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

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 Rivest, Shamir

* This work was supported by the Input Output Cryptocurrency Collaborative Research Chair, which has received funding from Input Output HK.

** 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).

and Adleman [23] in the late 1970s, which initiated a long line of research [14, 15, 21, 3, 27, 26, 12, 22, 25, 24, 20, 5, 16, 17]. 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 [16] (for the case of Poker protocols) and Royale [17] (for the case of protocols for general card games). 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 [27, 26, 3, 12], as pointed out in [22, 16]. 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 [17] for defining security of protocols for general card games with secret state in the Universal Composability framework of [8] 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.* [2, 1] 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.* [20] and Bentov *et al.* [5]. However, protocols for Poker (resp., for general card games) using this approach have only been formally analysed in Kaleidoscope [16] (resp., Royale [17]), 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 (Sections 3.2 and 3.3).
- 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 (Section 5.2).

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 [17]. 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 verifiable random function, obtaining fresh randomness for each Open Card operation.

1.2 Related Works

Our results are most closely relate to Royale [17], 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.* [5]. 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 Section 6. Due to significant improvements in communication complexity, our protocols enjoy much better efficiency for the recovery phase than Royale, since we employ the same compact checkpoint witnesses but 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

In this section we introduce the notation and definitions that will be used throughout the paper. We denote the security parameter by κ. For a randomized algorithm F, $y \xleftarrow{\$} F(x)$ denotes running F with input x and its random coins, obtaining an output y. If we need to specify the coins r, we will use the notation $y \leftarrow F(x;r)$. We denote sampling an element x uniformly at random from a set $\mathcal{X}$ by $x \xleftarrow{\$} \mathcal{X}$. For a distribution $\mathcal{Y}$, we denote sampling y according to the distribution $\mathcal{Y}$ by $y \xleftarrow{\$} \mathcal{Y}$. We say that a function f is negligible in n if for every positive polynomial p there exists a constant c such that $f(n) < \frac{1}{p(n)}$ when $n > c$. We denote by $\mathsf{negl}(\kappa)$ the set of negligible functions in κ. Two ensembles $X = \{X_{\kappa,z}\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ and $Y = \{Y_{\kappa,z}\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}$ of binary random variables are said to be *statistically indistinguishable*, denoted by $X \approx_s Y$, if for all z it

holds that $|\Pr[\mathcal{D}(X_{\kappa,z})=1]-\Pr[\mathcal{D}(Y_{\kappa,z})=1]|$ is negligible in κ for every probabilistic distinguisher $\mathcal{D}$. In case this only holds for non-uniform probabilistic polynomial-time (PPT) distinguishers we say that X and Y are *computationally indistinguishable* and denote it by $X \approx_c Y$.

2.1 Universal Composability

We prove our protocols secure in the Universal Composability (UC) framework introduced by Canetti in [8]. In this section, we present a brief description of the UC framework originally given in [11] and refer interested readers to [8] for further details. In this framework, protocol security is analyzed under the real-world/ideal-world paradigm, *i.e.*, by comparing the real world execution of a protocol with an ideal world interaction with the primitive that it implements. The model includes a *composition theorem*, that basically states that UC secure protocols can be arbitrarily composed with each other without any security compromises. This desirable property not only allows UC secure protocols to effectively serve as building blocks for complex applications but also guarantees security in practical environments, where several protocols (or individual instances of protocols) are executed in parallel, such as the Internet.

The UC framework gives that the entities involved in both the real and ideal world executions are modeled as PPT Interactive Turing Machines (ITM) that receive and deliver messages through their input and output tapes, respectively. In the ideal world execution, dummy parties (possibly controlled by an ideal adversary $\mathcal{S}$ referred to as the *simulator*) interact directly with the ideal functionality $\mathcal{F}$, which works as a trusted third party that computes the desired primitive. In the real world execution, several parties (possibly corrupted by a real world adversary $\mathcal{A}$) interact with each other by means of a protocol π that realizes the ideal functionality. The real and ideal executions are controlled by the *environment* $\mathcal{Z}$, an entity that delivers inputs and reads the outputs of the individual parties, the adversary $\mathcal{A}$ and the simulator $\mathcal{S}$. After a real or ideal execution, $\mathcal{Z}$ outputs a bit, which is considered as the output of the execution. The rationale behind this framework lies in showing that the environment $\mathcal{Z}$ (that represents everything that happens outside of the protocol execution) is not able to efficiently distinguish between the real and ideal executions, thus implying that the real world protocol is as secure as the ideal functionality.

We denote by $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa,z,\bar{r})$ the output of the environment $\mathcal{Z}$ in the real-world execution of a protocol π between n parties with an adversary $\mathcal{A}$ under security parameter κ, input z and randomness $\bar{r}=(r_{\mathcal{Z}},r_{\mathcal{A}},r_{P_1},\ldots,r_{P_n})$, where $(z,r_{\mathcal{Z}})$, $r_{\mathcal{A}}$ and r_{P_i} are respectively related to $\mathcal{Z}$, $\mathcal{A}$ and party i. Analogously, we denote by $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa,z,\bar{r})$ the output of the environment in the ideal interaction between the simulator $\mathcal{S}$ and the ideal functionality $\mathcal{F}$ under security parameter κ, input z and randomness $\bar{r}=(r_{\mathcal{Z}},r_{\mathcal{S}},r_{\mathcal{F}})$, where $(z,r_{\mathcal{Z}})$, $r_{\mathcal{S}}$ and $r_{\mathcal{F}}$ are respectively related to $\mathcal{Z}$, $\mathcal{S}$ and $\mathcal{F}$. The real world execution and the ideal executions are respectively represented by the ensembles $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}=\{\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa,z,\bar{r})\}_{\kappa\in\mathbb{N}}$ and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}=\{\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa,z,\bar{r})\}_{\kappa\in\mathbb{N}}$ with $z\in\{0,1\}^*$ and a uniformly chosen $\bar{r}$.

The UC framework also considers the $\mathcal{G}$-hybrid world, where the computation proceeds as in the real-world with the additional assumption that the parties have access to an auxiliary ideal functionality $\mathcal{G}$. In this model, honest parties do not communicate with the ideal functionality directly, instead the adversary delivers all the messages to and from the ideal functionality. We consider the communication channels to be ideally authenticated, so that the adversary may read but not modify these messages. Unlike messages exchanged between parties, which can be read by the adversary, the messages exchanged between parties and the ideal functionality are divided into a *public header* and a *private header*. The public header can be read by the adversary and contains non-sensitive information (such as session identifiers, type of message, sender and receiver). Whereas the private header cannot be read by the adversary and contains information such as the parties' private inputs. We denote the ensemble of environment outputs the execution of a protocol π in a $\mathcal{G}$-hybrid model as $\mathsf{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ (defined analogously to $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$). UC security is then formally defined as:

Definition 1. *An n-party ($n \in \mathbb{N}$) protocol π is said to UC-realize an ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if, for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that, for every environment $\mathcal{Z}$, the following relation holds:* $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$.

Adversarial Model: Our protocols are secure against *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.

Setup Assumptions: It is a well-known fact that UC-secure two-party and multiparty protocols for non trivial functionalities require a setup assumption [10]. The main setup assumption for our protocols is random oracle model [4], which can be modelled in the UC framework by giving parties access to a random oracle functionality $\mathcal{F}_{\mathrm{RO}}$, which is defined in Figure 2. Moreover, in order to obtain a generic and modular construction, we will write our protocols in terms of a digital signature functionality $\mathcal{F}_{\mathsf{DSIG}}$ (defined in Figure 3), a verifiable random function functionality $\mathcal{F}_{\mathsf{VRF}}$ (defined in Figure 1 and discussed below) and a smart contract functionality (defined in Section 2.2). In Figure 3, we present functionality $\mathcal{F}_{\mathsf{DSIG}}$ as defined in [9], where it is shown that any $\mathsf{EUF\text{-}CMA}$ signature scheme realizes $\mathcal{F}_{\mathsf{DSIG}}$. Notice that this fact implies that our protocols can be realized based on practical digital signature schemes such as ECDSA.

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. In Figure 1, we present functionality $\mathcal{F}_{\mathsf{VRF}}$ as defined in [18]. While a VRF achieving the standard VRF security definition or even the

Functionality $\mathcal{F}_{\mathsf{VRF}}$.

