Constant-Deposit Multiparty Lotteries on Bitcoin

Massimo Bartoletti[1(✉)] and Roberto Zunino[2]

[1] Università degli Studi di Cagliari, Cagliari, Italy
bart@unica.it
[2] Università degli Studi di Trento, Trento, Italy

Abstract. An active research trend is to exploit the consensus mechanism of cryptocurrencies to secure the execution of distributed applications. In particular, some recent works have proposed fair lotteries which work on Bitcoin. These protocols, however, require a deposit from each player which grows quadratically with the number of players. We propose a fair lottery on Bitcoin which only requires a constant deposit.

1 Introduction

Recent research on blockchain technologies studies how to extend the applications of cryptocurrencies from simple transfers of money to complex financial transactions. The goal is to make financial agreements or "smart contracts" [24] between mutually distrusting participants, and automatically enforce them via the consensus mechanism of the cryptocurrency, without relying on a trusted third party. In particular, some works propose to run smart contracts on top of existing cryptocurrencies (mostly, on Bitcoin). Many of these approaches, e.g. [1,6,8,16–18], implement *fair* computations, where a set of players contribute to compute a function without revealing their inputs; fairness, studied in various forms, guarantees e.g. that any player that aborts after learning the output pays a penalty to all players that did not learn the output. Other works implement decentralised authorization systems [10], and contracts which allow users to make statements, penalising those which make conflicting ones [22].

A particular kind of smart contract is the one which implements a lottery among a set a players. Intuitively, this is an application where each one of N players puts their bets in a pot, and a winner—uniformly chosen among the players—gets the whole pot. Secure protocols for multiparty lotteries on Bitcoin have been recently proposed by [2,4,5,8]. These protocols enjoy a *fairness* property, which roughly guarantees that: (i) each honest player will have (on average) a non-negative payoff, even in the presence of adversaries who play against; (ii) when all the players are honest, the protocol behaves as an ideal lottery: one player wins the whole pot, while all the others lose their bets.

To obtain the result, these protocols require that, to bet e.g. 1 coin, each one of the N players must block a *deposit* of $O(N^2)$ coins throughout the whole

M. Brenner et al. (Eds.): FC 2017 Workshops 2017, LNCS 10323, pp. 231–247, 2017.
https://doi.org/10.1007/978-3-319-70278-0_15

protocol[1]. Since the deposit grows quadratically with N, these protocols are only practical for a small number of players. In this paper we address this issue.

Contributions. We propose a fair protocol for multiparty lotteries, whose deposit does *not depend* on the number N of players. More specifically, our protocol is fair for any choice of the deposit value (including zero), and for any adversarial strategy. Furthermore, if the deposit value is positive, an adversary who tries to attack the protocol with the goal of altering the payoff of honest players, can only lose money on average. Our protocol is based on a *single-elimination tournament*, i.e. a tree of $N-1$ two-player matches where the loser of each match is eliminated. Overall, a complete run of the protocol requires $O(N)$ transactions on-chain and $O(\log N)$ time (assuming that the time to put transactions on the Bitcoin ledger dominates the time required for communications and local computations). Our protocol has been implemented as an Ethereum smart contract; an implementation on Bitcoin would require a variant of the mechanism for verifying the signature of transactions, to allow the malleability of input fields.

An extended version of this paper is available online at [7].

2 Statically Signing Chains of Transactions

The current signature mechanism of Bitcoin is known to be unsuitable for signing *chains* of transactions before they are put on the ledger[2]. Consider e.g. two players, a and b, and three transactions, T_0, T_1 and T_2, made as follows.

– transaction T_1 has T_0 as input, while T_2 has T_1 as input: hence the three transactions form a chain.
– the out-scripts of T_0 and T_1 require signatures by both players a and b.

The players want to put the chain of transactions on the ledger, assuming that T_0 is already there. Intuitively, the players have two possible ways of proceeding:

dynamic signing: both players sign T_1 and put it on the ledger. After that, they both sign T_2 and put it on the ledger.

static signing: a signs both T_1 and T_2 *before* these transactions are on the ledger, and sends her signatures to b. Then, b adds his own signatures, and puts both T_1 and T_2, one after the other, on the ledger.

Without the *segregated witnesses* feature [19], only dynamic signing is feasible. Of course, in static signing, the addition of b's signature to the in-script of T_1 alters its in-script.[3] Note that this will not invalidate a's signature of T_1 (because the signature does not consider the in-script), so T_1 can still be put on

[1] Concurrently and independently of our work, [20] proposes a lottery protocol for Bitcoin that requires zero deposit.

[2] See https://en.bitcoin.it/wiki/Transaction_Malleability.

[3] in-script and out-script are respectively referred as scriptPubKey and scriptSig in the Bitcoin documentation.

the ledger. However, altering the in-script changes the *hash* of T_1, which is used in T_2.in to refer to the previous transaction. Because of this, a's signature of T_2 is no longer valid, hence b can not put T_2 on the ledger.

A possible solution to this problem is to allow *partial* signatures, which e.g. neglect the in part of transactions, as already done for the in-script part. Indeed, even if T_2.in (i.e., the hash of T_1) is modified, the (partial) signature in T_2.in-script is still valid, because it neglects the in part. More in general, we define below a signature scheme for Bitcoin transactions, allowing users to choose which parts M of the transaction to include in the signature. In this way, once the transaction is signed, anyone can modify the parts not in M without invalidating the signature. The ability of modifying transactions while preserving their signatures is called *transaction malleability*: while in some circumstances it can cause security vulnerabilities [3], if used in a controlled manner it can extend the range of applications built upon Bitcoin [1]. Note that the unsigned parts of a transaction can be freely altered by adversaries; therefore, designing a secure protocol must take into account for this possibility. E.g., in the previous static signing example, b can alter T_2.in so to refer to some $\mathsf{T} \neq \mathsf{T}_1$ whose out-script can be satisfied by a's signature. In this way T becomes unredeemable. To protect against this attack, a could use a fresh key in T_1.out-script, so that nothing else can be redeemed by her signature.

We anticipate that our lottery protocol does not require the whole flexibility of the signature mechanism outlined below, but it only relies on the malleability of the in and in-script fields. While the malleability of in-script is already allowed by the segregated witnesses release, that of in fields would require support from the signature verification mechanism (e.g., a new signature flag or opcode).

