Constant-Deposit Multiparty Lotteries on Bitcoin for Arbitrary Number of Players and Winners

Shun Uchizono[1], Takeshi Nakai[2], Yohei Watanabe[1,3], and Mitsugu Iwamoto[1]

[1] The University of Electro-Communications, Tokyo, Japan
{uchizono, watanabe, mitsugu}@uec.ac.jp
[2] Toyohashi University of Technology, Aichi, Japan
nakai@cs.tut.ac.jp
[3] National Institute of Advanced Industrial Science and Technology, Tokyo, Japan.

Abstract. Secure lottery is a cryptographic protocol that allows multiple players to determine a winner from them uniformly at random, without any trusted third party. Bitcoin enables us to construct a secure lottery to guarantee further that the winner receives reward money from the other losers. Many existing works for Bitcoin-based lottery use deposits to ensure that honest players never be disadvantaged in the presence of adversaries. Bartoletti and Zunino (FC 2017) proposed a Bitcoin-based lottery protocol with a constant deposit, i.e., the deposit amount is independent of the number of players. However, their scheme is limited to work only when the number of participants is a power of two. We tackle this problem and propose a lottery protocol applicable to an arbitrary number of players based on their work. Furthermore, we generalize the number of winners; namely, we propose a secure (k, n)-lottery protocol. To the best of our knowledge, this is the first work to address Bitcoin-based (k, n)-lottery protocol. Notably, our protocols maintain the constant deposit property.

Keywords: Secure lottery · Bitcoin · Fairness · Elimination tournament.

1 Introduction

1.1 Backgrounds

Consider a bet in which each of the n players gambles one dollar. The champion is randomly chosen from them and he/she receives the sum of the bets, n dollars, as a reward. Secure lottery is a cryptographic protocol that allows us to play such games fairly [12–14,18]. That is, it ensures that no honest player is disadvantaged in the presence of adversarial players who do not follow procedures.

One of the crucial issues in constructing a secure lottery is how to deal with the abort attack, which terminates in the middle of a protocol to avoid losing. To counter the attack, we must enforce an adversary to tell the lottery result to all honest parties. Such a property is typically defined as *fairness*, which ensures

that at the end of a protocol, either all parties learn the output or none of them learn it. Unfortunately, it is known that fairness cannot be achieved without any additional assumption such as the honest majority or trusted third parties [11].

Another fundamental challenge is how to force losers to pay winners. Since a typical cryptographic protocol treats no monetary entity, we cannot require a protocol to guarantee such a property. To enforce the payoff, we need to introduce a setup for handling monetary operations, e.g., a trusted bank [19, 22], e-cash [6, 9, 17], or decentralized cryptocurrency.

Secure lottery based on cryptocurrency. Using cryptocurrency, e.g., Bitcoin [23] and Ethereum [24], we can construct a secure lottery protocol that forces losers to pay winners without relying upon any trusted third party even in the dishonest majority. Informally, in cryptocurrency-based protocols, parties cooperate to create some transactions at the beginning of the protocol and deposit or bet money. One of the transactions is corresponding to n dollars, the prize money. If a protocol guarantees that only the winner can learn the witness to redeem it, it implies that only the winner can receive the prize.

There is a line of work on achieving a variant of fairness using monetary penalties. The monetary penalty enforces adversaries to follow procedures to avoid losing money, and it allows us to achieve fairness. In secure multi-party computation, many works adopt such a definition, e.g., [5, 7, 8, 15, 16, 21].

Similarly, it is known that monetary penalties enable us to construct a secure lottery protocol even in the dishonest majority. Back and Bentov [2] showed a secure lottery based on Bitcoin in the two-party setting. Their protocol can enforce a payment from the loser to the winner. Moreover, it guarantees that an aborting party loses money and then another party obtains money as compensation. Afterward, Andrychowicz, Dziembowski, Malinowski, and Mazurek [1] and Bentov and Kumaresan [2], respectively, proposed Bitcoins-based secure lottery protocols that can be applied to an arbitrary number of parties.

In many works of Bitcoin-based secure lotteries, parties must input deposit to achieve fairness in addition to the bet. Indeed, the existing protocol made of Marcin [1] requires parties to input $O(n^2)$ deposits, where n is the number of parties. The deposit is guaranteed to be returned to every honest party at the end of the protocol. On the other hand, for adversarial parties, the deposit is not returned to them but is instead distributed to honest parties as compensation. Even though the protocol promises to refund deposits to honest parties, it is undesirable to require money other than bets. That is to say, too expensive deposits make it difficult for parties to participate in the protocol. Based on the backgrounds, Bartoletti and Zunino [4] proposed a secure lottery protocol with a constant deposit. Independently, Miller and Bentov [20] proposed a secure lottery without any deposit money. However, as pointed out by Bartoletti and Zunino [4], their scheme has an issue of depending on a Bitcoin specific opcode, MULTIINPUT. To be a generic scheme, it should not rely on a custom scripting language supported by a particular blockchain.

In this paper, we focus on Bartoletti-Zunino work [4]. Informally they realize a constant-deposit protocol based on a single-eliminate tournament, i.e., they use multiple matches between two players to determine one winner of the lottery. However, their protocol assumes that the tournament has a complete binary tree structure. In other words, it has an issue to be applicable only if the number of participants can be expressed in 2^L, where L is a positive integer that refers to the tree depth.

1.2 Our Contribution

This paper presents two contributions. The first one is to solve the issue of the restriction of the number of participants in the Bartolotti and Zunino scheme. That is, we propose $(1, n)$-lottery protocol for an arbitrary positive integer n. Our construction idea is we bias the winning percentage for each match to ensure that all participants are equal even if the tournament is not a complete binary tree.

The second contribution is to generalize the number of winners, namely, we propose a (k, n)-lottery protocol for arbitrary k and n. To realize the protocol, we first construct $(k, k+1)$-lottery protocol. Our (k, n)-lottery protocol is derived from a composition of $(k, k+1)$-lottery protocols. More precisely, in our protocol parties first run $(n-1, n)$-lottery protocol and determine one loser. Afterward, $n-1$ winners run $(n-2, n-1)$-lottery protocol and further determine one loser. Players repeat such a process until deciding $n-k$ losers. To the best of our knowledge, this is the first work to realize (k, n)-lottery protocol based on Bitcoin with a constant deposit.

1.3 Basic Notations

For any positive integer i, let $[i] \coloneqq \{0, 1, \ldots, i-1\}$. We denote by η a security parameter. We suppose that all players are probabilistic polynomial-time algorithms (PPTA) in a security parameter η.

We construct lottery protocols based on a tournament structure represented as a binary tree, as in [4]. Hereafter, we call a champion to distinguish it from the winners of matches in the tournament. In a binary tree, its leaf nodes refer to players, and the other nodes represent a match (or the winner) of two child nodes. Each node at level l in the tree is identified as a $(l+1)$-bit string. For a node π, we denote its child nodes as $\pi_{\text{left}} = \pi \parallel 0$ and $\pi_{\text{right}} = \pi \parallel 1$. Namely, π is the prefix of its child nodes. We write $\pi \sqsubset \pi'$ if π is a prefix of π'. We note that, since we handle an arbitrary number of players, the tournaments may not be the complete binary tree. Hence, the binary tree in our protocol is represented by $\mathit{\Pi} \subseteq \{\{0,1\}^l \mid 1 \leq l \leq L\}$, where the tree has L levels. We denote by P the set of players. Note that, since the players correspond to leaf nodes, it holds $P \subset \mathit{\Pi}$. For a bit string π, $|\pi|$ means the bit length of π. We denote by π_r the root node of a binary tree.

Organization. As a preparation for the introduction of our protocol, we first describe a bitcoin overview in Section 2. Section 3 presents several notations and

useful lemmas regarding tournament structures. In Section 4, we define secure lottery protocol. We show our constructions for $(1, n)$-lottery and (k, n)-lottery in Sections 5 and 6, respectively. In these sections, we prove security of the protocols according to the security definitions, shown in Section 4.

2 A Brief Introduction to Bitcoin

In a blockchain protocol, parties maintain a global *ledger* that holds ordered sets of records, i.e., blocks. To append a new block to the blockchain, parties must race and win to solve a cryptographic puzzle, as known as the *mining* process. The puzzle hardness is parameterized so that the intervals between the growth of blocks are approximately constant at a particular time (about 10 minutes in Bitcoin). Since each block contains a cryptographic hash function of the previous block, the state of each block is preserved by subsequent blocks. Furthermore, when the blockchain diverges into multiple states, proper parties accept the longest chain. Hence, if an adversary tries to rewrite data contained in a block, it needs to reconstruct the subsequent blocks in addition to the block. The adversary must further make the rewritten chain the longest to get other parties to accept it. However, it is infeasible unless the adversary possesses more than half the computing power of the entire network. That is, a blockchain realizes a tamper-resistant public bulletin board based on the assumption about the computing power of adversaries [3, 10].

Bitcoin is a decentralized cryptocurrency based on a blockchain. The Bitcoin ledger manages *transactions* on its blocks. Roughly speaking, a transaction Tx_1 refers to a sender, the amount transferred coins, and the recipient, i.e., it expresses information about "a sender S sends Q coins to a recipient R." The party R can send Q coins to the other party by making a new transaction Tx_2 that refers to Tx_1. Then, Tx_1 becomes a spent transaction and thereafter R cannot re-use it. We can check the balance of a party by referring to all *unspent* transactions corresponding to the party on the blockchain.