$\mathcal{F}_{\mathsf{VRF}}$ interacts with parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$ as follows:

- **Key Generation.** Upon receiving a message $(\textsc{KeyGen}, sid)$ from a party $\mathcal{P}_i$, hand $(\textsc{KeyGen}, sid, \mathcal{P}_i)$ to the adversary. Upon receiving $(\textsc{Verification Key}, sid, \mathcal{P}_i, \mathsf{VRF}.vk)$ from the adversary, if $\mathcal{P}_i$ is honest, verify that $\mathsf{VRF}.vk$ is unique, record the pair $(\mathcal{P}_i, \mathsf{VRF}.vk)$ and return $(\textsc{Verification Key}, sid, \mathsf{VRF}.vk)$ to $\mathcal{P}_i$. Initialize the table $T(\mathsf{VRF}.vk, \cdot)$ to empty.
- **Malicious Key Generation.** Upon receiving a message $(\textsc{KeyGen}, sid, \mathsf{VRF}.vk)$ from $\mathcal{S}$, verify that $\mathsf{VRF}.vk$ has not being recorded before; in this case initialize table $T(\mathsf{VRF}.vk, \cdot)$ to empty and record the pair $(\mathcal{S}, \mathsf{VRF}.vk)$.
- **VRF Evaluation.** Upon receiving a message $(\textsc{Eval}, sid, m)$ from $\mathcal{P}_i$, verify that some pair $(\mathcal{P}_i, \mathsf{VRF}.vk)$ is recorded. If not, then ignore the request. Then, if the value $T(\mathsf{VRF}.vk, m)$ is undefined, pick a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(\mathsf{VRF}.vk, m) = (y, \emptyset)$. Then output $(\textsc{Evaluated}, sid, y)$ to $\mathcal{P}_i$, where y is such that $T(\mathsf{VRF}.vk, m) = (y, S)$ for some S.
- **VRF Evaluation and Proof.** Upon receiving a message $(\textsc{EvalProve}, sid, m)$ from $\mathcal{P}_i$, verify that some pair $(\mathcal{P}_i, \mathsf{VRF}.vk)$ is recorded. If not, then ignore the request. Else, send $(\textsc{EvalProve}, sid, \mathcal{P}_i, m)$ to the adversary. Upon receiving $(\textsc{Eval}, sid, m, \pi)$ from the adversary, if value $T(\mathsf{VRF}.vk, m)$ is undefined, verify that π is unique, pick a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(\mathsf{VRF}.vk, m) = (y, \{\pi\})$. Else, if $T(\mathsf{VRF}.vk, m) = (y, S)$, set $T(\mathsf{VRF}.vk, m) = (y, S \cup \{\pi\})$. In any case, output $(\textsc{Evaluated}, sid, y, \pi)$ to $\mathcal{P}_i$.
- **Malicious VRF Evaluation.** Upon receiving a message $(\textsc{Eval}, sid, \mathsf{VRF}.vk, m)$ from $\mathcal{S}$ for some $\mathsf{VRF}.vk$, do the following. First, if $(\mathcal{S}, \mathsf{VRF}.vk)$ is recorded and $T(\mathsf{VRF}.vk, m)$ is undefined, then choose a random value y from $\{0,1\}^{\ell_{\mathsf{VRF}}}$ and set $T(\mathsf{VRF}.vk, m) = (y, \emptyset)$. Then, if $T(\mathsf{VRF}.vk, m) = (y, S)$ for some $S \neq \emptyset$, output $(\textsc{Evaluated}, sid, y)$ to $\mathcal{S}$, else ignore the request.
- **Verification.** Upon receiving a message $(\textsc{Verify}, sid, m, y, \pi, \mathsf{VRF}.vk')$ from some party P, send $(\textsc{Verify}, sid, m, y, \pi, \mathsf{VRF}.vk')$ to the adversary. Upon receiving $(\textsc{Verified}, sid, m, y, \pi, \mathsf{VRF}.vk')$ from the adversary do:
  1. If $\mathsf{VRF}.vk' = \mathsf{VRF}.vk$ for some $(\mathcal{P}_i, \mathsf{VRF}.vk)$ and the entry $T(\mathsf{VRF}.vk, m)$ equals (y, S) with $\pi \in S$, then set $f = 1$.
  2. Else, if $\mathsf{VRF}.vk' = \mathsf{VRF}.vk$ for some $(\mathcal{P}_i, \mathsf{VRF}.vk)$, but no entry $T(\mathsf{VRF}.vk, m)$ of the form $(y, \{\ldots, \pi, \ldots\})$ is recorded, then set $f = 0$.
  3. Else, initialize the table $T(\mathsf{VRF}.vk', \cdot)$ to empty, and set $f = 0$.

  Output $(\textsc{Verified}, sid, m, y, \pi, f)$ to P.

Fig. 1. Functionality $\mathcal{F}_{\mathsf{VRF}}$.

simulatable VRF notion of [13] is not sufficient to realize $\mathcal{F}_{\mathsf{VRF}}$, it has been shown in [18] 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 [19].

Functionality $\mathcal{F}_{\text{RO}}$

$\mathcal{F}_{\text{RO}}$ is parameterized by a range $\mathcal{D}$. $\mathcal{F}_{\text{RO}}$ keeps a list L of pairs of values, which is initially empty, and proceeds as follows:

- Upon receiving a value (sid, m) from a party P_i or from $\mathcal{S}$, if there is a pair $(m, \hat{h})$ in the list L, set $h = \hat{h}$. Otherwise, choose $h \xleftarrow{\$} \mathcal{D}$ and store the pair (m, h) in L. Reply to the activating machine with (sid, h).

Fig. 2. Functionality $\mathcal{F}_{\text{RO}}$ for the random oracle.

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

Given ideal adversary $\mathcal{S}$, parties $P_1, \ldots, P_n$ and a signer P_s, $\mathcal{F}_{\mathsf{DSIG}}$ performs:

- **Key Generation** Upon receiving a message (KEYGEN, sid) from some party P_s, verify that $sid = (P_s, sid')$ for some sid'. If not, then ignore the request. Else, hand (KEYGEN, sid) to the adversary $\mathcal{S}$. Upon receiving $(\text{VERIFICATION KEY}, sid, \mathsf{SIG}.vk)$ from $\mathcal{S}$, output $(\text{VERIFICATION KEY}, sid, \mathsf{SIG}.vk)$ to P_s, and record the pair $(P_s, \mathsf{SIG}.vk)$.
- **Signature Generation** Upon receiving a message (SIGN, sid, m) from P_s, verify that $sid = (P_s, sid')$ for some sid' . If not, then ignore the request. Else, send (SIGN, sid, m) to $\mathcal{S}$. Upon receiving $(\text{SIGNATURE}, sid, m, \sigma)$ from $\mathcal{S}$, verify that no entry $(m, \sigma, \mathsf{SIG}.vk, 0)$ is recorded. If it is, then output an error message to P_s and halt. Else, output $(\text{SIGNATURE}, sid, m, \sigma)$ to P_s, and record the entry $(m, \sigma, \mathsf{SIG}.vk, 1)$.
- **Signature Verification** Upon receiving a message $(\text{VERIFY}, sid, m, \sigma, \mathsf{SIG}.vk')$ from some party P_i, hand $(\text{VERIFY}, sid, m, \sigma, \mathsf{SIG}.vk')$ to $\mathcal{S}$. Upon receiving $(\text{VERIFIED}, sid, m, \phi)$ from $\mathcal{S}$ do:
  1. If $\mathsf{SIG}.vk' = \mathsf{SIG}.vk$ and the entry $(m, \sigma, \mathsf{SIG}.vk, 1)$ is recorded, then set $f = 1$. (This condition guarantees completeness: If the verification key $\mathsf{SIG}.vk'$ is the registered one and σ is a legitimately generated signature for m, then the verification succeeds.)
  2. Else, if $\mathsf{SIG}.vk' = \mathsf{SIG}.vk$, the signer P_s is not corrupted, and no entry $(m, \sigma', \mathsf{SIG}.vk, 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, \mathsf{SIG}.vk, 0)$. (This condition guarantees unforgeability: If $\mathsf{SIG}.vk'$ is the registered one, the signer is not corrupted, and never signed m, then the verification fails.)
  3. Else, if there is an entry $(m, \sigma, \mathsf{SIG}.vk', f')$ recorded, then let $f = f'$. (This condition guarantees consistency: All verification requests with identical parameters will result in the same answer.)
  4. Else, let $f = \phi$ and record the entry $(m, \sigma, \mathsf{SIG}.vk', \phi)$.