Signature scheme for transaction malleability. Let

$$M \subseteq \{\mathsf{in}[n], \mathsf{in\text{-}script}[n], \mathsf{value}[n], \mathsf{out\text{-}script}[n], \mathsf{lockTime} \mid n \geq 0\}$$

and denote with $M(\mathsf{T})$ the bitstring obtained by concatenating the parts of the transaction T mentioned in M. We then define:

$$\mathbf{sig}_k^M(\mathsf{T}) = (sig_k(M(\mathsf{T})), M) \qquad \mathbf{ver}_k(\mathsf{T}, (y, M)) = ver_k(M(\mathsf{T}), y)$$

Hereafter, we use σ as a meta-variable for the *partial signatures* $(sig_k(\ldots), M)$, and $\boldsymbol{\sigma}$ for arrays of such pairs (we will always use the same convention for arrays). When $\boldsymbol{k}$ and $\boldsymbol{\sigma}$ have the same size n, we define:

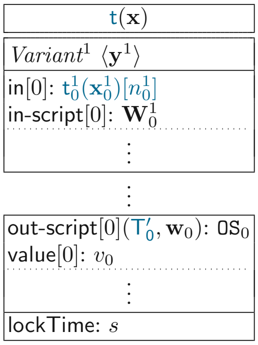
$$\mathbf{sig}_{\boldsymbol{k}}^M(\mathsf{T}) = (\mathbf{sig}_{\mathbf{k}[0]}^M(\mathsf{T}), \ldots, \mathbf{sig}_{\mathbf{k}[n-1]}^M(\mathsf{T})) \qquad \mathbf{ver}_{\boldsymbol{k}}(\mathsf{T}, \boldsymbol{\sigma}) = \bigwedge_i \mathbf{ver}_{\boldsymbol{k}[i]}(\mathsf{T}, \boldsymbol{\sigma}[i])$$

Transaction templates. The mechanism shown above allows to statically sign chains of transactions; further, we can also use it to statically sign chains of the form $\mathsf{T}_0\, \mathsf{T}_1(y)\, \mathsf{T}_2$, where the transaction $\mathsf{T}_1(y)$ depends on a parameter y such that (i) y is unknown at signing time (it will only be known later on), and (ii) y only affects those parts of $\mathsf{T}_1(y)$ not included in the partial signatures. Under these assumptions, instantiating y in a later moment will *not* invalidate any signature. More importantly, while there might be a large number of values for y (and so, a large number of chains that can be put on the ledger), only one partial static signature of T_1 is needed (as well as for T_0 and T_2).

Parametric descriptions like the chain above are useful when designing complex protocols, where the actual chain (or graph) of transactions to be put on the ledger depend on events known after signatures have already been computed. We now introduce a general notation for expressing transactions with parameters and variants, which hereafter we name *transaction templates*. Our notation shows all the possible forms of the malleable transaction parts which are used in a protocol. Further, we will show how to statically sign such transactions (in all their forms). We anticipate that, for our lottery protocol, the number of possible transactions is large, while the number of needed static signatures is small.

Hereafter, we fix $M = \{\mathsf{value}[n], \mathsf{out\text{-}script}[n], \mathsf{lockTime} \mid n \geq 0\}$ in our signature scheme, so making the in and in-script fields malleable.[4]

The general form of transaction templates $\mathsf{t}, \mathsf{t}', \ldots$ is shown on the right. The template $\mathsf{t}(\mathbf{x})$ is parametrized over an array of values $\mathbf{x}$, in a given domain. Further, for its in and in-script fields, the template describes a few *variants*, each of which may take some additional parameters $\mathbf{y}$. Note that out-scripts may only refer to the template parameters $\mathbf{x}$, while in and in-scripts may also refer to their own variant parameters $\mathbf{y}$. Further, the in field refers to another template. A template $\mathsf{t}(\mathbf{x})$ can be instantiated to a transaction $\mathsf{T} = \mathsf{t}(\mathbf{x}).\mathit{Variant}^i\langle\mathbf{y}^i\rangle$, by choosing the variant i and the parameters. Here, T.in is set to any redeemable transaction on the ledger which is an instantiation of the template in the in field of t.


$\mathsf{t}(\mathbf{x})$
$\mathit{Variant}^1\ \langle\mathbf{y}^1\rangle$
in[0]: $\mathsf{t}_0^1(\mathbf{x}_0^1)[n_0^1]$
in-script[0]: $\mathbf{W}_0^1$
$\vdots$
$\vdots$
out-script[0]$(\mathsf{T}_0', \mathbf{w}_0)$: $\mathtt{OS}_0$
value[0]: v_0
$\vdots$
lockTime: s

The procedure for signing transaction templates is detailed in [7].

3 The Tournament Protocol

We introduce our lottery protocol for $N = 2^L$ players; each player is represented by a bit-string in $\mathcal{P} = \{0,1\}^L$, ranged over by $a, b, \ldots$. We assume that each player bets 1฿ in the lottery, and blocks a deposit of d฿, for an arbitrary $d \geq 0$. Our protocol is based on a single-elimination tournament, where matches are organised as a complete binary tree of L levels. The tournament involves $N - 1$

[4] Note that only the transactions related to our protocol need to use this form of malleability. Instead, signers of transactions unrelated to the protocol can simply choose non-malleable signatures, unless they are prepared to defend against malleability-related attacks. For instance, if T and T' are standard transactions on the ledger with the same out-script, when T is redeemed by T_1 with a malleable in field, an adversary can also make T' redeemed, by putting on the ledger a copy of T_1 where the in field is changed to point to T'.

two-player matches: the winners of the matches at level $\ell \in 1..L-1$ play at the next level $\ell - 1$; the winner of the match at level 0 wins the whole N₿ stake.

Let $\Pi = \{\{0,1\}^n \mid n \leq L\}$ (i.e., sequences of n bits) be the set of tree *paths*. Intuitively, for every path in $\Pi \setminus \mathcal{P}$ we have a two-player match. For any two paths $\pi, \pi' \in \Pi$, we write $\pi \sqsubseteq \pi'$ when π is a prefix of π' ($\sqsubset$ for proper prefixes).

Key pairs and secrets. Our protocol requires players to exchange a certain number of Bitcoin transactions, together with their signatures. To this purpose, each player p generates all the following key pairs for every $a, b \in \mathcal{P}$ and for every π:

$K_p(\mathit{Bet}_p),\ K_p(\mathit{Collect}),\ K_p(\mathit{Init}, a)$	
$K_p(\mathit{Win}, \pi, a),\ K_p(\mathit{WinTO}, \pi, a)$	$\epsilon \neq \pi \sqsubseteq a$
$K_p(\mathit{Turn1}, \pi, a, b),\ K_p(\mathit{Turn1TO}, \pi, a, b),\ K_p(\mathit{Turn2TO}, \pi, a, b)$	$\pi \sqsubset a, b$
$K_p(\mathit{Turn2}, \pi, a)$	$\pi \sqsubset a$
$K_p(\mathit{Timeout1}, \pi, a, b),\ K_p(\mathit{Timeout2}, \pi, a, b)$	$\pi \sqsubset a, b$

The first component in each key pair above (e.g., *Collect*) is a distinct label. Note that each player generates $O(N^2)$ key pairs. We assume that the private part of a key pair $K_p(\cdots)$ is kept secret by p, while the public part is communicated to the other players. For each set of key pairs $K_p(X, \cdots)$, we denote with $\mathbf{K}(X, \cdots)$ the set of key pairs $\{K_p(X, \cdots) \mid p \in \mathcal{P}\}$. We denote with ϵ the empty sequence.

