Kaleidoscope: An Efficient Poker Protocol with Payment Distribution and Penalty Enforcement

Bernardo David* Rafael Dowsley† Mario Larangeira*

Abstract

The research on secure poker protocols without trusted intermediaries has a long history that dates back to modern cryptography's infancy. Two main challenges towards bringing it into real-life are enforcing the distribution of the rewards, and penalizing misbehaving/aborting parties. Using recent advances on cryptocurrencies and blockchain technologies, Andrychowicz *et al.* (IEEE S&P 2014 and FC 2014 BITCOIN Workshop) were able to address those problems. Improving on these results, Kumaresan *et al.* (CCS 2015) and Bentov *et al.* (ASIACRYPT 2017) proposed specific purpose poker protocols that made significant progress towards meeting the real-world deployment requirements. However, their protocols still lack either efficiency or a formal security proof in a strong model. Specifically, the work of Kumaresan *et al.* relies on Bitcoin and simple contracts, but is not very efficient as it needs numerous interactions with the cryptocurrency network as well as a lot of collateral. Bentov *et al.* achieve further improvements by using stateful contracts and off-chain execution: they show a solution based on general multiparty computation that has a security proof in a strong model, but is also not very efficient. Alternatively, it proposes to use tailor-made poker protocols as a building block to improve the efficiency. However, a security proof is unfortunately still missing for the latter case: the security properties the tailor-made protocol would need to meet were not even specified, let alone proven to be met by a given protocol. Our solution closes this undesirable gap as it concurrently: (1) enforces the rewards' distribution; (2) enforces penalties on misbehaving parties; (3) has efficiency comparable to the tailor-made protocols; (4) has a security proof in a simulation-based model of security. Combining techniques from the above works, from tailor-made poker protocols and from efficient zero-knowledge proofs for shuffles, and performing optimizations, we obtain a solution that satisfies all four desired criteria and does not incur a big burden on the blockchain.

1 Introduction

Shamir, Rivest and Adleman, soon after their seminal work on the RSA cryptosystem, started exploring new ideas on cryptography inspired by everyday activities such as playing games. In particular, they started investigating how to play poker remotely [43]. A poker game, despite its apparent triviality, in fact, relates to a set of very interesting problems for the distributed setting. For example, shuffling a deck of cards in the presence of the players is very different from securely shuffling with remote parties: in the latter case every player needs to participate in the shuffling procedure; otherwise, security may not be assured at all for the participants.

*Tokyo Institute of Technology. Emails: bernardo@bmdavid.com, 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).

Mental Poker, Cryptography and the Gambling Market: Since its origins the research on mental poker and card games worked as a drive for the research in cryptography. Therefore it is not a surprise that we can find a myriad of works pursuing the development of this area within the cryptographic literature. The original work of Shamir *et al.* inspired a number of follow-ups, starting in the eighties with the works on the feasibility of playing mental games, e.g., [37, 15, 29, 4, 51, 25, 26, 16, 17].

In fact, the study of this problem led to concrete and seminal results that have shaped modern cryptography. Vulnerabilities in the poker protocol of Shamir *et al.* [43] that were pointed out by Lipton [37] triggered developments in the formalization of security models that culminated with the establishment of the notion of *probabilistic security* by Goldwasser and Micali [29] as the standard security goal for cryptosystems. Another example is the general purpose secure multiparty computation [28]. In other words, research in mental poker has motivated the development of the cryptography in a broader sense.

The first protocols for mental poker faced several limitations due to poor efficiency. However, they made it clear that a full protocol for card games needs secure sub-protocols for several steps: generating cards, blindly shuffling cards, revealing cards, checking the outcome of the game, and so on. In the following decades several works dealt with this important problem, e.g., [36, 40, 5, 53, 12, 31, 52, 45, 42, 50, 47, 46]. These works improved the efficiency of one or more of these sub-protocols.

Coincidentally, in economic terms, poker has become a strong trend from the early 2000's in what is known as the "Poker Boom" [49] and was described in prestigious economic venues [22]. Much of the strong interest in online gambling has its advent due to the appearance of online casinos. The rising popularity of online games can be verified by the peaks of players flocking in the poker rooms registered in websites like `http://www.fulltilt.com/` and `https://www.pokerstars.com/`. This phenomenon still holds despite legal restrictions imposed by new US legislation [19, 18]. Even with these setbacks players continue to play in websites based in other countries. For example, a report from the Financial Times [1] describes how UK firms filled the vacuum left by the US counterparts in the estimated 40 billion dollars global market of international online gambling (with one of the major online casino reporting 22 millions users and revenue of 2.5 billions dollars).

The current model of online gambling is based on trusted casinos, which are responsible for generating the randomness used to shuffle the cards and for enforcing the proper execution of the game. In contrast, a real world poker game requires almost no trust among the players, or between players and third parties like casinos. In the current model, a malicious casino or an attacker working for a casino can greatly influence the outcome of the game by manipulating the randomness used for shuffling or by leaking additional information to the players. And such cases have already happened (see Section "Integrity and Fairness" of [48] for more details). This state of affairs represents a clear disadvantage from online poker in comparison with a game played face-to-face. Techniques from mental poker can be used to overcome this problem and securely play poker online without the need for trusted casinos.

In the domain of cryptocurrencies, users and practitioners did not fail to notice this strong interest in online gambling and the opportunities that cryptocurrencies offer for online gambling [44]. In terms of the development of cryptography and economic importance, the research in the field of mental poker combined with cryptocurrencies, has the potential to not only give novel theoretical results, but also concrete fruits in the real world.

Challenges Preventing Deployment: Given that poker is most commonly played with money at stake, two central problems that were not addressed by the previously mentioned

secure poker protocols, but need to be solved for allowing deployment are protecting against aborts and ensuring that winners get rewarded properly.

The first problem consists in players who leave the game prematurely (i.e. abort the protocol execution) causing the protocol to freeze. Castellà-Roca *et al.* [13] investigated this scenario and proposed a protocol. However, we have identified a flaw in this protocol that effectively enables the adversary to artificially increase the probability that a given card (or set of cards) is drawn from the deck.

The problem of ensuring that a player actually gets a reward if it wins has only been tackled very recently after the advent of cryptocurrencies and blockchain technologies. Andrychowicz *et al.* [3, 2] addressed this problem with the help of Bitcoin and blockchains. They concurrently also dealt with the abort problem in a far more satisfactory way by imposing financial penalties on the aborting parties and using the collected money to compensate the remaining players. They followed the approach of using an unfair multiparty computation protocol along with many simple smart contracts and Bitcoin deposits to ensure that the rewards are distributed to players whenever the relevant conditions are fulfilled, and to enforce financial penalties on aborting/misbehaving parties. A similar solution was independently proposed by Bentov and Kumaresan in [8] and [34]. Kumaresan *et al.* [35] improved on this general strategy and employed it in designing a specific purpose protocol for card games (with a focus on poker). However, this protocol is not efficient enough for practical purposes: for instance, it requires a lot of interaction with the cryptocurrency network and of collateral (for a more detailed discussion see [9, Section 7]).

A significant improvement was obtained by Bentov *et al.* [9] by leveraging the power of stateful contracts to greatly improve the efficiency and solve some of the bottlenecks in the previous protocol. The main difference between the two approaches boils down to use of smart contracts. The protocol presented in [35] is based on simple claim-or-refund transactions, requiring $O(n^2)$ rounds of interaction with the cryptocurrency network and an amount of collateral linear in the number of messages exchanged during the protocol. In contrast, the protocol of Bentov *et al.* [9] requires only $O(1)$ rounds of interaction with the cryptocurrency network and an amount of collateral equal to the compensation the players would receive. The central idea for improving the performance and decreasing the amount of collateral needed is to use a single stateful contract that keep all the deposits and to execute the unfair protocol off-chain. After the initial deposits, the stateful contract is only involved in two situations: for the cash distribution, or if a problem happens.

Bentov *et al.* presented a general possibility result in which the unfair multiparty computation protocol is based on enhanced trapdoor permutations and provided an UC-style security proof. Additionally, they argued that their paradigm can also be instantiated with tailor-made poker protocols to improve the performance; however, they did not provided a security proof for this case nor described the properties the underlying poker protocol should satisfy. This security gap is discussed below.

Lack of Strong Security Proofs: Even though efficient solutions are known for different components of card games, most have not been formally proven secure in a strong security model. As the experience from decades of cryptographic research clearly shows, in order to avoid proposing/using insecure protocols, it is imperative to use strong security models to formally prove the protocols' security.