Precisely, a transaction form has inputs and outputs. An input specifies a transaction to be used for this remittance. In the above example, the input of Tx_2 is Tx_1('s output). An output (script) specifies the recipient by describing a condition to use the transaction. Typically, the output script contains a signature verification with a public key of the recipient. When a party uses a transaction, he/she needs to write a witness on the transaction as an input script that satisfies the output script of the input transaction. See Fig. 1 that shows the transaction flow in the simplest case. Transaction Tx_2 redeems the previous transaction Tx_1 to use $\$v$. Then, witness w_1 written in the input script of Tx_2 must satisfy the condition ϕ_1, which is the output script of Tx_1. Similarly, to use $\$v$ with reference to Tx_2, it is necessary to create a transaction that holds w in its input script such that $\phi_2(w) = 1$. Hereafter, in the graphical description, an arrow connects the corresponding input and output.

In Bitcoin, by specifying some transactions as inputs, a party can create a transaction to transfer the sum of the coins. Similarly, a single transaction

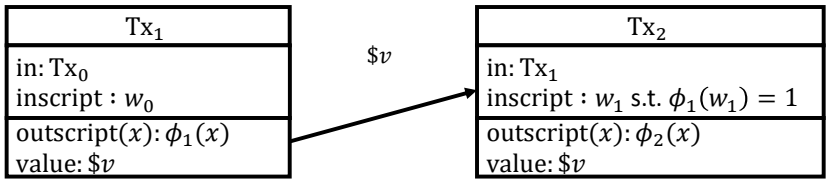

Fig. 1. Graphical description of a transaction flow.

can specify multiple recipients by holding multiple output scripts. Formally, we denote a m-input and l-output transaction in Bitcoin by

$$(\mathsf{in}[m], \mathsf{inscript}[m], \mathsf{value}[l], \mathsf{outscript}[l], \mathsf{lockTime}),$$

where $\mathsf{in}[i]$ is an identifier of the input transaction (i.e., the previous one), $\mathsf{inscript}[i]$ is the corresponding input script (i.e., a witness), $\mathsf{value}[i]$ is the number of coins, $\mathsf{outscript}[i]$ refers to the corresponding output script, and $\mathsf{lockTime}$ specifies the earliest time when the transaction appears on the ledger. Namely, the miners do not approve the transaction until the time specified by $\mathsf{lockTime}$. Note that the sum of the input coins must match the sum of the output coins.

A transaction excluding the input script $(\mathsf{in}[m], \mathsf{value}[l], \mathsf{outscript}[l], \mathsf{lockTime})$ is called the *simplified form*. Typically, as described above, the output script contains a signature verification algorithm to specify the recipient. The input script of the next transaction states a signature in its simplified form in order to prove the creator is the specified recipient.

3 Tournaments with Uniform Winning Probability

3.1 Tournaments with a Single Champion

First, we discuss the case where the champion is only one. In cases of tournaments based on complete binary trees, it is obvious that every party has an equivalent chance to be champion by equating win probabilities of all matches by 1/2. On the other hand, if it is not a complete binary tree, i.e., the number of matches differs from player to player, then it is necessary to bias the winning probabilities to make the tournament equal for all players. We here present several useful lemmas to make fair tournaments even in such cases. (We show the proofs in Appendix A.)

Let us consider a tournament that may not be a complete binary tree and consider the biased probabilities of each match to make it fair. Suppose a match π of which child nodes are π_{left} and π_{right}. We consider two subtrees such that its root nodes are π_{left} and π_{right}, and let v^{π}_{left} and v^{π}_{right} be the number of leaf nodes in these subtrees, respectively. (Note that, if the entire tournament form

is the complete binary tree, then $v^{\pi}_{\text{left}} = v^{\pi}_{\text{right}}$ always holds.) Based on the above notations, for a node π, we define $\mathsf{BiasedPr}(\pi) := v^{\pi}_{\text{left}}/(v^{\pi}_{\text{left}} + v^{\pi}_{\text{right}})$. We can construct a fair tournament based on any binary tree using this function from the following lemma.

Lemma 1 *For any tournament consisting of a binary tree, if the winning probabilities of each match π is set with $(\mathsf{BiasedPr}(\pi), 1 - \mathsf{BiasedPr}(\pi))$, then the tournament is equal for every player.*

3.2 Tournaments with Multiple Champions

Next, we discuss the case of multiple champions. In this case, we must pay attention to the joint winning probabilities of each set of players not only to the winning probability of individual players. For instance, in order to choose two champions, consider the case of dividing the players half into two groups and running a single champion tournament in each group. In this case, although each player has the same probability of being champion, the problem arises that players in the same group can never win simultaneously. To tackle this issue and deal with an arbitrary number of winners, we construct $(k, k+1)$-lottery protocol. Thus, we first discuss a single eliminate tournament that determines k champions from $k+1$ players, $(k, k+1)$-tournament. Afterward, we show that tournaments applicable to an arbitrary number of champions can be constructed by combining multiple $(k, k+1)$-tournaments.

To construct a $(k, k+1)$-tournament, we adopt the single-elimination tournament proceeding as follows: First two players p_1 and p_2 play a match π_b. The winning player is determined to be a champion, and the loser $l_1 \in \{p_1, p_2\}$ plays the next match π_2 with p_2. In a similar way, for $i = 1 \dots k-1$, player p_{i+1} and the previous match loser l_{i-1} plays a match π_i. The loser of $(k-1)$-th match π_{k-1} becomes the only loser of the tournament.

Lemma 2 *If the winning probabilities of match π_i between l_{i-1} and p_{i+1} is set with $(i/(i+1), 1 - i/(i+1))$ for $i = 1 \dots k-1$, then the tournament is equal for every player. Moreover, for any subset $S \subset P$ such that $|S| = k$, the probability of winning the parties in S simultaneously is also equivalent.*

We can construct a (k, n)-tournaments for arbitrary k and n by running $(n-j, n-j+1)$-tournament for $j = 1 \dots n-k$. Concretely, the winners of $(n-j', n-j'+1)$-tournament continue the next $(n-j'-1, n-j')$-tournament to further determine one loser, and the players continue such process until the remaining winners are k players.

Lemma 3 *For any positive integers k and n with $k < n$, if a (k, n)-tournament is composed of sequential executions of $(n-j, n-j+1)$-tournament for $j = 1 \dots n-k$, then the probability of being the winner of a tournament is equivalent for every player. Moreover, for any subset $S \subset P$ such that $|S| = k$, the probability of winning the parties in S simultaneously is also equivalent.*

4 Definition of Secure Lottery Protocol

Suppose a game in which each of n player bets $\$\alpha$. A secure (k, n)-lottery protocol is a cryptographic protocol to k champions who obtain $\$(n\alpha/k)$ from them *fairly*. This section presents the security model of this protocol.

Hereafter, we say that player p can freely redeem transaction Tx if p holds a witness that satisfies the output script of Tx. Let *wealth* of player p at round t mean the total amount of coins in transactions such that p can freely redeem at round t. Note that we ignore coins not involved in the protocol. Also, *payoff* of player p refers to the difference between wealth at the beginning of the protocol and at the end.

Before presenting formal descriptions, we discuss an intuitive understanding of security requirements. First, we focus on the case of $k = 1$, i.e., the champion is only one. As a premise, if all players behave honestly, it is necessary to determine the champion uniformly at random. Of course, it is ideal to achieve this property even in the presence of an adversary. However, such a requirement is somewhat too strong to achieve. For instance, an adversary may abort early after the start of a protocol. In this case, since the protocol terminates without determining the champion, it does not fulfill the condition of determining the champion uniformly at random. Thus, in the case where corrupted players exist, we relax the requirement. More concretely, a secure protocol ensures that the expected value of honest parties' payoffs is never negative for the arbitrary strategy of the adversary.

In the case of $k \geq 2$, the requirements are almost similar to the above, however, there is one additional condition that comes from having multiple champions. We require that, if all players are honest, for any set of players $W \subset P$ such that $|W| = k$, the probability that W becomes champions is the same. In other words, it ensures that not only the tournament is fair for individual players, but also is equal for each set of players. It is also necessary that, if an adversary violates this property, its expected payoff becomes negative. This requirement means that adversaries cannot prevent a certain set of players from becoming champions simultaneously without loss.

To capture the above requirements formally, we introduce several notations. Let $\sigma_{\mathcal{A}}$ denote a strategy set of a PPTA adversary $\mathcal{A}$, and let st_0 denote the ledger state at the beginning of the protocol. We denote by $\Omega(p, st_0, t, \sigma_{\mathcal{A}})$ a random variable of wealth of player p at round t. In the case where there is no corrupted party, we describe $\sigma_{\mathcal{A}} = \perp$. Let β and ϵ denote the round number at the beginning and at the end of the protocol, respectively. We define a random variable with respect to payoff as follows.

$$\Phi(p, st_0, \sigma_{\mathcal{A}}) = \Omega(p, st_0, \epsilon, \sigma_{\mathcal{A}}) - \Omega(p, st_0, \beta, \sigma_{\mathcal{A}}) \tag{1}$$

We denote by $E(\Phi(p, st_0, \sigma_{\mathcal{A}}))$ the expected value of the payoff.

