ROYALE: A Framework for Universally Composable Card Games with Financial Rewards and Penalties Enforcement

Bernardo David* Rafael Dowsley† Mario Larangeira*

Abstract

While many tailor made card game protocols are known, the vast majority of those suffer from three main issues: lack of mechanisms for distributing financial rewards and punishing cheaters, lack of composability guarantees and little flexibility, focusing on the specific game of poker. Even though folklore holds that poker protocols can be used to play any card game, this conjecture remains unproven and, in fact, does not hold for a number of protocols (including recent results). We both tackle the problem of constructing protocols for general card games and initiate a treatment of such protocols in the Universal Composability (UC) framework, introducing an ideal functionality that captures general card games constructed from a set of core card operations. Based on this formalism, we introduce Royale, the first universally composable protocol for securely playing general card games with financial rewards/penalties enforcement. We remark that Royale also yields the first universally composable poker protocol. Interestingly, such an instantiation of a poker game based on Royale performs better than most previous works without composability guarantees, which we highlight through a detailed concrete complexity analysis and benchmarks obtained from a prototype implementation.

1 Introduction

Online card games have become highly popular with the advent of online casinos, which act as trusted third parties performing the roles of both dealers and cashiers. However, this state of affairs is unsatisfactory, as a malicious casino (possibly compromised by an insider attack) can easily subvert game outcomes. In fact, such vulnerabilities not only constitute a looming threat but have indeed been exploited in the past [31].

The problem of playing card games among distrustful players without relying on a trusted third party, commonly referred to as *mental poker*, has been the subject a long line of research initiated in the early days of modern cryptography [28, 16, 17, 25, 3, 35, 14, 20, 34, 27, 30, 29]. However, the aforementioned mental poker protocols did not provide formal security definitions or proofs. In fact, concrete flaws in the protocols of [35, 34] (resp. [3, 14]) have been identified in [27] (resp. [18]). Moreover, even if some of these protocols can be proven secure, they do not ensure that aborting adversaries cannot prevent the game to reach an outcome or that honest players receive financial rewards according to such an outcome.

*Tokyo Institute of Technology. Emails: {bdavid,mario}@c.titech.ac.jp. This work was supported by the Input Output Cryptocurrency Collaborative Research Chair, which has received funding from Input Output HK.

†Aarhus University and Input Output HK. Email: rafael@cs.au.dk. 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).

Techniques for ensuring that players receive their rewards according to game outcomes have only been developed very recently by Andrychowicz *et al.* [2, 1], building on decentralized cryptocurrencies and blockchain protocols. Their techniques also prevent misbehavior (including aborts) by imposing financial penalties to adversaries who are caught deviating from the protocol. Basically, their techniques ensure that honest players either receive the rewards determined by the game outcome or, in case an outcome is not reached, a share of the financial penalty imposed to the adversary. Similar techniques were independently proposed by Bentov and Kumaresan in [6, 21].

Improvements on the techniques of [2, 1, 6, 21] were subsequently proposed by Kumaresan *et al.* [22], who also applied them to constructing protocols for secure card games with financial rewards/penalties. However, besides lacking formal security guarantees, their card games protocol has significant efficiency issues. Namely, it requires a lot of interaction with the cryptocurrency network and locks high amounts of money in collateral deposits (for a more detailed discussion see [7, Section 7]). The efficiency issue was addressed by Bentov *et al.* [7] by leveraging the power of stateful contracts. The protocol of Bentov *et al.* [7] requires only $O(1)$ rounds of interaction with the cryptocurrency network and an amount of collateral equal to the compensation the players would receive, while the protocol of [22] requires $O(n^2)$ rounds of interaction with the cryptocurrency network and an amount of collateral linear in the number of messages exchanged during the protocol. Nevertheless, neither of these works provided formal security definitions and proofs for their card game protocols.

The first security definitions and provably secure protocol for secure poker with financial rewards/penalties enforcement was recently proposed by David *et al.* [18] building on the approach of [7]. This work improves on the efficiency of the poker protocol proposed in [7] while addressing the lack of formal security guarantees. However, the definitions and protocols presented in [18] still have two main limitations: they lack composability guarantees and they are tailor-made for poker, not being fit for securely playing other games. Composability ensures that a protocol can be arbitrarily executed along with copies of itself and other protocols, serving as a building block of more complex applications. While folklore holds that a secure poker protocol can be used to play any other card game, it is known that re-purposing the sub-components of poker protocols for other games would void their security guarantees, specially due to composability issues. In this work, we aim at closing this gap by proposing a protocol for playing any card games proven secure in the Universal Composability [10] framework, which in turn implies an universally composable protocol for the specific case of poker.

1.1 Our Contributions

We initiate a composable treatment of card game protocols, introducing both the first ideal functionality for general card games and the first universally composable tailor-made protocol for *general* card games. Our functionality and matching protocol support core operations that can be used to construct a large number of different card games, as opposed to previous protocols, which focus specifically on the game of poker. Besides capturing a large number of card games, our protocol enforces financial rewards/penalties while achieving efficiency comparable to previous works without universal composability guarantees. In fact, for practical parameters, a DDH-based instantiation of our protocol is concretely more efficient than most previous works, most of which have no provable security guarantees. Our contributions are summarized as follows:

- The *first* ideal functionality for general[1] card games (Section 3).
- Royale, the *first* provably secure protocol for *general* card games (Section 4.2).
- Royale is proven to UC-realize our functionality (Theorem 4), being the *first universally composable* card game protocol (yielding the first universally composable poker protocol).
- An efficient mechanism for financial rewards/penalties enforcement in Royale, accompanied by a detailed efficiency analysis showing it outperforms most previous works for practical parameters (Section 5) and benchmarks obtained from a prototype implementation (Section 6).

As a first step in providing a composable treatment, we introduce an ideal functionality that captures the two main features of the card game protocols we wish to construct: financial rewards/penalties enforcement and support for different games. We follow the basic approach of [7] for enforcing financial rewards/penalties based on stateful contracts, which are modeled as a separate ideal functionality in the construction of our protocol. This approach requires each player to provide a deposit with an amount of *collateral* that is forfeited (and distributed among the other players) in case a player behaves maliciously during protocol execution. If a player suspects that another player is misbehaving (*e.g.* failing to send a message), a complaint is sent to the stateful contract functionality, which mediates protocol execution until the conflict is resolved or a culprit is found, resulting in the termination of the protocol after collateral deposit distribution. As pointed out in [7], such a stateful contract functionality can be implemented based on smart contracts on a blockchain-based system such as Ethereum [8].

In order to support different card games, our ideal functionality is parameterized by a program describing the flow of the game being modeled, differently from the ideal functionality introduced in [18], which only captures the flow of a poker game. This program determines the order in which the functionality carries out a number of operations that are used throughout the game, as well as the conditions under which a player wins or loses the game. Namely, the game rules can request a number of core card operations: public shuffling of closed cards on the table, private opening of cards (towards only one player, used for drawing cards), public opening of cards and shuffling of cards in a player's private hand (which can be used to securely swap cards among players). Moreover we provide an interface for the game rules to request public actions from the players (allowing players to broadcast their course of action), such as placing a bet or choosing a card from the table.

Finally, we construct Royale, a protocol for general card games that can be proven to UC-realize our functionality with the help of a stateful contract. Royale is constructed in a modular fashion based on generic signature, threshold encryption and non-interactive zero-knowledge proofs (NIZKs) that can be efficiently instantiated under standard computational assumptions (DDH) in the random oracle model. Since the stateful contract is ultimately implemented by a blockchain-based solution, one of the main bottlenecks in such a protocol is the amount of on-chain storage required for executing the stateful contract, which must analyze the protocol execution and determine whether a player has correctly executed the protocol or not when a complaint is issued. We achieve low on-chain storage complexity by providing compact *checkpoint witnesses* that allow the players to prove that the protocol has been correctly executed (or not), differently from [7], which requires large amounts of the protocol transcript to be sent to the contract. Moreover, we achieve good off-chain efficiency by leveraging recent advances in zero-knowledge arguments for correct of shuffles [4].

[1] Any card games that can be expressed in terms of a number of core operations provided by the functionality.

The individual card operations in our protocol are inspired by Kaleidoscope [18], which achieves the desired efficiency guarantees for the specific case of poker games. However, Kaleidoscope is based specifically on the DDH assumption and does not achieve universal composability guarantees, which seem unlikely to be achievable in its original form. Kaleidoscope's security proof involves a simulator that makes heavy use of extraction of witnesses in non-interactive zero knowledge (NIZK) arguments based on the Fiat-Shamir heuristic, which require rewinding a copy of the adversary in the security proof, an operation that is not allowed in proofs in the UC framework. While substituting such Fiat-Shamir NIZKs for an ideal functionality for zero-knowledge proofs would solve this issue, the efficiency of the resulting protocol would be greatly affected, since current constructions of UC-secure zero-knowledge proofs [9] are significantly less efficient than the simple NIZKs used in the original Kaleidoscope. We overcome this obstacle without sacrificing efficiency through subtle modifications to the protocol itself and a novel security proof that only requires the simulator to generate simulated proofs, eliminating the need for extracting witnesses (and consequently the need for rewinding).

1.2 Related Works

Even though there is a large number of previous works on protocols for secure card games, the problem of aborting adversaries and reward distribution for card games has only been (efficiently) addressed in the recent work of Bentov *et al.* [7]. Moreover, as previously discussed, formal security definitions and proofs for secure card game protocols were only recently introduced in Kaleidoscope [18]. Since we aim at addressing both the issues of composability and financial penalties/rewards distribution, we center our discussion on the work of Bentov et al. [7] and on Kaleidoscope [18], which more closely relate to this goal. We refer interested readers to [18] for a comprehensive discussion of efficiency and concrete security issues of previous works.

Enforcing Financial Rewards and Penalties: Most games of poker are played with money at stake, posing two central challenges that were overlooked in the first poker protocols but need to be solved in order to allow for practical deployment: (1) protecting against potentially aborting cheaters and (2) ensuring that winners receive their rewards. The first challenge arises because malicious players might misbehave, sending invalid messages or aborting the protocol execution, which causes the game to freeze and prevents honest players from reaching an outcome. The second challenge is related to the fact that previous poker protocols had no means to ensure winners get rewarded, simply because these protocols did not deal with financial assets, which traditionally could not be manipulated by cryptographic protocols as first observed in [2]. Both problems have only been tackled very recently with the advent of cryptocurrencies and blockchain technologies, which allow cryptographic protocols to manipulate financial assets.

These challenges were addressed by Bentov *et al.* [7] in an efficient way by further optimizing an approach previously developed and pursued in [2, 1, 6, 21, 22]. The central idea in the general purposed secure computation protocol of [7] is to execute an unfair protocol without any interaction with the cryptocurrency network, relying on a single stateful contract that handles funds distribution and financially punishes misbehaving parties. Before the unfair protocol is executed, the stateful contract receives deposits of funds that will be distributed according to the protocol output as well as of collateral funds that will be used to punish misbehaving parties and compensate honest parties. In case a party suspects cheating, it "complains" to the stateful contract, which will mediate protocol execution until a cheater is found or the complaint is solved (so that execution can proceed off-chain). In case a party is found to be cheating, its collateral funds are distributed among the honest parties and protocol execution

ends. If the protocol reaches an output, the stateful contract distributes the funds deposited at the onset of execution according to the output. Bentov *et al.* [7] apply this general approach to a tailor-made poker protocol, aiming at implementing a secure poker protocol with higher efficiency than their general purpose secure computation protocol. However, the tailor-made poker protocol presented in [7] is not formally proven secure and, even if found to be secure, has efficiency issues, as discussed in the remainder of this section.

Formal Security Guarantees: The vast majority of poker protocols [28, 16, 17, 25, 3, 35, 14, 20, 34, 27, 30, 29, 22, 7] claim different levels of security but do not provide formal securities. Besides making it hard to assess the exact security offered by such protocols, the lack of clear security definitions and proofs has led to concrete security flaws in many of these protocols [35, 34, 3, 14], as pointed out in [18]. While Bentov *et al.* [7] argue that their framework can be directly applied to tailor-made poker protocols to provide financial rewards/penalties enforcement with high efficiency, they do not provide a security proof for such a direct application of their framework to tailor-made protocols nor describe the properties the underlying poker protocol should satisfy.Their work specifically mentions the poker protocols of [30, 29] as potential building blocks. However, the protocols of [30, 29] are not formally proven secure and their security models are not formally defined and seem to be rather weak (judging by the informal descriptions presented by the authors). Using such protocols as building blocks in a black-box way without a clear definition of the security requirements they should fulfill and a matching security proof can potentially lead to security vulnerabilities being "inhereted" from the building blocks, and to composition issues. Moreover, even if proven secure, the protocols of [30, 29] face efficiency issues for practical parameters. In the specific case of poker, the lack of formal security definitions and proofs was only recently remedied by Kaleidoscope [18], which introduced both the first poker protocol that provably realizes a detailed simulation-based security definition and provides financial rewards/penalties enforcement.

Efficiency Issues: Since Royale is the first work to consider general card games, we compare the efficiency of each card operation provided by Royale to that of similar operations provided in previous works on poker protocols. The most costly operation in poker protocols is the shuffling of cards. Briefly, we recall that in the protocol of Barnett and Smart[3] (that serves as the basis for many subsequent protocols), shuffling a deck of cards requires each player to individually carry out cut-and-choose based zero-knowledge (ZK) proofs of shuffle correctness towards each of the other players. Such a technique introduces a tremendous communication and computational burden, while increasing the number of protocols rounds. The shuffling protocol by Wei and Wang [30] (cited as a potential building block in [7]) improves on this approach by modifying the cut-and-choose based ZK proofs of shuffle correctness such that each player only needs to execute one such proof in order to convince all other players that it behaved honestly, instead of requiring one individual proof for each of the other players. A subsequent work by Wei [29] (also cited as a potential building block in [7]) further improves on the complexity of the shuffle procedure by eliminating the need for cut-and-choose and instead relying on a technique that allows all players to shuffle only one instance of a card deck representation, while proving correctness of their shuffles. However, even though the techniques of [29] and [30] progressively reduce the communication and computational overheads, they still require a large number of rounds (more than $4n$ rounds, where n is the number of players). A recent work introduced the Kaleidoscope [18] protocol, which put forth a novel shuffling phase based on efficient zero-knowledge proofs of shuffle correctness, achieving better concrete efficiency both in terms of communication and computation than all previous works for practical parameters (considering

at most 18 players and 2000 cards), while only requiring n rounds (where n is the number of players). The shuffling procedure of a DDH-based instantiation of Royale (Section 4.4) inherits the same high efficiency of the Kaleidoscope shuffle while achieving universal composability. The computational, communication and round complexities of opening cards in Royale are very similar to those of previous works, which already achieved high efficiency for these operations. For a more detailed discussion of concrete complexity, we refer the reader to Section 5.