Out of the few protocols that have been suggested, it seems that only [35] and its follow-up work [9] present a more detailed security proof in a strong, simulation-based security model. We should, nonetheless, emphasize that in the case of the work in [9] only the general solution based

on enhanced trapdoor permutations enjoys such a strong security proof. While this general construction can be used to instantiate a poker protocol, it would not be as efficient as tailor-made poker protocols (it has inefficiencies in terms of computation and communication due to its generality). Bentov *et al.* [9, Section 7] argue that, instead of the general protocol, tailor-made poker protocols can be used as a building block and coupled with their techniques for dealing with aborts and cash distribution in order to obtain more efficient poker protocols. However a proof of security for this claim is not presented in their work, furthermore it is not even mentioned which security properties the tailor-made poker protocol would have to satisfy in order for the overall solution to be secure. They specifically mentioned as a potential candidate "the protocol of Wei and Wang [47, 46]". However, these citations refer to two different journal works appearing in different journals: Wei and Wang [47] that appeared in Journal of Mathematical Cryptology in 2012 and Wei [46] that appeared in Information Sciences in 2014. The work of [9] seems to consider them the same protocol, mentioning [47] during the description of their solution and then referring also to [46]. Unfortunately this state of the things leaves the reader confused about which protocol they consider to be adequate. In addition, the security models used in [47, 46] are not formally defined and seem to be rather weak (judging by the informal descriptions presented by the authors). One example is the "ideal game" in [47] (i.e., Game 8) which does not capture all the challenging difficulties involved in playing mental poker securely (a gap that we address by introducing the first detailed functionality for poker in the research literature in Section 3). Using such building blocks in a black-box way without a comprehensive definition of the security requirements and a matching security proof can potentially lead to security vulnerabilities that are "inhereted" from the building blocks, and to composition problems.

Concrete Flaws in Previous Protocols: Given the lack of formal security definitions and proofs in previous papers, it comes as no surprise that some proposals have security problems. It has been observed in [42] that the protocols of Zhao *et al.* [53, 52] are broken.

We have identified a flaw in the mechanism used for player dropout resistance (*i.e.* guaranteed output delivery) in the protocol proposed by Castellà-Roca *et al.* [13] that allows an adversary to manipulate the probability of a card being drawn from the deck. When a player leaves the game the protocol does not abort, it requires however a complete new deck to be shuffled for continuing the game. The players then proceed using the new deck and a veto mechanism that prevents the distributed cards from reappearing. The main problem is that the cards that were previously distributed to the player who dropped out are not vetoed, and thus are potentially reinserted into the game - allowing a potential attack vector for colluding parties. If one adversarial player has in his hand a card that can greatly increase the winning probability of another player, he is able to drop out of the game so that the card is reinserted into the new deck and potentially reappears - thus influencing the outcome. Note that this attack cannot happen in a face-to-face poker game.

Analyzing the protocol of Barnett and Smart [5], we have observed that the adversary can obtain a trapdoor that allows it to learn the order of all cards in the deck with probability 1, consequently learning exactly which cards are held by other players. This attack can be carried out by any adversary that corrupts at least one of the players during the key generation phase of the *Verifiable Threshold Masking Function* (VTMF) employed in that protocol. In this phase, each player $\mathcal{P}_i$ is required to generate a random secret-key share $sk_i \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts the group element $h_i = g^{sk_i}$, where g is a generator. An adversary corrupting a player $\mathcal{P}_j$ can choose a trapdoor $td \xleftarrow{\$} \mathbb{Z}_p$ and broadcast $h_j = \frac{g^{td}}{\sum_{i \neq j} h_i}$. The resulting public key for the VTMF will be computed as $pk = \sum_i h_i = \left(\sum_{i \neq j} h_i\right) \cdot \frac{g^{td}}{\sum_{i \neq j} h_i} = g^{td}$ with the corresponding secret key equal to

the trapdoor td known by the adversary, allowing it to reverse the masking operations done to each card and recover the order of the cards in the final shuffled deck. In Section 2.5, we present a key generation procedure for a threshold ElGamal cryptosystem that can be readily employed to solve this issue of [5].

General Requirements for a Useful Poker Protocol: The current state of the art is still unsatisfactory as there is no solution that meets all the following criteria that would be necessary in a deployment in a real world scenario in which money is at stake:

1. **Efficiency:** performance that is comparable to tailor-made poker protocols;
2. **Security:** a simulation-based, formal proof of security;
3. **Penalties:** avoiding aborts/misbehavior or penalizing the misbehaving players;
4. **Rewards:** securely distributing the rewards to the players.

The works that are closer to achieve these criteria are [35] and [9], which made fundamental progress towards providing viable solutions to satisfy conditions (3) and (4). Nevertheless, none of their solutions meet simultaneously conditions (1) and (2). The solutions in [35] as well as the general solution in [9] do not meet condition (1), while the solution in [9] using tailor-made protocol improves on condition (1) but does not address (2) as it lacks a security proof.

1.1 Our Contribution

We present our protocol, *Kaleidoscope*, named after the homonymous poker themed movie from the sixties [32]. Given the earlier discussion, our main goal in this work is to design a poker protocol that concurrently meets all four criteria above. In designing our solution we face two main challenges: 1. constructing an efficient off-chain protocol without sacrificing provable security guarantees as in previous tailor-made poker protocols, 2. reducing the amount of data stored in the blockchain, which is a highly constrained resource. In summary, our contributions are:

1. *First* full-fledged *simulation-based* security definition tailored for poker (Section 3);
2. First *fully-simulatable* poker protocol, which provably realizes our security definition (Section 4);
3. Improved concrete computational and communication complexities for off-chain card operations (around 10 times better than previous works) and reduced on-chain storage requirements for the penalties and rewards enforcement mechanism (estimated in Section 5).

As our goal is to provide a strong security guarantee, we first specify a poker functionality that encompasses the whole game execution, penalizes aborting parties and guarantees the distribution of the rewards. Such modeling of the whole poker game as an ideal functionality is, to the best of our knowledge, novel. Then we design a tailor-made protocol that provably realizes such functionality in a simulation-based security model.

Our protocol is designed with both off-chain and on-chain efficiency in mind. We focus on the case where players act honestly and the on-chain protocol execution is used as a last resort to recover from malicious actions. In this context, we design an off-chain protocol that is highly

efficient while providing *compact* witnesses to be posted to the blockchain for claiming rewards or enforcing penalties.

We design a highly optimized tailor-made poker protocol that meets criteria (1) and (2) inspired by the protocol of Barnett and Smart [5] and an efficient zero-knowledge proof of correctness for shuffle [6]. While we follow the general approach of [5], we make a series of subtle but significant modifications in order to both correct security issues and improve efficiency of several phases. We meet criteria (3) and (4) by building on top of the ingenious ideas in [35] and [9], optimizing their general rewards/penalties mechanism for the specific case of poker, for which we obtain compact witnesses of correct behavior.

1.2 Overview and Intuition of Our Protocol

Next we present a more detailed overview of our protocol. Due to the fact that it is not reasonable to assume that the majority of the players are honest in a poker game, the secure poker protocol will not be able to guarantee fairness. Instead, we follow the approach of imposing a financial penalty on the party that interrupts the correct execution of the protocol, and use this money to compensate the honest parties. A stateful contract is used to enforce these properties.

As it is highly desirable to decrease the burden on the blockchain as much as possible (thus improving the efficiency and decreasing the impact on other users), the execution of the protocol is performed mostly off-chain and the parties only go back on-chain for the cash distribution or if some problem happens. When the protocol goes back on-chain, the parties need to present witnesses to the stateful contract to validate the state of the game. It is important to decrease the size of these witnesses that need to be stored by the players, as well as the verification costs for the stateful contract. In this regard, a key characteristic of poker is that the future execution is independent from the past when conditioned on a few variables that keep track of the current status. Hence, if all participants sign these variables at a *checkpoint*, then this constitutes a *witness* witness that can be delivered to the stateful contract in order to prove the state of the game at this particular point. Therefore, at the checkpoints, the players can delete all other previous witnesses, saving space for the players and verification efforts for the stateful contract.

The general overview of the protocol is:

1. Initially the parties lock into the stateful contract functionality an amount of money equal to the sum of the collateral and the money that they will use for the bets. A few initialization procedures are also executed during this stage.

2. The parties then execute an unfair tailor-made poker protocol off-chain. We construct a highly optimized protocol building on the techniques of [5] along with an efficient zero-knowledge proof of correctness for shuffles from [6]. During this stage the players need to record a few witnesses that must be sent to the stateful contract in the case of problems that require its intervention. All messages are signed by the senders, and at some checkpoints a few variables that summarize the status of the game are signed by all players, constituting a *compact* witness of correct execution.

3. If the protocol finishes correctly off-chain, then the final payout amounts will have been signed by all players, and so the parties only come back on-chain for the cash distribution that is performed by the stateful contract.

4. If some problem happens and a player requests the intervention of the stateful contract, each party that does not want to get penalized handles their respective recorded witnesses to the stateful contract, which is then able to verify the latest status of the protocol

execution and continue the execution (on-chain) under its mediation. During the mediated execution, it penalizes any participant that does not follow the protocol rules or abort.

Note that on Step 2, the adopted technique is used in order to decrease the size of the witnesses that the players need to store after the checkpoint as well as to reduce the amount of on-chain verification that needs to be performed in case of intervention (thus reducing the burden on the blockchain, which affects all users of the cryptocurrency).

The safe deposit d that each of the n participants lock into the contract should be enough to pay the compensation amount q for all the other parties, i.e., $d \geq q(n-1)$. Obviously, the monetary compensation q should be related to the maximum possible bet amount m at each hand; otherwise the corrupted parties would have an incentive to abort the protocol if they notice that one hand will end up badly for them.