The outcome of a match is randomly determined with a "coin toss" protocol, as in [2]. Intuitively, the players generate two random secrets, and exchange their hashes; then, they reveal the secrets: the winner is determined by a function of the two secrets (i.e., the parity of the sum of the lengths of the two secrets). Since a player may be involved in L distinct matches, we assume that each p generates L secrets (i.e., long random sequences of bits), one for each $\pi \sqsubset p$. The secret of p at level π is denoted by s_p^π; its public hash $H(s_p^\pi)$ is denoted by h_p^π.

Overview of the protocol. Our protocol uses a number of transactions, the templates of which are in Fig. 1. The protocol is organised in three phases:

initialization: the players exchange the public data, e.g. the static signatures and hashed secrets. Then, they collect all the bets, and put on the ledger the transactions for the leaves of the tournament tree.

execution: this phase is organised in L *rounds*, one for each level of the tree. In each round ℓ, exactly 2^ℓ two-player matches are played, by the winners of the previous round. The possible executions of a single round are depicted in Fig. 3. The winner of the last round collects the whole stake.

garbage collection: this allows players to recover from some potential interference, to be discussed in the proof of Theorem 5.

We now comment the protocol in Fig. 2. We denote the duration of each round with $\tau_{\mathit{Round}} = 6\,\tau_{\mathit{Ledger}}$, following Fig. 3. The transaction templates of Fig. 1 define some timelocks, which depend on a time τ_1 (chosen in the initialization phase), corresponding to the start of the execution phase.

Init
certifies that all players have placed their bets (and deposits)

$\forall p \in \mathcal{P}: \begin{cases} \text{in}[p]\text{: Bet}_p \\ \text{in-script}[p]\text{: } \mathbf{sig}_{K_p(Bet_p)}(\bullet) \end{cases}$

$\forall p \in \mathcal{P}: \begin{cases} \text{out-script}[p](\mathsf{T},\boldsymbol{\sigma})\text{: } \mathbf{ver}_{\mathbf{K}(Init,p)}(\mathsf{T},\boldsymbol{\sigma}) \\ \text{value}[p]\text{: } 1 + d \end{cases}$ ₿

Win(π, a) with $\epsilon \neq \pi \sqsubset a$
certifies that a has won all the rounds until π (included)

Timeout1 $\langle b \rangle$
in: Timeout1(π, b, a)
in-script: $\mathbf{sig}_{\mathbf{K}(Timeout1,\pi,b,a)}(\bullet)$

Timeout2 $\langle b \rangle$
in: Timeout2(π, a, b)
in-script: $\mathbf{sig}_{\mathbf{K}(Timeout2,\pi,a,b)}(\bullet)$

Turn2fst $\langle b, \hat{s}_a, \hat{s}_b \rangle$
in: Turn2(π, a, b)
in-script: $\hat{s}_a, \hat{s}_b, \mathbf{sig}_{\mathbf{K}(Turn2,\pi,a)}(\bullet)$

Turn2snd $\langle b, \hat{s}_a, \hat{s}_b \rangle$
in: Turn2(π, b, a)
in-script: $\hat{s}_b, \hat{s}_a, \mathbf{sig}_{\mathbf{K}(Turn2,\pi,a)}(\bullet)$

out-script$(\mathsf{T},\boldsymbol{\sigma})$: $\mathbf{ver}_{\mathbf{K}(Win,\pi,a)}(\mathsf{T},\boldsymbol{\sigma}) \lor \mathbf{ver}_{\mathbf{K}(WinTO,\pi,a)}(\mathsf{T},\boldsymbol{\sigma})$
value: $(1+d)\, 2^{L-|\pi|}$ ₿

Win(a, a) (leaf)
contains the bet (and deposit) of a at the first round

in: Init$[a]$
in-script: $\mathbf{sig}_{\mathbf{K}(Init,a)}(\bullet)$

out-script$(\mathsf{T},\boldsymbol{\sigma})$: $\mathbf{ver}_{\mathbf{K}(Win,a,a)}(\mathsf{T},\boldsymbol{\sigma})$
value: $1 + d$ ₿

Win(ϵ, a) (root)
certifies that a has won the lottery

(Variants as for Win(π, a))

out-script$[a](\mathsf{T},\sigma)$: $\mathbf{ver}_{K_a(Collect)}(\mathsf{T},\sigma)$
value$[a]$: $N + d$ ₿

$\forall p \neq a: \begin{cases} \text{out-script}[p](\mathsf{T},\sigma)\text{: } \mathbf{ver}_{K_p(Collect)}(\mathsf{T},\sigma) \\ \text{value}[p]\text{: } d \end{cases}$ ₿

Turn1(π, a, b) with $\pi \sqsubset a, b$
certifies that a and b are playing in match π, where it is a's turn to reveal her secret

in[0]: Win$(\pi0, a)$
in-script[0]: $\mathbf{sig}_{\mathbf{K}(Win,\pi0,a)}(\bullet)$
in[1]: Win$(\pi1, b)$
in-script[1]: $\mathbf{sig}_{\mathbf{K}(Win,\pi1,b)}(\bullet)$

out-script$(\mathsf{T}, \hat{s}_a, \boldsymbol{\sigma})$:
$(\, H(\hat{s}_a) = h_a^\pi \land \mathbf{ver}_{\mathbf{K}(Turn1,\pi,a,b)}(\mathsf{T},\boldsymbol{\sigma})\,)$
$\lor\ \mathbf{ver}_{\mathbf{K}(Turn1TO,\pi,a,b)}(\mathsf{T},\boldsymbol{\sigma})$
value: $(1+d)\, 2^{L-|\pi|}$ ₿

Turn2(π, a, b) with $\pi \sqsubset a, b$
certifies that a and b are playing in match π, where a has revealed her secret, and now it is b's turn

Secret $\langle \hat{s}_a \rangle$
in: Turn1(π, a, b)
in-script: $\hat{s}_a$, $\mathbf{sig}_{\mathbf{K}(Turn1,\pi,a,b)}(\bullet)$

out-script$(\mathsf{T}, \hat{s}_a, \hat{s}_b, \boldsymbol{\sigma})$:
$(\, H(\hat{s}_a) = h_a^\pi \land H(\hat{s}_b) = h_b^\pi$
$\land\ \mathbf{ver}_{\mathbf{K}(Turn2,\pi,winner(a,b,\hat{s}_a,\hat{s}_b))}(\mathsf{T},\boldsymbol{\sigma})\,)$
$\lor\ \mathbf{ver}_{\mathbf{K}(Turn2TO,\pi,a,b)}(\mathsf{T},\boldsymbol{\sigma})$
value: $(1+d)\, 2^{L-|\pi|}$ ₿

Timeout1(π, a, b) with $\pi \sqsubset a, b$
certifies that a lost against b in match π because she did not reveal her secret in time

in: Turn1(π, a, b)
in-script: $\bot$, $\mathbf{sig}_{\mathbf{K}(Turn1TO,\pi,a,b)}(\bullet)$

out-script$(\mathsf{T},\boldsymbol{\sigma})$: $\mathbf{ver}_{\mathbf{K}(Timeout1,\pi,a,b)}(\mathsf{T},\boldsymbol{\sigma})$
value: $(1+d)\, 2^{L-|\pi|}$ ₿
lockTime: $\tau_1 + (L - |\pi| - 1)\tau_{Round} + 2\tau_{Ledger}$

