21 - Bringing Down the Complexity: Fast Composable Protocols for Card Games Without Secret State

Bernardo David[1,3(✉)], Rafael Dowsley[2,3], and Mario Larangeira[1,3]

[1] Tokyo Institute of Technology, Tokyo, Japan
{bernardo,mario}@c.titech.ac.jp
[2] Aarhus University, Aarhus, Denmark
[3] IOHK, Hong Kong, China
rafael@cs.au.dk

Abstract. While many cryptographic protocols for card games have been proposed, all of them focus on card games where players have some state that must be kept secret from each other, *e.g* closed cards and bluffs in Poker. This scenario poses many interesting technical challenges, which are addressed with cryptographic tools that introduce significant computational and communication overheads (*e.g.* zero-knowledge proofs). In this paper, we consider the case of games that do not require any secret state to be maintained (*e.g.* Blackjack and Baccarat). Basically, in these games, cards are chosen at random and then publicly advertised, allowing for players to publicly announce their actions (before or after cards are known). We show that protocols for such games can be built from very lightweight primitives such as digital signatures and canonical random oracle commitments, yielding constructions that far outperform all known card game protocols in terms of communication, computational and round complexities. Moreover, in constructing highly efficient protocols, we introduce a new technique based on verifiable random functions for extending coin tossing, which is at the core of our constructions. Besides ensuring that the games are played correctly, our protocols support financial rewards and penalties enforcement, guaranteeing that winners receive their rewards and that cheaters get financially penalized. In order to do so, we build on blockchain-based techniques that leverage the power of stateful smart contracts to ensure fair protocol execution.

1 Introduction

Cryptographic protocols for securely playing card games among mutually distrustful parties have been investigated since the seminal work of Shamir

B. David and M. Larangeira—This work was supported by the Input Output Cryptocurrency Collaborative Research Chair, which has received funding from Input Output HK.

R. Dowsley—This project has received funding from the European research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme (grant agreement No 669255).

W. Susilo and G. Yang (Eds.): ACISP 2018, LNCS 10946, pp. 45–63, 2018.
https://doi.org/10.1007/978-3-319-93638-3_4

et al. [20] in the late 1970s, which initiated a long line of research [3,4,8,10–12,14,17–19,21–24]. Not surprisingly, all of these previous works have focused on obtaining protocols suitable for implementing a game of Poker, which poses several interesting technical challenges. Intuitively, in order to protect a player's "poker face" and allow him to bluff, all of his cards might need to be kept private throughout (and even after) protocol execution. In previous works, ensuring this level of privacy required several powerful but expensive cryptographic techniques, such as the use of zero-knowledge proofs and threshold cryptography. However, not all popular card games require a secret state (*e.g.* private cards) to be maintained, which is the case of the popular games of Blackjack (or 21) and Baccarat. In this work, we investigate how to exploit this fundamental difference to construct protocols specifically for games without secret state that achieve higher efficiency than those for Poker.

Games Without Secret State: In games such as Baccarat and Blackjack, no card is privately kept by any player at any time. Basically, in such games, cards from a shuffled deck of closed cards (whose values are unknown to all players) are publicly opened, having their value revealed to all players. We say these are *games without secret state*, since no player possesses any secret state (*i.e.* private cards) at any point in the game, as opposed to games such as Poker, where the goal of the game is to leverage private knowledge of one's card's values to choose the best strategy. An immediate consequence of this crucial difference is that the heavy cryptographic machinery used to guarantee the secrecy and integrity of privately held cards can be eliminated, facilitating the construction of highly efficient card game protocols.

Security Definitions: Even though protocol for secure card games (and specially Poker) have been investigated for several decades, formal security definitions have only been introduced very recently in Kaleidoscope [12] (for the case of Poker protocols) and Royale [14] (for the case of protocols for general card games). Concrete security issues and cases of cheating when trusting online casinos for playing card games are also analysed in [12]. The lack of formal security definitions in previous works has not only made their security guarantees unclear but resulted in concrete security issues, such as the ones in [3,8,23,24], as pointed out in [12,19]. Hence, it is important to provide security definitions that capture the class of protocols for card games without secret state. Adapting the approach of Royale [14] for defining security of protocols for general card games with secret state in the Universal Composability framework of [6] is a promising direction to tackle this problem. Besides clearly describing the security guarantees of a given protocol, a security definition following the approach of Royale also ensures that protocols are *composable*, meaning that they can be securely used concurrently with copies of themselves or other protocols.

Enforcing Financial Rewards and Punishment: One of the main issues in previous protocols for card games is ensuring that winners receive their

rewards while preventing cheaters to keep the protocol from reaching an outcome. This problem was recently solved by Andrychowicz *et al.* [1,2] through an approach based on decentralized cryptocurrencies and blockchain protocols. They construct a mechanism that ensures that honest players receive financial rewards and financially punishes cheaters (who abort the protocol or provide invalid messages). The main idea is to have all players provide deposits of betting and collateral funds, forfeiting their collateral funds if they are found to be cheating. A cheater's collateral funds are then used to compensate honest players. Their general approach has been subsequently improved and applied to poker protocols by Kumaresan *et al.* [17] and Bentov *et al.* [4]. However, protocols for Poker (resp., for general card games) using this approach have only been formally analysed in Kaleidoscope [12] (resp., Royale [14]), where fine tuned *checkpoint witnesses* of correct protocol execution are also proposed as means of improving the efficiency of the mechanism for enforcing rewards/penalties. Such an approach can be carried over to the case of games without secret state.

1.1 Our Contributions

We introduce a general model for reasoning about the composable security of protocols for games without secret state and a protocol that realizes our security definitions with support to financial rewards/penalties. We also introduce optimizations of our original protocol that achieve better round and communication complexities at the expense of a cheap preprocessing phase (in either the Check-in or Create Shuffled Deck procedures). Our protocols do not require expensive card shuffling operations that rely on zero-knowledge proofs, achieving much higher concrete efficiency than all previous works that support card games with secret state (*e.g.* Poker). Our contributions are summarized below:

- The first ideal functionality for general card games without secret state: $\mathcal{F}_{\mathsf{CG}}$.
- An analysis showing that that Baccarat and Blackjack can be implemented by our general protocol ,*i.e.* in the $\mathcal{F}_{\mathsf{CG}}$-hybrid model (Sect. 3).
- A highly efficient protocol π_{CG} for card games which realizes $\mathcal{F}_{\mathsf{CG}}$ along with optimized Protocols $\pi_{\mathsf{CG-PRE}}$ and $\pi_{\mathsf{CG-VRF}}$ (Theorems 1, 2 and 3).
- A novel technique for coin tossing "extension" based on verifiable random functions (VRF) that is of independent interest (Sect. 5).

We start by defining $\mathcal{F}_{\mathsf{CG}}$, an ideal functionality that captures only games without secret state, which is adapted from the functionality for general card games with secret state proposed in Royale [14]. In order to show that such a restricted functionality still finds interesting applications, we show that the games of Blackjack and Baccarat can be implemented by $\mathcal{F}_{\mathsf{CG}}$. Leveraging the fact the $\mathcal{F}_{\mathsf{CG}}$ only captures games without secret state, we construct protocols that rely on cheap primitives such as digital signatures and canonical random oracle based commitments, as opposed to the heavy zero knowledge and threshold cryptography machinery employed in previous works. Most notably,

our approach eliminates the need for expensive card shuffling procedure relying on zero-knowledge proofs of shuffle correctness. In fact, no card shuffling procedure is needed in Protocol π_{CG} and Protocol $\pi_{\mathsf{CG-VRF}}$, where card values are selected on the fly during the Open Card procedure. Our basic protocol π_{CG} simply selects the value of each (publicly) opened card from a set of card values using randomness obtained by a simple commit-and-open coin tossing, which requires two rounds. Later we show that we perform the Open Card operation in one sigle round given a cheap preprocessing phase. In order to perform this optimization, we introduce a new technique that allows for a single coin tossing performed during the Check-in procedure to be later "extended" in a single round with the help of a VRF, obtaining fresh randomness for each Open Card operation.