Definition 1 *We say a $(1, n)$-lottery protocol Π is secure if Π fulfills the followings except a negligible probability in η:*

- *If all players are honest,* $E(\Phi(p, st_0, \bot)) = 0$ *and* $\Omega(p, st_0, \epsilon, \bot) \in \{-\alpha, \alpha(n-1)\}$ *for all* $p \in P$.
- *For all PPTA adversaries* $\mathcal{A}$*, i.e., if there exist corrupted players,* $E(\Phi(p, st_0, \sigma_{\mathcal{A}})) \geq 0$ *holds for all* $p \in H$.

Definition 2 *We say a* (k, n)*-lottery protocol* Π *is secure if* Π *fulfills the followings except a negligible probability in* η*:*

- *If all players are honest,* $E(\Phi(p, st_0, \bot)) = 0$ *and* $\Omega(p, st_0, \epsilon, \bot) \in \{-\alpha, (\alpha/k)(n-k)\}$ *for all* $p \in P$*. Furthermore,* $\Pr(\sum_{s \in S} \Omega(s, st_0, \beta, \bot) = k(n-k)) = \binom{n}{k}^{-1}$ *for all* $S = \{s_1, \ldots, s_k\} \subset P$.
- *For any PPTA adversary* $\mathcal{A}$*,* $E(\Phi(p, st_0, \sigma_{\mathcal{A}})) \geq 0$ *holds for all* $p \in H$.
- *For any PPTA adversary* $\mathcal{A}$*, if there exists* $S \subseteq H$ *such that* $|S| \leq k$ *and* $\Pr(\sum_{s \in S} \Phi(s, st_0, \sigma_{\mathcal{A}}) = |S|(n-k)) \neq \binom{n-|S|}{k-|S|}^{-1}$*, the protocol guarantees that* $\sum_{p \in C} E(\Phi(p, st_0, \sigma_{\mathcal{A}})) < 0$.

To achieve a secure protocol, we require players to input *deposit* in addition to the bets. The deposits play a roll of compensation for honest players when an adversary behaves maliciously. We say that a protocol is constant-deposit if the deposit amount of every player is a constant value independent from the number of players.

5 $(1, n)$-Lottery Protocol with Constant Deposits

This section presents a $(1, n)$-lottery protocol for an arbitrary positive integer n. We suppose that a bet amount of each party is $\alpha = 1$. Our protocol is based on single-elimination tournaments with binary tree structure. The tournament consists of $n-1$ two-player matches: the winners of the matches at level $l \in [L]$ play at the next level $l-1$, where L is the tree depth. The winner of the match at level 0 obtains $\$n$ as a reward. Bartoletti and Zunino's protocol set the winning probability to $1/2$ in each match. To construct a protocol for an arbitrary number of players, it is necessary to modify it so that all players are fair to win even if the tournament is not the complete binary tree. The main idea of our protocol is to bias the probability of winning in each match.

5.1 Building Block: Biased Coin-Tossing Protocol

We denote with τ_{Ledger} the sufficient time to write a transaction on the ledger and confirm it. (It is about 60 minutes in Bitcoin.) We denote by $K_p(\mathsf{Tx}, \pi, \mathcal{P})$ a key pair of player p for transaction Tx, which corresponds to a match π. $\mathcal{P}$ refers to players' identifiers corresponding to the match. We suppose that the private part of key pairs is kept secret by p. (Note that we write signing and verification keys without distinguishing between them.) We define $\mathbf{K}(\mathsf{Tx}, \pi, \mathcal{P}) := \{K_p(\mathsf{Tx}, \pi, \mathcal{P}) \mid p \in P\}$.

Let $(\mathsf{ver}, \mathsf{sig})$ be a signature scheme. Following Bartoletti and Zunino's work, we allow the partial signature that enables to exclude of the input field from

the signature subjects. It allows to generate a signature on a transaction before determining the input field of the transaction. Namely, we use the malleability of input fields. Hereafter, a signature written in the input field of transaction $\mathsf{Tx} = (\mathsf{in}[m], \mathsf{inscript}[m], \mathsf{value}[l], \mathsf{outscript}[l], \mathsf{lockTime})$ is for $(\mathsf{value}[l], \mathsf{outscript}[l], \mathsf{lockTime})$. Below, we omit the inputs of signatures and refer to it as $\mathsf{sig}_{K_p(\mathsf{Tx},\pi,\mathcal{P})}$. Also, $\mathsf{sig}_{\mathbf{K}(\mathsf{Tx},\pi,\mathcal{P})}$ means the multi-signature with $\mathbf{K}(\mathsf{Tx}, \pi, \mathcal{P})$.

As described the previous section, we construct a protocol based on a tournament structure. Thus, before presenting our lottery protocol, we show a protocol to realize a match between two parties. Since we deal with tournaments not the complete binary tree, it is necessary to bias some matches to ensure all players to have the same probability of winning the tournament. Hence, we construct a match protocol, called a biased coin tossing protocol, that can parameterize the winning probability.

To handle biased probabilities, we introduce a *winner function* to determine the winner in a match. Let a and b be players that hold secrets s_a and s_b, respectively. We consider a match such that the winner depends on s_a and s_b, and define the function to determine the winner as follows.

$$\mathrm{Winner}(s_a, s_b, v_a, v_b) = \begin{cases} a & \text{if } s_a + s_b \,(\mathrm{mod}\, v_a + v_b) < v_a, \\ b & \text{otherwise.} \end{cases} \tag{2}$$

where v_a and v_b are positive integers. Hereafter, we suppose that s_a and s_b are sampled from $[v_a + v_b]$ uniformly at random.[4] The output $x \in \{a, b\}$ means the winner of the match.

See Protocol 1 and Fig. 2 that shows a protocol of realizing a match π_i in a tournament. (Suppose π_a and π_b be the child nodes of π_i.) A match consists of three types of transactions, Win, $\mathsf{Turn1}$, and $\mathsf{Turn2}$. At the beginning of the protocol, suppose $\mathsf{Win}(\pi_a, a)$ and $\mathsf{Win}(\pi_b, b)$ being on the ledger, which implies that player a and b won the previous matches π_a and π_b, respectively. Now, they play a match π_i. $\mathsf{Turn1}$ is used to aggregate the coins of $\mathsf{Win}(\pi_a, a)$ for the preparation of the match. $\mathsf{Turn2}$ is a transaction of which input is $\mathsf{Turn1}$. See the output script of $\mathsf{Turn1}$. To redeem $\mathsf{Turn1}$, a player must write s_a on the input script of $\mathsf{Turn2}$. Thus, putting $\mathsf{Turn2}$ on the ledger implies to reveal a's secret $s_a^{\pi_i}$. $\mathsf{Win}(\pi_i, a)$ and $\mathsf{Win}(\pi_i, b)$ are transactions of which input is $\mathsf{Turn2}$. See the output script of $\mathsf{Turn2}$. To redeem $\mathsf{Turn2}$, a player must write $s_a^{\pi_i}$ and $s_b^{\pi_i}$ on the input script of the next transaction. Thus, to redeem $\mathsf{Turn2}$, player b must reveal his/her secret $s_b^{\pi_i}$. Furthermore, since $s_a^{\pi_i}$ and $s_b^{\pi_i}$ satisfy either $a = \mathrm{Winner}(s_a^{\pi_i}, s_b^{\pi_i}, v_a, v_b)$ or $b = \mathrm{Winner}(s_a^{\pi_i}, s_b^{\pi_i}, v_a, v_b)$, players can put only one of $\mathsf{Win}(\pi_i, a)$ and $\mathsf{Win}(\pi_i, b)$ on the ledger. The transaction put on the ledger refers to the winner of this match and is used as the input of $\mathsf{Turn1}$ in the next match.

[4] In our protocol, players commit the secrets at the beginning of the protocol by using a cryptographic hash function. Thus, more precisely, we need to extend the bit lengths of secrets to an appropriate length by adding multiples of $v_a + v_b$.

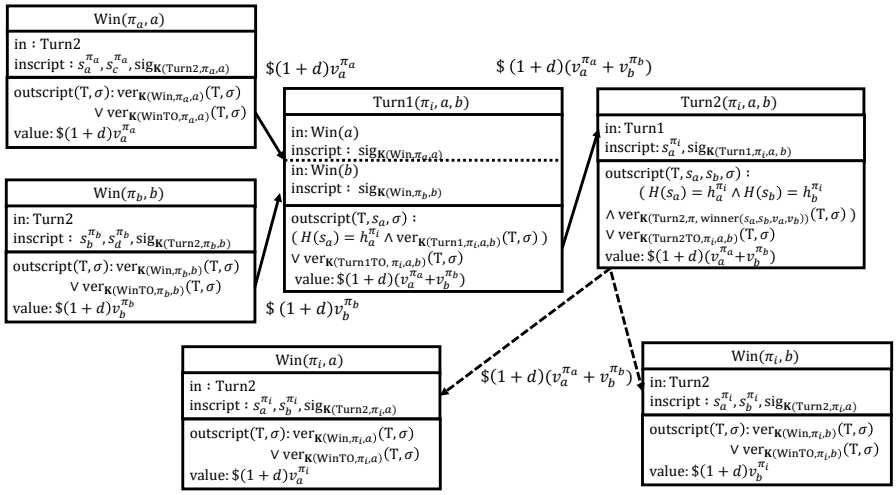

Fig. 2. Graphical description of biased coin-tossing (for match π_i).