Composability Issues: The need for arbitrary composability naturally arises in poker and general card game protocols with financial rewards/penalties enforcement, since those protocols need to use other cryptographic protocols, *e.g.* secure channels and cryptocurrency protocols. This need is specially pressing in the case of general card game protocols, where card operations are arbitrarily mixed and matched in order to create different games, which can potentially cause serious security issues in protocols without arbitrary composability guarantees. However, none of the previous works on poker or card games protocols have considered this issue, and Kaleidoscope [18], the only poker protocol with provable security guarantees, only achieves sequential composability. The Universal Composability [10] (UC) framework is widely used to reason about arbitrary composability for cryptographic protocols. The main obstacle to providing a proof of security for Kaleidoscope as well as other previous poker protocols lies in their use of non-interactive zero-knowledge (NIZK) proofs obtained from applying the Fiat-Shamir transformation to Sigma protocols. The proof strategy for such previous protocols requires the simulator to extract witnesses from NIZKs, which traditionally requires the simulator to rewind the adversary. Even though techniques for extracting witnesses from Sigma protocol-based zero-knowledge proofs of knowledge exist [9], they require further setup assumptions and incur in high communication, computational and round overheads. We introduce a novel proof strategy for Royale that only requires the simulator to generate simulated NIZKs instead of extracting witness, which allows the proof to proceed without the need for rewinding.

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 Non-Interactive Zero-Knowledge Proofs

We employ a number of non-interactive zero-knowledge proofs (NIZKs) for different relations, which we instantiate in the random oracle model (ROM) for the sake of efficiency. We adopt the

notation and security definitions for NIZKs of [33] (a full version of [32]). Let $\mathcal{R}$ be an efficiently computable binary relation. for pairs $(x, w) \in \mathcal{R}$, we call x the *statement* and w the *witness*, and denote by $\mathcal{L}_{\mathcal{R}} = \{x \mid \exists\, w\, s.t.\, (x, w) \in \mathcal{R}\}$ the language consisting of statements in $\mathcal{R}$. Before we define the security of NIZK Proofs of Membership for $\mathcal{R}$ and of NIZK Proofs of Knowledge for $\mathcal{R}$, we define the syntax common to both of these NIZKs. A NIZK scheme $\mathsf{NIZK}_{\mathcal{R}}$ for relation $\mathcal{R}$ is tuple of algorithms $(\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext})$ such that: Prov is a PPT algorithm that takes as input $(x, w) \in \mathcal{R}$ and outputs a proof π; Verify is a deterministic polynomial time algorithm that takes as input (x, π) and outputs 1 if the proof is valid and 0 otherwise; Sim is a PPT algorithm that takes as input an statement x and outputs a proof π and auxiliary string aux; Ext is a deterministic polynomial time algorithm that takes as input a pair (x, π) and outputs a witness w.

Definition 1 (NIZK Proof of Membership for relation $\mathcal{R}$ in the ROM)**.** A NIZK Proof of Membership scheme $\mathsf{NIZK}^{\mathsf{RO}}_{\mathcal{R}}$ for relation $\mathcal{R}$ is a tuple of algorithms $(\mathsf{Prov}^{\mathsf{RO}}, \mathsf{Verify}^{\mathsf{RO}}, \mathsf{Sim}^{\mathsf{RO}}, \mathsf{Ext}^{\mathsf{RO}})$ with access to a random oracle RO such that the following properties hold:

- *Completeness*: For any $(x, w) \in \mathcal{R}$,

$$\Pr\left[\, \rho \xleftarrow{\$} \{0,1\}^{\kappa}\ ;\ \pi \leftarrow \mathsf{Prov}^{\mathsf{RO}}(x, w; \rho) : \mathsf{Verify}^{\mathsf{RO}}(x, \pi) = 0 \,\right] \leq \mathsf{negl}(\kappa)$$

- *Zero-Knowledge*: For any PPT distinguisher $\mathcal{A}$ we have

$$\left|\Pr\left[\, \mathcal{A}^{\mathsf{RO},\mathcal{O}_1}(1^{\kappa}) = 1 \,\right] - \Pr\left[\, \mathcal{A}^{\mathsf{RO},\mathcal{O}_2}(1^{\kappa}) = 1 \,\right]\right| \leq \mathsf{negl}(\kappa),$$

  where oracle $\mathcal{O}_1$ returns π on queries $(x, w) \in \mathcal{R}$ for $(\pi, \mathsf{aux}) \leftarrow \mathsf{Sim}^{\mathsf{O}}(x)$ and oracle $\mathcal{O}_2$ returns $\pi \leftarrow \mathsf{Prov}^{\mathsf{RO}}(x, w; \rho)$ for $\rho \xleftarrow{\$} \{0,1\}^{\kappa}$.

- *Soundness*: For all adversaries $\mathcal{A}$,

$$\Pr\left[\, (x, \pi) \leftarrow \mathcal{A}^{\mathsf{RO}}(1^{\kappa})\ ;\ x \notin \mathcal{L}_{\mathcal{R}}\ \wedge\ \mathsf{Verify}^{\mathsf{RO}}(x, \pi) = 1 \,\right] \leq \mathsf{negl}(\kappa)$$

Since all of the NIZKs used in our protocols are in the ROM, we omit the random oracle RO in the superscript $\mathsf{NIZK}^{\mathsf{RO}}$. We denote the individual algorithms $(\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim})$ of a NIZK Proof of Membership $\mathsf{NIZK}_{\mathcal{R}}$ for relation $\mathcal{R}$ by $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Prov}$, $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Verify}$, $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim}$.

Using NIZK Proofs of Membership in UC: In our constructions, we will use NIZKs Proofs of Membership in the ROM in the UC framework. In this setting, every query to the random oracle RO done by the NIZK algorithms is sent instead to the random oracle ideal functionality $\mathcal{F}_{\mathrm{RO}}$ (defined in Figure 1). Moreover, when the simulator algorithm $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim}$ is used inside a UC simulator $\mathcal{S}$, all actions taken by $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim}$ in relation to random oracle RO are carried out by $\mathcal{S}$ in its internally simulated $\mathcal{F}_{\mathrm{RO}}$. It is well-known that a UC simulator cannot rewind the adversary, hence we require NIZKs Proofs of Membership used in our protocol to be *straight-line* simulatable in the random oracle model, meaning that the simulator algorithm $\mathsf{NIZK}_{\mathcal{R}}.\mathsf{Sim}$ can program the random oracle answers to queries by the adversary but does not require the simulated adversary to be rewinded.

2.2 Re-Randomizable Threshold Public Key Encryption

A re-randomizable threshold public key encryption (RTE) scheme is a central building block for our protocols. We use the definition of RTEs from [33]. Intuitively, we focus of the (n, n)-Threshold case, where n parties cooperate to generate a public key and a ciphertext generated under that public key can only be decrypted if all n cooperate again. A re-randomizable threshold public key encryption scheme RTE consists of a tuple of PPT algorithms (Setup, KeyGen, $\mathsf{CombinePK}$, Enc, ReRand, $\mathsf{ShareDec}$, $\mathsf{ShareCombine}$, $\mathsf{SimShareDec}$, $\mathsf{CombineSK}$, Dec, Trans) such that:

- $\mathsf{Setup}(1^\kappa)$ is a setup algorithm that takes as input the security parameter κ and outputs public parameters param. All the other algorithms implicitly take param as input. In our protocols we assume that param is known by all parties as a parameter of the protocol.
- $\mathsf{KeyGen}(\mathsf{param})$ is a key generation algorithm that takes as input parameters param and outputs a public key pk_i and a secret key sk_i.
- $\mathsf{CombinePK}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ is a deterministic public key combining algorithm that takes as input a set of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and outputs a combined public key pk.
- $\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$ is a encryption algorithm that takes as input a public key pk and a plaintext message m, and outputs a ciphertext ct.
- $\mathsf{ReRand}(\mathsf{pk}, \mathsf{ct})$ is a re-randomization algorithm that takes as input a public key pk and a ciphertext ct, and outputs a re-randomized ciphertext ct'.
- $\mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct})$ is a deterministic decryption share generation algorithm that takes as input a secret key share sk_i and a ciphertext ct, and outputs a decryption share d_i.
- $\mathsf{ShareCombine}(\mathsf{ct}, \mathsf{d}_1, \ldots, \mathsf{d}_n)$ is a deterministic decryption share combining algorithm that takes as input a ciphertext ct and a set of decryption shares $(\mathsf{d}_1, \ldots, \mathsf{d}_n)$, and outputs a plaintext message m.
- $\mathsf{SimshareDec}\,(\mathsf{sk}_j, \mathsf{ct}, \mathsf{m}, \mathsf{m}')$ is a deterministic simulator algorithm that takes as input a secret key share sk_j (for $j \in \{1, \ldots, n\}$), a ciphertext $\mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$, a plaintext message m and an alternative plaintext message m', outputting a decryption share d_j such that $\mathsf{m}' \leftarrow \mathsf{ShareCombine}(\mathsf{ct}, \mathsf{d}_1, \ldots, \mathsf{d}_n)$, where $\mathsf{d}_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct})$ for $i \in \{1, \ldots, n\} \setminus j$. Intuitively, given a ciphertext encrypting a known plaintext message and an alternative plaintext message, this algorithm outputs a decryption share that results in the ciphertext being decrypted to the alternative message.
- $\mathsf{CombineSK}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ is a deterministic secret key share combination algorithm that takes as input a set of secret key shares $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ and outputs a secret key sk.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$ is a deterministic decryption algorithm that takes as input a ciphertext ct and a secret key sk, and outputs a message m.
- $\mathsf{Trans}(\mathsf{ct}, \{\mathsf{sk}_i\}_{i \in \{1,\ldots,n\} \setminus j})$ is a deterministic algorithm that takes as input a ciphertext ct and a set of secret keys $\{\mathsf{sk}_i\}_{i \in \{1,\ldots,n\} \setminus j}$, and outputs a ciphertext ct'.

We define the security of such schemes as in [33]. Notice that we do not use algorithms $\mathsf{CombineSK}$, Dec and Trans in our protocol or proofs but they are necessary for defining the scheme's security.

Definition 2. A tuple of algorithms $\mathsf{RTE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{CombinePK}, \mathsf{Enc}, \mathsf{ReRand}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{SimShareDec}, \mathsf{CombineSK}, \mathsf{Dec}, \mathsf{Trans})$ is a secure re-randomizable threshold public key encryption if the following properties hold:

Key Combination Correctness: For every valid set of pairs of public and secret keys $\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{1,\ldots,n\}}$, and for $\mathsf{pk} \leftarrow \mathsf{CombinePK}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and $\mathsf{sk} \leftarrow \mathsf{CombineSK}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, the public and secret key pair $(\mathsf{pk}, \mathsf{sk})$ is a valid key pair. Moreover, for every plaintext message m in the plaintext message space of RTE and ciphertext $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$, it holds that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}) = \mathsf{ShareCombine}(\mathsf{ct}, \mathsf{ShareDec}(\mathsf{sk}_1, \mathsf{ct}), \ldots, \mathsf{ShareDec}(\mathsf{sk}_n, \mathsf{ct})).$$

IND-CPA Security: The public key cryptosystem $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is IND-CPA secure.

Ciphertext transformative indistinguishability: For every valid set of public and secret key pairs $\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{1,\ldots,n\}}$, and for $\mathsf{pk} \leftarrow \mathsf{CombinePK}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and $\mathsf{sk} \leftarrow \mathsf{CombineSK}(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, every plaintext message m in the plaintext message space of RTE, ciphertext $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}_j, \mathsf{m})$ and every $j \in \{1, \ldots, n\}$ it holds that

$$(\mathsf{param}, \mathsf{Trans}(\mathsf{ct}, \{\mathsf{sk}_i\}_{i\in\{1,\ldots,n\}\setminus j}) \approx_c (\mathsf{param}, \mathsf{Enc}(\mathsf{pk}, \mathsf{m}))$$

Unlinkability: First we define the following unlinkability experiment $\mathsf{Unlink}_{\mathcal{A}}(1^\kappa)$ executed with an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$:

1. Generate $\mathsf{param} \xleftarrow{\$} \mathsf{Setup}(1^\kappa)$
2. $(\mathcal{I}, \mathsf{st}_1) \xleftarrow{\$} \mathcal{A}_1(\mathsf{param})$ outputs a set $\mathcal{I} \subset \{1, \ldots, n\}$ of up to $n-1$ corrupted indices and state st_1;
3. For $i = 1, \ldots, n$, $(\mathsf{pk}_i, \mathsf{sk}_i) \xleftarrow{\$} \mathsf{KeyGen}(\mathsf{param})$;
4. $(\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{st}_2) \xleftarrow{\$} \mathcal{A}_2(\{\mathsf{pk}_i\}_{i\in\{1,\ldots,n\}}, \{\mathsf{sk}_j\}_{j\in\mathcal{I}}, \mathsf{st}_1)$;
5. Choose $b \xleftarrow{\$} \{0, 1\}$ and re-randomize ct_b to obtain a ciphertext $\mathsf{ct}' \xleftarrow{\$} \mathsf{ReRand}(\mathsf{pk}, \mathsf{ct}_b)$, where $\mathsf{pk} \leftarrow \mathsf{CombinePK}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$;
6. $b' \xleftarrow{\$} \mathcal{A}_3(\mathsf{ct}', \mathsf{st}_2)$;
7. Return 1 if $b' = b$, else return 0.