2 Preliminaries

In this section we settle notation and definitions used throughout the paper. $x \xleftarrow{\$} S$ denotes sampling uniformly at random an element x from a set S.

2.1 Security Model, Adversarial Model and Setup Assumptions

We prove our protocol secure in the real/ideal simulation paradigm with sequential composition. This is an intuitive paradigm that provides strong security guarantees for the protocols that are proven secure according to it. A real world and an ideal world are defined and compared. In the real world, the protocol π is executed with the parties, some of which are corrupted and controlled by the adversary $\mathcal{A}$. On the other hand, in the ideal world the protocol is replaced by an ideal functionality $\mathcal{F}$ and a simulator $\mathcal{S}$ interacts with it. The ideal functionality $\mathcal{F}$ describes the behavior that is expected from the protocol and acts as a trusted entity. A protocol π is said to securely realize the ideal functionality $\mathcal{F}$, if for every polynomial time adversary $\mathcal{A}$ in the real world, there is a polynomial time simulator $\mathcal{S}$ for the ideal world, such that the two worlds cannot be distinguished. In more detail, no probabilistic polynomial time distinguisher $\mathcal{D}$ can have non-negligible advantage in distinguishing the concatenation of the output of the honest parties and of the adversary $\mathcal{A}$ in the real world from the concatenation of the output of the honest parties (which come directly from $\mathcal{F}$) and of the simulator $\mathcal{S}$ in the ideal world. Protocols proven secure according to this paradigm can be sequentially composed. For more details about this security paradigm please check [11, 27].

We consider *malicious* adversaries that may deviate from the protocol in any arbitrary way. Moreover we consider the *static* case, where the adversary is only allowed to corrupt parties before protocol execution starts and parties remain corrupted (or not) throughout the execution. Our protocol uses the Random Oracle Model (ROM) [7].

2.2 Decision Diffie Hellman (DDH):

The DDH problem consists in deciding whether $c = ab$ or $c \xleftarrow{\$} \mathbb{Z}_p$ in a tuple (g, g^a, g^b, g^c) where g is a generator of a group $\mathbb{G}$ of order p, and $a, b \xleftarrow{\$} \mathbb{Z}_p$. The DDH assumption states that the DDH problem is hard for every probabilistic polynomial time distinguisher.

2.3 Digital Signatures

We will employ digital signatures with Existential Unforgeability under Adaptive Chosen Message Attacks (EUF-CMA) [30]. In general, a digital signature scheme is a tuple of three PPT algorithms $\mathsf{SIG} = (\mathsf{SIG.Gen}, \mathsf{SIG.Sign}, \mathsf{SIG.Vrf})$ such that:

- $\mathsf{SIG.Gen}(1^\lambda)$ takes in a security parameter and outputs a verification key $\mathsf{SIG}.vk$ and a signing key $\mathsf{SIG}.sk$.
- $\mathsf{SIG.Sign}_{\mathsf{SIG}.sk}(m)$ takes in a signing key $\mathsf{SIG}.sk$ and a message m, outputting a signature σ on message m under signing key $\mathsf{SIG}.sk$.
- $\mathsf{SIG.Vrf}_{\mathsf{SIG}.vk}(m, \sigma)$ takes in a verification key $\mathsf{SIG}.vk$, a message m and a signature σ, outputting 1 if the signature is valid and 0 otherwise.

2.4 Non-Interactive Zero-Knowledge Proofs for Discrete Logarithm Relations

In this section, we describe some non-interactive zero knowledge proofs (NIZK) for discrete logarithm relations constructed via the Fiat-Shamir heuristic that will be employed by our protocols.

Non-Interactice Zero-Knowledge Proof of Discrete Logarithm Equality (DLEQ): We will need a NIZK of knowledge of a value $\alpha \in \mathbb{Z}_p$ such that $x = g^\alpha$ and $y = h^\alpha$ given g, x, h, y. We denote this proof by $\mathsf{DLEQ}(g, x, h, y)$. Chaum and Pedersen [14] constructed a sigma protocol for this relation that works as follows:

1. The prover computes $a_1 = g^w$ and $a_2 = h^w$ where $w \xleftarrow{\$} \mathbb{Z}_p$ and sends a_1, a_2 to the verifier.
2. The verifier sends a challenge $e \xleftarrow{\$} \mathbb{Z}_p$ to the prover.
3. The prover sends a response $z = w - \alpha e$ to the verifier.
4. The verifier accepts the proof if $a_1 = g^z x^e$ and $a_2 = h^z y^e$ hold.

This sigma protocol can be transformed into a NIZK of knowledge of α in the random oracle model through the Fiat-Shamir heuristic [24, 39]. In this transformation, the challenge e is computed by the prover as $e = \mathsf{H}(x, y, a_1, a_2)$, where a_1, a_2 are computed as in the interactive protocol and $\mathsf{H}(\cdot)$ is a random oracle (that can be of course substituted by a cryptographic hash function). The proof consists of the challenge e along with response $z = w - \alpha e$ computed according to x, y and w. The verifier can check the proof by computing $a_1' = g^z x^e$ and $a_2' = h^z y^e$, and verifying that $\mathsf{H}(x, y, a_1', a_2') = e$. $\mathsf{DLEQ}(g, x, h, y)$ in our protocols references its Fiat-Shamir NIZK version, *i.e.* (e, z).

Non-Interactive Zero-Knowledge Proof of Discrete Logarithm Knowledge (DLOG): We will also need a simpler NIZK of knowledge of a value $\alpha \in \mathbb{Z}_p$ such that $x = g^\alpha$ given g, x. This proof is denoted by $\mathsf{DLOG}(g, x)$. Schnorr [41] constructed a sigma protocol for this relation that was later proven secure in [39]. Using the Fiat-Shamir heuristic [24, 39] to turn this sigma protocol into a NIZK, the prover computes a challenge $e = \mathsf{H}(g, x, a)$, where $a = g^w$ and $w \xleftarrow{\$} \mathbb{Z}_p$, and a response $z = w - \alpha e$. The final proof consists of (e, z) and a verifier can check that it is valid by first computing $a' = g^z x^e$ and then verifying that $c = \mathsf{H}(g, x, a')$. $\mathsf{DLOG}(g, x)$ in our protocols references its Fiat-Shamir NIZK version, *i.e.* (e, z).

2.5 (n, n)-Threshold ElGamal Cryptosystem

Threshold cryptosystems with (t, n)-threshold allow a group of n parties to jointly generate a public key that is then used to encrypt plaintext messages in such a way that they can only be recovered from the ciphertexts if at least t parties cooperate [20]. In our card deck generation procedure we employ a (n, n)-threshold version of the ElGamal cryptosystem [23] based on the constructions of [38, 21] with a verifiable decryption protocol as in the Verifiable Threshold Masking Functions (VTMF) of [5]. The final goal is to encode card information as Threshold ElGamal ciphertexts as in the VTMF based construction of [5]. However, we do not require the verifiable masking and verifiable re-masking (*rerandomization*) operations because the verification that these ciphertexts are correctly re-randomized is handled by the zero-knowledge proofs of correctness of a shuffle [6] presented in the next section. However, we do use the fact that this scheme is additively homomorphic (and thus rerandomizable) and a verifiable decryption procedure, where it is possible to verify that each user is providing a valid decryption share (as in a VTMF). We now present the (n, n)-Threshold ElGamal cryptosystem with verifiable decryption TEG and refer interested readers to [38, 21, 5] for a full discussion:

- **Key Generation** $\mathsf{TEG.Gen}(1^\lambda)$**:** Each party $\mathcal{P}_i$ generates a random secret-key share $\mathsf{TEG}.sk_i \xleftarrow{\$} \mathbb{Z}_p$ and broadcasts $h_i = g^{\mathsf{TEG}.sk_i}$ along with a proof $\mathsf{DLOG}(g, h_i)$ [1]. Once all n parties have broadcast their public key share h_i, each party $\mathcal{P}_i$ verifies the accompanying proofs $\mathsf{DLOG}(g, h_j)$ (aborting if invalid) and then saves all h_j, for $i \neq j$, reconstructing the public key by computing $\mathsf{TEG}.pk = h = \prod_{i=1}^{n} h_i = g^{\sum_{i=1}^{n} \mathsf{TEG}.sk_i}$.

- **Encryption** $\mathsf{TEG.Enc}_{\mathsf{TEG}.pk}(m, r)$**:** The encryption of a message $m \in \mathbb{G}$ under a public-key $\mathsf{TEG}.pk$ with randomness $r \in \mathbb{Z}_p$ is carried out as a regular ElGamal encryption. Namely, a ciphertext $c = (c_1 = g^r, c_2 = h^r m)$ is generated.

- **Re-Randomization** $\mathsf{TEG.ReRand}(c, r')$**:** A ciphertext $c = (c_1, c_2)$ is re-randomized with fresh randomness r' by computing $c' = (g^{r'} c_1, h^{r'} c_2)$.