5.2 Our Construction of $(1, n)$-lottery

The biased probability of each match in $(1, n)$**-Lottery**. First, we present the biased probability of each match. Let us consider a match π of which child nodes π_a and π_b. As in Section 3.1, we consider two subtrees such that its root nodes are π_a and π_b, and let v_a^π and v_b^π be the number of leaf nodes in these subtrees, respectively. From Lemma 1, we set the winner function in each match π of our $(1, n)$-lottery protocol as $\mathrm{Winner}(s_a^\pi, s_b^\pi, v_a^\pi, v_b^\pi)$.

Our protocol is applicable to an arbitrary binary tree. Let $\mathit{\Pi} \subseteq \{\{0,1\}^n \mid 1 \leq n \leq L\}$ be a binary tree applied to our protocol, and it has L levels. Based on the binary tree and the biased probability, our protocol proceeds as follows.

Precondition: For all $p \in P$, the ledger contains a transaction Bet_p with value $\$(1+d)$, and redeemable with key $K_p(\mathsf{Bet}_p)$.

Initialization phase:

1. For all player $p \in P$, p generates the following secret keys locally. Each player p generates all the following key pairs.
   - For all π such that π is leaf and every $p \in P$:
     $K_p(\mathsf{Bet}_p), K_p(\mathsf{CollectW}), K_p(\mathsf{Init}, a)$
   - For all π and every $p \in P$:
     $K_p(\mathsf{Win}, \pi, a), K_p(\mathsf{WinTo}, \pi, a)$
   - For all π such that π is neither leaf nor root and every $a, b \in P$ such that $a, b \sqsubset \pi$:
     $K_p(\mathsf{Turn1To}, \pi, a, b), K_p(\mathsf{Turn1}, \pi, a), K_p(\mathsf{Turn2To}, \pi, a, b), K_p(\mathsf{Turn2}, \pi, a)$,
     $K_p(\mathsf{Timeout1}, \pi, a, b), K_p(\mathsf{Timeout2}, \pi, a, b)$

Protocol 1 Biased Coin-Tossing $\Pi^{\mathrm{W}}_{a,b}(s_a, s_b, v_a, v_b)$

Setup:

The initialization phase was completed, and $\mathsf{Win}(\pi, a)$ and $\mathsf{Win}(\pi, b)$ have been put already on the ledger. Players a and b hold secrets s_a and s_b, respectively. Let τ be the round of the beginning of the protocol.

Procedure:

One of the players puts $\mathsf{Turn1}(\pi, a, b)$ on the ledger.
a writes s_a on the input script of $\mathsf{Turn2}(\pi, a, b)$, and put the transaction on the ledger.
if $\mathsf{Turn2}(\pi, a, b)$ does not appear within $\tau + 2\tau_{\mathrm{Ledger}}$ **then**
    One of the players puts $\mathsf{Timeout1}(\pi, a, b)$ on the ledger.
    One of the players puts $\mathsf{Win}(\pi, b)$ on the ledger.
b computes $w = \mathrm{Winner}(s_a, s_b, v_a, v_b)$
if $w = a$ **then**
    b puts $\mathsf{Win}(\pi, a)$ on the ledger.
if $w = b$ **then**
    b puts $\mathsf{Win}(\pi, b)$ on the ledger.
if $\mathsf{Win}(\pi, x \in \{a, b\})$ does not appear within $\tau + 4\tau_{\mathrm{Ledger}}$ **then**
    One of the players puts $\mathsf{Timeout2}(\pi, a, b)$ into the ledger.
    One of the players puts $\mathsf{Win}(\pi, a)$ on the ledger.

2. For all player $p \in P$, p generates secrets $s_p^{\pi_p}$ for each π_p, such that $(|\pi_p| < L)$, and broadcasts to the other players his/her public keys and hashes $h_p^{\pi_p} = H(s_p^{\pi_p})$.
3. If $h_p^{\pi_p} = h_{p'}^{\pi_{p'}}$ for some $(p, \pi_p) \neq (p', \pi_{p'})$, the players abort.
4. Parties agree the time τ_{Init} large enough to fall after the initialization phase.
5. Each player signs all transaction templates in Fig. 3 except for Init, and broadcasts the signatures.
6. Each player verifies the signatures received by the others. some signature is not valid or missing, the player aborts the protocol.
7. Each player signs Init, and sends the signature to the first player.
8. The first player puts the (signed) transaction Init on the ledger.
9. If Init does not appear within one τ_{Ledger} , then each p redeems Bet_p and aborts.
10. The players put the signed transactions $\mathsf{Win}(p, p)$ on the ledger, for all $p \in P$.

Tournament execution phase: For all levels $l = L \ldots 1$, players proceed as follows: Run $\Pi^{\mathrm{W}}_{a,b}(s_a^{\pi_a}, s_b^{\pi_b}, v_a^{\pi_a}, v_b^{\pi_b})$ for each π, such that $|\pi| = l - 1$, in parallel. Then, $v_a^{\pi_a}, v_b^{\pi_b}$ denote the biased probability determined in the manner shown in the above.[5]

Garbage collection phase: If there is some unredeemed $\mathsf{Win}(\pi, p)$ such that π is not the root on the ledger, players put $\mathsf{CollectOrphanWin}(\pi, p)$ on the

[5] Only the Win transaction corresponding to the winner of the final match uses the template for the root node. See Fig. 3, and $\mathsf{Win}(\pi_r, a)$ is the corresponding template.

ledger. (If all players behave honestly, this step is not carried out. It is a countermeasure for the transaction insertion attack, shown in Section 5.3.)

At step 2, players prepare all transactions that may be used in the protocol. Note that they then signs the transactions using signing keys of all players. Thus, after this step, it is not possible for some players to collude and forge transactions, except for input and input script fields. The number of transactions created in this step is $O(n^2)$, which is derived from the number of possible match combinations. See $\mathsf{Win}(\pi_r, a)$ in Fig. 3 that is a transaction for the champion. At the end of the tournament execution phase, only the champion is freely redeemable a $\mathsf{Win}(\pi_r, a)$ and can obtain $\$(n+d)$, which is the reward and deposit for the champion. Furthermore, $\mathsf{Win}(\pi_r, a)$ holds outputs to return deposits for each player.

5.3 Transaction Insertion Attack

In our scheme, as in the Bartoletti-Zunino scheme, an adversary can turn an honest player who should be the winner into the loser in a match. The details of the attack are described below.

Settings. Consider a match π with honest player a and malicious player b, where they are winners of the previous matches π_0 and π_1, respectively. Let player c be the loser of π_0, and let π' be the parent node of π. Player b has a freely redeemable transaction T_b with $\$(v_a^\pi + v_b^\pi)(1+d)$ in the external to the protocol.

Procedures. Suppose when honest player a puts $\mathsf{Turn2}(\pi, a, b)$ on the ledger in the biased coin tossing protocol for π, player b realizes that he has lost the match. Then, b redeems T_b through a transaction $\mathsf{Win}(\pi, b)$ by malleating its input and input script fields. (Note that in our scheme, we assume the input malleability.) Player b can now redeem both his transaction and $\mathsf{Win}(\pi', c)$ by putting $\mathsf{Turn1}(\pi_1, b, c)$ on the ledger. Player a can redeem the pending $\mathsf{Turn2}(\pi, a, b)$ (after its timeout has expired) using $\mathsf{Timeout2}(\pi, a, b)$, and then redeem that with $\mathsf{Win}(\pi, a)$. This transaction is now orphan, i.e. it can no longer be used in the next rounds because its $\mathsf{Win}(\pi', c)$ was already redeemed by b. However, the orphan transaction can be redeemed in the garbage collection phase by $\mathsf{CollectW}(\pi, a)$. Thus, player a can collect $\$(v_a^\pi + v_b^\pi)(1 + d)$ at the garbage collection phase.

As shown above, in order to realize this attack in match π, an adversary needs to invest additional coins $\$(v_a^\pi + v_b^\pi)(1+d)$ into the protocol. The affected honest player can collect $\$d$ by the root $\mathsf{Win}(\pi_r, a)$ and $\$(v_a^\pi + v_b^\pi)(1 + d)$ by the garbage collection. Informally, this adversarial scenario does not affect the security since the honest player who is applied this attack would rather gain due to the deposit. We present security proof of our protocol in the next subsection.

5.4 Security Proof

This section shows security proof of our $(1, n)$-lottery protocol. Our proof is based on the fact that the possible attack strategies for adversaries is only the transaction insertion attack or the rejection of revealing their secrets.