Timeout2(π, a, b) with $\pi \sqsubset a, b$
certifies that b lost against a in match π because she did not reveal her secret in time

in: Turn2(π, a, b)
in-script: $\bot$, $\bot$, $\mathbf{sig}_{\mathbf{K}(Turn2TO,\pi,a,b)}(\bullet)$

out-script$(\mathsf{T},\boldsymbol{\sigma})$: $\mathbf{ver}_{\mathbf{K}(Timeout2,\pi,a,b)}(\mathsf{T},\boldsymbol{\sigma})$
value: $(1+d)\, 2^{L-|\pi|}$ ₿
lockTime: $\tau_1 + (L - |\pi| - 1)\tau_{Round} + 4\tau_{Ledger}$

CollectOrphanWin(π, a) with $\epsilon \neq \pi \sqsubset a$
certifies that a was prevented by an adversary to participate in the rounds after π, but she can collect her winnings so far (see **??** for details)

in: Win(π, a)
in-script: $\mathbf{sig}_{\mathbf{K}(WinTO,\pi,a)}(\bullet)$

out-script$[a](\mathsf{T},\sigma)$: $\mathbf{ver}_{K_a(Collect)}(\mathsf{T},\sigma)$
value$[a]$: $2^{L-|\pi|} + d$ ₿

$\forall p$ with $a \neq p \sqsubseteq \pi: \begin{cases} \text{out-script}[p](\mathsf{T},\sigma)\text{: } \mathbf{ver}_{K_p(Collect)}(\mathsf{T},\sigma) \\ \text{value}[p]\text{: } d \end{cases}$ ₿

lockTime: $\tau_1 + (L - |\pi|)\tau_{Round} + \tau_{Ledger}$

Fig. 1. Transaction templates for the lottery protocol.

Precondition: for all players p, the ledger contains a transaction Bet_p with value $(1+d)$฿, and redeemable with key $K_p(Bet_p)$.

Initialization phase:

1. each player p generates all the key pairs and the secrets s_p^π as in Section 3, and broadcasts to the other players the public keys and hashes $h_p^\pi = H(s_p^\pi)$;
2. if $h_p^\pi = h_{p'}^{\pi'}$ for some $(p, \pi) \neq (p', \pi')$, the players abort;
3. choose the time τ_1 large enough to fall after the initialization phase;
4. each player signs all the transactions templates in Figure 1 except for Init, and broadcasts the signatures;
5. each player verifies the signatures received by the others; if some signature is not valid or missing, the player aborts the protocol;
6. each player p signs Init, and sends the signature to the first player;
7. the first player puts the (signed) transaction Init on the ledger;
8. if Init does not appear within one τ_{Ledger}, then each p redeems Bet_p and aborts;
9. the players put the signed transactions $\mathsf{Win}(p, p)$ on the ledger, for all $p \in \mathcal{P}$.

Execution phase:

for each level $\ell = L..1$:

for each π such that $|\pi| = \ell - 1$, in parallel, a two-player match is played:

10. let a and b be such that $\mathsf{Win}(\pi 0, a)$ and $\mathsf{Win}(\pi 1, b)$ are on the ledger;
11. the players put $\mathsf{Turn1}(\pi, a, b)$ on the ledger;
12. player a puts $\mathsf{Turn2}(\pi, a, b).Secret\langle s_a^\pi \rangle$ on the ledger;
13. the players wait until either $\mathsf{Turn2}(\pi, a, b)$ is confirmed, or $\mathsf{Timeout1}(\pi, a, b)$ is enabled. In the second case, they put $\mathsf{Timeout1}(\pi, a, b)$ on the ledger; once it is confirmed, they put $\mathsf{Win}(\pi, b).Timeout1\langle a \rangle$ on the ledger, and terminate the match at π;
14. player b computes $w = winner(a, b, s_a^\pi, s_b^\pi)$, the winner of the match at π;
    - if $w = a$, player b puts $\mathsf{Win}(\pi, a).Turn2fst\langle b, s_a^\pi, s_b^\pi \rangle$ on the ledger.
    - if $w = b$, player b puts $\mathsf{Win}(\pi, b).Turn2snd\langle a, s_a^\pi, s_b^\pi \rangle$ on the ledger.
15. the players wait until either $\mathsf{Win}(\pi, c)$ is confirmed (for some $c \in \{a, b\}$) , or $\mathsf{Timeout2}(\pi, a, b)$ is enabled. In the second case, they put $\mathsf{Timeout2}(\pi, a, b)$ on the ledger; once confirmed, they put $\mathsf{Win}(\pi, a).Timeout2\langle b \rangle$ on the ledger.

Garbage collection phase: if there is some unredeemed $\mathsf{Win}(\pi, p)$ with $\pi \neq \epsilon$, then the players put $\mathsf{CollectOrphanWin}(\pi, p)$ on the ledger.

Fig. 2. Tournament lottery protocol.

Initialization phase. In step 1, all the players generate the signatures and secrets, and exchange the related public data. Step 2 is needed to prevent attacks where a player does not compute a hash from her own secret, but replays the hash of another player. In step 3 we choose the time τ_1 to be large enough so that the initialization can be completed within τ_1. In steps 4–5 the players exchange all the static signatures needed in the execution phase. Each player p contributes his own part of the signature, using his own keys $K_p(\ldots)$. Steps 6–8 collect the bets from the transactions Bet_p in a single transaction Init. If Init is not confirmed

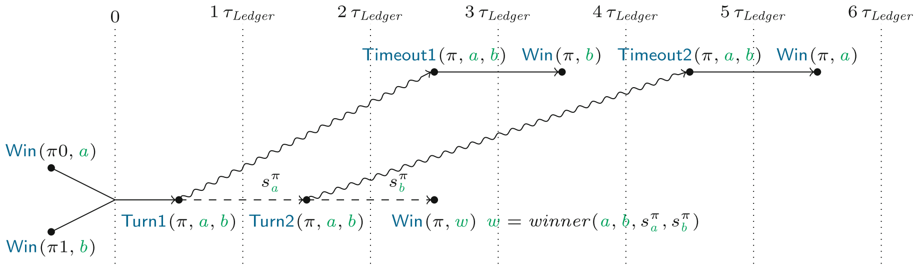


Fig. 3. Graph of the transactions in a tournament round. An edge from transaction T to T′ means that T′ redeems T. Solid edges mean that any player can redeem; wavy edges mean that any player can redeem, but only after a timeout. Dashed edges mean that only the player who knows the secret on the label can redeem.

on the ledger, e.g. because some player has already redeemed his bet, then all the other players redeem their original bets. In this way, they ensure that Init can no longer appear on the ledger, hence the protocol is aborted. Step 8 also prevents an attack where Init is maliciously delayed so to make honest players lose. Finally, step 9 sets up the first level of the tournament, by splitting the stake in the Init among all the leaves of the tree, i.e. Win(p, p).