Related Works. Our results are most closely relate to Royale [14], the currently most efficient protocol for general card games with secret state, which employs a mechanism for enforcing financial rewards and penalties following the stateful contract approach of Bentov *et al.* [4]. In our work, we restrict the model of Royale to capture only games without secret state but maintain the same approach for rewards/penalties enforcement based on stateful contracts. As an advantage of restricting our model to this specific class of games, we eliminate the need for expensive card suffling procedures while constructing very cheap Open Card procedures. Moreover, we are able to construct protocols that only require digital signatures and simple random oracle based commitments (as well as VRFs for one of our optimizations), achieving much higher efficiency than Royale, as shown in Sect. 6. Our protocols enjoy much better efficiency for the recovery phase than Royale, since we employ the same compact checkpoint witnesses but achieve much lower communication complexity, meaning that the protocol messages that must be sent to the stateful contract (*i.e.* posted on a blockchain) are much shorter than those of Royale.

2 Preliminaries

We denote the security parameter by κ. For a randomized algorithm F, $y \xleftarrow{\$} F(x;r)$ denotes running F with input x and its random coins r, obtaining an output y. If r is not specified it is assumed to be sampled uniformly at random. We denote sampling an element x uniformly at random from a set $\mathcal{X}$ (resp. a distribution $\mathcal{Y}$) by $x \xleftarrow{\$} \mathcal{X}$ (resp. $y \xleftarrow{\$} \mathcal{Y}$). We denote two *computationally indistinguishable* ensembles of binary random variables X and Y by $X \approx_c Y$.

Security Model: We prove our protocols secure in the Universal Composability (UC) framework introduced by Canetti in [6]. We consider *static malicious* adversaries, who can arbitrarily deviate from the protocol but must corrupt parties before execution starts, having the corrupted (or honest) parties remain so throughout the execution. It is a well-known fact that UC-secure

two-party and multiparty protocols for non trivial functionalities require a setup assumption [7]. We assume that parties have access to a random oracle functionality $\mathcal{F}_{\mathsf{RO}}$, a digital signature functionality $\mathcal{F}_{\mathsf{DSIG}}$, a verifiable random function functionality $\mathcal{F}_{\mathsf{VRF}}$ and a smart contract functionality $\mathcal{F}_{\mathsf{SC}}$. For further details on the UC framework as well as on the ideal functionalities, we refer the reader to [6] and to the full version of this paper [13].

Verifiable Random Functions: Verifiable random functions (VRF) are a key ingredient of one of our optimized protocols. In order to provide a modular construction in the UC framework, we model VRFs as an ideal functionality $\mathcal{F}_{\mathsf{VRF}}$ that captures the main security guarantees for VRFs, which are usually modeled in game based definitions. While a VRF achieving the standard VRF security definition or even the simulatable VRF notion of [9] is not sufficient to realize $\mathcal{F}_{\mathsf{VRF}}$, it has been shown in [15] that this functionality can be realized in the random oracle model under the CDH assumption by a scheme based on the 2-Hash-DH verifiable oblivious pseudorandom function construction of [16]. We refer interested readers to [15] and the full version of this paper [13] for the definition of functionality $\mathcal{F}_{\mathsf{VRF}}$ and further discussion of its implementation.

Stateful Contracts: We employ an ideal functionality $\mathcal{F}_{\mathsf{SC}}$ that models a *stateful contract*, following the approach of Bentov *et al.* [4]. We use the functionality $\mathcal{F}_{\mathsf{SC}}$ defined in [14] and presented in Fig. 1. This functionality is used to ensure correct protocol execution, enforcing rewards distribution for honest parties and penalties for cheaters. Basically, it provides a "Check-in" mechanism for players to deposit betting and collateral funds, a "Check-out" mechanism for ensuring that players receive their rewards according to the game outcome and a Recovery mechanism for identifying (and punishing) cheaters. After check-in, if a player suspects cheating, it can complain to $\mathcal{F}_{\mathsf{SC}}$ by requesting the Recovery phase to be activated, during which $\mathcal{F}_{\mathsf{SC}}$ mediates protocol execution, verifying that each player generates valid protocol messages. If any player is found to be cheating, $\mathcal{F}_{\mathsf{SC}}$ penalizes the cheaters, distributing their collateral funds among the honest players and ending the execution. It is important to emphasize that the $\mathcal{F}_{\mathsf{SC}}$ functionality can be easily implemented via smart contracts over a blockchain, such as Ethereum [5]. Moreover, our construction (Protocol π_{CG}) requires only simple operations, *i.e.* verification of signatures and of random oracle outputs. A regular honest execution of our protocol is performed entirely off-chain, without intervention of the contract.

3 Modeling Card Games Without Secret State

Before presenting our protocols, we must formally define security for card games without secret state. We depart from the framework introduced in Royale [14] for modeling general card games (which can include secret state), restricting the model to the case of card games without secret state. In order to showcase

Functionality $\mathcal{F}_{\mathsf{SC}}$

The functionality is executed with players $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and is parametrized by a timeout limit τ, and the values of the initial stake t, the compensation q and the security deposit $d \geq (n-1)q$. There is an embedded program GR that represents the game's rules and a protocol verification mechanism pv.

Players Check-in: When execution starts, $\mathcal{F}_{\mathsf{SC}}$ waits to receive from each player $\mathcal{P}_i$ the message $(\text{CHECKIN}, sid, \mathcal{P}_i, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i)$ containing the necessary coins and its signature verification key. Record the values and send $(\text{CHECKEDIN}, sid, \mathcal{P}_i, \mathsf{SIG}.vk_i)$ to all players. If some player fails to check-in within the timeout limit τ or if a message $(\text{CHECKIN-FAIL}, sid)$ is received from any player, then send $(\text{COMPENSATION}, \mathsf{coins}(d+t))$ to all players who checked in and halt.

Player Check-out: Upon receiving $(\text{CHECKOUT-INIT}, sid, \mathcal{P}_j)$ from $\mathcal{P}_j$, send $(\text{CHECKOUT-INIT}, sid, \mathcal{P}_j)$ to all players. Upon receiving $(\text{CHECKOUT}, sid, \mathcal{P}_j, \mathsf{payout}, \sigma_1, \ldots, \sigma_n)$ from $\mathcal{P}_j$, verify that $\sigma_1, \ldots, \sigma_n$ are valid signatures by the players $\mathcal{P}_1, \ldots, \mathcal{P}_n$ on $(\mathtt{CHECKOUT}|\mathsf{payout})$ with respect to $\mathcal{F}_{\mathsf{DSIG}}$. If all tests succeed, for $i = 1, \ldots, n$, send $(\text{PAYOUT}, sid, \mathcal{P}_i, \mathsf{coins}(w))$ to $\mathcal{P}_i$, where $w = \mathsf{payout}[i] + d$, and halt.