- **Verifiable Decryption** $\mathsf{TEG.Dec}_{\mathsf{TEG}.sk_1,\ldots,\mathsf{TEG}.sk_n}(c)$**:** Parse $c = (c_1, c_2)$. Each party $\mathcal{P}_i$ broadcast a decryption share $d_i = c_1^{\mathsf{TEG}.sk_i}$ and a proof $\mathsf{DLEQ}(g, h_i, c_1, d_i)$ showing that they have correctly used their secret-key share $\mathsf{TEG}.sk_i$. Once all n parties have broadcast their decryption share d_i, each party $\mathcal{P}_i$ checks that the $\mathsf{DLEQ}(g, h_j, c_1, d_j)$ proofs are correct for all $i \neq j$ (aborting otherwise) and retrieves the message by computing

$$\frac{c_2}{\prod_{i=1}^{n} d_i} = \frac{c_2}{c_1^{\sum_{i=1}^{n} \mathsf{TEG}.sk_i}} = \frac{m \cdot \mathsf{TEG}.pk^r}{g^{r \sum_{i=1}^{n} \mathsf{TEG}.sk_i}} = \frac{m\left(g^{\sum_{i=1}^{n} \mathsf{TEG}.sk_i}\right)^r}{g^{r \sum_{i=1}^{n} \mathsf{TEG}.sk_i}} = m.$$

2.6 Zero-Knowledge Proofs of Correctness of a Shuffle

A central component of our protocol is a zero-knowledge proof that an ordered set of ElGamal ciphertexts has been obtained by re-randomizing each ciphertext and permuting the resulting ciphertexts in a previous ordered set (an operation called a *Shuffle*). Formally, we want to prove knowledge of a permutation $\pi \in \Sigma_N$ and randomness $\mathbf{r} = (r_1, \ldots, r_N)$ such that for the vectors of ciphertexts $\mathbf{c} = (c_1, \ldots, c_N)$ and $\mathbf{c'} = (c'_1, \ldots, c'_N)$ we have $c'_i = \mathsf{TEG.ReRand}(c_{\pi(i)}, r_i)$. An efficient zero-knowledge argument for correctness of this kind of shuffle has been proposed in [6]

[1] This zero-knowledge proof of the knowledge of the exponent solves the issue in [5] that was pointed out in the introduction.

and it can be turned into the required zero-knowledge proof through the Fiat-Shamir heuristic [24, 39]. We denote this NIZK by $\mathsf{ZKSH}(\pi, \mathbf{r}, \mathbf{c}, \mathbf{c}')$ and refer interested readers to for details on its construction and proof.

The zero-knowledge argument of [6] proves correctness of a shuffle of ElGamal ciphertexts arranged in a $k \times l$ matrix. The resulting Fiat-Shamir NIZK requires $11k$ elements of the underlying group $\mathbb{G}$ and $5l$ elements of $\mathbb{Z}_p$. The computational complexity is of $2\log(k)kl$ exponentiations for the prover and $4kl$ exponentiations for the verifier. In our scenario, we are interested in proving the correctness of a shuffle of 52 ciphertexts. We choose the parameters $k = 4$ and $l = 13$, which give us 208 exponentiations for the prover and 208 exponentiations for the verifier, with a proof size of 44 elements of $\mathbb{G}$ and 65 elements of $\mathbb{Z}_p$.

The scheme of [6] requires a CRS containing the public-key used for generating the ciphertexts that are shuffled and parameters for a generalized Pedersen commitment (that takes as input a message consisting of multiple elements of $\mathbb{Z}_p$). In our protocols, the public key is jointly generated by the parties (as described above). Notice that the public parameters for the generalized Pedersen commitment used in the scheme of [6] are basically $n+1$ random group generators $G_1, \ldots, G_n, H$ (such that the discrete logarithm of each generator in relation to the others is unknown). Thus, each of these generators can be generated by exactly the same procedure used for distributed key generation in the threshold ElGamal cryptosystem presented above. In the simulation, the simulator can extract the witness of the DLOG NIZKs of discrete logarithm knowledge provided by each of the parties in order to learn the trapdoor needed for simulating the ZKSH NIZKs of correctness of a shuffle. Concretely, each party $\mathcal{P}_i$ generates each of the generators needed for the generalized Pedersen commitment by sampling a random $r_i \xleftarrow{\$} \mathbb{Z}_p$ and broadcasting $h_i = g^{r_i}$ along with a proof $\mathsf{DLOG}(g, h_i)$. In the simulation, first r_i is extracted from $\mathsf{DLOG}(g, h_i)$ (with the help of simulator of the DLOG NIZK), which gives the simlator of the ZKSH NIZK the trapdoor for the generalized Pedersen commitment that it needs.

3 Game Formalization

The most widely played poker variant is the Texas Hold'em, thus, before presenting our formalization, we introduce a brief overview of this variant.

Game Overview. Initially each player receives two covered cards and is free to look his own cards. Additionally five covered cards are placed in the center of the table. These are named *community cards* and are revealed during the subsequent rounds of the hand. The hand is played in at most 4 betting rounds and the goal is to obtain the highest-ranked hand among all players that play until the end. Each player forms his hand by picking any combination of five cards from his own cards and the community cards. The winner takes the bets, which is commonly referred as the *pot*. After the first betting round, three community cards are revealed, then another one is revealed after the second betting round, and finally the last covered community card is opened after the third round. Afterwards the players perform one last betting round before revealing their hands in order to determine the winner of the hand and distribute the money.

Betting Rounds. In each betting round the players proceed in turns and have some actions available:

- **check**: the player does not add any amount to the pot. This action is only allowed if no player has placed any bet before in the round.

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

The functionality is executed with n players with identities $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$ and is parametrized by the small sb and big bb blind bets amounts, the initial stake t, the maximum bet m per hand, the security deposit d and of the compensation amount q. There are c corrupted parties that are controlled by $\mathcal{S}$. Whenever a message is sent to $\mathcal{S}$ for confirmation or action selection, $\mathcal{S}$ should answer, but can always answers ABORT, in which case the compensation procedure is executed; this option will not be explicitly mentioned in the functionality description henceforth.

Players Check-in: Wait to receive a message $(\text{CHECKIN}, \mathsf{coins}(d+t))$ from each player. Announce the check-ins to the other players. Allow the players to dropout and reclaim their coins if a player fails to check-in ($\mathcal{S}$ is consulted for approval). Once all check-ins are done, order the players by picking a random permutation and denote the ordered sequence of players by $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$. Send the order to all players. This order is used in a circular way to update the roles of small and big blinds after each hand (among the players that are still in the game), and within each hand to proceed with the actions (among the players that are still active in the hand). Let psb (resp. pbb) denote the index of the player who is the small (resp. the big) blind, and initialize $psb = 1$ and $pbb = 2$. Initialize the vector that tracks the balance of the n players as $\mathsf{balance} = (t, \ldots, t)$ and initialize the vector that tracks the bets of the n players as $\mathsf{bets} = (0, \ldots, 0)$. Keep track of the active players in the game, initially marking them all active.

Player Check-out: Send $(\text{CHECKOUT}, i)$ to $\mathcal{S}$. If $\mathcal{S}$ answers $(\text{CHECKOUT}, i)$, mark $\mathcal{P}_i$ as inactive in the game, send $(\text{PAYOUT}, \mathsf{coins}(d + \mathsf{balance}[i]))$ to $\mathcal{P}_i$ and ignore future messages from $\mathcal{P}_i$. If no active player is left, stop the execution.

Compensation: For each honest player $\mathcal{P}_i$ who has not checked-out, send $(\text{COMPENSATION}, \mathsf{coins}(d + q + \mathsf{balance}[i] + \mathsf{bets}[i]))$ to him. Let ℓ denote the amount of remaining locked coins. Send $(\text{REMAINING}, \mathsf{coins}(\ell))$ to $\mathcal{S}$ and stop the execution.

Hand Execution: Initialize the set of hand's active players as the game's active players. Keep track of the chronological order of folds. Send SHUFFLE? to $\mathcal{S}$. If $\mathcal{S}$ answers SHUFFLE, shuffle a deck of cards $\mathcal{D}$ and draw two private cards $\mathsf{pc}_{i,1}$ and $\mathsf{pc}_{i,2}$ for each active player $\mathcal{P}_i$ as well as the community cards $\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3, \mathsf{cc}_4, \mathsf{cc}_5$. Proceed as follows:

- Send (PAYSB, psb) to $\mathcal{P}_{psb}$. If $\mathcal{P}_{psb}$ answers PAYSB, then decrease $\mathsf{balance}[psb]$ by sb, increase $\mathsf{bets}[psb]$ by sb and broadcast (SBPAID, psb) to the players.
- Send (PAYBB, pbb) to $\mathcal{P}_{pbb}$. If $\mathcal{P}_{pbb}$ answers PAYBB, then decrease $\mathsf{balance}[pbb]$ by bb, increase $\mathsf{bets}[pbb]$ by bb and broadcast (BBPAID, psb) to the players.
- Send HAND together with the private cards of the corrupted players to $\mathcal{S}$. If $\mathcal{S}$ answers HAND, send $(\text{HAND}, \mathsf{pc}_{i,1}, \mathsf{pc}_{i,2})$ to each $\mathcal{P}_i$.
- Execute a round of bets starting with the closest active successor of pbb.
- Send $(\text{FLOP}, \mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3)$ to $\mathcal{S}$. If $\mathcal{S}$ answers FLOP, send $(\text{FLOP}, \mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3)$ to the players.
- Execute a round of bets starting with the closest active successor of $psb - 1$.
- Send $(\text{TURN}, \mathsf{cc}_4)$ to $\mathcal{S}$. If $\mathcal{S}$ answers TURN, send $(\text{TURN}, \mathsf{cc}_4)$ to the players.
- Execute a round of bets starting with the closest active successor of $psb - 1$.
- Send $(\text{RIVER}, \mathsf{cc}_5)$ to $\mathcal{S}$. If $\mathcal{S}$ answers RIVER, send $(\text{RIVER}, \mathsf{cc}_5)$ to the players.
- Execute a round of bets starting with the closest active successor of $psb - 1$.
- Run the showdown starting with the last player who increased the bet in the last round, if there is one; otherwise, the closest active successor of $psb - 1$.