The advantage of $\mathcal{A}$ in this experiment is defined as

$$\mathsf{AdvUnlink}_{\mathcal{A}}(1^\kappa) = \left| \Pr\left[\, \mathsf{Unlink}_{\mathcal{A}}(1^\kappa) = 1 \,\right] - \frac{1}{2} \right|$$

For any PPT adversary $\mathcal{A}$, it should hold that $\mathsf{AdvUnlink}_{\mathcal{A}}(1^\kappa) \in \mathsf{negl}(1^\kappa)$.

Share-simulation indistinguishability: For every valid set of public and secret key pairs $\{\mathsf{pk}_i, \mathsf{sk}_i\}_{i\in\{1,\ldots,n\}}$, and $\mathsf{pk} \leftarrow \mathsf{CombinePK}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, for every plaintext messages m, m' in the message space of RTE, ciphertext $\mathsf{ct} \xleftarrow{\$} \mathsf{Enc}(\mathsf{pk}, \mathsf{m})$, index $j \in \{1, \ldots, n\}$ and every decryption share $\mathsf{d}_j \leftarrow \mathsf{ShareDec}(\mathsf{sk}_j, \mathsf{ct})$, the following holds

$$(\mathsf{param}, \mathsf{ct}, \mathsf{SimshareDec}(\mathsf{sk}_j, \mathsf{ct}, \mathsf{m}, \mathsf{m}')) \approx_c (\mathsf{param}, \mathsf{ct}, \mathsf{d}_j)$$

2.3 NIZKs for Relations over Re-Randomizable Threshold Public Key Encryption

In our protocols we will need a number of NIZKs for relations over the re-randomizable threshold public key encryption scheme we employ. For the sake of clarity, we define the generic relations for which we will need to prove statements in zero-knowledge and describe our protocols and simulators in terms of those.

$\mathcal{R}_1$ **- Correctness of public key share:** This relation shows that the prover knows the randomness used for generating a public and secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and the secret key sk_i.

$$\pi_{\mathcal{R}_1} \xleftarrow{\$} \mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Prov}\left\{(\mathsf{pk}_i), (r_i, \mathsf{sk}_i) : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param}; r_i)\right\}$$

$\mathcal{R}_2$ **- Correctness of decryption share:** This relation shows that the prover used the secret key sk_i corresponding to its public key pk_i for computing a decryption share d_i of a ciphertext ct.

$$\pi_{\mathcal{R}_2} \xleftarrow{\$} \mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Prov}\left\{(\mathsf{pk}_i, \mathsf{ct}, \mathsf{d}_i), (r_i, \mathsf{sk}_i) : (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param}; r_i), \mathsf{d}_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct})\right\}$$

$\mathcal{R}_3$ **- Correctness of shuffle:** This relation shows that the prover correctly shuffled a set of ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_m)$ by re-randomizing them with randomness $(r_1, \ldots, r_m)$ and permuting them with a permutation Π.

$$\pi_{\mathcal{R}_3} \leftarrow \mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Prov}\left\{\begin{array}{c}(\mathsf{pk}, (\mathsf{ct}_1, \ldots, \mathsf{ct}_m), (\mathsf{ct}'_1, \ldots, \mathsf{ct}'_m)), (\Pi, (r_1, \ldots, r_m)) \ : \\ \forall i \in \{1, \ldots, m\},\ \mathsf{ct}'_{\Pi(i)} \leftarrow \mathsf{ReRand}(\mathsf{pk}, \mathsf{ct}_i; r_i)\end{array}\right\}$$

2.4 Universal Composability

We prove our protocols secure in the Universal Composability (UC) framework introduced by Canetti in [10]. In this section, we present a brief description of the UC framework originally given in [13] and refer interested readers to [10] 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.

In the UC framework, 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 $\bar{r}$.

In addition to these two models of computation, 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, but 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). On the other hand, 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 that represents an 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 3. 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:

$$\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}} \approx_c \mathsf{HYBRID}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$$

We say that a protocol is *statistically secure*, if the same holds for all $\mathcal{Z}$ with unbounded computing power.

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

$\mathcal{F}_{\mathrm{RO}}$ is parameterized by a range $\mathcal{D}$. $\mathcal{F}_{\mathrm{RO}}$ 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).

Figure 1: Functionality $\mathcal{F}_{\mathrm{RO}}$.

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

$\mathcal{F}_{\mathsf{DSIG}}$ interacts with an ideal adversary $\mathcal{S}$, parties $P_1, \ldots, P_n$ and a signer P_s as follows:

- **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 $(m, \sigma, v, 1)$.

- **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.

Figure 2: Functionality $\mathcal{F}_{\mathsf{DSIG}}$.

2.4.1 Adversarial Model:

We consider malicious adversaries, who can arbitrarily deviate from the protocol. We consider *static* adversaries, meaning that the adversary has to corrupt parties before execution starts and the corrupted (or honest) parties remain so throughout the execution.

2.4.2 Setup Assumptions:

It is a well-known fact that UC-secure two-party and multiparty protocols for non trivial functionalities require a setup assumption [12]. The main setup assumption for our protocols is random oracle model [5], which can be modelled in the UC framework as the $\mathcal{F}_{\text{RO}}$-hybrid model (defined in Figure 1). 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 2) and a smart contract functionality (defined in Section 4). We present functionality $\mathcal{F}_{\mathsf{DSIG}}$ as defined in [11], where it is also shown that $\mathsf{EUF\text{-}CMA}$ signature schemes realize $\mathcal{F}_{\mathsf{DSIG}}$. Notice that this fact will be used to show that our protocols can be realized based on practical digital signature schemes such DSA and ECDSA.

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 can invoke 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 answers (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 a balance vector as $\mathsf{balance} = (t, \ldots, t)$ and a bets vector as $\mathsf{bets} = (0, \ldots, 0)$.

Check-out: Whenever GR requests the check-out of the players with payouts specified by the 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 stop the execution.

Compensation: This procedure is triggered whenever $\mathcal{S}$ answers a request for confirmation of an action with (ABORT, sid). For each active honest player $\mathcal{P}_i$, send $(\text{COMPENSATION}, sid, \mathsf{coins}(d + q + \mathsf{balance}[i] + \mathsf{bets}[i]))$ to him. 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 an answer $(\text{ACTION-RSP}, sid, \mathcal{P}_i, act-rsp)$ from $\mathcal{P}_i$, forward it to all other players and GR.

Create Card: Whenever GR requests the creation of a card with value v, choose a new identifier id, store the card (id, v) and send the message $(\text{NEWCARD}, sid, \mathsf{id}, v)$ to all players and GR.

Shuffle Cards: Whenever GR requests the cards with identifiers $(\mathsf{id}_1, \ldots, \mathsf{id}_m)$ to be shuffled, send the message $(\text{SHUFFLE}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{S}$. If $\mathcal{S}$ answers $(\text{SHUFFLE}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$, a random permutation Π is applied to the corresponding 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 $(\text{SHUFFLED}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to all players and GR.

Open Private Card: Whenever GR requests to reveal the card with identifier id to $\mathcal{P}_i$, send the message $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id})$ to $\mathcal{S}$. If $\mathcal{S}$ answers $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id})$, read the card (id, v) from the memory and send the message $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{P}_i$.

Open Public 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.

Shuffle Private Cards: Whenever GR requests a set of private cards $(\mathsf{id}_1, v_1), \ldots, (\mathsf{id}_m, v_m)$ from player $\mathcal{P}_i$ to be shuffled, send the message $(\text{PRIVATE-SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{S}$. If $\mathcal{S}$ answers $(\text{PRIVATE-SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m)$, a random permutation $(v'_1, \ldots, v'_m) = \Pi(v_1, \ldots, v_m)$ is applied to the values $(v_1, \ldots, v_m)$ to obtain the updated cards $(\mathsf{id}_1, v'_m), \ldots, (\mathsf{id}_m, v'_m)$. Send the message $(\text{PRIVATE-SHUFFLED}, sid, (\mathsf{id}_1, v'_1), \ldots, (\mathsf{id}_m, v'_m))$ to $\mathcal{P}_i$ and the message $(\text{PRIVATE-SHUFFLED}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to the other players and GR.

Figure 3: Functionality for card games.

3 Formalization of Secure Card Games

In this section we formalize the notion of a secure card game. Our intention is to capture a large variety of card games, therefore we opt to formalize it using a functionality $\mathcal{F}_{\mathsf{CG}}$ that has an embedded program GR that expresses the rules of the game under consideration. $\mathcal{F}_{\mathsf{CG}}$ offers procedures to check-in and check-out the players into the game, to create cards, to shuffle a specified set of cards, to open cards both to the public as well as privately to one player. Additionally, it offers a channel for GR to communicate with the players. It also offers a mechanism to financially compensate the honest players in case that some adversarial player misbehaves or aborts. GR mediates the execution of the game between the players and performs actions such as: (1) requesting the operations with cards; (2) determining who is the next player that should move and requesting an action from it; (3) processing the outcomes of the players actions and of the operations with cards; (4) updating the players' money balance and bet amounts; (5) invoking the check-out procedure to allow one player to leave the game; (6) requesting the compensation mechanism to be executed if some player does not respond in an appropriate way (the appropriate responses depend on the specifics rules of the game, which are known to the players as well). Later on, the game program GR will also be used to parametrize the protocol, where it will inform the players which card operations have to be executed for each step of the game. Notice that we choose to only capture basic card operations and basic game financial transactions in $\mathcal{F}_{\mathsf{CG}}$. However, it can be easily combined with other functionalities that capture further operations that can come in handy in different games and that can be UC realized with cryptographic protocols. In this case, GR can also request the players to perform operations related to the extra functionalities aggregated into $\mathcal{F}_{\mathsf{CG}}$. A prime example of such an extension is incorporating a coin tossing functionality into $\mathcal{F}_{\mathsf{CG}}$. Such a functionality can be easily (and efficiently) UC-realized in the random oracle model and can provide players with a common source of uniform randomness, which is useful in actions such as choosing the order in which players act at random. Figure 3 describes the functionality $\mathcal{F}_{\mathsf{CG}}$ that captures secure card games.

4 Secure Protocol for Playing Card Games

In this section we describe a protocol that realizes functionality $\mathcal{F}_{\mathsf{CG}}$ with the help of a smart contract. The role of the smart contract is to make sure that all players are executing the card operations (and other game actions) as specified by the game rules programmed in GR and punish (resp. compensate) malicious (resp. honest) players if some problem happens. The basic idea is to follow the secure computation with financial penalties framework initiated by [2, 1] and have each player send to the contract an amount of coins that will be used for betting in the protocol and another amount of coins used as collateral in case of misbehavior. If a player suspects that another player is cheating in the game or misbehaving in protocol execution, it sends a request to the smart contract, which verifies protocol execution and, in case a player was actually found to be misbehaving, financially punishes the malicious player by distributing its collateral coins among the honest players. We model this smart contract as a functionality $\mathcal{F}_{\mathsf{SC}}$, described in Section 4.1. Our protocol is described in Section 4.2.

4.1 The Stateful Smart Contract Functionality $\mathcal{F}_{\mathsf{SC}}$

We follow the approach of Bentov *et al.* [7] in describing a functionality $\mathcal{F}_{\mathsf{SC}}$ that models a *stateful contract*. Such a contract receives coins from the players in a check-in procedure and, after that, is only activated in case a player wishes to report misbehavior or wishes to leave the

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 $(\text{CHECKIN}, sid, \mathcal{P}_i, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i, \mathsf{pk}_i, \pi^i_{\mathcal{R}_1})$ containing the necessary coins, its signature verification key, its share of the threshold ElGamal public-key and the zero-knowledge proof of knowledge of the secret-key's share. Record the values and send $(\text{CHECKEDIN}, sid, \mathcal{P}_i, \mathsf{SIG}.vk_i, \mathsf{pk}_i, \pi^i_{\mathcal{R}_1})$ 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 have 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 $(\texttt{CHECKOUT}|\mathsf{payout})$ 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.

Figure 4: The stateful contract functionality that is used by the secure protocol for card games.

game, retrieving the coins that he owns at that point. While Bentov *et al.* describe a stateful contract functionality that models execution of general programs with secure cash distribution (*i.e.* the output of the computation determines how coins are distributed among honest players) and penalties for misbehavior, we focus on the specific case of card games. That means that our functionality only allows a program GR that specifies the game rules to execute specific card operations instead of general computation. The card operations supported by our protocol are the ones described in functionality $\mathcal{F}_{\mathsf{CG}}$. However, as discussed in Section 3, we can extend $\mathcal{F}_{\mathsf{CG}}$ by incorporating into it other functionalities for which UC protocols are known. In this case, GR is also allowed to specify the operations described in these functionalities and the

stateful contract modelled by GR is also responsible for ensuring that the protocols realizing these functionalities are correctly executed. We describe $\mathcal{F}_{\mathsf{SC}}$ in Figure 4.

4.2 Protocol π_{CG}

We will construct a Protocol π_{CG} that realizes $\mathcal{F}_{\mathsf{CG}}$ in a modular fashion. The main building block of this protocol is a re-randomizable threshold public key encryption (RTE) and associated non-interactive zero-knowledge proofs (NIZK). Moreover, we will rely on a random oracle functionality $\mathcal{F}_{\mathrm{RO}}$ to apply the Fiat-Shamir heuristic to sigma protocols used for instantiating these NIZKs. Additionally, a standard digital signature functionality $\mathcal{F}_{\mathsf{DSIG}}$ will be used as building block in this protocol. Later on, we will describe a concrete instantiation of the protocol under the DDH assumption. We describe Protocol π_{CG} in Figures 5, 6, 7, 8.

