

Our example lottery script demonstrates the advantage of using a relative offset in `OP_BEACON` to specify when the beacon is to be sampled prevents either party from ever gaining an advantage. Even if Alice has signed the wager transaction but Bob hasn't, her funds are not yet committed and she may sign them to a different transaction if she chooses to abort before Bob signs and the block is published. Neither party can gain any insight into the value of the beacon by

delaying, since it will always be sampled a fixed amount of time in the future after the transaction is signed and published in the block chain.

An alternative design in which `OP_BEACON` specified an absolute block index from which to sample the beacon might be vulnerable in this scenario if one party signed the transaction first and the other was able to delay until the beacon output was known. To prevent this, we would need a validation rule that that the block index used in a beacon calculation must occur after the block in which that transaction was public. Because the absolute index might be computed dynamically, however, we expect this will be more difficult to detect and prevent and hence lead to more dead transactions on the block chain, motivating us to propose relative offsets for our Bitcoin Improvement Proposal.

D Fair contract signing protocols

We can demonstrate the utility of asymmetric lotteries though the example of fair contract signing protocols, the original application of beacons [32]. Alice and Bob agree want to sign the same contract, but neither wants to sign first as they don't trust that the other will sign in return. Rabin proposed an elegant protocol involving a beacon that outputs a random integer in the range $[1, k]$. Alice begins by signing a message of the form, "I commit myself to the contract if the beacon outputs 1." Bob responds by signing the same message. For all $i \in [1, k]$, Alice and Bob exchange messages of this form, with neither party ever signing more than one contract which the other party hasn't signed.

If both parties are honest, they will sign contracts for all $i \in [1, k]$ and the contract will be signed regardless of the beacon output. If Bob tries to cheat by not sending Alice his i^{th} message after receiving her i^{th} message, his attack will only succeed with probability $\frac{1}{k}$ as Alice will not sign subsequent messages. Bob might attempt to force the Beacon to output i for the contract Alice has signed that he has not. This could be solved using a multi-stage MRL based on a Bitcoin beacon, since all k outcomes are equally likely, making Bob's attack cost $O(k \cdot \lg N)$ for N stages (or k for a single stage). Cheating can be made to be arbitrarily expensive by choosing an appropriately large values for k and N.

A later variant of the protocol proposed by Ben-Or et al. [5] demonstrates a natural application of asymmetric MRLs. In this version of the protocol, each party responds in each round with a contract of the form "I commit myself to the contract with probability p" where p is incremented by β from the last contract signed by the other party. Each party has an advantage of β in any given round. However, using our asymmetric MRL, we can create a lottery with three outcomes with probabilities p, β, and $1 - p - \beta$. We only need to defend against manipulation to produce the outcome with probability β (that the most recent signer is committed but not the other party), therefore we could write each contract to specify a an asymmetric multi-stage MRL which makes forcing the β event as expensive as possible.

These two example protocols, designed to solve the sample problem, provide a compelling example for asymmetric MRLs and the need to specifically design a secure lottery based on the larger protocol it is being used in.

E Impact of transaction fees on attacker model

Embedded in our attacker model in Section 4.2 is the assumption that block rewards are much larger than transaction fees. This affects the opportunity cost for a mining attacker to hold a block because transaction fees will (likely) still be available if the miner withholds a block and is able to find another before competing miners. Assuming the block reward is B and the transaction fees in a block are worth a total of τ, a miner with a proportion p of the total mining power in the network will suffer an expected loss of $B + (1-p)\tau$ when withholding a block based on the chance that they can still gain the transaction fees they are forgoing in a subsequent block. Currently, in the average block we have $B \approx 1,000 \cdot \tau$ and hence we can safely approximate $B + (1-p)\tau \approx B$ regardless of p.

In the future, it is possible for transaction fees may significantly increase and block reward fees are scheduled to slowly decline, halving again in 2017 and every 4 years thereafter. While this situation is likely at least a decade off, we can quickly reason about the case where $B \to 0$ and transaction fees provide the only motivation for mining. Security in this model will require a further assumption that $p < \frac{1}{2}$, meaning there is no majority miner or mining pool, as is customarily considered a requirement for Bitcoin mining to be secure. In this case, the cost of withholding a block is still at least $\frac{\tau}{2}$. Hence, the cost of bribery to withhold blocks will always be at least half of the average value earned by a block. All of our results apply equally in this model, although the cost of manipulation is potentially divided in half in the worst case.