Figure 1: Functionality for the game of poker.

Betting round: Initialize $ba = max_i \mathsf{bets}[i]$. Proceed in a circular way using the ordered sequence of the hand's active players until either there is only one player who has not folded, in which case the pot distribution is executed immediately, or all the following conditions hold: (1) all players have had a chance to act; (2) for each player $\mathcal{P}_i$ who haven't folded or performed an all-in, $\mathsf{bets}[i] = w$ for some fixed value w; (3) for any $\mathcal{P}_j$ who have performed an all-in it holds that $w \geq \mathsf{bets}[j]$. In the turn of $\mathcal{P}_i$, a message (BET, ba) is sent to him and he can answer with one of the following actions:

- FOLD: He is marked inactive in the hand and (FOLD, i) is sent to the players.
- CALL: Check if $\mathsf{balance}[i] > ba - \mathsf{bets}[i]$. If it is not, run the compensation procedure. Otherwise, decrease $\mathsf{balance}[i]$ by $ba - \mathsf{bets}[i]$, set $\mathsf{bets}[i] = ba$ and send (CALL, i) to the players.
- (RAISE, r): Check if $\mathsf{balance}[i] > r - \mathsf{bets}[i]$ and $ba < r \leq m$. If not, run the compensation procedure. Otherwise, decrease $\mathsf{balance}[i]$ by $r - \mathsf{bets}[i]$, set $\mathsf{bets}[i]$ and ba to r, and send (RAISE, i, r) to the players.
- ALL-IN: Check if $r = \mathsf{balance}[i] + \mathsf{bets}[i] \leq m$. If not, run the compensation procedure. Set $\mathsf{bets}[i] = r$ and $\mathsf{balance}[i] = 0$. If $r > ba$, set $ba = r$. $\mathcal{P}_i$ will not perform any further action during the betting rounds of this hand. Send (ALL-IN, i) to the players.
- CHECK: If it's the hand's first betting round or anyone has bet in the round, run the compensation procedure. Otherwise, send (CHECK, i) to the players.

Showdown: The showdown performs one round among the active players and then call the pot distribution procedure. In the turn of $\mathcal{P}_i$, a message OPEN is sent to him and he can answer with one of the following actions:

- OPEN: Send $(\text{OPEN}, i, \mathsf{pc}_{i,1}, \mathsf{pc}_{i,2})$ to $\mathcal{S}$. If $\mathcal{S}$ answers (OPEN, i), then send $(\text{OPEN}, i, \mathsf{pc}_{i,1}, \mathsf{pc}_{i,2})$ to the players.
- MUCK: He is marked inactive in the hand and (MUCK, i) is sent to the players.

Pot(s) Distribution: Using bets, the opened hands, the chronological order of folds/mucks and the standard rules of Texas Hold'em poker compute the vector pot so that $\mathsf{pot}[i]$ determines the amount of money that $\mathcal{P}_i$ will receive from the pot(s). Send $(\text{DISTRIBUTE}, \mathsf{pot})$ to $\mathcal{S}$. If $\mathcal{S}$ answers DISTRIBUTE, then increase each $\mathsf{balance}[i]$ by $\mathsf{pot}[i]$ and set $\mathsf{bets} = (0, \ldots, 0)$. For each $\mathcal{P}_i$ that have $\mathsf{balance}[i] = 0$, perform his check-out. For every other $\mathcal{P}_i$, send the message CONTINUE?. If $\mathcal{P}_i$ answers CONTINUE, continue. If $\mathcal{P}_i$ answers CHECKOUT, perform his check-out. Update the roles of small and big blinds, and start a new hand.

Figure 2: Functionality for the game of poker (continuation).

- **call**: the player just matches the amount already bet.
- **fold**: the player does not add any amount to the pot and is out of the hand.
- **raise**: the player raises the amount of the bet by adding more money to the pot than what is necessary to continue on the hand.
- **all-in**: the player bets all the money that he has available. He does not need to perform any additional action during the betting rounds of this hand.

A betting round finishes when: (1) all players have had a chance to act and (2) all players who haven't folded or performed an all-in have bet the same amount of money for the round and this amount is at least equal to the amount bet by each player who performed an all-in. Note that whenever a player plays a **raise** action, all the other players (except all-ins) which had already placed their actions should review its actions in order to (at least) match the same amount added to the pot, **fold** or **all-in**. The players disclose their actions publicly, and in order. The order and the publicity of the actions are important for the dynamics of the game, because the other players watching the actions would base their own next actions on what they observe during the rounds.

Blind bets and Game Order. Before any cards are dealt, covered and community cards, the two first players according to the game order, are requested to place bets, usually called *small* and *big blinds*. Naturally they are called *blind* because the players did not received any cards yet. Moreover, the roles of small and big blinds are rotated on every hand, therefore all the participants will eventually be requested to place the blind bets.

All-in and Side Pots. A player always have the possibility of betting all his remaining money, in which case he is allowed to continue in the hand until the end. However, in case that other players continue betting and increasing the pot, a side pot will be created for them. An all-in player only participates in the pots in which the other players are putting money to match the money that he bet.

Safe Deposits and Limits on Bets. Our solution is based on the principle of imposing financial penalties on misbehaving/aborting parties and then using the collected money to compensate the remaining parties. The safe deposit d is required from each participant, and in case of problems each other party receives a compensation amount q. Clearly, if there are n players, then it should hold that $d \geq q(n-1)$. In this setting, a limit on the amount of bets m at each hand should be imposed; otherwise, if the amount of bets grow too big, a malicious player has no incentive to behave properly anymore. Therefore, it is clear that the monetary compensation q should be bigger than the maximum possible bet amount m at each hand. Note that it is possible to set m so that all the betting money available can be bet in a single hand, but in this case the security deposits would be bigger. Studying the optimal way of setting these quantities in order to encourage a proper behavior of rational players is out of the scope of this paper.

The Poker Functionality. We formalize the earlier game in the ideal functionality $\mathcal{F}_{\mathsf{poker}}$, which is described in Figures 1 and 2.

4 Poker Protocol

In building our protocol, we depart from the construction of [5], which represents cards as ciphertexts of a threshold ElGamal cryptosystem. In this scheme, first the parties run a distributed key generation algorithm to obtain the public-key (while each holds a share of the secret-key). Next, they start a shuffling procedure that involves rerandomizing ciphertexts that encrypt the numbers assigned to each card (1 to 52) and shuffling the ciphertexts. The parties also provide to each other proofs that the shuffling procedure was conducted correctly, which is done via a cut-and-choose technique. When cards are intended to be revealed publicly, each party broadcasts a decryption share along with a zero-knowledge proof showing that it was computed correctly. If a covered card is to be given to one specific party, each party sends their decryption shares and proofs directly to that party through a private channel.

In order to improve the efficiency of the shuffling phase, we substitute the cut-and-choose technique used for attesting correctness of the shuffling operations by a zero-knowledge proof of a shuffle introduced in [6] and discussed in Section 2.6. This proof is compatible with ElGamal ciphertexts and achieves the same security level of the cut-and-choose technique employed by [5] with only a fraction of the computational and communication complexities. The procedures for distributing private covered cards and for publicly opening a covered card remain the same.

The main new feature of our protocols is a mechanism for detecting and (financially) punishing cheaters without requiring the whole protocol to the executed on chain. This mechanism

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

The functionality is executed with n players with identities $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$. It is parametrized by the small sb and big bb blind bets amount, the initial stake t, the maximum bet m per hand, the security deposit d, the compensation amount q, a protocol verification mechanism pv and a timeout limit τ.

Players Check-in: Wait to receive from each player with id_i $(\text{CHECKIN}, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i, h_i, \mathsf{DLOG}(g, h_i))$ 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 $(\text{CHECKEDIN}, \mathsf{id}_i, \mathsf{SIG}.vk_i, h_i, \mathsf{DLOG}(g, h_i))$ to all players. Allow the players to dropout and reclaim their coins if a player fails to check-in within the timeout limit τ. Once all check-ins are done, order the players by picking a random permutation and announce the ordered sequence of players by $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ to them. Mark all players as active.

Player Check-out: Upon receiving $(\text{CHECKOUT}, \mathsf{active}, \mathsf{balance}, \sigma)$ from $\mathcal{P}_i$, verify that σ contains valid signatures by all active players on active and $\mathsf{balance}$ and that $\mathsf{active}[i] = 0$. If everything is correct, for $w = \mathsf{balance}[i] + d$, send $(\text{PAYOUT}, \mathsf{coins}(w))$ to $\mathcal{P}_i$ and mark him as inactive. Send $(\text{CHECKEDOUT}, i, w)$ to the other players.