In this protocol, the players start by jointly generating a public key for the RTE scheme along with individual secret key shares. The main idea is to represent open cards as ciphertexts of the RTE scheme encrypting a card value $[1, \ldots, 52]$ without any randomness (or randomness 0) while closed cards are shuffled such that they are represented by a re-randomized ciphertext that is permuted in way that cannot be reversed by any proper subset of the players (so that no player/collusion of players can trace the shuffling back to the initial open cards). The shuffle operation is done by having each player act in sequence, taking turns in rerandomizing all ciphertexts representing cards and permuting the resulting rerandomized ciphertexts, while proving in zero-knowledge that these operations were executed correctly. When a closed (shuffled) card has to be revealed to a player, all other players send decryption shares of the ciphertext representing this card computed with their respective secret keys, along with proofs that these decryption shares have been correctly computed.

Throughout the protocol, after a card operation has been executed, the players jointly generate a checkpoint witness proving that the operation has been completed successfully. These checkpoint witnesses contain signatures by all users on the current state of the protocol (*i.e.* ciphertexts representing cards and additional information needed for verifying their decryption, such as public keys of all users). Moreover, the protocol keeps track of each player's balance and current bets in a betting game. Checkpoint witnesses are also generated when this data is updated. If a player suspects that any other player is cheating (or has aborted) during protocol execution, it complains to the smart contract, which requests all users to send their latest checkpoints. At this point, the protocol execution is mediated by the smart contract, which receives (and broadcasts) all protocol messages generated by the players. If the smart contract detects that a player is cheating in this execution (by examining the transcript), it punishes the misbehaving player by distributing its collateral coins among the honest players.

4.3 Security Analysis

We analyze the security of π_{CG} in the UC framework by constructing a simulator such that any environment has a negligible chance to distinguish between an ideal world execution with the simulator and $\mathcal{F}_{\mathsf{CG}}$ and a real world execution of π_{CG} with an adversary. The main idea behind our simulator is to replace all ciphertexts representing cards by ciphertexts containing a known plaintext message. It does that by using the simulator of $\mathsf{NIZK}_{\mathcal{R}_3}$ to generate a simulated proof of correct shuffle given arbitrary ciphertexts (on known messages) without being caught by the other parties. In the private and public card opening procedures, when the simulator learns the value of the opened card from $\mathcal{F}_{\mathsf{CG}}$, it uses the decryption share simulation algorithm $\mathsf{SimShareDec}$ along with the simulator of $\mathsf{NIZK}_{\mathcal{R}_2}$ to generate a decryption share such that the

Protocol π_{CG} (Part 1)

Let $\mathsf{RTE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{CombinePK}, \mathsf{Enc}, \mathsf{ReRand}, \mathsf{ShareDec}, \mathsf{ShareCombine}, \mathsf{SimShareDec}, \mathsf{CombineSK}, \mathsf{Trans})$ be a secure re-randomizable threshold public-key encryption scheme as defined in Section 2.2. For $i \in \{1,2,3\}$, let $\mathsf{NIZK}_{\mathcal{R}_i} = (\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext})$ be a NIZK proof of membership scheme for the relation $\mathcal{R}_i$, as defined in Section 2.3. Protocol π_{CG} is parametrized by a security parameter 1^κ, RTE parameters $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$, 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 $\texttt{NONCE} \xleftarrow{\$} \{0,1\}^\kappa$ 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 $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{F}_{\text{RO}}$ and $\mathcal{F}_{\mathsf{DSIG}}$ as follows:

- **Check-in:** Every player $\mathcal{P}_i$ proceeds as follows:
  1. Send (KEYGEN, sid) to $\mathcal{F}_{\mathsf{DSIG}}$, receiving $(\text{VERIFICATION KEY}, sid, \mathsf{SIG}.vk_i)$ as answer.
  2. Sample randomness $r_i \xleftarrow{\$} \{0,1\}^\kappa$ and generate a key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param}, r_i)$ and a NIZK of public key correctness
  $$\pi^i_{\mathcal{R}_1} \leftarrow \mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Prov}\left\{(\mathsf{pk}_i), (r_i, \mathsf{sk}_i) \ : \ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param}, r_i)\right\}.$$
  3. Send $(\text{CHECKIN}, sid, \mathcal{P}_i, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i, \mathsf{pk}_i, \pi^i_{\mathcal{R}_1})$ to $\mathcal{F}_{\mathsf{SC}}$.
  4. For each other $\mathcal{P}_j$, upon receiving $(\text{CHECKEDIN}, sid, \mathcal{P}_j, \mathsf{SIG}.vk_j, \mathsf{pk}_j, \pi^j_{\mathcal{R}_1})$ from $\mathcal{F}_{\mathsf{SC}}$, check that $\mathsf{NIZK}_{\mathcal{R}_1}.\mathsf{Verify}((\mathsf{pk}_j), \pi^j_{\mathcal{R}_1}) = 1$. Output $(\text{CHECKEDIN}, sid, \mathcal{P}_j)$.
  5. Upon receiving valid check-in from all parties, compute $\mathsf{pk} \leftarrow \mathsf{CombinePK}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$. Initialize the internal lists of open cards $\mathcal{C}_O$, of closed cards $\mathcal{C}_C$ and of private cards of each player $\mathcal{P}_i$, $\mathcal{C}_i$, as empty sets. 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 $j = 1, \ldots, n$.
  6. If $\mathcal{P}_i$ fails to receive a check-in of another party $\mathcal{P}_j$ within the timeout limit τ, it requests $\mathcal{F}_{\mathsf{SC}}$ to dropout and receive its coins back.
- **Checkpoint Witnesses:** After the execution of a procedure, the players store a checkpoint witness that consists of the lists $\mathcal{C}_O, \mathcal{C}_C, \mathcal{C}_1, \ldots, \mathcal{C}_n$, 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 zero-knowledge 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.
- **Create Card:** To create a card with value v, every player $\mathcal{P}_i$ selects the next unused card id id, stores (id, v) in $\mathcal{C}_O$ and outputs $(\text{NEWCARD}, sid, \mathsf{id}, v)$.

Figure 5: Part 1 of Protocol π_{CG}.

Protocol π_{CG} (Part 2)

- **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 $\mathsf{bets}[j] = b'_j$.

- **Shuffle Cards:** In order to shuffle a set of cards with id values $\mathsf{id}_1, \ldots, \mathsf{id}_m$, $\mathcal{P}_i$ initially removes all the (eventual) cards $(\mathsf{id}_k, v_{\mathsf{id}_k})$, for $k \in \{1, \ldots, m\}$, that are in the list of opened cards $\mathcal{C}_O$ from that list and add $(\mathsf{id}, \mathsf{ct}_{\mathsf{id}_k})$, for $\mathsf{ct}_{\mathsf{id}_k} \leftarrow \mathsf{Enc}(\mathsf{pk}, v_{\mathsf{id}_k}, 0)$, in $\mathcal{C}_C$. Define $(\mathsf{ct}^0_{\mathsf{id}_1}, \ldots, \mathsf{ct}^0_{\mathsf{id}_m}) = (\mathsf{ct}_{\mathsf{id}_1}, \ldots, \mathsf{ct}_{\mathsf{id}_m})$, where the right-hand side cards are stored, together with the respective id values, in the internal list $\mathcal{C}_C$. For $j = 1, \ldots, n$:
  1. If $j \neq i$, upon receiving the message $(\text{SHUFFLE}, sid, \mathcal{P}_j, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}^j_{\mathsf{id}_1}, \ldots, \mathsf{ct}^j_{\mathsf{id}_m}, \pi^j_{\mathcal{R}_3})$ from $\mathcal{P}_j$, verifies the NIZK $\pi^j_{\mathcal{R}_3}$ by checking that $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Verify}((\mathsf{pk}, (\mathsf{ct}^{j-1}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{j-1}_{\mathsf{id}_m}), (\mathsf{ct}^j_{\mathsf{id}_1}, \ldots, \mathsf{ct}^j_{\mathsf{id}_m})), \pi^j_{\mathcal{R}_3}) = 1$.
  2. If $j = i$, sample a random permutation Π such that $\mathsf{id}'_1, \ldots, \mathsf{id}'_1 = \Pi(\mathsf{id}_1, \ldots, \mathsf{id}_m)$ and, for $k = 1, \ldots, m$, sample $r_k \xleftarrow{\$} \{0,1\}^\kappa$ and compute $\mathsf{ct}^i_{\mathsf{id}_k} \leftarrow \mathsf{ReRand}(\mathsf{pk}, \mathsf{ct}^{i-1}_{\Pi(\mathsf{id}_k)}, r_k)$. Broadcast $(\text{SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}^i_{\mathsf{id}_1}, \ldots, \mathsf{ct}^i_{\mathsf{id}_m}, \pi^i_{\mathcal{R}_3})$, where the NIZK of shuffle correctness $\pi^i_{\mathcal{R}_3}$ is computed as follows

$$\mathsf{NIZK}_{\mathcal{R}_3}\left\{ \begin{array}{c} \left(\mathsf{pk}, (\mathsf{ct}^{i-1}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{i-1}_{\mathsf{id}_m}), (\mathsf{ct}^i_{\mathsf{id}_1}, \ldots, \mathsf{ct}^i_{\mathsf{id}_m})\right), (\Pi, (r_1, \ldots, r_m)) \ : \\ \forall k \in \{1, \ldots, m\},\ \mathsf{ct}^i_{\mathsf{id}_k} \leftarrow \mathsf{ReRand}(\mathsf{pk}, \mathsf{ct}^{i-1}_{\Pi(\mathsf{id}_k)}, r_k) \end{array} \right\}.$$

  Every player $\mathcal{P}_i$ updates its internal list of closed cards $\mathcal{C}_C$ to contain the pairs $(\mathsf{id}_1, \mathsf{ct}^n_{\mathsf{id}_1}), \ldots, (\mathsf{id}_m, \mathsf{ct}^n_{\mathsf{id}_m})$ in place of the original ones. Output $(\text{SHUFFLED}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$.

- **Shuffle Private Cards:** In order to shuffle a set of private cards with id values $\mathsf{id}_1, \ldots, \mathsf{id}_m$ belonging to player $\mathcal{P}_j$, player $\mathcal{P}_i$ proceed as follows:
  - If $i = j$, sample a random permutation Π such that $\mathsf{id}'_1, \ldots, \mathsf{id}'_1 = \Pi(\mathsf{id}_1, \ldots, \mathsf{id}_m)$ and, for $k = 1, \ldots, m$, sample $r_k \xleftarrow{\$} \{0,1\}^\kappa$ and compute $\mathsf{ct}'_{\mathsf{id}_k} \leftarrow \mathsf{ReRand}(\mathsf{pk}, \mathsf{ct}_{\Pi(\mathsf{id}_k)}, r_k)$. Broadcast $(\text{PRIVSHUFFLE}, sid, \mathcal{P}_j, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m}, \pi^j_{\mathcal{R}_3})$, where the NIZK proof of shuffle correctness $\pi^j_{\mathcal{R}_3}$ is computed as follows:

$$\mathsf{NIZK}_{\mathcal{R}_3}\left\{ \begin{array}{c} \left(\mathsf{pk}, (\mathsf{ct}_{\mathsf{id}_1}, \ldots, \mathsf{ct}_{\mathsf{id}_m}), (\mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m})\right), (\Pi, (r_1, \ldots, r_m)) \ : \\ \forall k \in \{1, \ldots, m\},\ \mathsf{ct}'_{\mathsf{id}_k} \leftarrow \mathsf{ReRand}(\mathsf{pk}, \mathsf{ct}_{\Pi(\mathsf{id}_k)}, r_k) \end{array} \right\},$$

  - If $i \neq j$, upon receiving $(\text{PRIVSHUFFLE}, sid, \mathcal{P}_j, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m}, \pi^j_{\mathcal{R}_3})$ from $\mathcal{P}_j$, verify the NIZK $\pi^i_{\mathcal{R}_3}$ by checking that $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Verify}((\mathsf{pk}, (\mathsf{ct}_{\mathsf{id}_1}, \ldots, \mathsf{ct}_{\mathsf{id}_m}), (\mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m})), \pi^i_{\mathcal{R}_3}) = 1$.

  $\mathcal{P}_j$ outputs $(\text{PRIVATE-SHUFFLED}, sid, (\mathsf{id}_1, v'_1), \ldots, (\mathsf{id}_m, v'_m))$, where the new values $v'_1, \ldots, v'_m$ of the cards associated to each id value are known to him, and the other parties output $(\text{PRIVATE-SHUFFLED}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$. All players update their internal list $\mathcal{C}_C$.

Figure 6: Part 2 of Protocol π_{CG}.

Protocol π_{CG} (Part 3)

- **Open Private Card:** To open a private card $\mathsf{ct}_{\mathsf{id}}$ towards player $\mathcal{P}_j$, proceed as follows:
  - For $i \neq j$, $\mathcal{P}_i$ computes $\mathsf{d}_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct}_{\mathsf{id}})$ and a NIZK of decryption share correctness

$$\pi^i_{\mathcal{R}_2} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Prov}\left\{ \begin{array}{c} (\mathsf{pk}_i, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_i), (r_i, \mathsf{sk}_i) \ : \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param}, r_i), \mathsf{d}_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct}_{\mathsf{id}}) \end{array} \right\},$$

    and sends $(\textsc{opencard}, sid, \mathcal{P}_i, \mathsf{id}, \mathsf{d}_i, \pi^i_{\mathcal{R}_2})$ to $\mathcal{P}_j$. Add id to $\mathcal{C}_j$.
  - Player $\mathcal{P}_j$, upon receiving $(\textsc{opencard}, sid, \mathcal{P}_i, \mathsf{id}, \mathsf{d}_i, \pi^i_{\mathcal{R}_2})$ from $\mathcal{P}_i$, verifies the NIZK decryption share correctness by checking that $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathsf{pk}_i, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_i), \pi^i_{\mathcal{R}_2}) = 1$. Additionally, upon receiving valid decryption shares from all other players, $\mathcal{P}_j$ computes $\mathsf{d}_j \leftarrow \mathsf{ShareDec}(\mathsf{sk}_j, \mathsf{ct}_{\mathsf{id}})$ and retrieves the value of the card by computing $v_{\mathsf{id}} \leftarrow \mathsf{ShareCombine}(\mathsf{d}_1, \ldots, \mathsf{d}_n)$. $\mathcal{P}_j$ adds id to $\mathcal{C}_j$ and outputs $(\textsc{card}, sid, \mathsf{id}, v_{\mathsf{id}})$.