  Output $(\text{VERIFIED}, sid, m, f)$ to P_i.

Fig. 3. Functionality $\mathcal{F}_{\mathsf{DSIG}}$ for digital signature.

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 $(\texttt{CHECKOUT}|\mathsf{payout})$ for the payout vector 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. 4. The stateful contract functionality used by the secure protocol for card games based on Royale [17].

2.2 Stateful Contracts

We employ an ideal functionality $\mathcal{F}_{\mathsf{SC}}$ that models a *stateful contract*, following the approach of Bentov *et al.* [6]. We use the functionality $\mathcal{F}_{\mathsf{SC}}$ defined in [17] and presented in Figure 4. 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 funds for betting and collateral, as well as registering signature verification keys that will be used throughout the protocol for verifying authenticity of players messages and generating checkpoint witnesses. After check-in, if a player suspects cheating, it can complain to $\mathcal{F}_{\mathsf{SC}}$ by requesting the Recovery mechanism to be activated, during which $\mathcal{F}_{\mathsf{SC}}$ mediates protocol execution, verifying that each player generates valid protocol messages. Instead of re-executing the whole protocol, the Recovery phase of $\mathcal{F}_{\mathsf{SC}}$ requires the players to provide checkpoint witnesses proving that the protocol has been correctly executed up to a certain point, only requiring $\mathcal{F}_{\mathsf{SC}}$ to mediate protocol execution from that point on. If any player is found to be cheating, $\mathcal{F}_{\mathsf{SC}}$ penalizes the cheaters, distributing their collateral funds among the honest players and finalizing the protocol. Finally, $\mathcal{F}_{\mathsf{SC}}$ provides a "Check-out" mechanism, which ensures that players receive their rewards according to the game outcome.

Implementation of $\mathcal{F}_{\mathsf{SC}}$. 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 [7]. Moreover, our construction (for protocol π_{CG}) requires only simple operations, *i.e.* verification of signatures and random oracle outputs. The regular operation of our protocol is performed entirely off-chain, without intervention of the contract. However in the event that any problem happen or in the case that any participant in the game claim problems in the execution, any player can publish their agreed status of the game in the chain, via short witnesses (to be detailed in the protocol description).

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 [17] 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 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. Both Blackjack and Baccarat games require 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 $\mathsf{GR}_{\mathsf{blackjack}}$.

3.1 Modeling General Games Without Secret State

We present an ideal functionality $\mathcal{F}_{\mathsf{CG}}$ for card games without secret state in Figure 5. Our ideal functionality is heavily based on the $\mathcal{F}_{\mathsf{CG}}$ for games with secret state presented in Royale [17]. 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.

3.2 Blackjack and its Formalization

Before detailing our proposed formalization, we briefly review the rules of the most played version of the Blackjack game. These rules are captured by the rules $\mathsf{GR}_{\mathsf{blackjack}}$ to be introduced later in this section.

Game Overview: The game has two types of parties: the *dealer* and the *players*. Before the dealer distributes the cards, the players place their respective bets. Next, the dealer starts by handing two face up cards to each player in a pre-established order. The dealer receives only one card, and therefore its hand is not complete yet. On turns, each player places its action, which can be:

- **hit**: The player asks for another card from the deck, and he can ask for more cards until he thinks that it is a good hand;
- **stand**: The player does not do any action;

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. 5. Functionality for card games without secret state $\mathcal{F}_{\mathsf{CG}}$ based on [17].

– **double down:** In case a player's cards combined value is at most 11, the player can double the bet and get a single card. In that case the player cannot add more cards to the set;

– **split:** If there are two card equal cards in the set, the player has the option to separate the sets and play them independently. The player has to bet the same amount for the new hand;
– **insurance bet:** In case the dealer has an Ace as its first card, before the players' actions, the dealer gives them an option of betting at most half of its already bet amount as an *insurance* for the case the second card is one with value 10.

Each card has a value associated: the cards from 2 to 10 have their face value, while the Jacks, Kings and Queens each have value 10. Finally, the Ace card can be considered 1 or 11 at the player discretion. The combined value of a player's cards is referred to as the player's hand. The goal is to get a hand as close as possible to 21, in other words to obtain a *blackjack*: the unbeatable hand. Over this value the player is said to be *bust* and out of the hand, *i.e.* looses its bet. Each player decides how many cards they ask. Note that all the cards are visible to all players.

Cases for pay-out. The hand is over when a player wins, *i.e.,* player's hand is 21, or the dealer looses, *i.e.* the dealer's cards combined value is over 21. In the first case, the amount bet by the winner player is doubled by the dealer, while the dealer collects the bets of all the players whose hand has a value lower than the dealer cards. Otherwise the dealer also pays the double to other players. In the case the dealer hits a *blackjack*, and no other player does it, it collects all the bets. In the case of paying an *insurance bet*, it is treated independently of the main bet. That is, the player looses the latter, but is rewarded with the double of the former.

Game Rules for Blackjack: We describe the rules $\mathsf{GR}_{\mathsf{blackjack}}$ for the Blackjack game in Figures 6 and 7. It captures all the actions and dynamics of the game.

3.3 Baccarat

Before detailing our proposed formalization for the game, we briefly review the most played version of Baccarat. The actions of the players and dynamics of the game are captured by the rules $\mathsf{GR}_{\mathsf{baccarat}}$ to be introduced later in this section.

Game Overview: Similarly to Blackjack, the cards have values associated with each one: the numbered cards have their face value, while Kings, Jacks and Queens have value 0, and finally the Ace card has value of 1. The goal of the game is to get a hand as close as possible to 9, and the hand value is computed modulo 10. The game starts by the players deciding their respective bets. As mentioned earlier the player has the option of betting in the *banker's* or *player's* hands, or for a *tie*. After the bets are placed, the dealer draws, and reveals, two pairs of cards, respectively, the player and the banker hands. Again, in contrast to blackjack, the dealer is actually the one who decides to draw an extra card based on the hands and the players do not decide anything.

Game rules for Baccarat: We describe the game rules $\mathsf{GR}_{\mathsf{baccarat}}$ for Baccarat in Figures 8. Later we show that $\mathsf{GR}_{\mathsf{baccarat}}$ is used in $\mathcal{F}_{\mathsf{CG}}$. Similarly to the case of $\mathsf{GR}_{\mathsf{blackjack}}$, the game actions that must be taken by each participant are requested and announced through the action mechanism of $\mathcal{F}_{\mathsf{CG}}$, while card openings are

Game Rules $\mathsf{GR}_{\mathsf{blackjack}}$

The rules have parameters for the minimal min and maximum max bet amounts, the security deposit d, the initial stake t, the compensation amount q, and an upper limit of sampled cards u. Moreover, the rules keep a counter for drawn cards $\mathsf{sampled}$ initially set for 0. The game is played among n players $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and a virtual dealer $\mathcal{P}_d$ whose actions are performed by $\mathsf{GR}_{\mathsf{blackjack}}$. $\mathsf{GR}_{\mathsf{blackjack}}$ creates a shuffled deck of cards shoe consisting of 6 regular card decks and uses a vector for side bets sides which are particular to Blackjack.

Betting Round: Keep track of the chronological order of actions.

- Execute a round of bets starting with the closest active successor of the dealer, by requesting the action PAY to each p_i, $0 < i \leq n$. Upon receiving the answer b_i from p_i increase $\mathsf{bets}[i]$ by b_i and decrease $\mathsf{balance}[i]$ by the same amount.
- For each player that $\mathcal{P}_i$ and the dealer $\mathcal{P}_n$ open a single card $\mathsf{pc}_{i,1}$ for $0 < i \leq n$ and $i = d$, next, open another round of cards $\mathsf{pc}_{i,2}$, but only for $0 < i \leq n$. For every card sampled, increase $\mathsf{sampled}$ by 1.