Recovery: Upon receiving a recovery request $(\text{REPORT}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$ from $\mathcal{P}_i$ containing some checkpoint witnesses and current phase witnesses, send to each $\mathcal{P}_j \neq \mathcal{P}_i$ $(\text{REQUEST}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$. Upon getting $(\text{RESPONSE}, \mathcal{P}_j, \mathsf{Checkpoint}_j, \mathsf{CurrPhase}_j)$ from some player $\mathcal{P}_j$ with checkpoint and phase witnesses (which are not necessarily relative to the same checkpoint as received from other players), forward the witnesses to the other parties. Upon getting replies from all players or reaching the timeout limit τ, determine the current phase by verifying the most recent checkpoint that has valid witnesses. Verify the last valid point of the protocol execution using the current phase witnesses and pv. If there exists some $\mathcal{P}_i$ who sent misbehaving messages (together with a signature) in the current phase, then for each $\mathcal{P}_j \neq \mathcal{P}_i$ who has not checked-out, send $(\text{COMPENSATION}, \mathsf{coins}(d + q + \mathsf{balance}[j] + \mathsf{bets}[j]))$ to him. Send any leftover coins after the compensation for $\mathcal{P}_i$ and halt. Otherwise, mediate the execution of the protocol until the next checkpoint. This is done by using $(\text{NXT-STP}, \mathsf{phase}, \mathsf{round})$ to request an action from the next party that is supposed to act and using pv to verify the answer $(\text{NXT-STP-RSP}, \mathsf{msg}_{\mathsf{phase},\mathsf{round}})$. All messages 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 $(\text{RECOVERED}, \mathsf{phase}, \mathsf{Checkpoint})$ to the parties once the next checkpoint is reached.

Figure 3: The stateful contract functionality that is used in the poker protocol.

requires that the parties first make a deposit of a number of coins used as "collateral", *i.e.* they lose these coins if they are detected as cheaters or abort. The protocol execution has a series of *checkpoints* where parties cooperate to generate a witness that the protocol has been executed correctly up to that point. This witness is basically a signature by all parties agreeing on the current state of the execution. If at any point a protocol malfunction occurs (a party either does not receive a message or receives a invalid message), the party that detected the malfunction posts a complaint to the blockchain along with the last checkpoint witness and the protocol messages generated after that checkpoint. All the other parties are required to do the same or face punishment otherwise. These witnesses are used to verify the current state of the protocol and then the execution continues in the blockchain until the next checkpoint. Any misbehavior or abort in this on-chain execution is punished financially. After the protocol execution reaches the next checkpoint and the parties obtain the corresponding witnesses, the protocol is again

executed off-chain.

Our poker protocol π_{Poker} interacts with a stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$, described in Figure 3, that models blockchain transactions used to keep collateral deposits and enforce punishment of players who misbehave, as well as ensuring that winners get their rewards. Protocol π_{Poker} is described in Figures 4, 5, 6 and 7, which describe its several steps.

Implementation of $\mathcal{F}_{\mathsf{SC}}$. It is important to emphasize that the $\mathcal{F}_{\mathsf{SC}}$ functionality can be easily implemented via smart contracts over a blockchain. More formally, using a public available *ledger*. Moreover, our construction (for protocol π_{Poker}) requires only simple operations, *i.e.*, verification of signatures and discrete logarithm operations over cyclic groups. The regular operation of our protocol is performed entirely off-chain, without intervention of the contract. However in the event that any problem happen or in the case that any participant in the game claim problems in the execution, any player can publish their agreed status of the game in the chain, via short witnesses (to be detailed in the protocol description). This approach reduces the information stored in the blockchain, and is an improvement in terms of efficiency in comparison to previously suggested protocols.

Protocol π_{Poker}

Protocol π_{Poker} is executed by n players with identities $(\mathsf{id}_1, \ldots, \mathsf{id}_n)$ interacting with the stateful contract functionality $\mathcal{F}_{\mathsf{SC}}$, and is parametrized by the small sb and big bb blind bets amount, the initial stake t, the maximum bet m per hand, the security deposit d and a timeout limit τ. We assume that the parties agree on a generator g of a group $\mathbb{G}$ of order p for the (n,n)-Threshold ElGamal cryptosystem TEG and also on a $\mathsf{EUF\text{-}CMA}$ secure digital signature scheme SIG. Moreover, a *nonce* unique to each protocol execution and protocol round (*e.g.* a hash of the public protocol transcript up to the current round) is implicitly attached to every signed message to avoid replay attacks.

Recovery Triggers: Whenever a signature or NIZK proof is received, its validity is tested. If the test fails, the party proceeds to the recovery phase. The same happens if a party does not receive an expected message until a timeout limit τ. These triggers will be omitted henceforth.

Players Check-in: For $i = 1, \ldots, n$, the party with id_i proceeds as follows:

1. generates the keys of the signature scheme $(\mathsf{SIG}.vk_i, \mathsf{SIG}.sk_i) \xleftarrow{\$} \mathsf{SIG.Gen}(1^\lambda)$.
2. generates TEG's key shares by sampling $\mathsf{TEG}.sk_i \xleftarrow{\$} \mathbb{Z}_p$, setting $h_i = g^{\mathsf{TEG}.sk_i}$ and generating a proof $\mathsf{DLOG}(g, h_i)$.
3. sends $(\text{CHECKIN}, \mathsf{coins}(d+t), \mathsf{SIG}.vk_i, h_i, \mathsf{DLOG}(g, h_i))$ to $\mathcal{F}_{\mathsf{SC}}$ and waits until getting from $\mathcal{F}_{\mathsf{SC}}$ the check-in confirmation $(\text{CHECKEDIN}, \mathsf{id}_j, \mathsf{SIG}.vk_j, h_j, \mathsf{DLOG}(g, h_j))$ of each player and the parties' order $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ that is used henceforth in the protocol. If not received until the timeout limit τ, contact $\mathcal{F}_{\mathsf{SC}}$ to dropout and reclaim the deposited coins.
4. verifies each $\mathsf{DLOG}(g, h_j)$ for $j \neq i$, reconstructs the initial public key $\mathsf{TEG}.pk = \prod_{j=1}^{n} h_j$, record all h_j, and initializes a vector $\mathsf{balance} = (t, \ldots, t)$, a vector $\mathsf{bets} = (0, \ldots, 0)$, a counter $psb = 1$ and a counter $pbb = 2$.

Player Check-out: If $\mathcal{P}_i$ was marked as checking out in the pot distribution phase, it sends a message $(\text{CHECKOUT}, \mathsf{active}, \mathsf{balance}, \sigma)$ to $\mathcal{F}_{\mathsf{SC}}$, where σ contains all signatures on active and $\mathsf{balance}$, waits for confirmation from $\mathcal{F}_{\mathsf{SC}}$ and stops execution.

Figure 4: Protocol π_{Poker}.

Hand Execution - Shuffle: As the first step in executing a hand, the parties generate a randomly shuffled deck of closed cards $\mathsf{c}_1, \ldots, \mathsf{c}_{52}$. For $i = 1, \ldots, n$, $\mathcal{P}_i$ proceeds as follows (w.l.o.g. we assume all parties are active, the adaptation to the other cases is the straightforward one):

1. If $\mathcal{P}_i = \mathcal{P}_1$, it sets $\mathbf{c}^0 = (\mathsf{c}_1^0, \ldots, \mathsf{c}_{52}^0)$ where $\mathsf{c}_j^0 = \mathsf{TEG.Enc}_{\mathsf{TEG}.pk}(j, 1)$ [a]. Otherwise, $\mathcal{P}_i$ considers the cards $\mathbf{c}^{i-1} = (\mathsf{c}_1^{i-1}, \ldots, \mathsf{c}_{52}^{i-1})$ received from $\mathcal{P}_{i-1}$.
2. $\mathcal{P}_i$ samples uniformly at random a permutation $\pi \in \Sigma_{52}$ and $\mathbf{r} = (r_1, \ldots, r_{52})$ where $r_j \xleftarrow{\$} \mathbb{Z}_p$, and sets $\mathsf{c}_j^i = \mathsf{TEG.ReRand}_{\mathsf{TEG}.pk}(\mathsf{c}_{\pi(j)}^{i-1}, r_j)$, obtaining a new set $\mathbf{c}^i = (\mathsf{c}_1^i, \ldots, \mathsf{c}_{52}^i)$. Notice that this new set of ciphertexts representing cards simply contains rerandomized versions of the previous ciphertexts in a random order.
3. $\mathcal{P}_i$ generates a zero-knowledge proof of correctness of shuffle $\mathsf{ZKSH}(\pi, \mathbf{r}, \mathbf{c}^{i-1}, \mathbf{c}^i)$ and broadcasts it with the shuffled deck $\mathbf{c}^i$. All other parties verify this zero-knowledge proof.