To choose τ_1, note that the initialization phase requires:

- at steps 1–6, to generate all the needed $O(N^3)$ signatures and NL secrets, and share the related public parts. This costs $O(N^3)$ time.
- at step 7, to put on the ledger the transaction Init. This costs 1 τ_{Ledger}.
- after that, at step 9, to put all the transactions Win(p, p). This costs 1 τ_{Ledger}, because it can be done in parallel.

Therefore, we choose τ_1 such that $\tau_1 \geq \mathsf{currentTime} + O(N^3) + 2\tau_{Ledger}$.

Execution phase. In this phase, the players play against each other. We recommend the reader to examine Fig. 3 for an overview of how matches are played. Matches correspond to the nodes of the tournament tree, and so they are indexed by tree paths π. The match at π involves the winners of the two matches $\pi 0$ and $\pi 1$ of the previous round (i.e., the children of π). These winners are, respectively, the players a and b in the transactions Win($\pi 0$, a) and Win($\pi 1$, b) which are on the ledger at the start of the match (step 10). The goal of steps 10–15 is to put on the ledger a transaction Win(π, w), where w is the winner at π.

Step 11 starts by redeeming both Win($\pi 0$, a) and Win($\pi 1$, b) through the transaction Turn1(π, a, b). Note that any player (not only a and b) can perform this step, since everyone has the required signatures. At step 12, player a is expected to reveal her secret s^{π}_{a}; otherwise, after a certain deadline, the other players can make a lose. If a chooses to reveal her secret, she must put on the ledger the transaction Turn2(π, a, b), which redeems Turn1(π, a, b), through an input script containing s^{π}_{a}. Otherwise, after $1\tau_{Ledger}$, the timelock on Timeout1(π, a, b) expires, allowing any other player to put Timeout1(π, a, b) on the ledger at step 13. After that, $\mathsf{Win}(\pi, b)$ can be put on the ledger by any player, so making a lose the match. At step 14, it is the turn of player b to reveal his secret s_b^π; otherwise, similarly to the previous steps, the other players can make b lose after some time. If b chooses to reveal his secret, he must first compute the winner w of the match—this is possible because b knows both secrets s_a^π and s_b^π. Then, he must put $\mathsf{Win}(\pi, w)$ on the ledger, which redeems $\mathsf{Turn2}(\pi, a, b)$, through an input script containing s_b^π. Otherwise, after $1\tau_{Ledger}$, the timelock on $\mathsf{Timeout2}(\pi, a, b)$ expires, allowing any other player to put $\mathsf{Timeout2}(\pi, a, b)$ on the ledger at step 13. After that, $\mathsf{Win}(\pi, a)$ can be put on the ledger by any player, so making b lose the match.

After the last round of the execution phase, the tournament protocol is over. At this point, there is exactly one transaction $\mathsf{Win}(\epsilon, a)$ on the ledger, for some a. This transaction can be redeemed by a at any time, by putting on the ledger a transaction with in-script $\mathbf{sig}_{K_a(Collect)}(\bullet)$. Note that only a has the private key needed for this signature. In this way a can obtain the whole stake of N₿.

Garbage collection phase. As discussed in the proof of Theorem 5, a dishonest player can try to cheat by forging Win transactions. When this happens, some legit Win transactions are left orphan on the ledger: garbage collection allows the players who contributed to these transactions to redeem their money back. In this way the protocol remains secure, as established later on by Theorem 5.

4 Security of the Tournament Protocol

We assume that all the algorithms used by the players run in PPTIME with respect to a security parameter η. A function $f : \mathbb{N} \to \mathbb{R}$ is said to be *negligible* iff, for some constant $c \in \mathbb{N}$, the inequation $|f(\eta)| \leq \eta^{-c}$ holds asymptotically. We assume that all the cryptographic primitives (e.g., digital signatures, hash functions) are secure, up-to a negligible probability of attack.

We assume that Bitcoin works as a robust public transaction ledger, where every player can append valid transactions (which are confirmed in τ_{Ledger}), while invalid transactions cannot appear. Recent results [13] show that, in a backbone Bitcoin protocol, this assumption holds when the honest miners hold the majority of the hashing power (despite the negative results in [11]). For simplicity, we assume that transactions require no fees. All our results hold even when there is only one honest player.

Basic properties. Consider an arbitrary lottery protocol with N players, where each player bets a certain amount *bet* of bitcoins to have the chance to win $N \cdot bet$. A *run* is a pair (β, λ), where β is the state of the blockchain when the protocol starts, and λ is the timed sequence of public events occurred in a (possibly partial) protocol execution. The component λ includes, e.g., the exchanged signatures and the transactions put on the ledger after β. Each player a uses a *strategy* Σ_a to choose which events to perform at any time in a run of the protocol. Roughly, $\Sigma_a(1^\eta, \beta, \lambda)$ is a PPTIME algorithm which can observe the whole past (β, λ), and choose the next moves (not necessarily those prescribed by the protocol). We further allow Σ_a to access the local state of a, including her private information. A strategy Σ_a is *honest* when it follows the protocol; a player is honest when she uses an honest strategy. A run is *maximal* for a when she has performed all the enabled actions prescribed by Σ_a.

We say that a transaction is *freely redeemable by* a when (i) a can use her knowledge (including private information) to compute the needed witness, and (ii) a can freely choose the output script of the redeeming transaction. The *wealth* of a after a certain run (β, λ), denoted by $\mathit{wealth}(a, \beta, \lambda)$, is the amount of bitcoins freely redeemable at that time by a, but not by any other player.

Lottery protocols usually require players to block a *deposit* of bitcoins throughout their execution (beyond the bet). Technically, we define the deposit of a as the minimum amount of bitcoins $\mathit{wealth}(a, \beta, \epsilon) - \mathit{bet}$ such that, starting from β, a can always perform a maximal run of the protocol (using an honest strategy), regardless of the behaviour of the other players. Then, we say that a lottery protocol is *d-deposit* if d is the maximum of the deposits of all players. Note that, by definition, it must be $d \geq 0$: otherwise, should a lose the lottery, there would not be enough bitcoins to pay the other players.

The following Theorem 1 states that the tournament protocol requires *constant* d₿ deposit; note instead that the protocols in [2,4,5,8] require $O(N^2)$₿ deposit.

Theorem 1. *The tournament protocol is d-deposit.*

Lemma 1. *For each level $\ell = L..1$ of the execution phase:*

1. *for every π such that $|\pi| = \ell$, the ledger contains a transaction* $\mathsf{Win}(\pi, a)$ *with value $(1 + d)\, 2^{(L-\ell)}$₿, for some a;*
2. *the round starts within time $\tau_1 + (L - \ell) \cdot \tau_{Round}$.*

Theorem 2 exploits Lemma 1 to establish an upper bound to the completion time of our protocol. Note that a single honest player a is enough to guarantee termination: indeed, even if the other players do not cooperate, a can always put all the required transactions on the ledger, after the respective timeouts.