Action round: Proceed with the ordered sequence of the players, and each one ends its round of actions by calling the STAND action, unless its cards sum 21 or above (according to the card values described in Section 3.2). In $\mathcal{P}_i$'s turn, $\mathsf{GR}_{\mathsf{blackjack}}$ requests is to indicate its action. The player $\mathcal{P}_i$ is expected to answer with one of the following actions:

- STAND: The player does not receive any new card. The next player in the sequence is called to act.
- HIT: Open a card. After the card is drawn, if the sum of all cards of $\mathcal{P}_i$ is over 21, $\mathsf{GR}_{\mathsf{blackjack}}$ does not open more cards to the player and sends an action request to the next player in the sequence. Otherwise, request an action again to the same player. For every card opened increment by 1 the counter $\mathsf{sampled}$.
- DOUBLE: if $\mathsf{balance}[i] - \mathsf{bets}[i] < 0$, do not accept the action. Otherwise set $\mathsf{bets}[i] = 2{\cdot}\mathsf{bets}[i]$, decrease $\mathsf{balance}[i]$ by $\mathsf{bets}[i]$, and open new card pc, updating $\mathsf{sampled}$. Do not accept any more actions, and proceed to the next player.
- SPLIT: If $\mathsf{pc}_{i,1}$ and $\mathsf{pc}_{i,2}$ are of the same type, $\mathcal{P}_i$ is allowed to request a split. In that case, check the following and proceed to the Dealer Action Phase:
  - if $\mathsf{balance}[i] \geq min$, then set an extra slot on vector bets, say $\mathsf{bets}[i']$, and set $\mathsf{bets}[i'] = \mathsf{bets}[i]$, and decrease balance by another bet amount, that is $\mathsf{balance}[i] = \mathsf{balance}[i] - \mathsf{bets}[i]$. Finally set $\mathsf{pc}_{i',1} = \mathsf{pc}_{i,2}$, open two cards $\mathsf{pc}_{i',2}$ and $\mathsf{pc}_{i,2}$ from shoe updating accordingly the the counter *sampled*. Proceed as a independent hand, $\mathsf{pc}_{i',1}$, $\mathsf{pc}_{i',2}$ and $\mathsf{bets}[i']$ with respect to balance $\mathsf{balance}[i]$ for $\mathcal{P}_i$;
  - if $\mathsf{balance}[i] < min$, $\mathcal{P}_i$ is not allowed to split and $\mathsf{GR}_{\mathsf{blackjack}}$ performs the Compensation procedure.

Fig. 6. Rules for the Blackjack game (Part 1 of 2).

Side bets round: Before the **Action round**, if $\mathsf{pc}_{d,1}$ is the Ace card, all the players in sequence are requested to choose if they want to make a single *insurance bet*. If the player $\mathcal{P}_i$ wants to make the bet it answers with a value $b_i > 0$, otherwise it answers with 0. In the case of bet, check if $b \leq \mathsf{bets}[i]/2$ and $\mathsf{balance}[i] - b > 0$, then update $\mathsf{sides}[i] = \mathsf{sides}[i] + b$, and $\mathsf{balance}[i] = \mathsf{balance}[i] - b$, and accept the bet. Otherwise, perform the Compensation procedure.
Dealer Actions: Open a card $\mathsf{pc}_{d,2}$ from shoe for $\mathcal{P}_d$ and set $\mathsf{sampled} = \mathsf{sampled}+1$. If $\mathsf{pc}_{d,1}$ and $\mathsf{pc}_{d,2}$ sum 17 or above, no extra card is opened. Otherwise open a new card pc from shoe, and set *sum* to the sum of the values associated with $\mathsf{pc}_{d,1}$, $\mathsf{pc}_{d,2}$ and pc.

1. If *sum* is 17 or above, proceed to the Pay-out phase. Otherwise, $\mathcal{P}_d$ has to HIT, then;
2. Sample a new card pc, add the value of it to *sum*, and go to step 1.

Pay-out: Using bets, and the sum of the opened cards for each player, compare the outcomes and computes the amount of money that $\mathcal{P}_i$ receives or looses for the following cases, according to the hand and actions:

- Tie: if the $\mathcal{P}_i$'s hand sums the same value of the dealer's hand, then set $\mathsf{balance}[i] = \mathsf{balance}[i] + \mathsf{bets}[i]$ (i.e., no payment is done);
- Insurance bet: if $\mathsf{pc}_{d,1}$ is an ace, and $\mathsf{pc}_{d,2}$ is a figure card, set $\mathsf{balance}[i] = \mathsf{balance}[1] + 2 \cdot \mathsf{sides}[i]$ and $\mathsf{balance}[d] = \mathsf{balance}[d] - \mathsf{sides}[i]$. Otherwise $\mathsf{balance}[d] = \mathsf{balance}[d] + \mathsf{sides}[i]$.
- Winner hand: if $\mathcal{P}_i$'s initial cards are a Ace (value 11) and a figure (value 10), then set the value $x = \frac{5}{2}$. Otherwise, check if $\mathcal{P}_i$ has acted with (DOUBLE, sid), then set $x = 3$, if not, then set $x = 2$. If $\mathcal{P}_i$'s hand is strictly higher than the dealer's hand, then set $\mathsf{balance}[i] = \mathsf{balance}[i] + x \cdot \mathsf{bets}[i]$ and $\mathsf{balance}[d] = \mathsf{balance}[d] - x \cdot \mathsf{bets}[i]$. If it is strictly lower, then set $\mathsf{balance}[d] = \mathsf{balance}[d] + \mathsf{bets}[i]$.

Restore the usual size of n for the bets and sides vectors (for the case of n players still player) and reset them to its initial state. If $\mathsf{sampled} \geq u$, request a re-shuffle of the cards, and $\mathsf{sampled} = 0$, otherwise does nothing.

Fig. 7. Rules for the Blackjack game (Part 2 of 2).

implemented by calling the card opening operation of $\mathcal{F}_{\mathsf{CG}}$ on the cards specified by the game rules.

Player's Hand Actions. First the dealer acts on behalf of the player's hand. Values between 1 and 5 will make the dealer draw a third card for the hand. In the case of 6 or 7 the player's hand *stands*, that is, no card is drawn. Furthermore, a *natural* is the hand of 8 and 9, the highest hands. Such high hand prevents the banker's hand to draw an extra card and the comparison is done with the cards initially drawn.

Banker's Hand Actions. Once the actions for the player's hand are done, the dealer acts on the banker's hand, again the decision of drawing an extra card or not. As mentioned, if the player's hand has a natural hand, the banker cannot

Game Rules $\mathsf{GR}_{\mathsf{baccarat}}$

The rules have parameters as the minimal min and maximum max bet amounts, and a upper limit of sampled cards u. Moreover, the rules keep a counter for drawn cards $\mathsf{sampled}$ initially set for 0. The game is played among n players and a single dealer, respectively $(\mathcal{P}_1, \ldots, \mathcal{P}_n, \mathcal{P}_d)$. The rule also assumes the existence of the $n+1$ and n size vectors for balances and bets respectively, $\mathsf{balance}$, for each player $\mathcal{P}_i$ and $\mathcal{P}_d$, and bets for each $\mathcal{P}_i$ only. Furthermore, keep a n-size vector hands, which is particular to Baccarat, for $\text{HAND}_i \in \{\text{player, banker, tie}\}$ of each player $\mathcal{P}_i$. $\mathsf{GR}_{\mathsf{baccarat}}$ creates a shuffled deck consisting of 12 regular card decks.

Betting Round: Each player is requested to place their respective bet b_i on $\text{HAND}_i \in \{\text{player, banker, tie}\}$. That is, each player $\mathcal{P}_i$ for $i = 1, \ldots, n$ is requested to place a bet b_i. For each bet b_i, if $b_i \geq min$, $b_i \leq max$, proceed to the Compensation procedure. Otherwise, update $\mathsf{balance}$ and bets vectors accordingly, that is $\mathsf{balance}[i] = \mathsf{balance}[i] - b_i$ and $\mathsf{bets}[i] = b_1$, and $\mathsf{hands}[i] = \text{HAND}_i$. Proceed to **Dealer Actions** phase.