Recovery: Upon receiving a recovery request $(\text{RECOVERY}, sid)$ from a player $\mathcal{P}_i$, send the message $(\text{REQUEST}, sid)$ to all players. Upon getting a message $(\text{RESPONSE}, sid, \mathcal{P}_j, \mathsf{Checkpoint}_j, \mathsf{proc}_j)$ from some player $\mathcal{P}_j$ with checkpoint witnesses (which are not necessarily relative to the same checkpoint as the ones received from other players) and witnesses for the current procedure; or an acknowledgement of the witnesses previous submitted by another player, forward this message to the other players. Upon receiving replies from all players or reaching the timeout limit τ, fix the current procedure by picking the most recent checkpoint that has valid witnesses (*i.e.* the most recent checkpoint witness signed by all players $\mathcal{P}_i$). Verify the last valid point of the protocol execution using the current procedure's witnesses, the rules of the game GR, and pv. If some player $\mathcal{P}_i$ misbehaved in the current phase (by sending an invalid message), then send $(\text{COMPENSATION}, \mathsf{coins}(d + q + \mathsf{balance}[j] + \mathsf{bets}[j]))$ to each $\mathcal{P}_j \neq \mathcal{P}_i$, send the leftover coins to $\mathcal{P}_i$ and halt. Otherwise, proceed with a mediated execution of the protocol until the next checkpoint using the rules of the game GR and pv to determine the course of the actions and check the validity of the answer. Messages $(\text{NXT-STP}, sid, \mathcal{P}_i, \mathsf{proc}, \mathsf{round})$ are used to request from player $\mathcal{P}_i$ the protocol message for round round of procedure proc according to the game's rules specified in GR, who answer with messages $(\text{NXT-STP-RSP}, sid, \mathcal{P}_i, \mathsf{proc}, \mathsf{round}, \mathsf{msg})$, where msg is the requested protocol message. All messages $(\text{NXT-STP}, sid, \ldots)$ and $(\text{NXT-STP-RSP}, sid, \ldots)$ are delivered to all players. If during this mediated execution a player misbehaves or does not answer within the timeout limit τ, penalize him and compensate the others as above, and halt. Otherwise send $(\text{RECOVERED}, sid, \mathsf{proc}, \mathsf{Checkpoint})$, to the parties once the next checkpoint $\mathsf{Checkpoint}$ is reached, where proc is the procedure for which $\mathsf{Checkpoint}$ was generated.

Fig. 1. The stateful contract functionality used by the secure protocol for card games based on Royale [14].

the applicability of our model to popular games, we further present game rule programs for Blackjack and Baccarat, which paramterize our general card game functionality for realizing these games.

Modeling General Games Without Secret State. We present an ideal functionality $\mathcal{F}_{\mathsf{CG}}$ for card games without secret state in Fig. 2. Our ideal functionality is heavily based on the $\mathcal{F}_{\mathsf{CG}}$ for games with secret state presented in Royale [14]. We define a version of $\mathcal{F}_{\mathsf{CG}}$ that only captures games without secret state, allowing us to realize it with a lightweight protocol. This version has the same structure and procedures as the $\mathcal{F}_{\mathsf{CG}}$ presented in Royale, except for the procedures that require secret state to be maintained. Namely, we model game rules with an embedded program GR that encodes the rules of the game to be implemented. $\mathcal{F}_{\mathsf{CG}}$ offers mechanisms for GR to specify the distribution of rewards and financially punish cheaters. Additionally, it offers a mechanism for GR to communicate with the players in order to request actions (*e.g.* bets) and publicly register their answers to such requests. In contrast to the model of Royale and previous protocols focusing on poker, $\mathcal{F}_{\mathsf{CG}}$ only offers two main card operations: shuffling and *public* opening of cards. Restricting $\mathcal{F}_{\mathsf{CG}}$ to these operations captures the fact that only games without secret state can be instantiated and allows for realizing this functionality with very efficient protocols. Notice that all actions announced by players are publicly broadcast by $\mathcal{F}_{\mathsf{CG}}$ and that players cannot draw closed cards (which might never be revealed in the game, constituting a secret state). As in Royale, $\mathcal{F}_{\mathsf{CG}}$ can be extended with further operations (*e.g.* randomness generation), incorporating ideal functionalities that model these operations. However, differently from Royale, these operations cannot rely on the card game keeping a secret state.

Formalizing and Realizing Blackjack and Baccarat. In order to illustrate the usefulness of our general functionality $\mathcal{F}_{\mathsf{CG}}$ for games without secret state, we show that it can be used to realize the games of Blackjack and Baccarat. In the full version of this work [13], we define game rule programs $\mathsf{GR}_{\mathsf{blackjack}}$ and $\mathsf{GR}_{\mathsf{blackjack}}$ for Blackjack and Baccarat, respectively, which parameterize $\mathcal{F}_{\mathsf{CG}}$ to realize these games. Both these games requires a special player that acts as the "dealer" or "house", providing funds that will be used to reward the other players in case they win bets. We remark that the actions taken by this special player are pre-determined in both $\mathsf{GR}_{\mathsf{blackjack}}$ and $\mathsf{GR}_{\mathsf{blackjack}}$, meaning that the party representing the "dealer" or "house" does not need to provide inputs (*e.g.* bets or actions) to the protocol, except for providing its funds. While $\mathsf{GR}_{\mathsf{blackjack}}$ and $\mathsf{GR}_{\mathsf{blackjack}}$ model the behavior of this special player as an individual party (which would be required to provide the totality of such funds), these programs can be trivially modified to require each player to provide funds that will be pooled to represent the "dealer's" or "house's" funds, since all of their actions are deterministic and already captured by $\mathsf{GR}_{\mathsf{blackjack}}$ and $\mathsf{GR}_{\mathsf{blackjack}}$.

Functionality $\mathcal{F}_{\mathsf{CG}}$

The functionality is executed with players $\mathcal{P}_1, \ldots, \mathcal{P}_n$ and is parameterized by a timeout limit τ, and the values of the initial stake t, the security deposit d and of the compensation q. There is an embedded program GR that represents the rules of the game and is responsible for mediating the execution: it requests actions from the players, processes their answers, and invokes the procedures of $\mathcal{F}_{\mathsf{CG}}$. $\mathcal{F}_{\mathsf{CG}}$ provides a check-in procedure that is run in the beginning of the execution, a check-out procedure that allows a player to leave the game (which is requested by the player via GR) and a compensation procedure that is invoked by GR if some player misbehaves/aborts. It also provides a channel for GR to request public actions from the players and card operations as described below. GR is also responsible for updating the vectors $\mathsf{balance}$ and bets. Whenever a message is sent to $\mathcal{S}$ for confirmation or action selection, $\mathcal{S}$ should answer, but can always answer (ABORT, sid), in which case the compensation procedure is executed; this option will not be explicitly mentioned in the functionality description henceforth.

Check-in: Executed during the initialization, it waits for a check-in message $(\text{CHECKIN}, sid, \mathsf{coins}(d+t))$ from each $\mathcal{P}_i$ and sends $(\text{CHECKEDIN}, sid, \mathcal{P}_i)$ to the remaining players and GR. If some player fails to check-in within the timeout limit τ, then allow the players that checked-in to dropout and reclaim their coins. Initialize vectors $\mathsf{balance} = (t, \ldots, t)$ and $\mathsf{bets} = (0, \ldots, 0)$.

Check-out: Whenever GR requests the players's check-out with payouts specified by vector payout, send $(\text{CHECKOUT}, sid, \mathsf{payout})$ to $\mathcal{S}$. If $\mathcal{S}$ answers $(\text{CHECKOUT}, sid, \mathsf{payout})$, send $(\text{PAYOUT}, sid, \mathcal{P}_i, \mathsf{coins}(d + \mathsf{payout}[i]))$ to each $\mathcal{P}_i$ and halt.