Theorem 2. *Assume that at least one player is honest, while the others can be adversaries with arbitrary strategies. Then:*

1. *after τ_1, either* Init *is on the ledger, or the protocol is aborted without any honest players losing their wealth;*
2. *after* Init *is on the ledger, a transaction* $\mathsf{Win}(\epsilon, p)$ *is put on the ledger within $6\, L\, \tau_{Ledger}$, for some p (who is the winner of the lottery).*

Payoff distribution. We now quantify the payoff of each player in a *single* run of the protocol where all the players are honest. The *payoff* of a player at a given point of an execution is the wealth difference between that point and the beginning of the protocol. Formally, given a run (β, λ) for a, this amounts to:

$$\Phi(a, \beta, \lambda) = \mathit{wealth}(a, \beta, \lambda) - \mathit{wealth}(a, \beta, \epsilon)$$

Then, Theorem 3 states that, once the Init transaction has been put on the ledger, there are only two possible outcomes of the protocol: either a player loses 1Ƀ (her bet), or she wins $N - 1$Ƀ (the bets of all the other players).

Theorem 3. *If all players are honest, then, for all players a and for all maximal runs (β, λ) of a such that $\mathsf{Init} \in \lambda$, we have $\Phi(a, \beta, \lambda) \in \{-1Ƀ,\ N - 1Ƀ\}$.*

Theorem 4 below describes the probability distribution of the payoff of an honest player in contexts where the other players are adversaries. In particular, we will assume that adversaries follow *rational* strategies which, on average, will not make them lose money (but for a negligible amount). In order to define rational strategies, we introduce an auxiliary notion. Given a set of strategies $\mathbf{\Sigma}$ for all players and a blockchain state β, we denote with $\mathrm{E}_{\Phi}(a, \mathbf{\Sigma}, \beta, \eta)$ the *expected payoff* of a over all the runs (β, λ) which are maximal for each player p using $\mathbf{\Sigma}[p]$. Then, we say that player a is *rational in* $\mathbf{\Sigma}$ iff for all β, there exists a negligible f such that, for all η, $\mathrm{E}_{\Phi}(a, \mathbf{\Sigma}, \beta, \eta) \geq f(\eta)$.

Theorem 4 states that the expected payoff of each player p in a given set of honest players $\mathcal{H}$ is either -1 or $N - 1$ with probabilities, respectively, ${}^{N-1}/_{N}$ or ${}^{1}/_{N}$, up-to a negligible error. This holds when either all the players are honest (and the deposit is arbitrary, potentially zero), or the adversaries are rational and the deposit is greater than zero.

Theorem 4. *Let $\mathcal{H} \subseteq \mathcal{P}$ be a set of players, and let $\mathbf{\Sigma}$ be such that $\mathbf{\Sigma}[a]$ is honest for all $a \in \mathcal{H}$. If (i)$\mathcal{H} = \mathcal{P}$, or (ii) $d > 0$ and $\mathbf{\Sigma}[b]$ is rational for all $b \in \mathcal{P} \setminus \mathcal{H}$, then the payoff of each $p \in \mathcal{H}$ is distributed as follows, for all β:*

$$Pr(\Phi(p, \beta, \lambda) = v \mid \mathsf{Init} \in \lambda \ \textit{maximal}) = \begin{cases} \frac{N-1}{N} + f_1(\eta) & \textit{if } v = -1 \\ \frac{1}{N} + f_2(\eta) & \textit{if } v = N - 1 \\ f_3(\eta) & \textit{otherwise} \end{cases}$$

where f_1, f_2, f_3 are negligible functions, and λ is a random variable, sampled so that (β, λ) is maximal with respect to $\mathbf{\Sigma}$.

In the presence of adversaries (i.e., $\mathcal{H} \neq \mathcal{P}$), the hypothesis (*ii*) is necessary. Indeed, if adversaries are not rational, they can simply increase the payoff of honest players by giving them money, or voluntarily losing by timeout. Instead, if $d = 0$, a rational adversary can interfere with the protocol and cause the payoff distribution to differ from the one given by Theorem 4. Remarkably, we will show in Theorem 5 that even if the adversary can alter the payoff *distribution*, she can not diminish the payoff *average*, which is at least negligible in all cases. Hence, the protocol is still secure.

Honest strategies are rational. Theorem 5 below establishes that, even in the case of adversaries with *arbitrary* strategies, for any value of the deposit (including zero), our lottery protocol is secure, i.e. a player which follows the protocol does not lose money, on average.

Theorem 5. *Honest strategies are rational in any set of strategies* $\mathbf{\Sigma}$.

Proof (Sketch). Without loss of generality, assume that only one player, say a, is honest, while the other $N - 1$ players are adversaries, with arbitrary (not necessarily rational) strategies $\mathbf{\Sigma}$. We need to prove that the average payoff of a is nonnegative, up to a negligible quantity. Before Init is put on the ledger, a can redeem her bet, so her payoff is zero. Hence, we only need to consider the case where Init has been put on the ledger. Hereafter, we inductively define *proper* transactions as follows: T is proper either if $\mathsf{T} = \mathsf{Init}$, or all the inputs of T are proper. Note that, in a run of the protocol where all the players are honest, all the transactions put on the ledger are proper.

We start by studying the possible attack strategies, which determine how adversaries put new transactions on the ledger, and how they redeem existing transactions. Adversaries can move their wealth through transactions unrelated to the protocol. Further, they can put on the ledger any transaction obtained by instantiating some transaction template of the protocol. In doing that, they can exploit the malleability of in fields, and make them redeem some previous transaction unrelated to the protocol, consuming part of their wealth in the process. This results in an improper transaction. Its presence on the ledger is not a problem per se, unless it can be exploited to interfere with a proper protocol transaction—e.g., by preventing it to be redeemed, and causing the tournament behavior to diverge from the protocol. So, we now turn our attention to how proper transactions can be redeemed.

We first note that each out script of the protocol transactions (except for the *final* transactions $\mathsf{Win}(\epsilon, p)$ and $\mathsf{CollectOrphanWin}(\pi, p)$) requires a signature from every player, including the honest a. Hence, adversaries can only redeem those transactions through the signatures exchanged during the initialization phase, i.e. using some instantiation of the protocol templates. Further, every transaction template uses its own public keys, so when a protocol transaction T is redeemed by T', then (exactly) one of the following cases applies:

(a) T is Init and T' is a leaf $\mathsf{Win}(p, p)$, or
(b) T has an outgoing edge to T', according to Fig. 3, or
(c) T is $\mathsf{Win}(\pi, p)$ with $\pi \neq \epsilon$, and T' is $\mathsf{CollectOrphanWin}(\pi, p)$, or
(d) T is a final transaction.