- **Open Public Card:** In order to open a public card $\mathsf{ct}_{\mathsf{id}}$, each player $\mathcal{P}_i$ proceed as follows:
  - compute $\mathsf{d}_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct}_{\mathsf{id}})$ and a NIZK of decryption share correctness

$$\pi^i_{\mathcal{R}_2} \leftarrow \mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Prov}\left\{ \begin{array}{c} (\mathsf{pk}_i, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_i), (r_i, \mathsf{sk}_i) \ : \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{param}, r_i), \mathsf{d}_i \leftarrow \mathsf{ShareDec}(\mathsf{sk}_i, \mathsf{ct}_{\mathsf{id}}) \end{array} \right\},$$

    and broadcast $(\textsc{opencard}, sid, \mathcal{P}_i, \mathsf{id}, \mathsf{d}_i, \pi^i_{\mathcal{R}_2})$.
  - Upon receiving $(\textsc{opencard}, sid, \mathcal{P}_j, \mathsf{id}, \mathsf{d}_j, \pi^j_{\mathcal{R}_2})$ from $\mathcal{P}_j$, verify $\pi^j_{\mathcal{R}_2}$ by checking that $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathsf{pk}_j, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_j), \pi^j_{\mathcal{R}_2}) = 1$. Upon receiving valid decryption shares from all players, retrieve the value of card $\mathsf{ct}_{\mathsf{id}}$ by computing $v_{\mathsf{id}} \leftarrow \mathsf{ShareCombine}(\mathsf{d}_1, \ldots, \mathsf{d}_n)$. Add $(\mathsf{id}, v_{\mathsf{id}})$ to $\mathcal{C}_O$ and output $(\textsc{card}, sid, \mathsf{id}, v_{\mathsf{id}})$.

- **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 $(\textsc{act}, sid, \mathcal{P}_i, act - desc)$ and $\mathcal{P}_i$ executes any necessary operations. $\mathcal{P}_i$ broadcasts $(\textsc{action-rsp}, sid, \mathcal{P}_i, act - rsp, \sigma_i)$, where $act - rsp$ is his answer and σ_i his signature on $act - rsp$, and outputs $(\textsc{action-rsp}, sid, \mathcal{P}_i, act - rsp)$. Upon receiving this message, all other players check the signature, and if it is valid output $(\textsc{action-rsp}, sid, \mathcal{P}_i, act - rsp)$. If a player $\mathcal{P}_j$ believes cheating is happening, he proceeds to the recovery procedure.

- **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 $(\textsc{checkout-init}, sid, \mathcal{P}_j)$ to $\mathcal{F}_{\mathsf{SC}}$.
  2. Upon receiving $(\textsc{checkout-init}, sid, \mathcal{P}_j)$ from $\mathcal{F}_{\mathsf{SC}}$, each $\mathcal{P}_i$ (for $i = 1, \ldots, n$) sends $(\textsc{sign}, sid, (\texttt{CHECKOUT}|\mathsf{payout}))$ to $\mathcal{F}_{\mathsf{DSIG}}$ (where payout is a vector containing the amount of money that each player will receive according to GR), obtaining $(\textsc{signature}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), \sigma_i)$ as answer. Player $\mathcal{P}_i$ sends σ_i to $\mathcal{P}_j$.
  3. For all $i \neq j$, $\mathcal{P}_j$ sends $(\textsc{verify}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), \sigma_i, \mathsf{SIG}.vk_i)$ to $\mathcal{F}_{\mathsf{DSIG}}$, where payout is computed locally by $\mathcal{P}_j$. If $\mathcal{F}_{\mathsf{DSIG}}$ answers all queries $(\textsc{verify}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), \sigma_i, \mathsf{SIG}.vk_i)$ with $(\textsc{verified}, sid, (\texttt{CHECKOUT}|\mathsf{payout}), 1)$, $\mathcal{P}_j$ sends $(\textsc{checkout}, sid, \mathsf{payout}, \sigma_1, \ldots, \sigma_n)$ to $\mathcal{F}_{\mathsf{SC}}$. Otherwise, it proceeds to the recovery procedure.
  4. Upon receiving $(\textsc{payout}, sid, \mathcal{P}_i, \mathsf{coins}(w))$ from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{P}_i$ outputs this message and halts.

Figure 7: Part 3 of Protocol π_{CG}.

Protocol π_{CG} (Part 4)

- **Compensation:** Upon receiving $(\text{COMPENSATION}, sid, \mathcal{P}_i, \mathsf{coins}(w))$ from $\mathcal{F}_{\mathsf{SC}}$, output this message and halt.
- **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.

Figure 8: Part 4 of Protocol π_{CG}.

ciphertext representing that card is decrypted to the intended card value. The security of Protocol π_{CG} is formally captured by the following theorem:

Theorem 4. *Let* RTE *be a secure re-randomizable threshold public-key encryption scheme as defined in Section 2.2. For* $i \in \{1, 2, 3\}$*, let* $\mathsf{NIZK}_{\mathcal{R}_i} = (\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext})$ *be a NIZK Proof of Membership scheme for the relation* $\mathcal{R}_i$ *as defined in Section 2.3. For every static active adversary* $\mathcal{A}$ *who corrupts at most* $n-1$ *parties, there exists a simulator* $\mathcal{S}$ *such that, for every environment* $\mathcal{Z}$*, the following relation holds:*

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

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 $\mathcal{A}$ in the real world and with $\mathcal{S}$ in the ideal world. $\mathcal{S}$ 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 $\mathcal{Z}$.

We now describe the simulator $\mathcal{S}$. Let $\mathcal{P}_h$ denote one of honest parties (any arbitrary one), which we have special procedures during the simulation. As in the protocol, the simulator $\mathcal{S}$ is parametrized by a security parameter 1^κ, RTE parameters $\mathsf{param} \leftarrow \mathsf{Setup}(1^\kappa)$, 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 $\mathcal{F}_{\text{RO}}$:** $\mathcal{S}$ simulates the answers to the random oracle queries from $\mathcal{A}$ exactly as $\mathcal{F}_{\text{RO}}$ 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 Card, Check-out, Compensation:** $\mathcal{S}$ simulates the execution of the respective procedures 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.
- **Shuffle Cards:** In this procedure, $\mathcal{S}$ will replace all ciphertexts representing cards (*i.e.* encrypting a card number) with ciphertexts encrypting 1, in such a way that $\mathcal{S}$ later knows the encrypted values and can generate decryption shares that result in decrypting each ciphertext to an arbitrary value. Upon receiving $(\text{SHUFFLE}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ proceeds as follows to simulate each round of the shuffle cards procedure:
  - Each honest party define $\mathsf{ct}^0_{\mathsf{id}_1}, \ldots, \mathsf{ct}^0_{\mathsf{id}_m}$ using the procedures from π_{CG}.
  - Upon receiving a message $(\text{SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}^i_{\mathsf{id}_1}, \ldots, \mathsf{ct}^i_{\mathsf{id}_m}, \pi^{i-1}_{\mathcal{R}_3})$ on behalf of $\mathcal{P}_i$ from $\mathcal{A}$, $\mathcal{S}$ verifies that $\pi^i_{\mathcal{R}_3}$ is valid using the steps of Protocol π_{CG} that the honest parties would use. If this check fails or the message from the next player that is supposed to shuffle is not received before the timeout limit τ, $\mathcal{S}$ proceeds to the recovery procedure. If the recovery procedure fails, $\mathcal{S}$ sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to the compensation procedure. If the recovery procedure succeeds, proceed with the simulation.
  - If it is the turn of an honest party (except $\mathcal{P}_h$) to shuffle, apart from performing the checks of the previous step, $\mathcal{S}$ simulates the shuffle procedure of π_{CG} and sends the resulting messages to $\mathcal{A}$.
  - If it is $\mathcal{P}_h$'s turn to shuffle, apart from performing the checks of the first step, $\mathcal{S}$ computes ciphertexts $\mathsf{ct}^h_{\mathsf{id}_j} \leftarrow \mathsf{Enc}(\mathsf{pk}, 1)$, for $j = 1, \ldots, m$ (with fresh randomness for each ciphertext). Next, $\mathcal{S}$ executes $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Sim}(\mathsf{pk}, (\mathsf{ct}^{h-1}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{h-1}_{\mathsf{id}_m}), (\mathsf{ct}^h_{\mathsf{id}_1}, \ldots, \mathsf{ct}^h_{\mathsf{id}_m}))$ to generate a proof of shuffle correctness $\pi^h_{\mathcal{R}_3}$ and sends $(\text{SHUFFLE}, sid, \mathcal{P}_h, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}^h_{\mathsf{id}_1}, \ldots, \mathsf{ct}^h_{\mathsf{id}_m}, \pi^h_{\mathcal{R}_3})$ to $\mathcal{A}$.

  If all rounds of the shuffle cards procedure are successfully completed, $\mathcal{S}$ sends $(\text{SHUFFLE}, sid, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{F}_{\mathsf{CG}}$.

- **Shuffle Private Cards:** In this procedure, $\mathcal{S}$ relays corrupted players' private shuffling requests to $\mathcal{F}_{\mathsf{CG}}$ and simulates the honest party's private shuffling requests. Upon receiving $(\text{PRIVATE-SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ proceeds as follows:

  - If $\mathcal{P}_i$ is corrupted, $\mathcal{S}$ waits for a message $(\text{PRIVSHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m}, \pi^i_{\mathcal{R}_3})$ from $\mathcal{P}_i^{\mathcal{A}}$ and verifies $\pi^i_{\mathcal{R}_3}$ by checking that $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Verify}((\mathsf{pk}, (\mathsf{ct}_{\mathsf{id}_1}, \ldots, \mathsf{ct}_{\mathsf{id}_m}), (\mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m})), \pi^i_{\mathcal{R}_3}) = 1$, where $(\mathsf{ct}_{\mathsf{id}_1}, \ldots, \mathsf{ct}_{\mathsf{id}_m})$ are the corresponding ciphertexts before the private shuffle. If this check succeeds, $\mathcal{S}$ sends $(\text{PRIVATE-SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{F}_{\mathsf{CG}}$. If this check fails or the message is not received before the timeout limit τ, $\mathcal{S}$ proceeds to the recovery procedure. If the recovery procedure fails, $\mathcal{S}$ proceeds to the compensation procedure after sending (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$. If the recovery procedure succeeds, $\mathcal{S}$ sends $(\text{PRIVATE-SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{F}_{\mathsf{CG}}$.
  - If $\mathcal{P}_i$ is honest, $\mathcal{S}$ simulates the honest steps of π_{CG} for doing a private shuffle and sends $(\text{PRIVSHUFFLE}, sid, \mathcal{P}_h, \mathsf{id}_1, \ldots, \mathsf{id}_m, \mathsf{ct}'_{\mathsf{id}_1}, \ldots, \mathsf{ct}'_{\mathsf{id}_m}, \pi^h_{\mathcal{R}_3})$ to $\mathcal{A}$. Next, $\mathcal{S}$ sends $(\text{PRIVATE-SHUFFLE}, sid, \mathcal{P}_i, \mathsf{id}_1, \ldots, \mathsf{id}_m)$ to $\mathcal{F}_{\mathsf{CG}}$.

- **Open Private Card:** Upon receiving a message $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id})$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ proceeds as follows:

  - If $\mathcal{P}_i$ is corrupted, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id})$ to $\mathcal{F}_{\mathsf{CG}}$, obtaining $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id}, v)$ as answer. Next, $\mathcal{S}$ simulates an opening of the private card with id represented by ciphertext $\mathsf{ct}_{\mathsf{id}}$ towards party $\mathcal{P}_i$ in such a way that decrypting $\mathsf{ct}_{\mathsf{id}}$ results in v. For all honest parties other than $\mathcal{P}_h$, $\mathcal{S}$ generates the decryption shares honestly using the procedures of π_{CG} and send the resulting messages to $\mathcal{A}$. Additionally, $\mathcal{S}$ generates the decryption share of $\mathcal{P}_h$, d_h, using $\mathsf{SimShareDec}$ such that decrypting $\mathsf{ct}_{\mathsf{id}}$ yields v. Next, $\mathcal{S}$ generates a simulated NIZK $\pi^h_{\mathcal{R}_2}$ showing that the decryption share d_h was correctly computed by executing $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Sim}(\mathsf{pk}_h, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_h)$. These simulated decryption share d_h and proof $\pi^h_{\mathcal{R}_2}$ are recorded and reused if the same ciphertext is opened again. $\mathcal{S}$ sends $(\text{OPENCARD}, sid, \mathcal{P}_h, \mathsf{id}, \mathsf{d}_h, \pi^h_{\mathcal{R}_2})$ to $\mathcal{A}$.
  - If $\mathcal{P}_i$ is honest, $\mathcal{S}$ waits for messages $(\text{OPENCARD}, sid, \mathcal{P}_j, \mathsf{id}, \mathsf{d}_j, \pi^j_{\mathcal{R}_2})$ from each malicious party $\mathcal{P}_j$ controlled by $\mathcal{A}$. Upon receiving each of these messages, $\mathcal{S}$ verifies $\pi^j_{\mathcal{R}_2}$ by checking that $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathsf{pk}_j, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_j), \pi^j_{\mathcal{R}_2}) = 1$. If all of the messages are received and all NIZKs $\pi^j_{\mathcal{R}_2}$ are valid, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id})$ to $\mathcal{F}_{\mathsf{CG}}$. If any of these messages is not received before the timeout limit τ or any of the NIZKs $\pi^j_{\mathcal{R}_2}$ are invalid, $\mathcal{S}$ proceeds to the recovery procedure. If the recovery procedure succeeds, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathcal{P}_i, \mathsf{id})$ to $\mathcal{F}_{\mathsf{CG}}$. Otherwise, $\mathcal{S}$ sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to the compensation procedure.