Dealer Actions: Two cards $\mathsf{pc}_{p,1}$ and $\mathsf{pc}_{p,2}$ are opened for $\mathcal{P}_d$ as the Player's hand. Next, two more cards are opened as the Banker's hand, $\mathsf{pc}_{b,1}$ and $\mathsf{pc}_{b,2}$, updating the $\mathsf{sampled}$ counter for each drawn card. Accordingly to the rules and card values described in the early Section 3.3, it may be necessary to draw the card $\mathsf{pc}_{p,3}$, the extra card for the Player's hand, again, updating $\mathsf{sampled}$. Depending on the values and the rule described in Table 1, it may be necessary to draw the Banker's extra card. In that case, request $\mathcal{P}_d$ to draw $\mathsf{pc}_{b,3}$, and update $\mathsf{sampled}$ accordingly. Proceed to **Pay-out** phase.

Pay-out: There are three possible outcomes, $\mathcal{P}_i$ wins whenever it has bet on one of the three possible outcome. Otherwise it looses its bet entirely, namely $\mathsf{balance}[d] = \mathsf{balance}[d] + \mathsf{bets}[i]$ and $\mathsf{bets}[i] = 0$. Consider $\mathcal{P}_i$ is rewarded according to one of the following three cases:

- Tie: Set $\mathsf{balance}[i] = \mathsf{balance}[i] + 8 \cdot \mathsf{bets}[i]$ and $\mathsf{balance}[d] = \mathsf{balance}[d] - 8 \cdot \mathsf{bets}[i]$;
- Banker's hand: Set $\mathsf{balance}[i] = \mathsf{balance}[i] + 1.95 \cdot \mathsf{bets}[i]$ and $\mathsf{balance}[d] = \mathsf{balance}[d] - 1.95 \cdot \mathsf{bets}[i]$;
- Player's hand: Set $\mathsf{balance}[i] = \mathsf{balance}[i] + 2 \cdot \mathsf{bets}[i]$ and $\mathsf{balance}[d] = \mathsf{balance}[d] - 2 \cdot \mathsf{bets}[i]$.

After making the payments, allow the entrance and exit of players. If $\mathsf{sampled} \geq u$, then request a re-shuffle and set the sampled cards counter to its original state $\mathsf{sampled} = 0$. Next, set the vector bets and hands to its original state. Finally, if there is a single player still playing, then proceed to the **Betting round**. Otherwise, end the game.

Fig. 8. Rules for the Baccarat game.

draw an extra card independently of the current hand value, and the comparison of the hands are done with the cards that are on the table.

In general the criteria for drawing are based on initial banker's and player's hands, and the third card drawn for the player. More concretely, in the case that the player's hand is not a natural and the value of the banker's hand is 0, 1 or 2, the dealer is requested to draw an extra card. If the hand is worth 7 or

above, the hand stands. The rule for the values in between, that is 3, 4, 5 and 6, depends on the third card drawn for the player's hand if there is one. For completeness, the rule is described in Table 1.

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

Banker's	**Player's Third Card** ($\mathsf{pc}_{p,3}$)										
Hand	**N**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
9	S	S	S	S	S	S	S	S	S	S	S
8	S	S	S	S	S	S	S	S	S	S	S
7	S	S	S	S	S	S	S	S	S	S	S
6	S	S	S	S	S	S	S	D	D	S	S
5	D	S	S	S	S	D	D	D	D	S	S
4	D	S	S	D	D	D	D	D	D	S	S
3	D	D	D	D	D	D	D	D	D	S	D
2	D	D	D	D	D	D	D	D	D	D	D
1	D	D	D	D	D	D	D	D	D	D	D
0	D	D	D	D	D	D	D	D	D	D	D

Table 1. Rules for drawing a third card for the Banker depending on the values of the Banker's hand and the Player's third card ($\mathsf{pc}_{p,3}$), where "N" denotes that a third card $\mathsf{pc}_{p,3}$ was not drawn for the player. The action of drawing a third card for the banker is denoted by "D" means, while "S" denotes that the Banker's hand stands, *i.e.* no third card is drawn for the Banker.

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 Figures 9, 10 and 11.

4.1 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

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. 9. 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. 10. 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 $\mathcal{F}_{\text{RO}}$, 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 $\mathcal{F}_{\text{RO}}$, 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 $(\text{RECOVERY}, sid)$ to $\mathcal{F}_{\mathsf{SC}}$ if it starts the procedure.
  - Upon receiving a message $(\text{REQUEST}, sid)$ from $\mathcal{F}_{\mathsf{SC}}$, every player $\mathcal{P}_i$ sends $(\text{RESPONSE}, sid, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{proc}_i)$ 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 $(\text{NXT-STP}, sid, \mathcal{P}_i, \mathsf{proc}, \mathsf{round})$ from $\mathcal{F}_{\mathsf{SC}}$, player $\mathcal{P}_i$ sends $(\text{NXT-STP-RSP}, sid, \mathcal{P}_i, \mathsf{proc}, \mathsf{round}, \mathsf{msg})$ 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 $(\text{NXT-STP-RSP}, sid, \mathcal{P}_j, \mathsf{proc}, \mathsf{round}, \mathsf{msg})$ 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 $(\text{RECOVERED}, sid, \mathsf{proc}, \mathsf{Checkpoint})$ 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. 11. Part 3 of Protocol π_{CG}.

simulation for the mechanisms for requesting players actions and enforcing financial rewards/penalties follows the same approach as in Royale [17]. 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:*

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

Proof. In order to prove the security of π_{CG}, we construct a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between interactions with an adversary $\mathcal{A}$ running Protocol π_{CG} in the real world and with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CG}}$ in the ideal world. The basic operations that implement financial rewards/penalties enforcement (*i.e.* Checkpoint Witnesses, Recovery Trigger, Tracking Balance and Bets, Executing Actions, Check-in, Check-out, Recovery and Compensation) are simulated using the same approach as in Royale [17], where the simulator executes the procedures of an honest player in the simulated execution with its internal copy of the adversary, allowing $\mathcal{F}_{\mathsf{CG}}$ to proceed if the adversary acts honestly. For the sake of completeness, we describe the procedures for simulating these operations as presented in Royale [17]. The new techniques introduced in this work are reflected in the simulation of the Shuffle Cards and Open Card operations, which are discussed at length.

We now describe the simulator $\mathcal{S}$, which interacts with an internal copy of the adversary $\mathcal{A}$, the environment $\mathcal{Z}$ and the ideal functionality $\mathcal{F}_{\mathsf{CG}}$. $\mathcal{S}$ writes all the messages received from $\mathcal{Z}$ in $\mathcal{A}$'s input tape, simulating $\mathcal{A}$'s environment. Also, $\mathcal{S}$ writes all messages from $\mathcal{A}$'s output tape to its own output tape, forwarding them to $\mathcal{Z}$. Let $\mathcal{H}$ denote one of honest parties (any arbitrary one), for which $\mathcal{S}$ will execute special procedures during the simulation. As in the protocol, $\mathcal{S}$ is parameterized 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. $\mathcal{S}$ simulates an execution with an internal copy of the adversary $\mathcal{A}$ (that controls the malicious parties), and generates the protocol messages from the honest parties. $\mathcal{S}$ proceeds as follows to simulate each procedure of Protocol π_{CG}:

- **Simulating $\mathcal{F}_{\mathrm{RO}}$:** $\mathcal{S}$ simulates the answers to the random oracle queries from $\mathcal{A}$ exactly as $\mathcal{F}_{\mathrm{RO}}$ would (and stores the lists of queries/answers), except when stated otherwise in $\mathcal{S}$'s description.
- **Simulating $\mathcal{F}_{\mathsf{DSIG}}$:** $\mathcal{S}$ simulates queries from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{DSIG}}$ exactly as $\mathcal{F}_{\mathsf{DSIG}}$ would.
- **Simulating $\mathcal{F}_{\mathsf{SC}}$:** $\mathcal{S}$ simulates queries from $\mathcal{A}$ to $\mathcal{F}_{\mathsf{SC}}$ exactly as $\mathcal{F}_{\mathsf{SC}}$ would.
- **Checkpoint Witnesses, Recovery Trigger, Tracking Balance and Bets, Executing Actions, Check-in, Create Shuffled Deck, Check-out, Compensation:** $\mathcal{S}$ simulates the execution of the respective steps of π_{CG} for the honest parties. If the procedure finishes correctly in this internal simulation, then $\mathcal{S}$ forwards the necessary messages to $\mathcal{F}_{\mathsf{CG}}$ to let it continue.
- **Recovery:** When the recovery phase is activated, $\mathcal{S}$ proceeds by following the steps of an honest party in π_{CG} by sending its most up to date checkpoint and current procedure witnesses to the simulated $\mathcal{F}_{\mathsf{SC}}$ and proceeding by sending the next messages of the honestly simulated execution of π_{CG} to $\mathcal{F}_{\mathsf{SC}}$ instead of sending them directly to $\mathcal{A}$. However, when a card operation (shuffling or opening of cards) is required while execution is mediated by the simulated $\mathcal{F}_{\mathsf{SC}}$, the messages required by such operation are simulated as described below. If the recovery phase succeeds, $\mathcal{S}$ sends $\mathcal{F}_{\mathsf{CG}}$ a message

allowing the execution to proceed normally and, if it fails, $\mathcal{S}$ sends a failure message to $\mathcal{F}_{\mathsf{CG}}$ notifying that the operation has failed as described below in the steps for simulating each operation. In case of a failure, $\mathcal{S}$ proceeds to simulate the compensation phase.