Compensation: This procedure is triggered whenever $\mathcal{S}$ answers a request for confirmation of an action with (ABORT, sid). Send $(\text{COMPENSATION}, sid, \mathsf{coins}(d + q + \mathsf{balance}[i] + \mathsf{bets}[i]))$ to each active honest player $\mathcal{P}_i$. Send the remaining locked coins to $\mathcal{S}$ and stop the execution.

Request Action: Whenever GR requests an action with description $act - desc$ from $\mathcal{P}_i$, send a message $(\text{ACTION}, sid, \mathcal{P}_i, act - desc)$ to the players. Upon receiving $(\text{ACTION-RSP}, sid, \mathcal{P}_i, act - rsp)$ from $\mathcal{P}_i$, forward it to all other players and GR.

Create Shuffled Deck: Whenever GR requests the creation of a shuffled deck of cards containing cards with values $v_1, \ldots, v_m$, choose the next m free identifiers $\mathsf{id}_1, \ldots, \mathsf{id}_m$, representing cards as pairs $(\mathsf{id}_1, v_1), \ldots, (\mathsf{id}_m, v_m)$. Choose a random permutation Π that is applied to the values $(v_1, \ldots, v_m)$ to obtain the updated cards $(\mathsf{id}_1, v'_1), \ldots, (\mathsf{id}_m, v'_m)$ such that $(v'_1, \ldots, v'_m) = \Pi(v_1, \ldots, v_m)$. Send the message $(\text{SHUFFLED}, sid, v_1, \ldots, v_m, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to all players and GR.

Open Card: Whenever GR requests to reveal the card (id, v) in public, read the card (id, v) from the memory and send the message $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{S}$. If $\mathcal{S}$ answers $(\text{CARD}, sid, \mathsf{id}, v)$, forward this message to all players and GR.

Fig. 2. Functionality for card games without secret state $\mathcal{F}_{\mathsf{CG}}$ based on [14].

Protocol π_{CG} (Part 1)

Protocol π_{CG} is parametrized by a security parameter 1^κ, a timeout limit τ, the values of the initial stake t, the compensation q, the security deposit $d \geq (n-1)q$ and an embedded program GR that represents the rules of the game. In all queries (SIGN, sid, m) to $\mathcal{F}_{\mathsf{DSIG}}$, the message m is implicitly concatenated with `NONCE` and `cnt`, where `NONCE` $\xleftarrow{\$} \{0,1\}^\kappa$ is a fresh nonce (sampled individually for each query) and `cnt` is a counter that is increased after each query. Every player $\mathcal{P}_i$ keeps track of used `NONCE` values (rejecting signatures that reuse nonces) and implicitly concatenate the corresponding `NONCE` and `cnt` values with message m in all queries $(\text{VERIFY}, sid, m, \sigma, \mathsf{SIG}.vk')$ to $\mathcal{F}_{\mathsf{DSIG}}$. Protocol π_{CG} is executed by players $\mathcal{P}_1, \ldots, \mathcal{P}_n$ interacting with functionalities $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{F}_{\text{RO}}$ and $\mathcal{F}_{\mathsf{DSIG}}$ as follows:

- **Checkpoint Witnesses:** After the execution of a procedure, the players store a checkpoint witness that consists of the lists $\mathcal{C}_O$ and $\mathcal{C}_C$, the vectors $\mathsf{balance}$ and bets as well as a signature by each of the other players on the concatenation of all these values. Each signature is generated using $\mathcal{F}_{\mathsf{DSIG}}$ and all players check all signatures using the relevant procedure of $\mathcal{F}_{\mathsf{DSIG}}$. Old checkpoint witnesses are deleted. If any check fails for $\mathcal{P}_i$, he proceeds to the recovery procedure.
- **Recovery Triggers:** All signatures and proofs in received messages are verified by default. Players are assumed to have loosely synchronized clocks and, after each round of the protocol starts, players expect to receive all messages sent in that round before a timeout limit τ. If a player $\mathcal{P}_i$ does not receive an expected message from a player $\mathcal{P}_j$ in a given round before the timeout limit τ, $\mathcal{P}_i$ considers that $\mathcal{P}_j$ has aborted. After the check-in procedure, if any player receives an invalid message or considers that another player has aborted, it proceeds to the recovery procedure.
- **Check-in:** Every player $\mathcal{P}_i$ proceeds as follows:
  1. Send (KEYGEN, sid) to $\mathcal{F}_{\mathsf{DSIG}}$, receiving $(\text{VERIFICATION KEY}, sid, \mathsf{SIG}.vk_i)$.
  2. Send $(\text{CHECKIN}, sid, \mathcal{P}_i, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i)$ to $\mathcal{F}_{\mathsf{SC}}$.
  3. Upon receiving $(\text{CHECKEDIN}, sid, \mathcal{P}_j, \mathsf{SIG}.vk_j)$ from $\mathcal{F}_{\mathsf{SC}}$ for all $j \neq i$, $j = 1, \ldots, n$, initialize the internal lists of open cards $\mathcal{C}_O$ and closed cards $\mathcal{C}_C$. We assume parties have a sequence of unused card id values (*e.g.* a counter). Initialize vectors $\mathsf{balance}[j] = t$ and $\mathsf{bets}[j] = 0$ for $j = 1, \ldots, n$. Output $(\text{CHECKEDIN}, sid)$.
  4. If $\mathcal{P}_i$ fails to receive $(\text{CHECKEDIN}, sid, \mathcal{P}_j, \mathsf{SIG}.vk_j)$ from $\mathcal{F}_{\mathsf{SC}}$ for another party $\mathcal{P}_j$ within the timeout limit τ, it requests $\mathcal{F}_{\mathsf{SC}}$ to dropout and receive its coins back.
- **Compensation:** This procedure is activated if the recovery phase of $\mathcal{F}_{\mathsf{SC}}$ detects a cheater, causing honest parties to receive refunds plus compensation and the cheater to receive the remainder of its funds after honest parties are compensated. Upon receiving $(\text{COMPENSATION}, sid, \mathcal{P}_i, \mathsf{coins}(w))$ from $\mathcal{F}_{\mathsf{SC}}$, a player $\mathcal{P}_i$ outputs this message and halts.

Fig. 3. Part 1 of Protocol π_{CG}.

Protocol π_{CG} (Part 2)