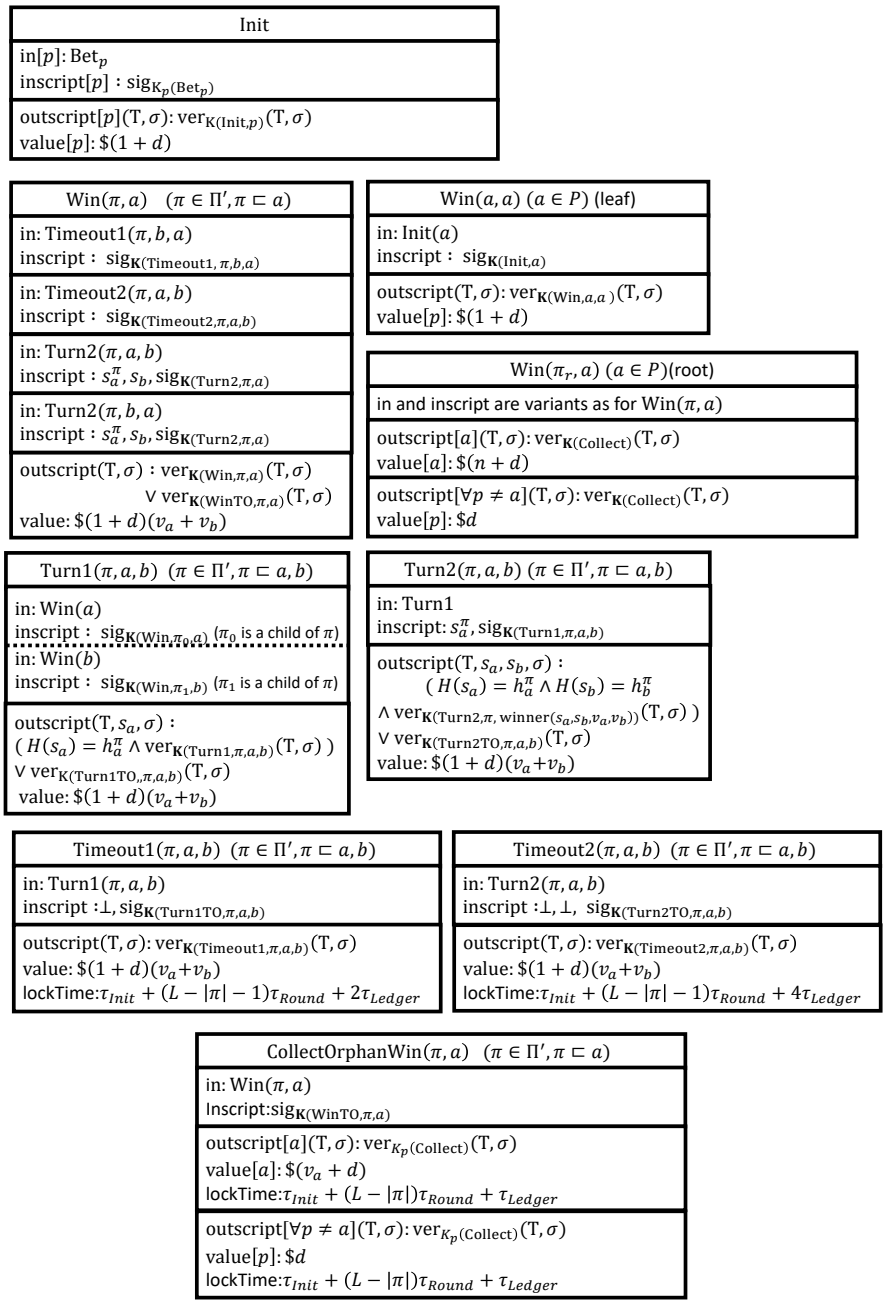

Fig. 3. Transaction templates used in our protocols. Let Π' be the set of nodes excluding leafs and the root. (Part I) Transaction templates for our $(1, n)$-lottery protocol. The dashed line in the inscript field indicates that both inscripts are redeemed at the same time. On the other hand, a solid line indicates that only one of the inscriptions is redeemed.

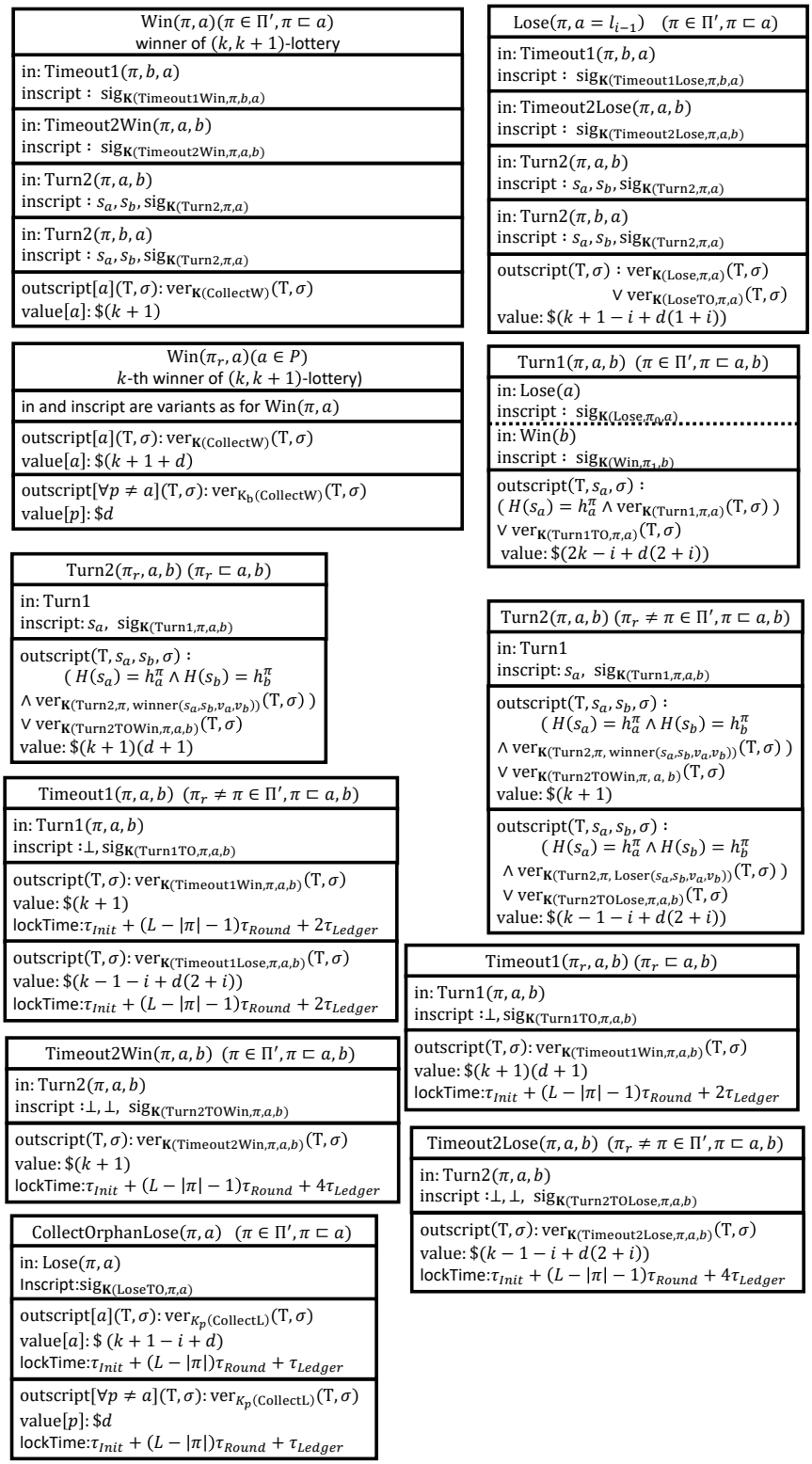

Fig. 4. Transaction templates for our $(k, k+1)$-lottery protocol. Let π_i denote i-th match of the protocol. We omit Win and Init descriptions since they are almost the same in Fig. 3. The differences from Fig. 3 are just changes of the values \$$(1+d)$ to \$$(k+d)$.

Theorem 1 *Our $(1, n)$-lottery protocol is secure and constant deposit.*

Proof (Sketch). We prove that our protocol fulfills the definition 1. In the case of $C = \emptyset$, it is obvious from Lemma 1. If an adversary deviates from the procedure or aborts at some step in the initialization phase, players terminate the protocol. In this case, all honest players do not lose money since no money transfers occur. Thus, we suppose that the initialization phase completes correctly. Below, we discuss two cases in the tournament execution phase, (i) an adversary rejects to reveal its secrets and (ii) an adversary applies the transaction insertion attack, described in Section 5.3.

In the case of (i), the biased coin-tossing protocol guarantees that the player who did not reveal the secret is treated as a loser. Thus, no honest player is lost nevertheless an adversary refuses to disclose its secret in any matches.

In the case of (ii), let us consider the case where an adversary applies the transaction insertion attack to a player p at match π. The player p obtains payoff $\$(v_p^\pi + d - 1)$ by CollectOrphanWin at the end of the protocol, as described Section 5.3. Note that, in this case, the player p does not reveal his/her secret corresponding to match π. Furthermore, at the beginning of the match, we can express the expected payoff of p as follows

$$\frac{v_p^\pi}{v_{p^r}^{\pi_r}} \times \$(n-1) + (1 - \frac{v_p^\pi}{v_{p^r}^{\pi_r}}) \times \$(-1) = \$(v_p^\pi - 1) \tag{3}$$

The inequality $v_p^\pi + d - 1 > v_p^\pi - 1$ implies that $E(\Phi(p, st_0, \sigma_{\mathcal{A}})) > 0$ if $d > 0$. Also, this property holds for an arbitrary positive integer d, our protocol satisfies constant-deposit. From the above, $(1, n)$-lottery protocol is secure. □

6 (k, n)-Lottery Protocol with Constant Deposits

This section shows our (k, n)-lottery protocol for arbitrary k and n and $(k, k+1)$-lottery protocol for arbitrary k and $k + 1$. We compose a (k, n)-lottery protocol from a composition of $(k, k + 1)$-lottery protocols as follows:

First n parties run $(n - 1, n)$-lottery and determine one loser. Thereafter, the remaining $n - 1$ winners run $(n - 2, n - 1)$-lottery and further determine one loser. Parties repeat the similar process until removing $n - k$ players, i.e., resulting in k winners.