For example, if T is a $\mathsf{Turn1}$, then T' must provide a signature made with the keys of $\mathsf{Turn1}$ or $\mathsf{Turn1TO}$. So, as per item (b), T' can only be redeemed by $\mathsf{Turn2}$ or $\mathsf{Timeout1}$. By the above reasoning, and by carefully inspecting the protocol (Fig. 2) and the used transactions (Fig. 1), we see that improper transactions can not interfere with the protocol steps where a proper transaction T is redeemed by a *single-input* template instantiation T'. Indeed, when such redemption happens, T' must be a proper protocol transaction as well. However, this reasoning does *not* extend to the case where the redeeming transaction T' has *multiple* inputs. In our protocol, this is only possible when T' is a $\mathsf{Turn1}$. Indeed, consider the case when a proper $\mathsf{T}_0 = \mathsf{Win}(0\pi, b)$ is on the ledger, as well as a proper $\mathsf{T}_1 = \mathsf{Win}(1\pi, a)$. If T_0 is redeemed by $\mathsf{Turn1}$ (as per item (b)), however, we have no guarantees that such $\mathsf{Turn1}$ is redeeming both T_0 and T_1—because it is possible that $\mathsf{Turn1}$ is instead redeeming the proper T_0 together with an improper transaction $\mathsf{Win}(1\pi, m)$, which was forged by the adversaries. When this interference happens, the protocol continues with an improper $\mathsf{Turn1}$, and T_1 is left on the ledger as an "orphan". Therefore, player a will not be able to participate in the current match. Note that, since $\mathsf{Turn1}$ is the only multiple-input protocol transaction, this interference can only happen at the *start* of a match. After a match is started, the honest player a has at least 1/2 probability to win the match, since a will always respect deadlines (so to avoid losing the match by timeout), and she chose her secret s^a_π in a uniformly random way during initialization. So, either the adversaries lose by timeout, or reveal their secrets and the match proceeds in a fair way.

We can now estimate the average payoff of the honest player a, by tracking her composite bet throughout the tournament rounds (i.e., the sum gained by a so far, that she must invest in further rounds). We start by noting that, at the beginning of each round, at least one of the following must hold:

1. a has lost a previous match.
2. there is an unspent $\mathsf{T} = \mathsf{Win}(\pi, a)$ on the ledger, and the adversaries *do not* interfere: hence, T is redeemed by $\mathsf{Turn1}$, and a participates in the match. In this case, a has at least 1/2 probability to double her composite bet.
3. there is an unspent $\mathsf{T} = \mathsf{Win}(\pi, a)$ on the ledger, and the adversaries *do* interfere: so, T is not redeemed (unlike its sibling Win), and a cannot participate in the match. The transaction T is left "orphan" on the ledger; after $1\,\tau_{Ledger}$, player a can collect the composite bet she earned so far, by putting $\mathsf{CollectOrphanWin}(\pi, a)$ on the ledger. In this way a can redeem her current composite bet.

Since a is honest, she will reveal her secret for a match only *after* $\mathsf{Turn1}$ has been put on the ledger (i.e., when adversaries can no longer interfere in the match). Note that the adversaries do not know the match result when they have to choose whether to interfere or not. Therefore, the whole tournament is similar to a game where a tosses L fair coins in sequence, doubling up her bet every time she wins the flip, and losing the whole stake at the first loss. Her opponent can choose to stop her before any of the coin tosses, but in such case she is allowed to collect what she won so far. Since this coin game is fair, also the average payoff of a in the tournament protocol is nonnegative. $\square$

5 Related Work

Several lottery protocols have been investigated outside the cryptocurrency setting, e.g. by [12,14,15,21,23]. In the last few years, some authors have proposed protocols that work on Bitcoin or similar cryptocurrencies.

Concurrently and independently of our work, Miller and Bentov [20] proposed a lottery protocol, that similarly to ours exploits a tournament tree and requires zero deposit. Two variants of the protocol are presented: the first one only relies on the SegWit feature [19], while the second one proposes a new signature verification opcode, called `MULTIINPUT`. The first variant requires players to statically sign a tree of $O(2^N)$ transactions. To reduce this overhead, our protocol relies on a more flexible signature verification scheme, that allows malleability of in fields, resulting in $O(N)$ transactions. This malleability introduces the interference issues discussed in Sect. 4. Such interferences do not make our protocol insecure, because the average payoff of honest players is non-negative, even for $d = 0$ (Theorem 5), thanks to the garbage collection phase. However, such interferences are still undesirable, because adversaries can prevent honest players from completing the tournament. The second variant of the protocol in [20] achieves $O(N)$ transactions and avoids interferences through a "controlled" malleability of in fields. This is obtained through the new `MULTIINPUT` opcode, which allows to malleate in fields (to achieve $O(N)$ transactions), but only within a pre-specified set (to avoid interferences).

Table 1 summarises the comparison between our protocol and [20] (MB), and also with the protocols in [2] (ADMM), [8] (BK). We also consider a variant of ours and [2], called "2 players iterated", which implement an N-players lottery by running $N-1$ instances of a two-players protocol. Similarly to our tournament protocol, these instances are composed in a tree: only the winners of a level can play at the next one, and the winner of the root collects all the bets. In the iterated versions, the initialization phase is performed for *every* match (using independent keys/secrets), while in the non-iterated version the initialization is done only once, at the beginning.

Table 1. Comparison of cryptocurrency-based lottery protocols.

	ADMM [2] N players	ADMM [2] 2 players iter.	BK [8] N players	MB [20] v1 N players	MB [20] v2 N players	Tournament N players	Tournament 2 players iter.
Deposit	$N(N-1)$	N	$O(N^2)$	0	0	$d \geq 0$	$d \geq 0$
Completion time	$4\,\tau_{Ledger}$	$4\,L\,\tau_{Ledger}$	$O(N)$	$O(L)$	$4\,L\,\tau_{Ledger}$	$(2+6\,L)\,\tau_{Ledger}$	$7\,L\,\tau_{Ledger}$
Off-chain trans	$O(N^2)$	$O(N)$	—	$O(2^N)$	$O(N^2)$	$O(N^2)$	$O(N)$
On-chain trans	$O(N)$	$O(N)$	$O(N^2)$	$O(N^2)$	$O(N)$	$O(N)$	$O(N)$
All-or-nothing	Yes	No	Yes	Yes	Yes	Yes, if $d > 0$	No
Bitcoin features				SegWit	SegWit MULTIINPUT	SegWit in-malleability	SegWit in-malleability

The first row in Table 1 quantifies the deposit: this is constant ($d \geq 0$) in our protocol, zero in [20], while in the others it grows with the number of players. More specifically, the deposit is $O(N^2)$ in [8] and in the non-iterated version of [2], while in the iterated version it is N: intuitively, an N-deposit at the last round is needed to guarantee that the final winner can collect the whole N stake.

The second row quantifies the completion time of the protocol, excluding the communication and computation time (which is marginal in practice, compared to the time required to put transactions on the ledger). Only the non-iterated version of [2] requires constant time; in [8] the time is linear in N, while in the other protocols the time is proportional to $L = \log N$.

The number of off-chain and on-chain transactions required by each protocol is shown in the third and fourth rows. Not that all protocols require a linear number of on-chain transactions, except for [8] and the first version of [20], which require $O(N^2)$ transactions.