- **Open Public Card** Upon receiving a message $(\text{CARD}, sid, \mathsf{id}, v)$ from $\mathcal{F}_{\mathsf{CG}}$, $\mathcal{S}$ simulates an opening of the public card with id represented by ciphertext $\mathsf{ct}_{\mathsf{id}}$ towards $\mathcal{A}$ in such a way that v is obtained, proceeding as follows:

  - For all honest parties other than $\mathcal{P}_h$, $\mathcal{S}$ generates the decryption shares honestly using the procedures of π_{CG} and send the resulting messages to $\mathcal{A}$. Additionally, $\mathcal{S}$ generates the decryption share of $\mathcal{P}_h$, d_h, using $\mathsf{SimShareDec}$ such that decrypting $\mathsf{ct}_{\mathsf{id}}$ yields v. Next, $\mathcal{S}$ generates a simulated NIZK $\pi^h_{\mathcal{R}_2}$ showing that the decryption share d_h was correctly computed by executing $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Sim}(\mathsf{pk}_h, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_h)$. These simulated

decryption share d_h and proof $\pi^h_{\mathcal{R}_2}$ are recorded and reused if the same ciphertext is opened again. $\mathcal{S}$ sends $(\text{OPENCARD}, sid, \mathcal{P}_h, \mathsf{id}, \mathsf{d}_h, \pi^h_{\mathcal{R}_2})$ to $\mathcal{A}$.

– $\mathcal{S}$ waits for messages $(\text{OPENCARD}, sid, \mathcal{P}_j, \mathsf{id}, \mathsf{d}_j, \pi^j_{\mathcal{R}_2})$ from each malicious party $\mathcal{P}_j$ controlled by $\mathcal{A}$. Upon receiving each of these messages, $\mathcal{S}$ verifies $\pi^j_{\mathcal{R}_2}$ by checking that $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Verify}((\mathsf{pk}_j, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_j), \pi^j_{\mathcal{R}_2}) = 1$. If all of the messages are received and all NIZKs $\pi^j_{\mathcal{R}_2}$ are valid, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{F}_{\mathsf{CG}}$. If any of these messages is not received before the timeout limit τ or any of the NIZKs $\pi^j_{\mathcal{R}_2}$ are invalid, $\mathcal{S}$ proceeds to the recovery procedure. If the recovery procedure succeeds, $\mathcal{S}$ sends $(\text{CARD}, sid, \mathsf{id}, v)$ to $\mathcal{F}_{\mathsf{CG}}$. Otherwise, $\mathcal{S}$ sends (ABORT, sid) to $\mathcal{F}_{\mathsf{CG}}$ and proceeds to the compensation procedure.

Simulation Indistinguishability: Notice that the simulator $\mathcal{S}$ only deviates from a real execution of Protocol π_{CG} in the shuffle cards, shuffle private cards, open private card and open public card procedures. In the case of the shuffle procedure, $\mathcal{S}$ acts exactly as an honest party in a real execution of π_{CG} except for when $\mathcal{P}_h$ is performing a shuffling operation (*i.e.* rerandomizing the ciphertexts representing cards, permuting them and generating proofs of shuffle correctness). In this case, $\mathcal{S}$ deviates from protocol execution by replacing the ciphertexts it receives from the previous party (or $\mathcal{A}$) by arbitrary ciphertexts containing the message 1 and generating a proof of shuffle correctness $\pi^h_{\mathcal{R}_3}$ by running $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Sim}(\mathsf{pk}, (\mathsf{ct}^{h-1}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{h-1}_{\mathsf{id}_m}), (\mathsf{ct}^{h}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{h}_{\mathsf{id}_m}))$. If the environment $\mathcal{Z}$ can distinguish an execution with ciphertexts $\mathsf{ct}^h_{\mathsf{id}_i} \leftarrow \mathsf{Enc}(\mathsf{pk}, 1)$ from an execution with the ciphertexts generated by an honest execution of π_{CG}, it can be used to break the unlinkability or IND-CPA security properties of RTE, since these ciphertexts are indistinguishable given these two properties. If $\mathcal{Z}$ can distinguish an execution with the proof of shuffle correctness $\pi^h_{\mathcal{R}_3}$ obtained by running $\mathsf{NIZK}_{\mathcal{R}_3}.\mathsf{Sim}(\mathsf{pk}, (\mathsf{ct}^{h-1}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{h-1}_{\mathsf{id}_m}), (\mathsf{ct}^{h}_{\mathsf{id}_1}, \ldots, \mathsf{ct}^{h}_{\mathsf{id}_m}))$ from an execution with the proof of shuffle correctness obtained by honestly running π_{CG}, it can be used to break the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{R}_3}$. In the case of the shuffle private cards procedure, $\mathcal{S}$ instructs $\mathcal{F}_{\mathsf{CG}}$ to abort execution in case the adversary $\mathcal{A}$ provided an invalid message, which also causes an abort in the real execution of π_{CG}. In the cases of the open public card and open private card procedures, $\mathcal{S}$ deviates from the protocol by computing an arbitrary decryption share d_h using $\mathsf{SimShareDec}$ such that decrypting $\mathsf{ct}_{\mathsf{id}}$ yields v (obtained from $\mathcal{F}_{\mathsf{CG}}$) and generating a simulated NIZK $\pi^h_{\mathcal{R}_2}$ showing that the decryption share was correctly computed by running $\mathsf{NIZK}_{\mathcal{R}_2}.\mathsf{Sim}(\mathsf{pk}_h, \mathsf{ct}_{\mathsf{id}}, \mathsf{d}_h)$. If the environment distinguishes an execution with the simulated decryption share d_h from an honest execution of π_{CG}, it can be used to break the share-simualtion indistinguishability of RTE. If $\mathcal{Z}$ distinguishes an execution with the simulated NIZK $\pi^h_{\mathcal{R}_2}$ from an honest execution of π_{CG}, it can be used to break the zero-knowledge property of $\mathsf{NIZK}_{\mathcal{R}_3}$. Hence, the ideal execution with $\mathcal{S}$ is indistinguishable from a real execution of π_{CG} as long as RTE is a secure re-randomizable threshold public-key encryption scheme as defined in Section 2.2 and, for $i \in \{1, 2, 3\}$, $\mathsf{NIZK}_{\mathcal{R}_i} = (\mathsf{Prov}, \mathsf{Verify}, \mathsf{Sim}, \mathsf{Ext})$ is a NIZK proof of membership scheme for the relation $\mathcal{R}_i$ as defined in Section 2.3. □

4.4 A DDH-Based Instantiation

In this section, we describe an instantiation of the Protocol π_{CG} that is secure under the popular DDH assumption in the random oracle model (*i.e.* substituting $\mathcal{F}_{\text{RO}}$ for a cryptographic hash function). The main components we need to construct in order to instantiate our protocol are the re-randomizable threshold public-key encryption scheme RTE and the NIZKs Proof of Membership schemes $\mathsf{NIZK}_{\mathcal{R}_1}, \mathsf{NIZK}_{\mathcal{R}_2}, \mathsf{NIZK}_{\mathcal{R}_3}$ for relations $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$. It was shown in [33,

Appendix C.2], that the threshold version of the classical ElGamal cryptosystem is a secure re-randomizable threshold public-key encryption scheme under the DDH assumption. Moreover, it was also shown in [33, Appendix C.2] that there exist NIZKs $\mathsf{NIZK}_{\mathcal{R}_1}, \mathsf{NIZK}_{\mathcal{R}_2}, \mathsf{NIZK}_{\mathcal{R}_3}$ for relations $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3$ secure under the DDH assumption. Namely, $\mathsf{NIZK}_{\mathcal{R}_1}$ can be implemented by the sigma protocol of Schnorr [26], $\mathsf{NIZK}_{\mathcal{R}_2}$ can be implemented by the protocol of Chaum and Pedersen [15] and $\mathsf{NIZK}_{\mathcal{R}_3}$ can be implemented by the protocol of Bayer and Groth [4]. Notice that the zero-knowledge argument of shuffle correctness of Bayer and Groth [4] requires a common reference string that consists of random group elements such that the discrete logarithm of these elements in a given base is unknown. We point out that such a common reference string can be trivially constructed before π_{CG} is run by coin tossing, which can be UC-realized based on UC-secure commitments [10, 12]. Even though these protocols are interactive, they can be made non-interactive through the Fiat-Shamir heuristic [19, 24]. Notice that their simulators are straight-line since they only need to program the random oracle. As for the digital signature functionality $\mathcal{F}_{\mathsf{DSIG}}$, it is known that $\mathsf{EUF\text{-}CMA}$ signature schemes realize $\mathcal{F}_{\mathsf{DSIG}}$, meaning that it can be realized based on practical digital signature schemes secure under the DDH assumption such DSA and ECDSA. If we use the resulting DDH-based instantiation to implement poker, we obtain a poker protocol very similar to the Kaleidoscope poker protocol [18], which only achieves sequential composability. Hence, we can use our Protocol π_{CG} to obtain an universally composable protocol for the game of poker with rewards and penalties enforcement that achieves basically the same efficiency as Kaleidoscope, which is the currently best (but not UC-secure) protocol for this specific functionality.

5 Concrete Complexity

Royale is both the first cryptographic protocol to support general card games based on a set of core card operations and one of the very few to be constructed based on generic primitives (with the exception of [3]), making it hard to compare its efficiency with previous works that are based on specific computational assumptions and focused on poker. Therefore, we estimate and compare the computational, communication and round complexities of each individual card operation in the works that introduce the previously most efficient (but unproven) poker protocols with the card operations in the DDH-based instantiation of Royale (described in Section 4.4). For the comparison, we focus on the works of Barnett and Smart [3] (specifically its DDH-based instantiation), and the protocols proposed as a building block for the (unproven) tailor-made poker protocol of Bentov *et al.* [7]: Wei and Wang [30] and Wei [29]. We remark that these previous works have not been formally proven secure. Moreover, differently from Royale, even if these previous works can be proven to implement a game of poker, using their card operations arbitrarily might cause security issues, as they are not designed to be composable. We have a particular interest on the *shuffle* procedure of these protocols, given that it is the most costly operation. As previously discussed, the card operations in this DDH-based instantiation have exactly the same efficiency as the individual card operations of the Kaleidoscope [18] poker protocol, so this protocol is not explicitly considered in our comparison. Our complexity estimates in terms of the number of players and card decks are presented in Table 1 and Table 4, while complexity estimates for Shuffle Cards phase with concrete practical parameters are presented in Table 2, Table 3, Table 5 and Table 6.

Instantiating the Building Blocks: In order to provide a fair comparison with previous works, we instantiate the protocols of Barnett and Smart [3], Wei and Wang [30] and Wei [29]

with the same efficient random oracle based building blocks used in our DDH-based instantiation of Royale. These include employing canonical random oracle-based commitments and non-interactive zero-knowledge (NIZK) proofs based on the Fiat-Shamir heuristic. In terms of protocol specific parameters, for the protocols of [3] and [30] a cut-and-choose security parameter of $s = 40$ is considered, while for the protocol of [29], we consider the parameter $k = 4$. Instantiating the NIZK of shuffle correctness required by Royale with the construction of [4] requires a careful choice of parameters, as the total number of cards is represented as $m = m_1 m_2$ and the choice of m_1 and m_2 affects both the computational and communication complexities. Namely, the communication complexity of the resulting NIZK is of $11m_1\mathbb{G} + 5n\mathbb{Z}_p$ (where $\mathbb{G}$ is the size of a group element and $\mathbb{Z}_p$ is the size of a ring element), while computational complexity for the prover (resp. verifier) is $2\log(m_1)m$ (resp. $4m$) exponentiations. Even though the choice of m_1 and m_2 can be optimized to obtain either shorter or faster proofs, in our general comparison we assume that $m_1 = m_2 = \lceil\sqrt{m}\rceil$, which not always represents the optimal choice of parameters.

	Barnett-Smart [3]	Bentov *et al.* [7] (with Wei-Wang [30])	Bentov *et al.* [7] (with Wei [29])	Royale
Shuffle Cards	$240m(n-1)$ $+161m$	$(44n+1)m$	$81m + 2n + 25$	$(2\log(\lceil\sqrt{m}\rceil)$ $+4n-2)m$
Open Private Card $\mathcal{P}_j$ (drawer), $\mathcal{P}_i$ (others)	$4(n-1)+1$, 3	$4n-3$, 3	$4n-3$, 3	$4n-3$, 3
Open Public Card	$4(n-1)+4$	$4n$	$4n$	$4n$

Table 1: Computational complexity in terms of modular exponentiations, where n is the number of players and m is the number of cards.

Computational Complexity: We estimate the computational complexity in terms of the number of modular exponentiations required for each card operation, since these operations tend to dominate the complexity in such cryptographic protocols. We consider the amount of local computation performed by each player. Our estimates are presented in Table 1. As previously observed, the Open Public Card and Open Private Card of all protocols we consider in our comparison have roughly the same concrete complexity, while the Shuffle Cards phase is the main bottleneck, clearly dominating complexity in these protocols. Notice that the two most efficient protocols in our comparison are Royale and the protocol of Wei [29], which actually achieves better asymptotic efficiency than Royale. However, we remark that Royale achieves *better concrete efficiency* than the protocol of Wei [29] for *practical* parameters. Estimates of the number of modular exponentiations required of each player in the Shuffle Cards phase of Royale with numbers of users and of cards commonly found in practical applications (*i.e.* popular games) are presented in Table 2, while the same estimates for the protocol of Wei [29] are presented in Table 3. Namely, for a maximum number of 18 players, Royale achieves better concrete computational complexity than the protocol of Wei [29] unless more than 2000 cards are used, which easily covers the number of cards used in commonly played card games. For example, in a game with 6 players and a standard deck of 52 cards (*e.g.* Poker), the Shuffle Cards phase of the protocol of Wei [29] requiresppproximately 3 times more exponentiations than Royale and, in a game with 12 players and 312 cards (*e.g.* Baccarat), the Shuffle Cards phase of the protocol of Wei [29] requires approximately 1.45 times more exponentiations than Royale.

m \ n	2	4	6	8	10	12
52	624	1040	1456	1872	2288	2704
104	1456	2288	3120	3952	4784	5616
156	2184	3432	4680	5928	7176	8424
208	2912	4576	6240	7904	9568	11232
260	4160	6240	8320	10400	12480	14560
312	4992	7488	9984	12480	14976	17472

Table 2: Concrete computational complexity of the Shuffle Cards phase of Royale in terms of modular exponentiations for practical numbers of players (n) and cards (m).

m \ n	2	4	6	8	10	12
52	4241	4245	4249	4253	4257	4261
104	8453	8457	8461	8465	8469	8473
156	12665	12669	12673	12677	12681	12685
208	16877	16881	16885	16889	16893	16897
260	21089	21093	21097	21101	21105	21109
312	25301	25305	25309	25313	25317	25321

Table 3: Concrete computational complexity of the Shuffle Cards phase of Wei [29] in terms of modular exponentiations for practical numbers of players (n) and cards (m).