- **Check-out:** A player $\mathcal{P}_j$ can initiate the check-out procedure and leave the protocol at any point that GR allows, in which case all players will receive the money that they currently own plus their collateral refund. The players proceed as follows:
  1. $\mathcal{P}_j$ sends $(\text{CHECKOUT-INIT}, sid, \mathcal{P}_j)$ to $\mathcal{F}_{\mathsf{SC}}$.
  2. Upon receiving $(\text{CHECKOUT-INIT}, sid, \mathcal{P}_j)$ from $\mathcal{F}_{\mathsf{SC}}$, each $\mathcal{P}_i$ (for $i = 1, \ldots, n$) sends $(\text{SIGN}, sid, (\texttt{CHECKOUT}|\mathsf{payout}))$ to $\mathcal{F}_{\mathsf{DSIG}}$ (where payout is a vector containing the amount of money that each player will receive according to GR), obtaining $(\text{SIGNATURE}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), \sigma_i)$ as answer. Player $\mathcal{P}_i$ sends σ_i to $\mathcal{P}_j$.
  3. For all $i \neq j$, $\mathcal{P}_j$ sends $(\text{VERIFY}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), \sigma_i, \mathsf{SIG}.vk_i)$ to $\mathcal{F}_{\mathsf{DSIG}}$, where payout is computed locally by $\mathcal{P}_j$. If $\mathcal{F}_{\mathsf{DSIG}}$ answers all queries $(\text{VERIFY}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), \sigma_i, \mathsf{SIG}.vk_i)$ with $(\text{VERIFIED}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), 1)$, $\mathcal{P}_j$ sends $(\text{CHECKOUT}, sid, \mathsf{payout}, \sigma_1, \ldots, \sigma_n)$ to $\mathcal{F}_{\mathsf{SC}}$. Otherwise, it proceeds to the recovery procedure.
  4. Upon receiving $(\text{PAYOUT}, sid, \mathcal{P}_i, \mathsf{coins}(w))$ from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{P}_i$ outputs this message and halts.
- **Executing Actions:** Each $\mathcal{P}_i$ follows GR that represents the rules of the game, performing the necessary card operations in the order specified by GR. If GR request an action with description $act - desc$ from $\mathcal{P}_i$, all the players output $(\text{ACT}, sid, \mathcal{P}_i, act - desc)$ and $\mathcal{P}_i$ executes any necessary operations. $\mathcal{P}_i$ broadcasts $(\text{ACTION-RSP}, sid, \mathcal{P}_i, act - rsp, \sigma_i)$, where $act - rsp$ is his answer and σ_i his signature on $act - rsp$, and outputs $(\text{ACTION-RSP}, sid, \mathcal{P}_i, act - rsp)$. Upon receiving this message, all other players check the signature, and if it is valid output $(\text{ACTION-RSP}, sid, \mathcal{P}_i, act - rsp)$. If a player $\mathcal{P}_j$ believes cheating is happening, he proceeds to the recovery procedure.
- **Tracking Balance and Bets:** Every player $\mathcal{P}_i$ keeps a local copy of the vectors $\mathsf{balance}$ and bets, such that $\mathsf{balance}[j]$ and $\mathsf{bets}[j]$ represent the balance and current bets of each player $\mathcal{P}_j$, respectively. In order to keep $\mathsf{balance}$ and bets up to date, every player proceeds as follows:
  - At each point that GR specifies that a betting action from $\mathcal{P}_i$ takes place, player $\mathcal{P}_i$ broadcasts a message $(\text{BET}, sid, \mathcal{P}_i, bet_i)$, where bet_i is the value of its bet. It updates $\mathsf{balance}[i] = \mathsf{balance}[i] - b_i$ and $\mathsf{bets}[i] = \mathsf{bets}[i] + b_i$.
  - Upon receiving a message $(\text{BET}, sid, \mathcal{P}_j, bet_j)$ from $\mathcal{P}_j$, player $\mathcal{P}_i$ sets $\mathsf{balance}[j] = \mathsf{balance}[j] - b_j$ and $\mathsf{bets}[j] = \mathsf{bets}[j] + b_j$.
  - When GR specifies a game outcome where player $\mathcal{P}_j$ receives an amount pay_j and has its bet amount updated to b'_j, player $\mathcal{P}_i$ sets $\mathsf{balance}[j] = \mathsf{balance}[j] + pay_j$ and $\mathsf{bets}[j] = b'_j$.
- **Create Shuffled Deck:** When requested by GR to create a shuffled deck of cards containing cards with values $v_1, \ldots, v_m$, each player $\mathcal{P}_i$ chooses the next m free identifiers $\mathsf{id}_1, \ldots, \mathsf{id}_m$ and, for $j = 1, \ldots, m$, stores $(\mathsf{id}_j, \perp)$ in $\mathcal{C}_O$ and v_j in $\mathcal{C}_C$. $\mathcal{P}_i$ outputs $(\text{SHUFFLED}, sid, v_1, \ldots, v_m, \mathsf{id}_1, \ldots, \mathsf{id}_m)$.

Fig. 4. Part 2 of Protocol π_{CG}.

Protocol π_{CG} (Part 3)

- **Open Card:** Every player $\mathcal{P}_i$ proceeds as follows to open card with id id:
  1. Organize the card values in $\mathcal{C}_C$ in alphabetic order obtaining an ordered list $\mathcal{C}_C = \{v_1, \ldots, v_m\}$.
  2. Sample a random $r_i \xleftarrow{\$} \{0,1\}^\kappa$ and send (sid, r_i) to $\mathcal{F}_{\mathrm{RO}}$, receiving (sid, h_i) as response. Broadcast (sid, h_i).
  3. After all (sid, h_j) for $j \neq i$ and $j = 1, \ldots, n$ are received, broadcast (sid, r_i).
  4. For $j = 1, \ldots, n$ and $j \neq i$, send (sid, r_j) to $\mathcal{F}_{\mathrm{RO}}$, receiving (sid, h'_j) as response and checking that $h_j = h'_j$. If all checks succeed, compute $k = \sum_i r_i \mod m$, proceeding to the Recovery phase otherwise. Define the opened card value as v_k, remove v_k from $\mathcal{C}_C$ and update $(\mathsf{id}, \perp)$ in $\mathcal{C}_O$ to (id, v_k).

- **Recovery:** Player $\mathcal{P}_i$ proceeds as follows:
  - Starting Recovery: Player $\mathcal{P}_i$ sends $(\textsc{recovery}, sid)$ to $\mathcal{F}_{\mathsf{SC}}$ if it starts the procedure.
  - Upon receiving a message $(\textsc{request}, sid)$ from $\mathcal{F}_{\mathsf{SC}}$, every player $\mathcal{P}_i$ sends $(\textsc{response}, sid, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{proc}_i)$ to $\mathcal{F}_{\mathsf{SC}}$, where $\mathsf{Checkpoint}_i$ is $\mathcal{P}_i$'s latest checkpoint witness and proc_i are $\mathcal{P}_i$'s witnesses for the protocol procedure that started after the latest checkpoint; or acknowledges the witnesses sent by another party if it is the same as the local one.
  - Upon receiving a message $(\textsc{nxt-stp}, sid, \mathcal{P}_i, \mathsf{proc}, \mathsf{round})$ from $\mathcal{F}_{\mathsf{SC}}$, player $\mathcal{P}_i$ sends $(\textsc{nxt-stp-rsp}, sid, \mathcal{P}_i, \mathsf{proc}, \mathsf{round}, \mathsf{msg})$ to $\mathcal{F}_{\mathsf{SC}}$, where msg is the protocol message that should be sent at round round of procedure proc of the protocol according to GR.
  - Upon receiving a message $(\textsc{nxt-stp-rsp}, sid, \mathcal{P}_j, \mathsf{proc}, \mathsf{round}, \mathsf{msg})$ from $\mathcal{F}_{\mathsf{SC}}$, every player $\mathcal{P}_i$ considers msg as the protocol message sent by $\mathcal{P}_j$ in round of procedure proc and take it into consideration for future messages.
  - Upon receiving a message $(\textsc{recovered}, sid, \mathsf{proc}, \mathsf{Checkpoint})$ from $\mathcal{F}_{\mathsf{SC}}$, every player $\mathcal{P}_i$ records $\mathsf{Checkpoint}$ as the latest checkpoint and continues protocol execution according to the game rules GR.

Fig. 5. Part 3 of Protocol π_{CG}.

4 The Framework

Our framework can be used to implement any card game without secret state where cards that were previously randomly shuffled are publicly revealed. Instead of representing cards as ciphertexts as in previous works, we exploit the fact that publicly opening a card from a set of previously randomly shuffled cards is equivalent to randomly sampling card values from an initial set of card values. The main idea is that each opened card has its value randomly picked from a list of "unopened cards" using randomness generated by a coin tossing protocol executed by all parties. This protocol requires no shuffling procedure per se and requires 2 rounds for opening each card (required for executing coin tossing). Later on, we will show that this protocol can be optimized in different ways, but its simple structure aids us in describing our basic approach.