– **Open Card:** Upon receiving $(\text{CARD}, sid, \mathsf{id}, v)$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ simulates honest parties' behavior exactly as in π_{CG} but proceeds as follows for $\mathcal{H}$:
  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 $h_{\mathcal{H}} \xleftarrow{\$} \{0,1\}^{\kappa}$ and, if no pair $(q, h_{\mathcal{H}})$ exists in the simulated $\mathcal{F}_{\text{RO}}$'s internal list of queries/answers, broadcast $(sid, h_{\mathcal{H}})$. Otherwise, output fail and halt.
  3. After all (sid, h_j) for $j \neq \mathcal{H}$ and $j = 1, \ldots, n$ are received, check that there exist pairs (r_j, h_j) in the list of queries and answers of the simulated $\mathcal{F}_{\text{RO}}$. If one such pair does not exist (meaning that $\mathcal{A}$ did not obtain h_j from the simulated $\mathcal{F}_{\text{RO}}$), $\mathcal{S}$ sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to the Recovery procedure after the adversary sends (or fails to send) a (sid, r_j), as this message will be invalid. Otherwise, $\mathcal{S}$ computes $r_{\mathcal{H}}$ such that $r_{\mathcal{H}} + \sum_{j=1}^{n,\, j\neq\mathcal{H}} r_j \mod m = k$ where $v_k = v$ (for v provided by $\mathcal{F}_{\mathsf{CG}}$). If a pair $(r_{\mathcal{H}}, h)$ exists in the simulated $\mathcal{F}_{\text{RO}}$'s internal list of queries/answers, $\mathcal{S}$ outputs fail and halts. Otherwise, upon receiving a query $(sid, r_{\mathcal{H}})$ to the simulated $\mathcal{F}_{\text{RO}}$, $\mathcal{S}$ answers with $(sid, h_{\mathcal{H}})$. $\mathcal{S}$ broadcasts $(sid, r_{\mathcal{H}})$, forcing the open card operation to result in the value v provided by $\mathcal{F}_{\mathsf{CG}}$ for that card identified by id.
  4. If all messages (sid, r'_j) from $\mathcal{A}$ are received before timeout τ, $\mathcal{S}$ checks that there exist pairs (r'_j, h_j) in the list of queries/answers of the simulated $\mathcal{F}_{\text{RO}}$, where (sid, h_j) are the messages received from $\mathcal{A}$ in the previous step. If all checks succeed, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{F}_{\mathsf{CG}}$. Otherwise, it sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to simulate the Recovery phase.

Simulation Indistinguishability: Notice that, unless $\mathcal{S}$ does not output fail, it behaves exactly as an honest player in π_{CG} and proceeds in such a way that, for every card value revealed by $\mathcal{F}_{\mathsf{CG}}$, the same card value is set in the internal execution with the copy of the adversary $\mathcal{A}$. Since $\mathcal{S}$ is able to observe the queries of all the other players in the simulated execution with $\mathcal{A}$, it can compute $r_{\mathcal{H}}$ such that any honest player executing the steps of π_{CG} will obtain the value v associated with id as provided by $\mathcal{F}_{\mathsf{CG}}$. Moreover, $\mathcal{S}$ can program the answer to $r_{\mathcal{H}}$ in such a way that $(r_{\mathcal{H}}, h_{\mathcal{H}})$ is a valid query/answer pair for $\mathcal{F}_{\text{RO}}$, passing the checks executed by other players. Notice that, once $\mathcal{S}$ computes $r_{\mathcal{H}}$, $\mathcal{A}$ cannot provide a r_j such that $r_{\mathcal{H}} + \sum_{j=1}^{n,\, j\neq\mathcal{H}} r_j \mod |\mathcal{C}_C| \neq \mathsf{id}$, since this value would not pass the verification where other players query $\mathcal{F}_{\text{RO}}$ with (sid, r_j) and verify that its answer (sid, h'_j) matches the h_j previously received, causing the protocol to proceed to the Recovery phase (which $\mathcal{S}$ also simulates). Hence, the execution with $\mathcal{S}$ proceeds exactly as in Protocol π_{CG} unless it outputs fail. We will show that this only happens with negligible probability. Notice that $\mathcal{S}$ only outputs

fail if a pair $(q, h_{\mathcal{H}})$ (resp. $(r_{\mathcal{H}}, h)$) exists in the simulated $\mathcal{F}_{\text{RO}}$'s internal list of queries/answers for a value $h_{\mathcal{H}} \xleftarrow{\$} \{0,1\}^{\kappa}$ (resp. $r_{\mathcal{H}} \xleftarrow{\$} \{0,1\}^{\kappa}$) chosen at random in Step 2 (resp. Step 3) of the Open Card operation simulation. Since throughout the simulation $\mathcal{S}$ picks all answers h for the simulated $\mathcal{F}_{\text{RO}}$ uniformly at random, the probability that $h_{\mathcal{H}} \xleftarrow{\$} \{0,1\}^{\kappa}$ has already been picked is negligible in the security parameter κ. An analogous argument shows that the probability that $r_{\mathcal{H}} \xleftarrow{\$} \{0,1\}^{\kappa}$ has already been queried to $\mathcal{F}_{\text{RO}}$ is negligible in κ. Hence, the ideal world execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CG}}$ is indistinguishable from an execution of π_{CG} with $\mathcal{A}$.

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.

5.1 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 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 Figure 12. 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:*

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

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 $(\textsc{shuffled}, sid, v_1, \ldots, v_m, \mathsf{id}_1, \ldots, \mathsf{id}_m)$.
- **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. 12. Protocol $\pi_{\mathsf{CG-PRE}}$ (only phases that differ from Protocol π_{CG} are described).

Proof. (Sketch) In order to prove this theorem we adapt the simulator constructed for π_{CG} (in the proof of Theorem 1) to take into consideration the preprocessing of commitments that takes place in the Shuffle Cards phase. Our adapted simulator $\mathcal{S}$ proceeds exactly as the simulator for π_{CG} except in the following operations:

- **Create Shuffled Deck:** Upon receiving $(\textsc{shuffle}, sid, v_1, \ldots, v_m, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ from $\mathcal{F}_{\mathsf{CG}}$ (meaning that GR requests a shuffling operation), $\mathcal{S}$ proceeds exactly as an honest party in protocol π_{CG} except for a special honest party $\mathcal{H}$. When simulating $\mathcal{H}$, $\mathcal{S}$ 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} but, for $l = 1, \ldots, m$, $\mathcal{S}$ samples a random $h_{\mathcal{H},l} \xleftarrow{\$} \{0,1\}^\kappa$. If no pair $(q, h_{\mathcal{H},l})$ exists in the simulated $\mathcal{F}_{\mathrm{RO}}$'s internal list of queries/answers, $\mathcal{S}$ broadcasts $(sid, h_{\mathcal{H},1}, \ldots, h_{\mathcal{H},l})$. Otherwise, $\mathcal{S}$ outputs fail and halts. If messages $(sid, h_{j,1}, \ldots, h_{j,l})$ are received from all (corrupted) parties before timeout τ, $\mathcal{S}$ sends $(\textsc{shuffle}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{F}_{\mathsf{CG}}$. Otherwise, it sends $(\textsc{abort}, sid)$ to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to simulate the Recovery procedure.
- **Open Card:** Upon receiving $(\textsc{card}, sid, \mathsf{id}, v)$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ simulates honest parties' behavior exactly as in π_{CG} but proceeds as follows for the special honest party $\mathcal{H}$:
  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. Check that there exist pairs $(r_{j,l}, h_{j,l})$ in the list of queries and answers of the simulated $\mathcal{F}_{\text{RO}}$, where $h_{i,l}$ is the next available (still closed) commitment generated in the Shuffle Cards operation. If one such pair does not exist (meaning that $\mathcal{A}$ did not obtain $h_{j,l}$ from the simulated $\mathcal{F}_{\text{RO}}$), $\mathcal{S}$ sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to the Recovery procedure after the adversary sends (or fails to send) a $(sid, r_{j,l})$, as this message will be invalid. Otherwise, $\mathcal{S}$ computes $r_{\mathcal{H},l}$ such that $r_{\mathcal{H},l} + \sum_{j=1}^{n,\, j \neq \mathcal{H}} r_{j,l} \mod m = k$ where $v_k = v$ (for v provided by $\mathcal{F}_{\mathsf{CG}}$). If a pair $(r_{\mathcal{H},l}, h)$ exists in the simulated $\mathcal{F}_{\text{RO}}$'s internal list of queries/answers, $\mathcal{S}$ outputs fail and halts. Otherwise, upon receiving a query $(sid, r_{\mathcal{H},l})$ to the simulated $\mathcal{F}_{\text{RO}}$, $\mathcal{S}$ answers with $(sid, h_{\mathcal{H},l})$. $\mathcal{S}$ broadcasts $(sid, r_{\mathcal{H},l})$, forcing the open card operation to result in the value v provided by $\mathcal{F}_{\mathsf{CG}}$ for that card identified by id.
3. If all messages $(sid, r'_{j,l})$ from $\mathcal{A}$ are received before timeout τ, $\mathcal{S}$ checks that there exist pairs $(r'_{j,l}, h_{j,l})$ in the list of queries/answers of the simulated $\mathcal{F}_{\text{RO}}$, where $(sid, h_{j,l})$ are the messages received from $\mathcal{A}$ in the Shuffle Cards operation. If all checks succeed, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{F}_{\mathsf{CG}}$. Otherwise, it sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to simulate the Recovery phase.

Simulation Indistinguishability: Notice that the simulator $\mathcal{S}$ proceeds as the simulator constructed in proving Theorem 1, with the sole difference that it receives (resp. generates) a batches of commitments $(sid, h_{j,1}, \ldots, h_{j,l})$ (resp. $(sid, h_{\mathcal{H},1}, \ldots, h_{\mathcal{H},l})$) in during the Shuffle Cards operations instead of during the Open Card operation. As before, $\mathcal{S}$ behaves exactly as an honest player and proceeds in a way that the Open Card operation outputs the same card value as in $\pi_{\mathsf{CG-PRE}}$, unless it outputs fail. The same argument of the proof of Theorem 1 can be used to show that $\mathcal{S}$ only outputs fail with negligible probability in the security parameter κ and that the ideal execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CG}}$ is indistinguishable from the execution of π_{CG} with $\mathcal{A}$.

5.2 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.

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,

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 $(\textsc{KeyGen}, sid)$ to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\textsc{Verification Key}, sid, \mathsf{VRF}.vk_i)$ in response. Sample a random $\mathtt{seed}_i \xleftarrow{\$} \{0,1\}^\kappa$ and send $(sid, \mathtt{seed}_i)$ to $\mathcal{F}_{\mathrm{RO}}$, 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 $\mathcal{F}_{\mathrm{RO}}$, 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 $(\textsc{shuffled}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$.
- **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 $(\textsc{EvalProve}, sid, \mathtt{seed}|\mathtt{cnt})$ to $\mathcal{F}_{\mathsf{VRF}}$, receiving $(\textsc{Evaluated}, sid, y_i, \pi_i)$ in response. Broadcast (sid, y_i, π_i).
  3. For $j = 1, \ldots, n$ and $j \neq i$, send $(\textsc{Verify}, sid, \mathtt{seed}|\mathtt{cnt}, y_j, \pi_j, \mathsf{VRF}.vk_j)$ to $\mathcal{F}_{\mathsf{VRF}}$, checking that $\mathcal{F}_{\mathsf{VRF}}$ answers with $(\textsc{Verified}, sid, \mathtt{seed}|\mathtt{cnt}, y_j, \pi_j, 1)$. 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. 13. Protocol $\pi_{\mathsf{CG-VRF}}$ (only phases that differ from Protocol π_{CG} are described).

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 Figure 13. 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:*

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

Proof. (Sketch) In order to prove this theorem we adapt the simulator constructed for π_{CG} (in the proof of Theorem 1) to take into consideration the coin tossing for seed generation and VRF key registration that takes place in the Check-in phase as well as the new procedure for opening cards. Additionally our adapted simulator $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{VRF}}$ by exactly following the steps of that

functionality except when stated otherwise in its description. $\mathcal{S}$ proceeds exactly as the simulator for π_{CG} except in the following operations:

- **Check-in:** When execution starts, $\mathcal{S}$ proceeds exactly as an honest party in protocol $\pi_{\mathsf{CG-VRF}}$ (including when simulating a special honest party $\mathcal{H}$), learning $\mathtt{seed}$ and the verification keys $\mathsf{VRF}.vk_i$ of each party $\mathcal{P}_i$.
- **Open Card:** Upon receiving $(\text{CARD}, sid, \mathsf{id}, v)$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ simulates honest parties' behavior exactly as in π_{CG} but proceeds as follows for $\mathcal{H}$:
  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. Let $T(\mathsf{VRF}.vk, \cdot)$ denote the table of queries/answers for verification key $\mathsf{VRF}.vk$ of the simulated $\mathcal{F}_{\mathsf{VRF}}$, which is initially empty. For $j = 1, \ldots, n$ such that $j \neq \mathcal{H}$, if $T(\mathsf{VRF}.vk_j, (\mathtt{seed}|\mathtt{cnt}))$ is undefined, $\mathcal{S}$ samples a random random $y_j \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{VRF}}}$ and a random unique π_j setting $T(\mathsf{VRF}.vk_j, (\mathtt{seed}|\mathtt{cnt})) = (y_j, \{\pi_j\})$. Next, $\mathcal{S}$ computes a random $y_{\mathcal{H}}$ such that $y_{\mathcal{H}} + \sum_{j=1}^{n,\, j\neq\mathcal{H}} y_j \mod m = k$ where $v_k = v$ (for v provided by $\mathcal{F}_{\mathsf{CG}}$). $\mathcal{S}$ sets $T(\mathsf{VRF}.vk_{\mathcal{H}}, \mathtt{seed}|\mathtt{cnt}) = (y_{\mathcal{H}}, \{\pi_{\mathcal{H}}\})$ and sends $(sid, y_{\mathcal{H}}, \pi_{\mathcal{H}})$ to $\mathcal{A}$.
  3. If all messages (sid, y_j, π_j) from $\mathcal{A}$ are received before timeout τ, $\mathcal{S}$ follows the steps of $\mathcal{F}_{\mathsf{VRF}}$ to verify that y_j is a valid output according to $\mathsf{VRF}.vk_j$, π_j and $T(\mathsf{VRF}.vk_j, \cdot)$. If all checks succeed, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{F}_{\mathsf{CG}}$. Otherwise, it sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and halts.

Simulation Indistinguishability: Notice that $\mathcal{S}$ proceeds exactly as an honest player would in Protocol $\pi_{\mathsf{CG-VRF}}$ with the exception of the way it simulates the answers $(\text{EVALUATED}, sid, y_{\mathcal{H}}, \pi_{\mathcal{H}})$ of $\mathcal{F}_{\mathsf{VRF}}$ to the queries from player $\mathcal{H}$ of the form $(\text{EVALPROVE}, sid, \mathtt{seed}|\mathtt{cnt})$. The difference is that instead of sampling a random $y_{\mathcal{H}}$, $\mathcal{S}$ samples a random $y_{\mathcal{H}} = k - \sum_{j=1}^{n,\, j\neq\mathcal{H}} y_j \mod m$. Notice, however, that $y_{\mathcal{H}}$ is distributed as any randomly sampled y, since $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{VRF}}$ following its exact steps (except for these specific queries from $\mathcal{H}$) and, as per definition of $\mathcal{F}_{\mathsf{VRF}}$, all $y_j \xleftarrow{\$} \{0,1\}^{\ell_{\mathsf{VRF}}}$, for $j = 1, \ldots, n$ such that $j \neq \mathcal{H}$. Hence, the ideal execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{CG}}$ is indistinguishable from the real execution of $\pi_{\mathsf{CG-VRF}}$ with $\mathcal{A}$.