Communication Complexity: We estimate the communication complexity in terms of the number of elements of $\mathbb{G}$ and elements of $\mathbb{Z}_p$ exchanged in each phase of the protocols. In contrast to the case of computational complexity, we consider the total amount of data exchanged over the network by all players during each phase of the analyzed protocols. As it is the case with computational complexity, the concrete communication complexity of the Open Public Card and Open Private Card phases in all analyzed protocols is roughly the same, while the Shuffle Cards phase constitutes the main bottleneck and dominates complexity. Notice that the two most efficient protocols in our comparison are Royale and the protocol of Wei [29]. However, in this case, Royale actually achieves both better asymptotic communication complexity and *better concrete efficiency* than the protocol of Wei [29]. Estimates of the total number of elements of $\mathbb{G}$ and of $\mathbb{Z}_p$ exchanged between all players in the Shuffle Cards phase of Royale with numbers of players and of cards commonly found in practical applications (*i.e.* popular games) are presented in Table 5, while the protocol of Wei [29] is considered in Table 6. For example, in a game with 6 players and a standard deck of 52 cards (*e.g.* Poker), the Shuffle Cards phase of the protocol of Wei [29] exchanges approximately 8 times more elements of $\mathbb{G}$ and twice more elements of $\mathbb{Z}_p$ than Royale and, in a scenario with 12 players and 312 cards (*e.g.* Baccarat), the Shuffle Cards phase of the protocol of Wei [29] also exchanges approximately 8 times more elements of $\mathbb{G}$ and twice more elements of $\mathbb{Z}_p$ than Royale.

Round Complexity: As in the previous cases, the Open Public Card and Open Private Card phases of the protocols in our comparison achieve roughly the same efficiency, while the Shuffle Cards phase dominates round complexity. Royale's Shuffle Card phase requires only n rounds (where n is the number of players) while the protocol of Wai and Wang [30] requires $4n + 1$ rounds and the protocol of Wei [29] requires $4n + 3$ rounds. Hence, Royale presents a clear

	Barnett-Smart [3]	Bentov *et al.* [7] (with Wei-Wang [30])	Bentov *et al.* [7] (with Wei [29])	Royale
Shuffle Cards	$164nm\ \mathbb{G}$, $122nm\ \mathbb{Z}_p$	$45nm\ \mathbb{G}$, $(2n^2+80n+2nm)\ \mathbb{Z}_p$	$n(17m+5)\ \mathbb{G}$, $n(m+18)\ \mathbb{Z}_p$	$n(2m+\lceil\sqrt{m}\rceil)\ \mathbb{G}$, $5n\lceil\sqrt{m}\rceil\ \mathbb{Z}_p$
Open Private Card	$3(n-1)\ \mathbb{G}$, $2(n-1)\ \mathbb{Z}_p$	$(n-1)\ \mathbb{G}$, $2(n-1)\ \mathbb{Z}_p$	$(n-1)\ \mathbb{G}$, $2(n-1)\ \mathbb{Z}_p$	$(n-1)\ \mathbb{G}$, $2(n-1)\ \mathbb{Z}_p$
Open Public Card	$3n\ \mathbb{G}$, $2n\ \mathbb{Z}_p$	$n\ \mathbb{G}$, $2n\ \mathbb{Z}_p$	$n\ \mathbb{G}$, $2n\ \mathbb{Z}_p$	$n\ \mathbb{G}$, $2n\ \mathbb{Z}_p$

Table 4: Communication complexity in terms of group elements $\mathbb{G}$ and ring elements $\mathbb{Z}_p$, where n is the number of players and m is the number of cards.

m \ n	2	4	6	8	10	12
52	220 $\mathbb{G}$, 80 $\mathbb{Z}_p$	440 $\mathbb{G}$, 160 $\mathbb{Z}_p$	660 $\mathbb{G}$, 240 $\mathbb{Z}_p$	880 $\mathbb{G}$, 320 $\mathbb{Z}_p$	1100 $\mathbb{G}$, 400 $\mathbb{Z}_p$	1320 $\mathbb{G}$, 480 $\mathbb{Z}_p$
104	430 $\mathbb{G}$, 110 $\mathbb{Z}_p$	860 $\mathbb{G}$, 220 $\mathbb{Z}_p$	1290 $\mathbb{G}$, 330 $\mathbb{Z}_p$	1720 $\mathbb{G}$, 440 $\mathbb{Z}_p$	2150 $\mathbb{G}$, 550 $\mathbb{Z}_p$	2580 $\mathbb{G}$, 660 $\mathbb{Z}_p$
156	640 $\mathbb{G}$, 130 $\mathbb{Z}_p$	1280 $\mathbb{G}$, 260 $\mathbb{Z}_p$	1920 $\mathbb{G}$, 390 $\mathbb{Z}_p$	2560 $\mathbb{G}$, 520 $\mathbb{Z}_p$	3200 $\mathbb{G}$, 650 $\mathbb{Z}_p$	3840 $\mathbb{G}$, 780 $\mathbb{Z}_p$
208	848 $\mathbb{G}$, 150 $\mathbb{Z}_p$	1696 $\mathbb{G}$, 300 $\mathbb{Z}_p$	2544 $\mathbb{G}$, 450 $\mathbb{Z}_p$	3392 $\mathbb{G}$, 600 $\mathbb{Z}_p$	4240 $\mathbb{G}$, 750 $\mathbb{Z}_p$	5088 $\mathbb{G}$, 900 $\mathbb{Z}_p$
260	1058 $\mathbb{G}$, 170 $\mathbb{Z}_p$	2116 $\mathbb{G}$, 340 $\mathbb{Z}_p$	3174 $\mathbb{G}$, 510 $\mathbb{Z}_p$	4232 $\mathbb{G}$, 680 $\mathbb{Z}_p$	5290 $\mathbb{G}$, 850 $\mathbb{Z}_p$	6348 $\mathbb{G}$, 1020 $\mathbb{Z}_p$
312	1266 $\mathbb{G}$, 180 $\mathbb{Z}_p$	2532 $\mathbb{G}$, 360 $\mathbb{Z}_p$	3798 $\mathbb{G}$, 540 $\mathbb{Z}_p$	5064 $\mathbb{G}$, 720 $\mathbb{Z}_p$	6330 $\mathbb{G}$, 900 $\mathbb{Z}_p$	7596 $\mathbb{G}$, 1080 $\mathbb{Z}_p$

Table 5: Concrete communication complexity of the Shuffle Cards phase of Royale for practical numbers of players (n) and cards (m).

advantage in terms of round complexity, which results in better performance in high latency networks, such as the Internet.

Checkpoint Witnesses and On-Chain Storage Complexity: In a real world implementation of Royale, a very promising way to realize the smart contract functionality $\mathcal{F}_{\mathsf{SC}}$ is to employ a smart contract system running on top of a blockchain with an associated cryptocurrency. As a consequence, the information sent by the players to $\mathcal{F}_{\mathsf{SC}}$ has to be stored in space-constrained blocks, raising a concern about on-chain storage complexity. First, we remark that Royale is designed in such a way that only the Check-in, Check-out and Recovery phases cause any information to be sent to $\mathcal{F}_{\mathsf{SC}}$ (and consequently stored in the blockchain), with the Recovery phase only being activated if a player misbehaves. In the Check-in phase, signatures verification keys and public key shares (plus associated proofs of validity) for each players are registered with the smart contract, amounting to storing $(2\ \mathbb{G} + 2\ \mathbb{Z}_p)n$ bits, where n is the number of players. Table 7 presents the size of this on-chain storage for typical number of players. In the Check-out phase, the vector payout (of size $|\mathsf{payout}|$) along with signatures by each player are sent to the smart contract, amounting to $|\mathsf{payout}| + 2n\ \mathbb{Z}_p$ being stored. Assuming that the

m \ n	2	4	6	8	10	12
52	2028 $\mathbb{G}$, 140 $\mathbb{Z}_p$	3796 $\mathbb{G}$, 280 $\mathbb{Z}_p$	5564 $\mathbb{G}$, 420 $\mathbb{Z}_p$	7332 $\mathbb{G}$, 560 $\mathbb{Z}_p$	9100 $\mathbb{G}$, 700 $\mathbb{Z}_p$	10868 $\mathbb{G}$, 840 $\mathbb{Z}_p$
104	4056 $\mathbb{G}$, 244 $\mathbb{Z}_p$	7592 $\mathbb{G}$, 488 $\mathbb{Z}_p$	11128 $\mathbb{G}$, 732 $\mathbb{Z}_p$	14664 $\mathbb{G}$, 976 $\mathbb{Z}_p$	18200 $\mathbb{G}$, 1220 $\mathbb{Z}_p$	21736 $\mathbb{G}$, 1464 $\mathbb{Z}_p$
156	6084 $\mathbb{G}$, 348 $\mathbb{Z}_p$	11388 $\mathbb{G}$, 696 $\mathbb{Z}_p$	16692 $\mathbb{G}$, 1044 $\mathbb{Z}_p$	21996 $\mathbb{G}$, 1392 $\mathbb{Z}_p$	27300 $\mathbb{G}$, 1740 $\mathbb{Z}_p$	32604 $\mathbb{G}$, 2088 $\mathbb{Z}_p$
208	8112 $\mathbb{G}$, 452 $\mathbb{Z}_p$	15184 $\mathbb{G}$, 904 $\mathbb{Z}_p$	22256 $\mathbb{G}$, 1356 $\mathbb{Z}_p$	29328 $\mathbb{G}$, 1808 $\mathbb{Z}_p$	36400 $\mathbb{G}$, 2260 $\mathbb{Z}_p$	43472 $\mathbb{G}$, 2712 $\mathbb{Z}_p$
260	10140 $\mathbb{G}$, 556 $\mathbb{Z}_p$	18980 $\mathbb{G}$, 1112 $\mathbb{Z}_p$	27820 $\mathbb{G}$, 1668 $\mathbb{Z}_p$	36660 $\mathbb{G}$, 2224 $\mathbb{Z}_p$	45500 $\mathbb{G}$, 2780 $\mathbb{Z}_p$	54340 $\mathbb{G}$, 3336 $\mathbb{Z}_p$
312	12168 $\mathbb{G}$, 660 $\mathbb{Z}_p$	22776 $\mathbb{G}$, 1320 $\mathbb{Z}_p$	33384 $\mathbb{G}$, 1980 $\mathbb{Z}_p$	43992 $\mathbb{G}$, 2640 $\mathbb{Z}_p$	54600 $\mathbb{G}$, 3300 $\mathbb{Z}_p$	65208 $\mathbb{G}$, 3960 $\mathbb{Z}_p$

Table 6: Concrete communication complexity of the Shuffle Cards phase of Wei [29] for practical numbers of players (n) and cards (m).

vector payout can be represented by n fields of 64 bits (allowing for representing high values in terms of one hundredth of millionth of coin), the Check-Out Phase requires $64n + 2n\mathbb{Z}_p$ bits of on-chain storage. Table 8 presents the size of this on-chain storage for typical number of players.

In the Recovery phase, the most up-to-date checkpoint witness is sent by the complaining player to the smart contract, which subsequently registers all other player's messages for the phase to be executed after this checkpoint witness was generated. The checkpoint witness itself includes the lists of closed cards $\mathcal{C}_O$, open cards $\mathcal{C}_C$ and private cards of each player $\mathcal{C}_i$, vectors $\mathsf{balance}$ and bets, as well as signatures by each player on these contents. We assume that vectors $\mathsf{balance}$ and bets are each represented by n fields of 64 bits, allowing for representing high values in terms of one hundredth of millionth of coin. Notice that each open card in $\mathcal{C}_O$ can be represented by a pair (id, v) and that each private card in $\mathcal{C}_i$ can be represented by an id value id. If we consider a standard deck of 52 cards, each card value v can be represented by 6 bits, while id values id with 9 bits can represent 512 cards, covering the vast majority of popular card games. On the other hand, each closed card in $\mathcal{C}$ is represented by a pair $(\mathsf{id}, \mathsf{ct}_{\mathsf{id}})$, where $\mathsf{ct}_{\mathsf{id}}$ consists of two elements of $\mathbb{G}$ (each requiring at least 256 of storage). Hence, the worst case for checkpoint witness size is that where all cards are still closed, resulting in the largest possible representation of lists $\mathcal{C}_C$, $\mathcal{C}_O$ and $\mathcal{C}_i$. In this case, the checkpoint witness has size $2m\ \mathbb{G} + |\mathsf{id}|m + |\mathsf{balance}| + |\mathsf{bets}| + 2n\ \mathbb{Z}_p$ bits, which translates to $2m\ \mathbb{G} + 9m + 128n + 2n\ \mathbb{Z}_p$ bits using our estimated sizes of id, $\mathsf{balance}$ and bets, where n is the number of players and m is the number of cards. Apart from the checkpoint witnesses, the Recovery Phase requires the messages of the phase executed after the latest checkpoint to be sent to the smart contract, amounting to on-chain storage equal to the communication complexity of each phase (as estimated above). Table 15 presents the size of this on-chain storage for typical number of players and cards.