After all parties have participated in the shuffling procedure, the shuffled deck for the current hand is set to be $\mathcal{D} = \mathbf{c}^n$. All parties sign it by computing $\sigma_{\mathcal{D}}^i = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk}(\texttt{DECK} - \texttt{READY}, \mathcal{D})$, broadcasts $\sigma_{\mathcal{D}}^i$ and verifies all signatures. *Checkpoint Witness:* The previous checkpoint witness concatenated with the deck $\mathcal{D}$ and corresponding signatures $\sigma_{\mathcal{D}}^i$.

Hand Execution - Blinds: After the shuffle is done, all parties wait for the small blind, *i.e.* for $\mathcal{P}_{psb}$ to broadcast a signature $\sigma_{sb}^{psb} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_{psb}}(\texttt{SB})$ as well as signatures on vectors $\mathsf{balance}$ and bets, where $\mathsf{balance}[psb]$ is decreased by sb coins, $\mathsf{bets}[psb]$ is increased by sb coins, while all other coordinates remain the same. Upon receiving the signatures, each party $\mathcal{P}_i$ broadcasts a signature $\sigma_{sb}^i = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\texttt{SB})$ as well as signatures on $\mathsf{balance}$ and bets. All signatures are verified. Proceed analogously for the big blind. *Checkpoint Witness:* The previous checkpoint witness with the updated $\mathsf{balance}$ and bets (and signatures on them) concatenated with all signatures σ_{sb}^i and σ_{bb}^i.

Hand Execution - Drawing Cards and Private Cards Distribution: Two private cards $\mathsf{pc}_{i,1}, \mathsf{pc}_{i,2}$ for each active party $\mathcal{P}_i$ as well as the community cards $\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3, \mathsf{cc}_4, \mathsf{cc}_5$ are drawn from $\mathcal{D}$ according to the rules of poker. For $i = 1, \ldots, n$, $\mathcal{P}_i$ proceeds as follows to open cards $\mathsf{pc}_{j,1}, \mathsf{pc}_{j,2}$ towards $\mathcal{P}_j$ for $j = 1, \ldots, n$ and to obtain its own private cards (here all parties act in parallel):

1. $\mathcal{P}_i$ computes its decryption shares for $\mathsf{pc}_{j,1}, \mathsf{pc}_{j,2}$ by parsing $\mathsf{pc}_{j,k}$ as $(c_{j,k,1}, c_{j,k,2})$ and computing $d_{j,k,i} = c_{j,k,1}^{\mathsf{TEG}.sk_i}$ and a NIZK $\mathsf{DLEQ}(g, h_i, c_{j,k,1}, d_{j,k,i})$ for $k \in \{1, 2\}$. $\mathcal{P}_i$ sends the decryption shares $d_{j,1,i}, d_{j,2,i}$ along with their corresponding proofs to $\mathcal{P}_j$ through a private channel.
2. Once it has received all $d_{i,1,j}, d_{i,2,j}$ and corresponding DLEQ proofs from the other parties, $\mathcal{P}_i$ checks that the proofs are valid. Finally, $\mathcal{P}_i$ learns its private cards by computing $\mathsf{pc}'_{i,k} = \frac{c_{i,k,2}}{\prod_{i=1}^n d_{i,k,j}}$ for $k \in \{1, 2\}$.
3. After retrieving its private cards, $\mathcal{P}_i$ broadcasts $\sigma_{pc}^i = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\texttt{PRIVATE} - \texttt{CARDS})$. Remember the signature implicitly includes a nonce unique to this protocol execution and specific round. Once signatures σ_{pc}^j from all parties have been received, verify them.

Checkpoint Witness: The previous checkpoint witness, except for the signatures σ_{sb}^i and σ_{bb}^i, concatenated with all σ_{pc}^i.

[a] Notice that these initial ciphertexts just encrypt the number of each card (in increasing order) under deterministic randomness 1, allowing $\mathcal{P}_2$ to locally compute the initial set of ciphertexts for verification.

Figure 5: Protocol π_{Poker} (continuation).

Hand Execution - Main Flow: After cards are drawn and private cards are distributed, all parties proceed to the main flow of playing a hand, where a number of community cards will be opened and a number of betting rounds will be played, both according to the community card opening and betting round procedures. All parties continue the main flow by proceeding as follows:

- Execute a betting round starting with the closest active successor of $\mathcal{P}_{pbb}$.
- Execute a community card opening procedure for flop cards $\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3$.
- Execute a betting round starting with the closest active successor of $\mathcal{P}_{psb-1}$.
- Execute a community card opening procedure for turn card cc_4.
- Execute a betting round starting with the closest active successor of $\mathcal{P}_{psb-1}$.
- Execute a community card opening procedure for river card cc_5.
- Execute a betting round starting with the closest active successor of $\mathcal{P}_{psb-1}$.
- Proceed to showdown starting with the last player who increased the bet in the last round, if there is one; otherwise, the closest active successor of $\mathcal{P}_{psb-1}$.

Betting Round: In the steps of π_{Poker} that require a betting round starting from party $\mathcal{P}_s$, each party $\mathcal{P}_i$ communicates its betting action $\text{ACTION}_i \in \{\text{FOLD}, \text{CALL}, (\text{RAISE}, r), \text{ALL-IN}, \text{CHECK}\}$ (as defined in $\mathcal{F}_{\mathsf{poker}}$) in a round robin manner starting from $\mathcal{P}_s$ and following the order $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ received from $\mathcal{F}_{\mathsf{SC}}$, proceeding as follows until the conditions specified in $\mathcal{F}_{\mathsf{poker}}$ for finishing the betting round are met:

- When it is $\mathcal{P}_i$'s turn to state its bet, $\mathcal{P}_i$ updates vectors bets and $\mathsf{balance}$ according to its action ACTION_i, *i.e.* it increases (resp. decreases) $\mathsf{bets}[i]$ (resp. $\mathsf{balance}[i]$) by the amount of coins required by ACTION_i as defined in $\mathcal{F}_{\mathsf{poker}}$. $\mathcal{P}_i$ generates a signature $\sigma^i_{bet} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\text{ACTION}_i, \mathsf{bets}[i], \mathsf{balance}[i])$ and broadcasts $(\text{ACTION}_i, \mathsf{bets}[i], \mathsf{balance}[i], \sigma^i_{bet})$.
- Upon receiving $(\text{ACTION}_j, \mathsf{bets}[j], \mathsf{balance}[j], \sigma^j_{bet})$ from party $\mathcal{P}_j$ for $j \neq i$, $\mathcal{P}_i$ checks the validity of σ^j_{bet}. Next, $\mathcal{P}_i$ verifies that $\mathsf{bets}[j]$ and $\mathsf{balance}[j]$ are consistent with ACTION_j according to the rules defined in $\mathcal{F}_{\mathsf{poker}}$. If not, $\mathcal{P}_i$ proceeds to the recovery phase. If both checks succeed, $\mathcal{P}_i$ updates its local copy of bets and $\mathsf{balance}$ with the new values of $\mathsf{bets}[j]$ and $\mathsf{balance}[j]$, and proceeds in the betting round.

When the conditions for ending the betting round specified in $\mathcal{F}_{\mathsf{poker}}$ are met, each party $\mathcal{P}_i$ broadcasts a signature $\sigma^i_{betstate} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\mathsf{bets}, \mathsf{balance})$ on its local copy of vectors bets and $\mathsf{balance}$. $\mathcal{P}_i$ waits until all signatures $\sigma^j_{betstate}$ are received from every other party $\mathcal{P}_j$ for $j \neq i$ and verifies that they are valid signatures on their local vectors bets and $\mathsf{balance}$ (verifying that all parties agree on the final bets and $\mathsf{balance}$). *Checkpoint Witness:* The previous checkpoint witness with the updated vectors bets and $\mathsf{balance}$, along with all signatures $\sigma^i_{betstate}$ on the updated vectors.

Community Card Opening: In the steps of π_{Poker} where a community card $\mathsf{cc} \in \{\mathsf{cc}_1, \mathsf{cc}_2, \mathsf{cc}_3, \mathsf{cc}_4, \mathsf{cc}_5\}$ has to be opened, party $\mathcal{P}_i$, for $i = 1, \ldots, n$, proceeds as follows:

1. $\mathcal{P}_i$ parses $\mathsf{cc} = (cc_1, cc_2)$ and broadcasts its decryption shares $d_i = cc_1^{\mathsf{TEG}.sk_i}$ along with a NIZK $\mathsf{DLEQ}(g, h_i, cc_1, d_i)$.
2. After all decryption shares d_j and corresponding DLEQ NIZKs are received from all parties, $\mathcal{P}_i$ verifies if all NIZKs are valid. $\mathcal{P}_i$ opens cc by computing $\frac{cc_2}{\prod_{i=1}^{n} d_j}$.
3. After opening cc, $\mathcal{P}_i$ broadcasts a signature $\sigma^i_{\mathsf{cc}} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk}(\texttt{COMMUNITY} - \texttt{OPEN}, \mathsf{cc})$ in order to communicate it has successfully opened cc. Once all signatures σ^j_{cc} from other parties have been received, $\mathcal{P}_i$ verifies that they are all valid.

Checkpoint Witness: The previous checkpoint witness together with all signatures σ^i_{cc}.

Figure 6: Protocol π_{Poker} (continuation).