When the game rules GR specify that a card must be created, it is added to a list of cards that have not been opened $\mathcal{C}_C$. When a card is opened, the parties execute a commit-and-open coin tossing protocol to generate randomness that is used to uniformly pick a card from the list of unopened cards $\mathcal{C}_C$, removing the selected card from $\mathcal{C}_C$ and adding it to a list of opened cards $\mathcal{C}_O$. This technique works since every card is publicly opened and no player gets to privately learn the value of a card with the option of not revealing it to the other players, which allows the players to keep the list of unopened cards up-to-date. We implement the necessary commitments with the canonical efficient random oracle based construction, where a commitment is simply an evaluation of the random oracle on the commitment message concatenated with some randomness and the opening consists of the message and randomness themselves. This simple construction achieves very low computational and communication complexities as computing a commitment (and verifying and opening) requires only a single call to the random oracle and the commitment (and opening) can be represented by a string of the size of the security parameter. Besides being compact, these commitments are publicly verifiable, meaning that any third party party can verify the validity of an opening, which comes in handy for verifying that the protocol has been correctly executed.

In order to implement financial rewards/penalties enforcement, our protocol relies on a stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$ that provides a mechanism for the players to deposit betting and collateral funds, enforcing correct distribution of such funds according to the protocol execution. If the protocol is correctly executed, the rewards corresponding to a game outcome are distributed among the players. Otherwise, if a cheater is detected, $\mathcal{F}_{\mathsf{SC}}$ distributes the cheater's collateral funds among honest players, who also receive a refund of their betting and collateral funds. After each game action (*e.g.* betting and card opening), all players cooperate to generate a *checkpoint witness* showing that the protocol has been correctly executed up to that point. This compact checkpoint witness is basically a set of signatures generated under each player's signing key on the opened and unopened cards lists and vectors representing the players' balance and bets. In case a player suspects cheating, it activates the recovery procedure of $\mathcal{F}_{\mathsf{SC}}$ with its latest checkpoint witness, requiring players to provide their most up-to-date checkpoint witnesses to $\mathcal{F}_{\mathsf{SC}}$ (or agree with the one that has been provided). After this point, $\mathcal{F}_{\mathsf{SC}}$ mediates protocol execution, receiving from all players the protocol messages to be sent after the latest checkpoint witness, ensuring their validity and broadcasting them to all players. If the protocol proceeds until next checkpoint witness is generated, the execution is again carried out directly by the players without involving $\mathcal{F}_{\mathsf{SC}}$. Otherwise, if a player is found to be cheating (by failing to provide their messages or providing invalid ones), $\mathcal{F}_{\mathsf{SC}}$ refunds the honest parties and distributes among them the cheater's collateral funds. Protocol π_{CG} is presented in Figs. 3, 4 and 5.

Security Analysis: The security of protocol π_{CG} in the Universal Composability framework is formally stated in Theorem 1. In order to prove this

theorem we construct a simulator such that an ideal execution with this simulator and functionality $\mathcal{F}_{\mathsf{CG}}$ is indistinguishable from a real execution of π_{CG} with any adversary. The main idea behind this simulator is that it learns from $\mathcal{F}_{\mathsf{CG}}$ the value of each opened card, "cheating" in the commit-and-open coin tossing procedure in order to force it to yield the right card value. The simulator can do that since it knows the values that each player has committed to with the random oracle based commitments and it can equivocate the opening of its own commitment, forcing the coin tossing to result in an arbitrary output, yielding an arbitrary card value. The simulation for the mechanisms for requesting players actions and enforcing financial rewards/penalties follows the same approach as in Royale [14]. Namely, the simulator follows the steps of an honest user and makes $\mathcal{F}_{\mathsf{CG}}$ fail if a corrupted party misbehaves, subsequently activating the recovery procedure that results in cheating parties being penalized and honest parties being compensated.

Theorem 1. *For every static active adversary $\mathcal{A}$ who corrupts at most $n-1$ parties, there exists a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the following relation holds:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{CG}},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{F}_{\mathrm{RO}},\mathcal{F}_{\mathsf{DSIG}},\mathcal{F}_{\mathsf{SC}}}_{\pi_{\mathsf{CG}},\mathcal{A},\mathcal{Z}}.$$

The proof is presented in the full version of this work [13].

5 Optimizing Our Protocol

In this section, we construct optimized protocols that improve on the round complexity of the open card operation, which represents the main efficiency bottleneck of our framework. The basic protocol constructed in the previous section requires a whole "commit-then-open" coin tossing to be carried out for each card that is opened. Even though this coin tossing can be implemented efficiently in the random oracle model, its inherent round complexity implies that each card opening requires 2 rounds. We show how the open card operation can be executed with only 1 round while also improving communication complexity but incurring a higher local space complexity (linear in the number of cards) for each player in the Shuffle Card operation. Next, we show how to achieve the same optimal round complexity with a low constant local space complexity.

Lower Round and Communication Complexities: A straightforward way to execute the Open Card operation in one round is to pre-process the necessary commitments during the Shuffle Cards operation. Basically, in order to pre-process the opening of m cards, all players broadcast m commitments to random values in the Shuffle Cards phase. Later on, every time the Open Card operation is executed, each player broadcasts an opening to one of their previously sent commitments. Besides making it possible to open cards in only one round, this simple technique reduces the communication complexity of

Protocol $\pi_{\mathsf{CG-PRE}}$

- **Create Shuffled Deck:** When requested by GR to create a shuffled deck of cards containing cards with values $v_1, \ldots, v_m$, each player $\mathcal{P}_i$ creates $\mathcal{C}_O = \{(\mathsf{id}_1, \perp), \ldots, (\mathsf{id}_m, \perp)\}$ and $\mathcal{C}_C = \{v_1, \ldots, v_m\}$ following the instructions of π_{CG}. Moreover, for $l = 1, \ldots, m$, $\mathcal{P}_i$ samples a random $r_{i,l} \xleftarrow{\$} \{0,1\}^\kappa$ and sends $(sid, r_{i,l})$ to $\mathcal{F}_{\mathrm{RO}}$, receiving (sid, h_i) in response. $\mathcal{P}_i$ broadcasts $(sid, h_{i,1}, \ldots, h_{i,m})$. After all $(sid, h_{j,1}, \ldots, h_{j,m})$ for $j \neq i$ and $j = 1, \ldots, n$ are received, $\mathcal{P}_i$ outputs $(\text{SHUFFLED}, sid, v_1, \ldots, v_m, \mathsf{id}_1, \ldots, \mathsf{id}_m)$.
- **Open Card:** Each player $\mathcal{P}_i$ proceeds as follows to open card with id id:
  1. Organize the card values in $\mathcal{C}_C$ in alphabetic order obtaining an ordered list $\mathcal{C}_C = \{v_1, \ldots, v_m\}$.
  2. Broadcast $(sid, r_{i,l})$, where $h_{i,l}$ is the next available (still closed) commitment generated in the Shuffle Cards operation.
  3. For $j = 1, \ldots, n$ and $j \neq i$, send $(sid, r_{j,l})$ to $\mathcal{F}_{\mathrm{RO}}$, receiving $(sid, h'_{j,l})$ in response and checking that $h_{j,l} = h'_{j,l}$. If all checks succeed, compute $k = \sum_i r_i \mod m$, proceeding to the Recovery phase otherwise. Define the opened card value as v_k, remove v_k from $\mathcal{C}_C$ and update $(\mathsf{id}, \perp)$ in $\mathcal{C}_O$ to (id, v_k).