6.1 Building Block: Modified Biased Coin-Tossing Protocol

We adopt the single-elimination tournament as described in Lemma 2 to construct a $(k, k + 1)$-lottery protocol. That is, it is a tournament where the winner of each match becomes the champion of $(k, k + 1)$-lottery, and the loser moves on to the next match. Protocol 1 is insufficient to implement such a tournament since it does not enable the loser to proceed to the next match. Hence, we here modify the protocol to resolve this problem.

Protocol 2 Modified Biased Coin-Tossing $\Pi^{\mathrm{WL}}_{a,b}(s_a, s_b, v_a, v_b)$

Setup:

The initialization phase was successfully completed, and $\mathsf{Lose}(\pi, a)$ and $\mathsf{Win}(\pi, b)$ have been put already on the ledger. Players a and b hold secrets s_a^π and s_b^π, respectively. Let τ be the round of the beginning of the protocol.

Procedure:

One of the players puts $\mathsf{Turn1}(\pi, a, b)$ on the ledger.
a writes s_a^π on the input script of $\mathsf{Turn2}(\pi, a, b)$, and put the transaction on the ledger.
if $\mathsf{Turn2}(\pi, a, b)$ does not appear within $\tau + 2\tau_{\mathrm{Ledger}}$ **then**
  One of the players puts $\mathsf{Timeout1}(\pi, a, b)$ on the ledger.
  One of the players puts $\mathsf{Win}(\pi, b)$ and $\mathsf{Lose}(\pi, a)$ on the ledger.
b computes $w = \mathrm{Winner}(s_a, s_b, v_a, v_b)$
if $w = a$ **then**
  b puts $\mathsf{Win}(\pi, a)$ and $\mathsf{Lose}(\pi, b)$ on the ledger.
if $w = b$ **then**
  b puts $\mathsf{Win}(\pi, b)$ and $\mathsf{Lose}(\pi, a)$ on the ledger.
if $\mathsf{Win}(\pi, x \in \{a, b\})$ does not appear within $\tau + 4\tau_{\mathrm{Ledger}}$ **then**
  One of the players puts $\mathsf{Timeout2Win}(\pi, a, b)$ into the ledger.
  One of the players puts $\mathsf{Win}(\pi, a)$ on the ledger.
if $\mathsf{Lose}(\pi, x \in \{a, b\})$ does not appear within $\tau + 4\tau_{\mathrm{Ledger}}$ **then**
  One of the players puts $\mathsf{Timeout2Lose}(\pi, a, b)$ into the ledger.
  One of the players puts $\mathsf{Lose}(\pi, a)$ on the ledger.

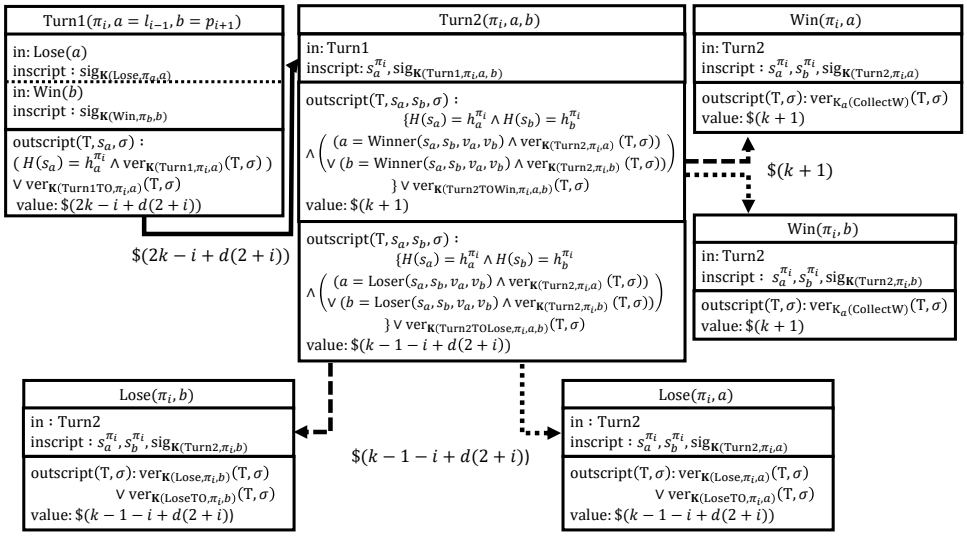

Fig. 5. Graphical description of modified biased coin-tossing (for match π_i). We denote with π_a and π_b child nodes of π_i. Note that Win and Lose redeemed by Turn1 are omitted.

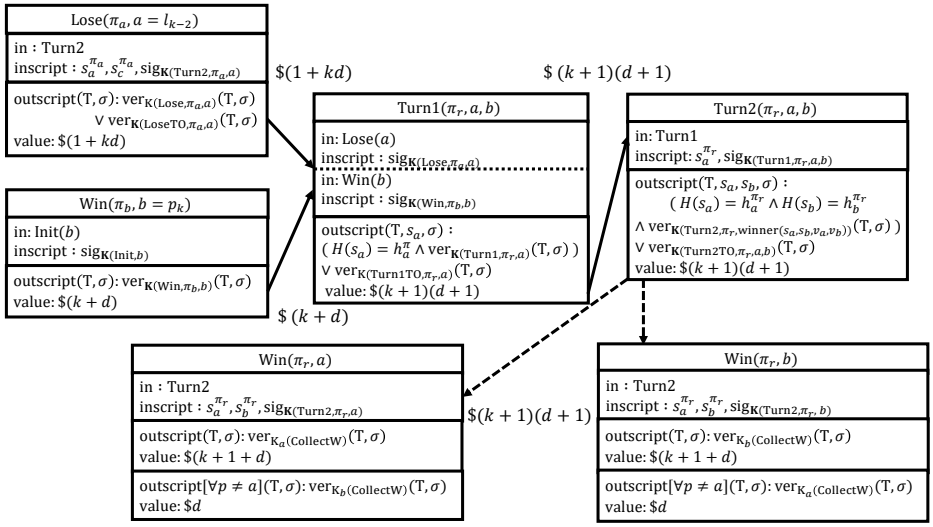

Fig. 6. Graphical description of biased coin-tossing (for match π_r) for the final match of $(k, k+1)$-lottery.

See Protocol 2 and Fig. 5 that show the modified protocol. The Loser function described in the Lose transaction returns the inverse of Winner function, i.e., it specifies the loser. That is, unlike Protocol 5.1, the loser also puts Lose transaction of which input is Turn2, and receives coins used in the next match. Moreover, since Turn2 has two output scripts, we set the timeouts for each of Win and Lose by preparing Timeout2Win and Timeout2Lose transactions. If Win or Lose is not published within the time limit, it is dealt with by publishing Timeout2Win or Timeout2Lose respectively. Fig. 7 shows flows of procedures when a timeout occurs.

6.2 Our Construction of $(k, k+1)$-Lottery Protocol

Let the bet mount be $\$k$ for each player in this section.

The biased probability of each match in $(k, k+1)$-Lottery. Suppose a match between p_{i+1} and l_{i-1} in i-th match π_i, where l_{i-1} is the loser of $(i-1)$-th match. From Lemma 2, for $i = 1 \ldots k-1$, the winning probability of p_{i+1} in π_i is set as $i/(i+1)$.

Based on the biased probability, our protocol proceeds as follows.

Precondition: for all players, the ledger contains a transaction Bet_p with value $\$(1+d)$, and redeemable with key $K_p(\mathsf{Bet}_p)$.

Initialization phase:

1. For all player $p \in P$, p generates the following secret keys locally.

- For all π such that $|\pi| = L$:
  $K_p(\mathsf{Bet}_p), K_p(\mathsf{CollectW}), K_p(\mathsf{CollectL}), K_p(\mathsf{Init}, a)$
- For all π such that $1 \leq |\pi| \leq L$:
  $K_p(\mathsf{Win}, \pi, a), K_p(\mathsf{WinTo}, \pi, a), K_p(\mathsf{Lose}, \pi, a), K_p(\mathsf{LoseTo}, \pi, a)$
- For all π such that $1 \leq |\pi| < L$:
  $K_p(\mathsf{Turn1To}, \pi, a, b), K_p(\mathsf{Turn1}, \pi, a, b),$
  $K_p(\mathsf{Turn2ToWin}, \pi, a, b), K_p(\mathsf{Turn2ToLose}, \pi, a, b), K_p(\mathsf{Turn2}, \pi, a),$
  $K_p(\mathsf{Timeout1Win}, \pi, a, b), K_p(\mathsf{Timeout1Lose}, \pi, a, b),$
  $K_p(\mathsf{Timeout2Win}, \pi, a, b), K_p(\mathsf{Timeout2Lose}, \pi, a, b)$

2. For all player $p \in P$, p generates secrets $s_p^{\pi_p}$ for each π_p, such that $(|\pi_p| < L)$, and broadcasts to the other players his/her public keys and hashes $h_p^{\pi_p} = H(s_p^{\pi_p})$.
3. If $h_p^{\pi_p} = h_{p'}^{\pi_{p'}}$ for some $(p, \pi_p) \neq (p', \pi_{p'})$, the players abort.
4. Parties agree the time τ_{Init} large enough to fall after the initialization phase. (This step is necessary to determine $\mathsf{lockTime}$ values built in the subsequent steps.)
5. Each player signs all the transaction templates in Fig. 3 and 4 except for Init and broadcasts the signatures.
6. Each player verifies the signatures received by the others. some signature is not valid or missing, the player aborts the protocol.
7. Each player signs Init, and sends the signature to the first player.
8. The first player puts the (signed) transaction Init on the ledger.
9. If Init does not appear within one τ_{Ledger} , then each p redeems Bet_p and aborts.
10. The players put the signed transactions $\mathsf{Win}(p, p)$ on the ledger, for all $p \in P$.