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 [17], which is the currently most efficient protocol for general card games (with secret state) that enforces financial rewards and penalties. We focus on the Shuffle Cards and Open Card operations, which represent the main bottlenecks in card game protocols. Interestingly, our main protocol π_{CG} and optimized protocol $\pi_{\mathsf{CG-VRF}}$ do not require

a Create Shuffled Deck operation and this operation in Protocol $\pi_{\mathsf{CG-PRE}}$ is orders of magnitude more efficient than in previous protocols. 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). Our optimized protocols implement the Open Card operation with a single round and different computational and local complexities. It is interesting to observe the trade-off between the computational and local space complexities of our two optimizations, which offer advantages for different scenarios.

Operation	Protocol	Computational	Communication	Space	Rounds
Open Card	π_{CG}	n H	$2n\kappa$	0	2
	$\pi_{\mathsf{CG-PRE}}$	$(n-1)$ H	$n\kappa$	$nm\kappa$	1
	$\pi_{\mathsf{CG-VRF}}$	$3n$ H $+(4n-1)$ Exp	$3n\kappa + n\ \lvert\mathbb{Z}\rvert)$	$n\ \lvert\mathbb{G}\rvert + \kappa$	1
	Royale [17]	n 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 H	$nm\kappa$	0	1
	$\pi_{\mathsf{CG-VRF}}$	0	0	0	0
	Royale [17]	n 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

Table 2. 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.

We estimate the computational complexity of the Shuffle Cards and Open Card operations of our protocols in terms of the number of random oracle 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 (which also represent random oracle outputs), 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 input and output lengths of κ bits. Moreover, we assume that $\mathcal{F}_{\mathsf{VRF}}$ is implemented by the 2-Hash-DH verifiable oblivious pseudorandom function construction of [19] as discussed in Section 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. In terms of communication/space complexity, 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 [17].

Our concrete complexity estimates are presented in Table 2. First, we remark that none of our protocols requires an expensive Create Shuffled Deck procedure involving zero knowledge proofs of shuffle correctness, which is the main bottleneck in previous works such as Royale [17], the currently most efficient protocol for card games with secret state. 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. 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 [17], while the communication and space complexities are much lower.

7 Conclusion

In this work, we observe that a certain class of card games do not require any secret state to be maintained, initiating an universally composable treatment of such games by introducing an ideal functionality $\mathcal{F}_{\mathsf{CG}}$ that captures this class. In contrast to previous works on general card games (which require expensive zero knowledge proofs), we show that $\mathcal{F}_{\mathsf{CG}}$ can be realized by the very simple and efficient Protocol π_{CG} based on cheap coin tossing in the random oracle model. Moreover, π_{CG} supports financial rewards and penalties with very low on-chain storage requirements. We develop a novel technique for extending coin tossing with VRFs, which is at the core of our optimized protocol $\pi_{\mathsf{CG-VRF}}$ and might be of independent interest. As a direct application of our model and constructions of efficient protocols for card games without secret state, we show that our general protocol can be used to instantiate the popular games of Blackjack and Baccarat, for which we also introduce the first formal definitions in the UC framework (ideal functionalities $\mathcal{F}_{\mathsf{blackjack}}$ and $\mathcal{F}_{\mathsf{baccarat}}$). Finally, we provide a detailed complexity analysis and comparison, showing that our protocol for card games without secret state achieves far better performance than previous protocols for general games.

While we construct our protocols in terms of canonical random oracle based commitments, our approach can be instantiated more generally based on any UC secure commitment scheme (or even coin tossing functionality) that allows for openings (and final outputs) to be publicly verified by a third party who does

not necessarily participate in the protocol but wishes to verify the blockchain. Even though such a general construction might not achieve the same concrete efficiency of our random oracle based protocols, it can be instantiated under other setup assumptions (*e.g.* a common reference string) without necessarily requiring random oracles. Constructing such a generalized version of our protocols is left as a future work.

References

1. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Fair two-party computations via bitcoin deposits. In Rainer Böhme, Michael Brenner, Tyler Moore, and Matthew Smith, editors, *FC 2014 Workshops*, volume 8438 of *LNCS*, pages 105–121. Springer, Heidelberg, March 2014.
2. Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure multiparty computations on bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 443–458. IEEE Computer Society Press, May 2014.
3. Adam Barnett and Nigel P. Smart. Mental poker revisited. In Kenneth G. Paterson, editor, *9th IMA International Conference on Cryptography and Coding*, volume 2898 of *LNCS*, pages 370–383. Springer, Heidelberg, December 2003.
4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.
5. Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. To appear on Asiacrypt 2017. Available at `http://eprint.iacr.org/2017/875`.
6. Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 410–440. Springer, Heidelberg, December 2017.
7. Vitalik Buterin. White paper. 2013. `https://github.com/ethereum/wiki/wiki/White-Paper`, Accessed on 5/12/2017.
8. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.
9. Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004)*, page 219. IEEE Computer Society, 2004.
10. Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.
11. Ignacio Cascudo, Ivan Damgård, Bernardo Machado David, Irene Giacomelli, Jesper Buus Nielsen, and Roberto Trifiletti. Additively homomorphic UC commitments with optimal amortized overhead. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 495–515. Springer, Heidelberg, March / April 2015.
12. Jordi Castellà-Roca, Francesc Sebé, and Josep Domingo-Ferrer. Dropout-tolerant ttp-free mental poker. In Sokratis Katsikas, Javier López, and Günther Pernul, editors, *Trust, Privacy, and Security in Digital Business: Second International Conference, TrustBus 2005, Copenhagen, Denmark, August 22-26, 2005. Proceedings*, pages 30–40, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

13. Melissa Chase and Anna Lysyanskaya. Simulatable VRFs with applications to multi-theorem NIZK. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 303–322. Springer, Heidelberg, August 2007.
14. Claude Crépeau. A secure poker protocol that minimizes the effect of player coalitions. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 73–86. Springer, Heidelberg, August 1986.
15. Claude Crépeau. A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 239–247. Springer, Heidelberg, August 1987.
16. Bernardo David, Rafael Dowsley, and Mario Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. Cryptology ePrint Archive, Report 2017/899, 2017. `http://eprint.iacr.org/2017/899`.
17. Bernardo David, Rafael Dowsley, and Mario Larangeira. 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`.
18. Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol. Cryptology ePrint Archive, Report 2017/573, 2017. (To appear in Eurocrypt 2018) `https://eprint.iacr.org/2017/573`.
19. Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and T-PAKE in the password-only model. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 233–253. Springer, Heidelberg, December 2014.
20. Ranjit Kumaresan, Tal Moran, and Iddo Bentov. How to use bitcoin to play decentralized poker. In Indrajit Ray, Ninghui Li, and Christopher Kruegel:, editors, *ACM CCS 15*, pages 195–206. ACM Press, October 2015.
21. Christian Schindelhauer. A toolbox for mental card games. Technical report, University of Lübeck, 1998.
22. Francesc Sebe, Josep Domingo-Ferrer, and Jordi Castella-Roca. On the security of a repaired mental poker protocol. *Information Technology: New Generations, Third International Conference on*, 00:664–668, 2006.
23. Adi Shamir, Ronald L Rivest, and Leonard M Adleman. Mental poker. In *The mathematical gardner*, pages 37–43. Springer, 1981.
24. Tzer-jen Wei. Secure and practical constant round mental poker. *Information Sciences*, 273:352–386, 2014.
25. Tzer-jen Wei and Lih-Chung Wang. A fast mental poker protocol. *Journal of Mathematical Cryptology*, 6(1):39–68, 2012.
26. Weiliang Zhao and V. Varadharajan. Efficient ttp-free mental poker protocols. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 1, pages 745–750 Vol. 1, April 2005.
27. Weiliang Zhao, Vijay Varadharajan, and Yi Mu. 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 '03, pages 105–109, Darlinghurst, Australia, Australia, 2003. Australian Computer Society, Inc.