6 Benchmarks

In this section we present benchmarks of Royale obtained with a prototype implementation of the DDH-based instantiation described in Section 4.4, showcasing the efficiency of our protocol for practical parameters. Our prototype implementation was done in Haskell using NIST curve

P-256. Experiments were conducted on a Intel i7 7500U CPU with 16 GB RAM running Linux Kernel 4.14.16. We analyze the network communication and execution times of each of Royale's main phases (with high on-chain storage/communication/computation requirements), considering scenarios with different numbers of cards and players. We focus on the following phases of Royale: Check-Out, Check-Out, Shuffle Cards, Shuffle Private Cards, Open Private Card and Open Public Card. Moreover, we analyse on-chain storage requirements for the Checkpoint Witnesses used in the Recovery Phase considering an implementation of the smart contract functionality $\mathcal{F}_{\mathsf{SC}}$ based on a smart contract verifies individual steps of Royale (*i.e.* checking NIZK, signature and encryption validity). While this approach requires the blockchain system to implement such special purpose transaction, it significantly reduces our protocol's footprint on the blockchain, even in case the recovery phase is activated.

We evaluate the execution time required by the aforementioned phases of Royale in milliseconds (ms) and consider network delays in term of Round Trip Times (RTT). Our analysis shows that Royale achieves high computational efficiency, with network delays representing the main bottleneck. We analyze the on-chain storage required by Royale in terms of the size in kilobytes (KB) of the data stored by the smart contract in each phase, which is zero for all phases, except for Check-in, Check-out and Recovery. Our analysis shows that the fixed on-chain footprints of the Check-in, Check-out and Recovery phases is reasonably small for practical parameters. While the Recovery phase always requires storage of the must up-to-date checkpoint witness (whose size we estimate in KB), it also requires player's messages for the current phase to be stored, whose size we also analyse in terms of KB as the network communication required for each phase.

Check-In In the check-in phase, all players locally generate signature key pairs and RTE public and secret key shares, which are registered with the smart contract functionality. Next, all players verify each others' share and locally compute an aggregated public key. The local execution times and the on-chain storage required by the Check-in Phase for different number of users is presented in Table 7. Notice that all communication in this phase is done via the smart contract and that it does not depend on the number of cards.

n:	2	4	6	8	10	12
Time:	0.38 ms	0.62 ms	0.84 ms	1.14 ms	1.3 ms	1.99 ms
On-Chain Storage:	0.25 KB	0.51 KB	0.76 KB	1.02 KB	1.27 KB	1.52 KB

Table 7: Check-in Phase execution time in milliseconds (ms) and on-chain storage size in Kilobytes (KB) for n players.

n:	2	4	6	8	10	12
Size	0.14 KB	0.28 KB	0.42 KB	0.56 KB	0.70 KB	0.84 KB

Table 8: Check-out Phase on-chain storage size in Kilobytes (KB) for n players.

Check-Out The Check-Out phase only requires players to sign vector payout and verify each other's signature, resulting in very low computational overhead. The on-chain storage required by the Check-out Phase for different number of users is presented in Table 8 (notice this does

not depend on the number of cards). The implementation represents vector payout by n fields of 64 bits.

m \ n	2	4	6	8	10	12
52	131.06 ms + 1 RTT	262.12 ms + 2 RTT	393.18 ms + 3 RTT	524.24 ms + 4 RTT	655.3 ms + 5 RTT	786.36 ms + 6 RTT
104	322.47 ms + 1 RTT	644.93 ms + 2 RTT	967.4 ms + 3 RTT	1289.86 ms + 4 RTT	1612.33 ms + 5 RTT	1934.8 ms + 6 RTT
208	634.66 ms + 1 RTT	1269.32 ms + 2 RTT	1903.98 ms + 3 RTT	2538.64ms + 4 RTT	3173.3 ms + 5 RTT	3807.96 ms + 6 RTT

Table 9: Execution time for the Shuffle Card phase with m cards and n players in milliseconds (ms) and Round Trip Time (RTT).

m \ n	2	4	6	8	10	12
52	13.73 KB	27.45 KB	41.18 KB	54.91 KB	68.63 KB	82.36 KB
104	24.49 KB	48.98 KB	73.48 KB	97.97 KB	122.46 KB	146.95 KB
208	40.73 KB	81.47 KB	122.2 KB	162.94 KB	203.67 KB	244.41 KB

Table 10: Network communication in the Shuffle Cards phase with m cards and n players in Kilobytes (KB).

Shuffle Cards The execution time and network communication for the Shuffle Cards phase with with m cards and n players are presented in Table 9 and Table 10, respectively. The execution time is presented as the sum of the local computation time required of each player and the network Round Trip Time necessary for delivering this phase's messages. All players execute the same actions in a round robin manner, *i.e.* verifying a proof of shuffle correctness by the previous player, rerandomizing and permuting ciphertexts, generating a proof of shuffle correctness and, after all parties shuffle the cards, generating a signature for the checkpoint witness.

m \ n	2	4	6	8	10	12
52	6.93 KB	7.05 KB	7.18 KB	7.3 KB	7.43 KB	7.55 KB
104	12.31 KB	12.43 KB	12.56 KB	12.68 KB	12.81 KB	12.93 KB
208	20.43 KB	20.55 KB	20.68 KB	20.8 KB	20.93 KB	21.05 KB

Table 11: Network communication in the Shuffle Private Cards phase with m cards and n players in Kilobytes (KB).

Shuffle Private Cards The execution time and network communication for the Shuffle Private Cards phase with with m cards and n players are presented in Table 12 and Table 11, respectively. The execution time is presented as the sum of the local computation time required of each player and the network Round Trip Time necessary for delivering this phase's messages. Differently from the Shuffle Cards procedure, the players $\mathcal{P}_i$ only check the shuffle generated

by $\mathcal{P}_j$ and broadcast their signatures on the checkpoint witness, later verifying each other's signatures.

m	$\mathcal{P}_j$ (Owner)	$\mathcal{P}_i$ (Others)
52	59.01 ms + 1 RTT	6.52 ms + 1 RTT
104	149.6 ms + 1 RTT	11.64 ms + 1 RTT
208	294.03 ms + 1 RTT	23.3 ms + 1 RTT

Table 12: Execution time for the Shuffle Private Cards phase with m cards in milliseconds (ms) and Round Trip Time (RTT).

Open Public Card The execution time and network communication for the Open Public Card phase with n players (and opening one card) are presented in Table 13. In this phase, all players generate and broadcast their decryption shares for the ciphertext representing the card to be opened, verify the decryption shares provided by each other and, if all chekcs succeed, generate checkpoint witness signatures.

n:	2	4	6	8	10	12
Time:	0.89 ms + 1 RTT	1.06 ms + 1 RTT	1.5 ms + 1 RTT	1.94 ms + 1 RTT	2.51 ms + 1 RTT	2.82 ms + 1 RTT
Comm.:	0.57 KB	1.89 KB	3.98 KB	6.82 KB	10.42 KB	14.78 KB

Table 13: Open Public Card Phase execution time in milliseconds (ms) and network communication in Kilobytes (KB) for n players (and opening one card).

Open Private Card The execution time and network communication for the Open Private Card phase with n players (and opening one card) are presented in Table 14. Notice that, in this phase, only $\mathcal{P}_j$ (the drawer) receives and verifies the decryption shares from players $\mathcal{P}_i$, who only generate their own shares and checkpoint witness signatures.

n:	2	4	6	8	10	12
p_j (Drawer) Time:	0.8 ms + 1 RTT	1 ms + 1 RTT	1.3 ms + 1 RTT	1.91 ms + 1 RTT	2.17 ms + 1 RTT	2.6 ms + 1 RTT
$\mathcal{P}_i$ (Others) Time:	0.18 ms + 1 RTT	0.18 ms + 1 RTT	0.18 ms + 1 RTT	0.18 ms + 1 RTT	0.18 ms + 1 RTT	0.18 ms + 1 RTT
Comm.:	0.38 KB	1.52 KB	3.41 KB	6.06 KB	9.47 KB	13.64 KB

Table 14: Open Private Card Phase execution time in milliseconds (ms) and network communication in Kilobytes (KB) for n players (and opening one card).

Checkpoint Witnesses Checkpoint witnesses size for our implementation is presented in Table 15. As previously discussed, we consider the size of checkpoint witnesses in the worst case, where all cards are closed (which results in the largest representation of checkpoint witnesses).

Notice that, for the setting of a regular game of poker with 52 cards and 6 players, we obtain a worst case checkpoint witness of less than 4 KB, while for the larger setting of a game of Baccarat with 312 cards and 12 users we obtain a worst case checkpoint witness of less than 15 KB. In case the Recovery Phase is activated by any player who suspects misbehavior, players are also required to send to the smart contract the next messages to be generated in the protocol rounds after the latest checkpoint witness. Hence, the total on-chain storage requirement in the Recovery phase consists of the size of one checkpoint witness and the communication complexity of the protocol phase executed immediately after point for which the checkpoint witness was generated. The average on-chain storage for recovering from misbehavior in each phase can be obtained by adding that phase's communication complexity (as analyzed above) to the size of a checkpoint witness. In case one or more players disagree on the latest checkpoint witness first submitted to the smart contract, they may provide their own checkpoint witnesses, adding to the total on-chain storage requirement.

m \ n	2	4	6	8	10	12
52	3.61 KB	3.77 KB	3.92 KB	4.08 KB	4.23 KB	4.39 KB
104	7.06 KB	7.22 KB	7.38 KB	7.53 KB	7.69 KB	7.84 KB
208	13.97 KB	14.13 KB	14.28 KB	14.44 KB	14.59 KB	14.75 KB

Table 15: Checkpoint Witnesses on-chain storage size in Kilobytes (KB) for m cards and n players.

Acknowledgements

We would like to thank Vincent Hanquez for implementing Royale and helping us in the benchmark process.

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 *Lecture Notes in Computer Science*, pages 105–121, Christ Church, Barbados, March 7, 2014. Springer, Heidelberg, Germany.

[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, Berkeley, CA, USA, May 18–21, 2014. IEEE Computer Society Press.

[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 *Lecture Notes in Computer Science*, pages 370–383, Cirencester, UK, December 16–18, 2003. Springer, Heidelberg, Germany.

[4] Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology* –

EUROCRYPT 2012, volume 7237 of *Lecture Notes in Computer Science*, pages 263–280, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

[5] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press.

[6] Iddo Bentov and Ranjit Kumaresan. How to use bitcoin to design fair protocols. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 421–439, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany.

[7] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. To appear on Asiacrypt 2017. Available at `http://eprint.iacr.org/2017/875`.

[8] Vitalik Buterin. White paper. 2013. `https://github.com/ethereum/wiki/wiki/White-Paper`, Accessed on 5/12/2017.

[9] Jan Camenisch, Stephan Krenn, and Victor Shoup. A framework for practical universally composable zero-knowledge protocols. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 449–467, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

[10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, NV, USA, October 14–17, 2001. IEEE Computer Society Press.

[11] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004)*, page 219. IEEE Computer Society, 2004.

[12] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[13] 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: 18th International Conference on Theory and Practice of Public Key Cryptography*, volume 9020 of *Lecture Notes in Computer Science*, pages 495–515, Gaithersburg, MD, USA, March 30 – April 1, 2015. Springer, Heidelberg, Germany.

[14] 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.

[15] David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *Advances in Cryptology – CRYPTO'92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105, Santa Barbara, CA, USA, August 16–20, 1993. Springer, Heidelberg, Germany.

[16] Claude Crépeau. A secure poker protocol that minimizes the effect of player coalitions. In Hugh C. Williams, editor, *Advances in Cryptology – CRYPTO'85*, volume 218 of *Lecture Notes in Computer Science*, pages 73–86, Santa Barbara, CA, USA, August 18–22, 1986. Springer, Heidelberg, Germany.

[17] Claude Crépeau. A zero-knowledge poker protocol that achieves confidentiality of the players' strategy or how to achieve an electronic poker face. In Odlyzko [23], pages 239–247.

[18] 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`.

[19] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Odlyzko [23], pages 186–194.

[20] P. Golle. Dealing cards in poker games. In *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II*, volume 1, pages 506–511 Vol. 1, April 2005.

[21] Ranjit Kumaresan and Iddo Bentov. How to use bitcoin to incentivize correct computations. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14: 21st Conference on Computer and Communications Security*, pages 30–41, Scottsdale, AZ, USA, November 3–7, 2014. ACM Press.

[22] 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: 22nd Conference on Computer and Communications Security*, pages 195–206, Denver, CO, USA, October 12–16, 2015. ACM Press.

[23] Andrew M. Odlyzko, editor. *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[24] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398, Saragossa, Spain, May 12–16, 1996. Springer, Heidelberg, Germany.

[25] Christian Schindelhauer. A toolbox for mental card games. Technical report, University of Lübeck, 1998.

[26] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[27] 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.

[28] Adi Shamir, Ronald L Rivest, and Leonard M Adleman. Mental poker. In *The mathematical gardner*, pages 37–43. Springer, 1981.

[29] Tzer-jen Wei. Secure and practical constant round mental poker. *Information Sciences*, 273:352–386, 2014.

[30] Tzer-jen Wei and Lih-Chung Wang. A fast mental poker protocol. *Journal of Mathematical Cryptology*, 6(1):39–68, 2012.

[31] Wikipedia. Online Poker. `https://en.wikipedia.org/wiki/Online_poker`, 2017. [Online; accessed 29-August-2017].

[32] Bingsheng Zhang and Hong-Sheng Zhou. Brief announcement: Statement voting and liquid democracy. In Elad Michael Schiller and Alexander A. Schwarzmann, editors, *36th ACM Symposium Annual on Principles of Distributed Computing*, pages 359–361, Washington, DC, USA, July 25–27, 2017. Association for Computing Machinery.

[33] Bingsheng Zhang and Hong-Sheng Zhou. Digital liquid democracy: How to vote your delegation statement. Cryptology ePrint Archive, Report 2017/616, 2017. `http://eprint.iacr.org/2017/616`.

[34] 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.

[35] 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.