Fig. 6. Protocol $\pi_{\mathsf{CG-PRE}}$ (only phases that differ from Protocol π_{CG} are described).

the Open Card operation, since each player only broadcasts one opening per card (but no commitment). However, it requires each player to store $(n-1)m$ commitments (received from other players) as all well as m openings (for their own commitments). Protocol $\pi_{\mathsf{CG-PRE}}$ is very similar to Protocol π_{CG}, only differing in the Shuffle Card and Open Card operations, which are presented in Fig. 6. The security of this protocol is formally stated in Theorem 2.

Theorem 2. *For every static active adversary $\mathcal{A}$ who corrupts at most $n-1$ parties, there exists a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the following relation holds:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{CG}}, \mathcal{S}, \mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{F}_{\mathrm{RO}}, \mathcal{F}_{\mathsf{DSIG}}, \mathcal{F}_{\mathsf{SC}}}_{\pi_{\mathsf{CG-PRE}}, \mathcal{A}, \mathcal{Z}}.$$

The proof is very similar to that of Theorem 1, a sketch is presented in the full version of this work [13].

Lower Round and Space Complexities via Coin Tossing Extension: Even though the previous optimization reduces the round complexity of our original protocol, it introduces a high local space complexity overhead, since each party needs to store the preprocessed commitments. In order to achieve low round complexity without a space complexity overhead, we show that a single coin tossing can be "extended" to open an unlimited number of cards. With this technique, we first run a coin tossing in the Check-in phase, later extending it to obtain new randomness used to pick each card that is opened.

Protocol $\pi_{\mathsf{CG-VRF}}$

- **Check-in:** When requested by GR to shuffle cards with identifiers $(\mathsf{id}_1, \ldots, \mathsf{id}_m)$ to be shuffled, each $\mathcal{P}_i$ proceeds as follows:
  1. Execute the steps of the Check-in phase of π_{CG}.
  2. Send (KeyGen, sid) to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\text{Verification Key}, sid, \mathsf{VRF}.vk_i)$ in response. Sample a random $\mathtt{seed}_i \xleftarrow{\$} \{0,1\}^\kappa$ and send $(sid, \mathtt{seed}_i)$ to $\mathcal{F}_{\text{RO}}$, receiving (sid, h_i) in response. Broadcast $(sid, \mathsf{VRF}.vk_i, h_i)$.
  3. After all $(sid, \mathsf{VRF}.vk_j, h_j)$ for $j \neq i$ and $j = 1, \ldots, n$ are received, broadcast $(sid, \mathtt{seed}_i)$.
  4. For $j = 1, \ldots, n$ and $j \neq i$, send $(sid, \mathtt{seed}_j)$ to $\mathcal{F}_{\text{RO}}$, receiving (sid, h'_j) in response and checking that $h_j = h'_j$. If all checks succeed, compute $\mathtt{seed} = \sum_i \mathtt{seed}_i$, proceeding to the Recovery phase otherwise. Set $\mathtt{cnt} = 1$ and broadcast message $(\text{shuffled}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$.
- **Open Card:** Every player $\mathcal{P}_i$ proceeds as follows to open card with id id:
  1. Organize the card values in $\mathcal{C}_C$ in alphabetic order obtaining an ordered list $\mathcal{C}_C = \{v_1, \ldots, v_m\}$.
  2. Send $(\text{EvalProve}, sid, \mathtt{seed}|\mathtt{cnt})$ to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\text{Evaluated}, sid, y_i, \pi_i)$ in response. Broadcast (sid, y_i, π_i).
  3. For $j = 1, \ldots, n$ and $j \neq i$, send $(\text{Verify}, sid, \mathtt{seed}|\mathtt{cnt}, y_j, \pi_j, \mathsf{VRF}.vk_j)$ to $\mathcal{F}_{\mathsf{VRF}}$, checking that $\mathcal{F}_{\mathsf{VRF}}$ answers with $(\text{Verified}, sid, \mathtt{seed}|\mathtt{cnt}, y_j, \pi_j, 1)$. If all checks succeed, compute $k = \sum_i y_i \mod m$, proceeding to the Recovery phase otherwise. Define the opened card value as v_k, remove v_k from $\mathcal{C}_C$, update $(\mathsf{id}, \bot)$ in $\mathcal{C}_O$ to (id, v_k) and increment the counter $\mathtt{cnt}$.

Fig. 7. Protocol $\pi_{\mathsf{CG-VRF}}$ (only phases that differ from Protocol π_{CG} are described).

We develop a new technique for extending coin tossing based on verifiable random functions, which is at the core of our optimized protocol. The main idea is to first have all parties broadcast their VRF public keys and execute a single coin tossing used to generate a seed. Every time a new random value is needed, each party evaluates the VRF under their secret key using the seed concatenated with a counter as input, broadcasting the output and accompanying proof. Upon receiving all the other parties' VRF output and proof, each party verifies the validity of the output and defines the new random value as the sum of all outputs. Protocol $\pi_{\mathsf{CG-VRF}}$ is very similar to Protocol π_{CG}, only differing in the Shuffle Card and Open Card operations, which are presented in Fig. 7. The security of this protocol is formally stated in Theorem 3.

Theorem 3. *For every static active adversary $\mathcal{A}$ who corrupts at most $n - 1$ parties, there exists a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the following relation holds:*

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathsf{CG}}, \mathcal{S}, \mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{F}_{\text{RO}}, \mathcal{F}_{\mathsf{DSIG}}, \mathcal{F}_{\mathsf{VRF}}, \mathcal{F}_{\mathsf{SC}}}_{\pi_{\mathsf{CG-VRF}}, \mathcal{A}, \mathcal{Z}}.$$

The proof is very similar to that of Theorem 1, a sketch is presented in the full version of this work [13].

6 Concrete Complexity Analysis

In this section, we analyse our protocols' computational, communication, round and space complexities, showcasing the different trade-offs obtained by each optimization. We compare our protocols with Royale [14], which is the currently most efficient protocol for general card games (with secret state) that enforces financial rewards and penalties. We focus on the Create Shuffled Deck and Open Card operations, which represent the main bottlenecks in card game protocols. Interestingly, our protocols eliminate the need for expensive zero knowledge proofs of shuffle correctness in the Create Shuffled Card, which are the most expensive components in previous works. Protocol π_{CG} only requires a simple coin tossing to perform the Open Card procedure at the cost of one extra round (in comparison to previous protocols), while our optimized protocols $\pi_{\mathsf{CG-PRE}}$ and $\pi_{\mathsf{CG-VRF}}$ implement this operation with a single round.

Table 1. Complexity comparison of the Shuffle Cards and Open Card operation of Protocols π_{CG}, $\pi_{\mathsf{CG-PRE}}$ and $\pi_{\mathsf{CG-VRF}}$ with n and m cards, excluding checkpoint witness signature generation costs. The cost of calling the random oracle is denoted by H and the cost of a modular exponentiation is denoted by Exp. The size of elements of $\mathbb{G}$ and $\mathbb{Z}$ are denoted by $|\mathbb{G}|$ and $|\mathbb{Z}|$, respectively.