Tournament execution phase: For levels $i = k - 1 \ldots 2$, players proceed as follows: Run $\Pi^{\mathrm{WL}}_{a,b}(s_a^{\pi_i}, s_b^{\pi_i}, v_a^{\pi_i}, v_b^{\pi_i})$ for each π, such that $|\pi| = i - 1$.
For level $i = 1$, players proceed as follows: Run $\Pi^{\mathrm{W}}_{a,b}(s_a^{\pi_i}, s_b^{\pi_i}, v_a^{\pi_i}, v_b^{\pi_i})$. Then, v_a^{π}, v_b^{π} denote the biased probability determined in the manner shown in the previous subsection.

Garbage collection phase: If there is some unredeemed $\mathsf{Win}(\pi, p)$ such that π is a leaf on the ledger, players put $\mathsf{CollectOrphanWin}(\pi, p)$ on the ledger. Similarly, if there is some unredeemed $\mathsf{Lose}(\pi, p)$ on the ledger, players put $\mathsf{CollectOrphanLose}(\pi, p)$ on the ledger.

At the end of the tournament execution phases, all champions can freely redeem $\mathsf{Win}(\pi, a)$ or $\mathsf{Win}(\pi_r, a)$ as rewards. Also, $\mathsf{Win}(\pi_r, a)$ guarantees that every honest party can collect their deposits. As in Protocol 5.1, the number of transactions prepared at step 5 is $O(n^2)$.

Theorem 2 *Our $(k, k+1)$-lottery protocol is secure and constant deposit.*

Proof (Sketch). We prove that our protocol fulfills Definition 2. In the case of $C = \emptyset$, it is obvious from Theorem 2. As in the proof of Theorem 1, we suppose

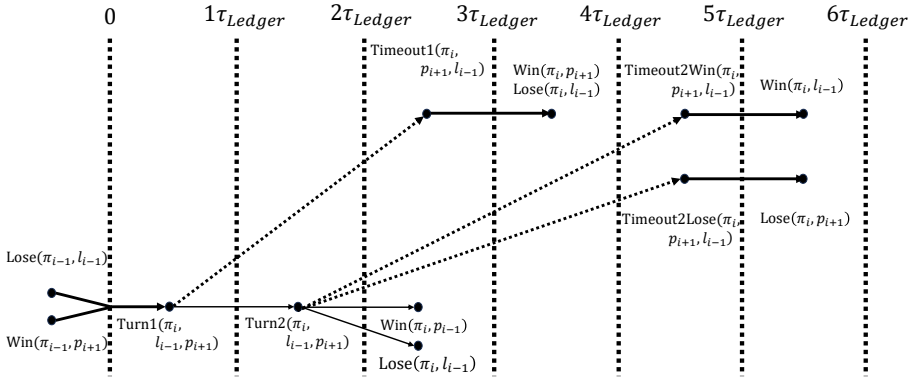

Fig. 7. Graph of the transactions in a tournament round. An arrow from transaction T to T' means that T redeems T'. Thick arrows mean any player can redeem; dashed edges mean any player can redeem, but only after a timeout. Thin arrows mean that only the player who knows the secret on the label can redeem it. $\tau_{Round} := 6\tau_{Ledger}$ refers to the number of rounds in each match.

that the initialization phase completes correctly and focuses on the tournament execution phase.

Below, we discuss two cases in the tournament execution phase: (i) an adversary rejects to reveal its secrets, and (ii) an adversary applies the transaction insertion attack, described in Section 5.3. The proof of case (i) is omitted since the same argument holds for Theorem 1. For case (ii), we consider further dividing it into the following two cases: (a) an adversary applies the transaction insertion attack to player p_{i+1} at match π_i, where π_i is the first match for p_{i+1}, (b) an adversary applies the attack to player l_{i+1} at match π_i, where l_{i+1} is the loser of the previous match.

Then, the player p_{i+1} obtains payoff $\$(k+d)$ at the end of the protocol. Also, player l_{i-1} at match π_i obtains payoff $\$(1-i+d)$ by CollectOrphanWin at the end of the protocol. Thus, to confirm that the honest party does not lose by the attack, it requires that the obtained payoff is more than the expected payoff at match π_i. In the case of (a), for any π_i, the expected payoff of player p_i is 0 because p_i because it is fair to the players from Theorem 2.

In the case of (b), The expected payoff of honest l_{i-1} is as follows.

$$\frac{k+1-i}{k+1} \times \$1 + (\frac{i}{k+1}) \times \$(-k) = \$(1-i) \tag{4}$$

From this Eq.(4), we can see $E(\Phi(p, st_0, \sigma_{\mathcal{A}})) - E(\Phi(p, st_0, \perp)) > 0$ since $1-i+d > 1-i$ for $i \in [k]$ if $d > 0$. It implies that the l_{i-1}'s expected payoff when the adversary applies the transaction insertion attack is larger than their expected payoff when all parties behave honestly.

Next, we confirm that every subset has the same winning probability for all $S = \{s_1, \ldots, s_k\} \subset P$. It is obvious if all honest parties behave honestly

since one loser is determined uniformly at random. To change the probability, adversaries can make two attacks: rejections of their secret or the transaction insertion attack. In both cases, since the expected payoff of the adversary is negative, the protocol fulfills the requirement. This $(k, k+1)$-lottery protocol is secure from the above. □

6.3 Construction of (k, n)-Lottery from $(k, k+1)$-Lottery

Let a bet amount of each party be $\alpha = n!/k!$. There are two technical challenges to realizing a secure (k, n)-lottery based on this strategy. The first one is to connect each $(k', k'+1)$-lottery protocol such that malicious parties cannot escape the protocol in the middle. This issue is derived from the fact that if parties run several lottery protocols sequentially, corrupted players can abort without losing at the initialization process of the next lottery. To circumvent this issue, we aggregate the initialization processes of all protocols in the first $(n-1, n)$-lottery protocol. That is, players prepare all of the secrets, signing (verification) keys, and transactions used in the entire (k, n)-lottery in the initialization of $(n-1, n)$-lottery protocol. By this modification, parties can skip all initialization phases after the completion of $(n-1, n)$-lottery protocol. Note that the number of transactions created in the initialization phase is $O(n^3)$, which can be derived from the number of possible match combinations.

Further, we also slightly change the tournament execution phase, except for the last $(k, k+1)$ lottery protocol. More concretely, we modify each match protocol, i.e., Fig. 5 and 6, such that Win transactions connect two tournament execution phases. See Appendix B for the modification details.

As the second challenge, it is necessary to ensure that (k, n)-lottery composed of sequential executions of $(n-j, n-j+1)$-lottery for $j \in [n-k]$ is indeed fair. We present the security proof of our (k, n)-lottery protocol below.

Theorem 3 *Our (k, n)-lottery protocol is secure and constant deposit.*

Proof (Sketch). We prove that our (k, n)-lottery protocol fulfills Definition 2.

As in the proof of our $(k, k+1)$-lottery protocol, we focus on the payoff obtained by a player who is affected by the transaction insertion attack at π_i^j, where π_i^j is i-th match of j-th $(n-j, n-j+1)$-lottery protocol. We denote by w_i^j and l_i^j the winner and loser of match π_i^j, respectively. As in the proof of Theorem 2, we consider further dividing case (ii) into the following two cases: (a) an adversary applies the transaction insertion attack to player l_{i-1}^j at match π_i, (b) the attack to player w_{i+1}^{j-1} at match π_i^j. In the case of (b), the player l_{i-1}^j obtains payoff $\$((n-1)!/\{(k-1)!(n-j+1)(n-j)\} \times (nj - ni - j^2 + j) + d)$ at the end of the protocol. Thus, to confirm that the honest party does not lose by the attack, it requires that the obtained payoff is more than the expected payoff at match π_i^j. The expected payoff of honest l_{i-1}^j is as follows.

$$\frac{(n-j+1-i)k}{(n-j+1)(n-j)} \times \$\frac{(n-1)!(n-k)}{k!} + (1 - \frac{(n-j+1-i)k}{(n-j+1)(n-j)}) \times \$(-\frac{(n-1)!}{k!})$$
$$= \$\frac{(n-1)!}{(k-1)!(n-j+1)(n-j)}(nj - ni - j^2 + j).$$

Note that $((n-j+1-i)k)/((n-j+1)(n-j))$ is the winning probability of l^j_{i-1}. From then on, we could prove similar to the proof of our $(k, k+1)$-lottery protocol. A similar calculation in the case of (a) shows no loss. Hence, our (k, n)-lottery is secure. □

Acknowledgment This work was supported by JSPS KAKENHI Grant Numbers JP18H05289, JP21H03395, JP21H03441, JP22H03590, JP23H00468, JP23H00479, 23K17455, JP23K16880, JST CREST JPMJCR22M1, JPMJCR23M2, and MEXT Leading Initiative for Excellent Young Researchers.