The fifth row describes whether a protocol has an ideal behaviour, where only one player wins the whole stake, while the others lose their bets. More specifically, we call a protocol "all-or-nothing" if, assuming rational adversaries, the payoff of honest players is either -1 or $N-1$. The iterated versions of the protocols are not "all-or-nothing": indeed, a rational adversary can simply stop playing after winning a match, collecting the partial winnings and making impossible for any other player to obtain the whole NɃ stake (hence forcing some honest player to gain $-1 < v < N - 1$Ƀ). Instead, our (non-iterated) protocol is "all-of-nothing" when $d > 0$ (Theorem 4).

The last row describes which Bitcoin features a protocol requires to be actually implemented. All protocols make use of non-standard transactions, which are currently handled by a small fraction of the miners. Note that some recent works [6] address the issue of implementing complex protocols on Bitcoin by using only standard transactions. Both our protocol and the ones in [8,20] also rely on the SegWit feature [19]. Additionally, our protocol requires the malleability of in-fields, as discussed in Sect. 2, while the second version of the protocol in [20] requires the `MULTIINPUT` opcode. This opcode would also allow to avoid the interferences outlined in the proof of Theorem 5. The protocol in [8] assumes resilience to malleability attacks, which can be obtained through [19].

The work [16] proposes a general model for secure multiparty computations on cryptocurrencies, which goes beyond the features provided by Bitcoin. Applying this model to lotteries, we would obtain a protocol where the deposit grows linearly in the number of dishonest players. This approach might also allow for reducing the number of rounds from $\log N$ to a constant number.

6 Conclusions

We have presented a lottery protocol based on Bitcoin, where N players can place a bet, and one of them, uniformly chosen, wins all the bets. Our protocol is parametric w.r.t. the deposit $d \geq 0$ that the players have to block throughout the protocol. For any value of d, our protocol ensures that honest players have a negligible average payoff, even in the presence of arbitrary adversaries (Theorem 5). Further, for $d > 0$, the payoff is distributed like an ideal lottery (Theorem 4): that is, the winner gets the sum of all the bets with probability close to $1/N$, while the other players lose their bets with probability close to $N-1/N$. This holds unless the adversaries follow strategies which (on average) make them lose money, and make honest players gain money. According to the terminology in [2], our protocol implements a *fair* lottery.

Although our protocol has been crafted for Bitcoin, the underlying ideas can be used to implement fair lotteries on other frameworks for smart contracts. This could allow to relax the rationality assumption of Theorem 4 when the deposit is zero. For instance, the implementations in Ethereum [9] of Miller and Bentov[5] and of Atzei[6] follow the structure of rounds of the tournament protocol.

Acknowledgments. The authors thank Patrick McCorry, Andrew Miller, and Iddo Bentov for their comments on a preliminary version of this paper. This work is partially supported by Aut. Reg. of Sardinia P.I.A. 2013 "NOMAD".

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł.: Fair two-party computations via Bitcoin deposits. In: Böhme, R., Brenner, M., Moore, T., Smith, M. (eds.) FC 2014. LNCS, vol. 8438, pp. 105–121. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44774-1_8
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. In: IEEE S&P, pp. 443–458 (2014)
3. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, Ł.: On the malleability of Bitcoin transactions. In: Brenner, M., Christin, N., Johnson, B., Rohloff, K. (eds.) FC 2015. LNCS, vol. 8976, pp. 1–18. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48051-9_1
4. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on Bitcoin. Commun. ACM **59**(4), 76–84 (2016)
5. Back, A., Bentov, I.: Note on fair coin toss via Bitcoin. http://www.cs.technion.ac.il/~idddo/cointossBitcoin.pdf (2013)
6. Banasik, W., Dziembowski, S., Malinowski, D.: Efficient zero-knowledge contingent payments in cryptocurrencies without scripts. In: Askoxylakis, I., Ioannidis, S., Katsikas, S., Meadows, C. (eds.) ESORICS 2016. LNCS, vol. 9879, pp. 261–280. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45741-3_14
7. Bartoletti, M., Zunino, R.: Constant-deposit multiparty lotteries on Bitcoin. IACR Cryptology ePrint Archive, 2016/955 (2016). http://eprint.iacr.org/2016/955
8. Bentov, I., Kumaresan, R.: How to use Bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-44381-1_24
9. Buterin, V.: Ethereum: a next generation smart contract and decentralized application platform (2013). https://github.com/ethereum/wiki/wiki/White-Paper
10. Crary, K., Sullivan, M.J.: Peer-to-peer affine commitment using Bitcoin. In: ACM Conference on Programming Language Design and Implementation, pp. 479–488 (2015)
11. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: Christin, N., Safavi-Naini, R. (eds.) FC 2014. LNCS, vol. 8437, pp. 436–454. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45472-5_28
12. Fouque, P.-A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 90–104. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45472-1_7
13. Garay, J., Kiayias, A., Leonardos, N.: The Bitcoin backbone protocol: analysis and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 281–310. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_10

[5] https://github.com/amiller/zero-collateral-lottery.

[6] https://github.com/natzei/constant-deposit-lottery.

14. Goldschlag, D.M., Stubblebine, S.G.: Publicly verifiable lotteries: applications of delaying functions. In: Hirchfeld, R. (ed.) FC 1998. LNCS, vol. 1465, pp. 214–226. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055485
15. Goldschlag, D.M., Stubblebine, S.G., Syverson, P.F.: Temporarily hidden bit commitment and lottery applications. Int. J. Inf. Secur. **9**(1), 33–50 (2010)
16. Kiayias, A., Zhou, H.-S., Zikas, V.: Fair and robust multi-party computation using a global transaction ledger. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 705–734. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_25
17. Kumaresan, R., Bentov, I.: How to use Bitcoin to incentivize correct computations. In: ACM CCS, pp. 30–41 (2014)
18. Kumaresan, R., Moran, T., Bentov, I.: How to use Bitcoin to play decentralized poker. In: ACM CCS, pp. 195–206 (2015)
19. Lombrozo, E., Lau, J., Wuille, P.: Segregated witness (consensus layer), BIP 141. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki
20. Miller, A., Bentov, I.: Zero-collateral lotteries in Bitcoin and Ethereum (2014). http://arxiv.org/abs/1612.05390
21. Rivest, R.L.: Electronic lottery tickets as micropayments. In: Hirschfeld, R. (ed.) FC 1997. LNCS, vol. 1318, pp. 307–314. Springer, Heidelberg (1997). https://doi.org/10.1007/3-540-63594-7_87
22. Ruffing, T., Kate, A., Schröder, D.: Liar, liar, coins on fire!: penalizing equivocation by loss of Bitcoins. In: ACM CCS, pp. 219–230 (2015)
23. Syverson, P.F.: Weakly secret bit commitment: applications to lotteries and fair exchange. In: IEEE CSFW, pp. 2–13 (1998)
24. Szabo, N.: Formalizing and securing relationships on public networks. First Monday **2**(9) (1997)