Operation	Protocol	Computational	Communication	Space	Rounds
Open card	π_{CG}	$n\ \mathsf{H}$	$2n\kappa$	0	2
	$\pi_{\mathsf{CG-PRE}}$	$(n-1)\ \mathsf{H}$	$n\kappa$	$nm\kappa$	1
	$\pi_{\mathsf{CG-VRF}}$	$3n\ \mathsf{H}$ $+(4n-1)\ Exp$	$3n\kappa + n\ \lvert\mathbb{Z}\rvert)$	$n\ \lvert\mathbb{G}\rvert + \kappa$	1
	Royale [14]	$n\ \mathsf{H}$ $+4n\ Exp$	$n\ \lvert\mathbb{G}\rvert + 2n\ \lvert\mathbb{Z}\rvert$	$2m\ \lvert\mathbb{G}\rvert$	1
Create shuffled deck	π_{CG}	0	0	0	0
	$\pi_{\mathsf{CG-PRE}}$	$m\ \mathsf{H}$	$nm\kappa$	0	1
	$\pi_{\mathsf{CG-VRF}}$	0	0	0	0
	Royale [14]	$n\ \mathsf{H} +$ $(2\log(\lceil\sqrt{m}\rceil)$ $+4n -$ $2)m\ Exp$	$n(2m + \lceil\sqrt{m}\rceil)\ \mathbb{G}$ $+5n\lceil\sqrt{m}\rceil\ \mathbb{Z}$	0	n

We estimate the computational complexity of the Shuffle Cards and Open Card operations of our protocols in terms of the number of RO calls and modular exponentiations. We present complexity estimates excluding the cost of generating the checkpoint witness signatures, since these costs are the same in both Royale and our protocols (1 signature generation and $n-1$ signature verifications). The communication and space complexities are estimated in terms of the number of strings of size κ, and elements from $\mathbb{G}$ and $\mathbb{Z}$. In order to estimate concrete costs, we assume that $\mathcal{F}_{\mathrm{RO}}$ is implemented by a hash function with κ

bits outputs. Moreover, we assume that $\mathcal{F}_{\mathsf{VRF}}$ is implemented by the 2-Hash-DH verifiable oblivious pseudorandom function construction of [16] as discussed in Sect. 2. This VRF construction requires 1 modular exponentiation to generate a key pair, 3 modular exponentiations and 3 calls to the random oracle to evaluate an input and generate a proof, and 4 modular exponentiations and 3 calls to the random oracle to verify an output given a proof. A verification key is one element of a group $\mathbb{G}$ and the output plus proof consist of 3 random oracle outputs and an element of a ring $\mathbb{Z}$ of same order as $\mathbb{G}$. The estimates for Royale are taken from [14].

Our concrete complexity estimates are presented in Table 1. Notice that our basic protocol π_{CG} and our optimized protocol $\pi_{\mathsf{CG-VRF}}$ do not require a Create Shuffled Deck operation at all, while Protocol $\pi_{\mathsf{CG-PRE}}$ requires a cheap Create Shuffled Cards operation where a batch of commitments to random values are performed. In fact, our protocols eliminate the need for expensive zero knowledge proofs of shuffle correctness, which is the main bottleneck in previous works such as Royale [14], the currently most efficient protocol for card games with secret state. Protocol $\pi_{\mathsf{CG-PRE}}$ improves on the round complexity of the Open Card operation of protocol π_{CG}, requiring only 1 round and the same computational complexity but incurring in a larger space complexity as each player must locally store $nm\kappa$ bits to complete this operation, since they need to store a number of pre-processed commitments that depends on both the number of players and the number of cards in the game. We solve this local storage issue with Protocol $\pi_{\mathsf{CG-VRF}}$, which employs our "coin tossing extension" technique to achieve local space complexity independent of the number of cards, which tends to be much larger than the number of players. We remark that the computational complexity of the Open Card operation of $\pi_{\mathsf{CG-VRF}}$ is equivalent to that of Royale [14], while the communication and space complexities are much lower.

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: 2014 IEEE Symposium on Security and Privacy, pp. 443–458. IEEE Computer Society Press, May 2014
3. Barnett, A., Smart, N.P.: Mental poker revisited. In: Paterson, K.G. (ed.) Cryptography and Coding 2003. LNCS, vol. 2898, pp. 370–383. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40974-8_29
4. Bentov, I., Kumaresan, R., Miller, A.: Instantaneous decentralized poker. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 410–440. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_15
5. Buterin, V.: White paper (2013). https://github.com/ethereum/wiki/wiki/White-Paper. Accessed 12 May 2017

6. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
7. Canetti, R., Fischlin, M.: Universally composable commitments. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_2
8. Castellà-Roca, J., Sebé, F., Domingo-Ferrer, J.: Dropout-tolerant TTP-free mental poker. In: Katsikas, S., López, J., Pernul, G. (eds.) TrustBus 2005. LNCS, vol. 3592, pp. 30–40. Springer, Heidelberg (2005). https://doi.org/10.1007/11537878_4
9. Chase, M., Lysyanskaya, A.: Simulatable VRFs with applications to multi-theorem NIZK. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 303–322. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74143-5_17
10. Crépeau, C.: A secure poker protocol that minimizes the effect of player coalitions. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 73–86. Springer, Heidelberg (1986). https://doi.org/10.1007/3-540-39799-X_8
11. Crépeau, C.: A zero-knowledge poker protocol that achieves confidentiality of the players' strategy *or* how to achieve an electronic poker face. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 239–247. Springer, Heidelberg (1987). https://doi.org/10.1007/3-540-47721-7_18
12. David, B., Dowsley, R., Larangeira, M.: Kaleidoscope: an efficient poker protocol with payment distribution and penalty enforcement. Cryptology ePrint Archive, Report 2017/899 (2017). http://eprint.iacr.org/2017/899
13. David, B., Dowsley, R., Larangeira, M.: 21 - bringing down the complexity: fast composable protocols for card games without secret state. Cryptology ePrint Archive, Report 2018/303 (2018). https://eprint.iacr.org/2018/303
14. David, B., Dowsley, R., Larangeira, M.: ROYALE: a framework for universally composable card games with financial rewards and penalties enforcement. Cryptology ePrint Archive, Report 2018/157 (2018). https://eprint.iacr.org/2018/157
15. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: an adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573 (2017). https://eprint.iacr.org/2017/573. (to appear in Eurocrypt 2018)
16. Jarecki, S., Kiayias, A., Krawczyk, H.: Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014. LNCS, vol. 8874, pp. 233–253. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45608-8_13
17. Kumaresan, R., Moran, T., Bentov, I.: How to use bitcoin to play decentralized poker. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 195–206. ACM Press, New York (2015)
18. Schindelhauer, C.: A toolbox for mental card games. Technical report, University of Lübeck (1998)
19. Sebe, F., Domingo-Ferrer, J., Castella-Roca, J.: On the security of a repaired mental poker protocol. In: Third International Conference on Information Technology: New Generations, pp. 664–668 (2006)
20. Shamir, A., Rivest, R.L., Adleman, L.M.: Mental poker. In: Klarner, D.A. (ed.) The Mathematical Gardner, pp. 37–43. Springer, Boston (1981). https://doi.org/10.1007/978-1-4684-6686-7_5
21. Wei, T.: Secure and practical constant round mental poker. Inf. Sci. **273**, 352–386 (2014)
22. Wei, T., Wang, L.-C.: A fast mental poker protocol. J. Math. Cryptol. **6**(1), 39–68 (2012)

23. Zhao, W., Varadharajan, V.: Efficient TTP-free mental poker protocols. In: International Conference on Information Technology: Coding and Computing (ITCC 2005) - Volume II, vol. 1, pp. 745–750, April 2005
24. Zhao, W., Varadharajan, V., Mu, Y.: A secure mental poker protocol over the internet. In: Proceedings of the Australasian Information Security Workshop Conference on ACSW Frontiers 2003 - Volume 21, ACSW Frontiers 2003, pp. 105–109, Darlinghurst, Australia. Australian Computer Society Inc. (2003)