References

1. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy. pp. 443–458 (2014). https://doi.org/10.1109/SP.2014.35
2. Back, A., Bentov, I.: Note on fair coin toss via bitcoin. CoRR **abs/1402.3698** (2014), http://arxiv.org/abs/1402.3698
3. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Advances in Cryptology – CRYPTO 2017. pp. 324–356. Springer International Publishing (2017)
4. Bartoletti, M., Zunino, R.: Constant-deposit multiparty lotteries on bitcoin. In: International Conference on Financial Cryptography and Data Security. pp. 231–247. Springer (2017)
5. Baum, C., David, B., Dowsley, R.: Insured mpc: Efficient secure computation with financial penalties. In: Financial Cryptography and Data Security. pp. 404–420. Springer International Publishing, Cham (2020)
6. Belenkiy, M., Chase, M., Erway, C.C., Jannotti, J., Küpçü, A., Lysyanskaya, A., Rachlin, E.: Making p2p accountable without losing privacy. In: Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society. p. 31–40. Association for Computing Machinery (2007). https://doi.org/10.1145/1314333.1314339
7. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Advances in Cryptology – CRYPTO 2014. pp. 421–439 (2014)
8. Bentov, I., Kumaresan, R., Miller, A.: Instantaneous decentralized poker. In: Advances in Cryptology – ASIACRYPT 2017. pp. 410–440. Springer International Publishing (2017)
9. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology. pp. 199–203. Springer US (1983)
10. Choudhuri, A.R., Goyal, V., Jain, A.: Founding secure computation on blockchains. In: Advances in Cryptology – EUROCRYPT 2019. pp. 351–380. Springer International Publishing (2019)

11. Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing. p. 364–369. STOC '86, Association for Computing Machinery, New York, NY, USA (1986). https://doi.org/10.1145/12130.12168
12. Goldschlag, D.M., Stubblebine, S.G.: Publicly verifiable lotteries: Applications of delaying functions. In: Financial Cryptography. pp. 214–226. Springer Berlin Heidelberg (1998)
13. Hall, C., Schneier, B.: Remote electronic gambling. In: Computer Security Applications Conference, Annual. p. 232. IEEE Computer Society (1997). https://doi.org/10.1109/CSAC.1997.646195
14. Konstantinou, E., Liagkou, V., Spirakis, P., Stamatiou, Y.C., Yung, M.: Electronic national lotteries. In: Financial Cryptography. pp. 147–163. Springer Berlin Heidelberg (2004)
15. Kumaresan, R., Bentov, I.: How to use bitcoin to incentivize correct computations. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. p. 30–41. Association for Computing Machinery (2014). https://doi.org/10.1145/2660267.2660380
16. Kumaresan, R., Moran, T., Bentov, I.: How to use bitcoin to play decentralized poker. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. p. 195–206. Association for Computing Machinery, New York, NY, USA (2015). https://doi.org/10.1145/2810103.2813712
17. Küpçü, A., Lysyanskaya, A.: Usable optimistic fair exchange. In: Topics in Cryptology - CT-RSA 2010. pp. 252–267. Springer Berlin Heidelberg (2010)
18. Kushilevitz, E., Rabin, T.: Fair e-lotteries and e-casinos. In: Topics in Cryptology — CT-RSA 2001. pp. 100–109. Springer Berlin Heidelberg (2001)
19. Lindell, A.Y.: Legally-enforceable fairness in secure two-party computation. In: Malkin, T. (ed.) Topics in Cryptology – CT-RSA 2008. pp. 121–137. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
20. Miller, A., Bentov, I.: Zero-collateral lotteries in bitcoin and ethereum. In: 2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW). pp. 4–13 (2017). https://doi.org/10.1109/EuroSPW.2017.44
21. Nakai, T., Shinagawa, K.: Constant-round linear-broadcast secure computation with penalties. Theoretical Computer Science **959**, 113874 (2023). https://doi.org/10.1016/j.tcs.2023.113874
22. Nakai, T., Shinagawa, K.: Secure multi-party computation with legally-enforceable fairness. In: Information and Communications Security. pp. 161–178. Springer Nature Singapore (2023)
23. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review (2008)
24. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)

A Proofs of Lemmas

Proof of Lemma 1: Let $\pi_{l\in[L]}$ such that $|\pi_l| = l$ be the l-th match for player p. Suppose v_p^l/v_p^{l-1} be the probability that p wins at π_l. Then, the probability that p wins the tournament holds:

$$\frac{1}{v_p^l} \times \frac{v_p^l}{v_p^{l-1}} \times \cdots \times \frac{v_p^1}{v_p^0} = \frac{1}{n_p^0} = \frac{1}{N}.$$

This is also true for any player. □

Proof of Lemma 2: Let π_i such that $|\pi_i| = i \in [k]$ be the i-th match for player p_{i+1} and l_{i-1}, where l_{i-1} is the loser of $(i-1)$-th match The probability that p wins the tournament holds:

$$1 - \frac{i}{i+1} \times \frac{i+1}{i+2} \times \cdots \times \frac{k}{k+1} = \frac{k}{k+1}.$$

This is also true for any player. Moreover, the probability of winning the parties in S simultaneously equals the probability of losing $p \notin S$. Thus, the probability of winning the parties in S simultaneously is equivalent for any $S \subset P$ such that $|S| = k$. □

Proof of Lemma 3: For any $j \in [k]$, the winning probability in $(n-j, n-j+1)$-lottery can be expressed by $(n-j-1)/(n-j)$, as shown in Lemma 2. Since the probability of each $(k', k'+1)$-lottery is independent, the probability that a player wins the entire (k, n)-lottery can be written as:

$$\frac{n-1}{n} \times \frac{n-2}{n-1} \times \cdots \times \frac{k}{k+1} = \frac{k}{n}.$$

Moreover, since the losers are chosen uniformly at random in each $(k', k'+1)$-lottery, it is obvious that the winning probability of any set of k players is equivalent.

B Transaction Templates for Constructing (k, n)-Lottery

To combine multiple $(k, k+1)$-lottery protocols, we modify Win transactions. See Fig. 8 that shows the point of connection between j-th lottery and $(j+1)$-th lottery protocols. The output scripts of $\mathsf{Win}(\pi^j, a)$ in j-th lottery are used as input of $\mathsf{Win}(\pi^{j+1}, a)$ in $(j+1)$-th lottery protocol. Furthermore, $\mathsf{Win}(\pi_r^j, a)$ redistributes $\$d$ to $\mathsf{Win}(\pi, a)$ for deposits of the next lottery. With this modification, $K_p(\mathsf{WinInit}, \pi, a)$ and $K_p(\mathsf{Return}, \pi, a)$ are added to the key pairs prepared in the initialization phase.

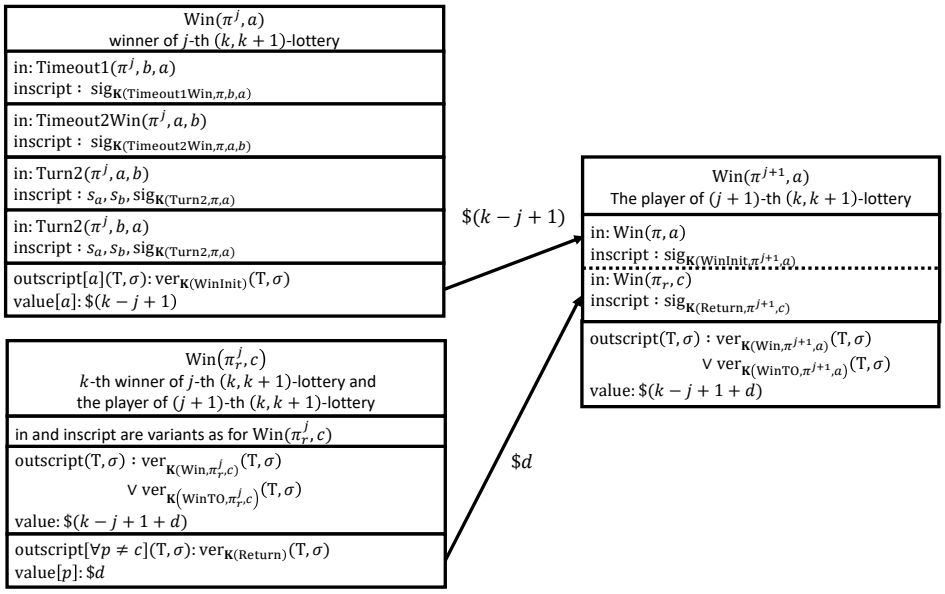

Fig. 8. Graphical description of the connection between j-th lottery and $(j+1)$-th lottery protocols