Showdown: The parties proceed in a round-robin way. If a party $\mathcal{P}_i$ wishes to open its private cards $\mathsf{pc}_{i,1}, \mathsf{pc}_{i,2}$ during showdown, $\mathcal{P}_i$ broadcasts the decryption shares $d_{i,1,j}, d_{i,2,j}$ along with their corresponding DLEQ proofs, for $j = 1, \ldots, n$. For every party $\mathcal{P}_i$ who opens its private cards during showdown, the other parties $\mathcal{P}_j$ decrypt $\mathsf{pc}_{i,1}, \mathsf{pc}_{i,2}$ by following the same procedure used for reconstructing their own private cards. If decryption fails, $\mathcal{P}_j$ proceed to the recovery phase. If a party $\mathcal{P}_i$ wishes to muck during showdown, it broadcasts a signature $\sigma^i_{muck} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\texttt{MUCK})$, the other parties verify the signature. Once all parties have either opened or mucked, the parties proceed to the pot distribution.

Pot Distribution: Each party $\mathcal{P}_i$ uses the opened cards, chronological order of folded/mucked hands and current vectors balance and bets to locally compute the updated balance for all parties according to the rules of poker. It also zeros out bets. $\mathcal{P}_i$ broadcast signatures on balance and bets. Upon receiving these values from each party $\mathcal{P}_j$, $\mathcal{P}_i$ verifies that it is a valid signature on its own local updated vectors balance and bets. A party $\mathcal{P}_i$ who wishes to continue playing broadcasts a signature $\sigma^i_{cont} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\texttt{CONTINUE})$. A party $\mathcal{P}_i$ who no longer wishes to play or who has $\mathsf{balance}[i] = 0$ broadcasts a signature $\sigma^i_{chko} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_j}(\texttt{CHECKOUT})$. Each party $\mathcal{P}_i$ checks that all other parties' signatures are valid. For all parties $\mathcal{P}_j$ who choose to check-out, mark party $\mathcal{P}_j$ as inactive. After determining which parties remain active and which check out, each party $\mathcal{P}_i$ constructs a vector active such that $\mathsf{active}[j] = 1$ if party $\mathcal{P}_j$ is active in the next hand or $\mathsf{active}[j] = 0$ if $\mathcal{P}_j$ is checking out. $\mathcal{P}_i$ broadcasts a signature $\sigma^i_{act} = \mathsf{SIG.Sign}_{\mathsf{SIG}.sk_i}(\mathsf{active})$. $\mathcal{P}_i$ checks that signatures σ^j_{act} by all other parties $\mathcal{P}_j$ are valid signatures on the same active vector, otherwise it proceeds to the recovery phase. If there were check-outs, update the public key as $\mathsf{TEG}.pk = \prod_{j=1 \text{ s.t. j is active}}^{n} h_j$. Increment *psb* and *pbb* using the order among the active players. A signature on these values are also generated by each party and checked by the others. *Checkpoint Witness:* Vectors balance, bets and active, counters *psb* and *pbb*, as well as all signatures on these values.

Recovery Request: If a party $\mathcal{P}_i$ enters the recovery phase at any step of a given phase, it sends a message $(\text{REPORT}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$ to $\mathcal{F}_{\mathsf{SC}}$, where $\mathsf{Checkpoint}_i$ is the checkpoint witness from the previous phase and $\mathsf{CurrPhase}_i$ is the transcript of the current phase so far (*i.e.* only the messages that have been received and sent by $\mathcal{P}_i$ after the last checkpoint).

Responding to a Recovery Request: Upon receiving a message $(\text{REQUEST}, \mathcal{P}_i, \mathsf{Checkpoint}_i, \mathsf{CurrPhase}_i)$ from $\mathcal{F}_{\mathsf{SC}}$ containing the checkpoint witness and current phase transcript included in the REPORT message of $\mathcal{P}_i$, every other party $\mathcal{P}_j$ sends a message $(\text{RESPONSE}, \mathcal{P}_j, \mathsf{Checkpoint}_j, \mathsf{CurrPhase}_j)$ to $\mathcal{F}_{\mathsf{SC}}$ containing their own most recent checkpoint witness and the transcript of the current phase. Once all parties have responded to the recovery request, all parties have learned each other checkpoint witnesses and the transcripts of the current phase. For $i = 1, \ldots, n$, party $\mathcal{P}_i$ proceeds as follows:

- Upon receiving $(\text{NXT-STP}, \mathsf{phase}, \mathsf{round})$ from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{P}_i$ computes its message $\mathsf{msg}_{\mathsf{phase},\mathsf{round}}$ for the round specified by round of the phase specified by phase and sends $(\text{NXT-STP-RSP}, \mathsf{msg}_{\mathsf{phase},\mathsf{round}})$ to $\mathcal{F}_{\mathsf{SC}}$ following the protocol rules.
- Upon receiving $(\text{RECOVERED}, \mathsf{phase}, \mathsf{Checkpoint})$ from $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{P}_i$ records the checkpoint witness of the phase specified by phase and returns to the regular execution of next phase as described in the protocol by communicating directly to the other parties.

Figure 7: Protocol π_{Poker} (continuation).

Theorem 1. *Assuming that the DDH problem is hard and that the digital signature scheme* SIG *is* EUF-CMA *secure, protocol* π_{Poker} *securely computes* $\mathcal{F}_{\mathsf{poker}}$ *in the* $\mathcal{F}_{\mathsf{SC}}$*-hybrid, random oracle model in the presence of malicious static adversaries.*

Proof. In order to prove Theorem 1 we construct a non-uniform expected probabilistic polynomial time simulator (ideal adversary) $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\mathsf{poker}}$ and internal copies of c corrupted parties. $\mathcal{S}$ simulates the actions of the $n-c$ honest parties and the functionality $\mathcal{F}_{\mathsf{SC}}$. Let $\mathcal{H}$ denote the set of honest parties and $\mathcal{C}$ denote the set of corrupted parties in the internal execution run by $\mathcal{S}$. $\mathcal{S}$ is parameterized by the small sb and big bb blind bets amounts, the initial stake t, the maximum bet m per hand, the security deposit d and a timeout limit τ. $\mathcal{S}$ proceeds as follows:

Player Check-in: $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{SC}}$ internally as well as the parties. Whenever a corrupted party $\mathcal{P}_c \in \mathcal{C}$ sends a message $(\text{CHECKIN}, \mathsf{coins}(d+t), \mathsf{SIG}.vk_c, h_c, \mathsf{DLOG}(g, h_c))$ to $\mathcal{F}_{\mathsf{SC}}$, $\mathcal{S}$ verifies $\mathsf{DLOG}(g, h_c)$ and, if it is valid, uses the coins to perform the check-in of that party with $\mathcal{F}_{\mathsf{poker}}$. Whenever a honest party $\mathcal{P} \in \mathcal{H}$ checks-in with $\mathcal{F}_{\mathsf{poker}}$, $\mathcal{S}$ is informed and simulates π_{Poker}'s check-in procedure for that party. $\mathcal{S}$ records all keys and uses the public key shares to construct the initial Threshold ElGamal public key $\mathsf{TEG}.pk$. Note that due to the usage of $\mathsf{DLOG}(g, h_c)$ it is guaranteed that the corrupted parties always know secret-key shares corresponding to the public-key shares that they send. Hence, for any proper subset of all players, from their point of view the final public-key is such that they do not know the corresponding secret-key and so the Threshold ElGamal encryption scheme can be used to obtain indistinguishable ciphertexts. If some party fails to check-in within the timeout limit, $\mathcal{S}$ allows the parties to dropout from $\mathcal{F}_{\mathsf{poker}}$ and reclaim their coins.

Hand Execution - Shuffle: $\mathcal{S}$ executes the shuffling protocol exactly as in the real protocol (emulating the actions of the honest parties), with the exception that it uses the simulator of the ZKSH proofs to extract their witnesses learning the exact order of the cards in the final shuffled deck $\mathcal{D} = (\mathsf{c}_1^n, \ldots, \mathsf{c}_{52}^n)$. $\mathcal{S}$ learns the permutation and fresh randomness used to rerandomize and permute the ciphertexts representing cards in each step of the shuffling phase. Thus, $\mathcal{S}$ is able to follow the ciphertext encrypting each card number (from 1 to 52) to its final position in the deck. In other words, $\mathcal{S}$ learns the plaintext messages of $(\mathsf{c}_1^n, \ldots, \mathsf{c}_{52}^n)$ since it knows both the exact contents of $(\mathsf{c}_1^0, \ldots, \mathsf{c}_{52}^0)$ and the exact randomness used to rerandomize and permute these initial ciphertexts (extracted from the ZKSH NIZKs). $\mathcal{S}$ follows the same recovery triggers of the real world protocol, proceeding to the recovery phase if an invalid NIZK or signature is detected or if it does not receive a message from one of the internal corrupted parties before the timeout. If the shuffling phase is successfully completed, $\mathcal{S}$ answers SHUFFLE to the query SHUFFLE? from $\mathcal{F}_{\mathsf{poker}}$ and proceeds.

Hand Execution - Blinds: $\mathcal{S}$ executes the small and big blinds phases exactly as in the real protocol, emulating internal honest parties $\mathcal{P}_h \in \mathcal{H}$ in case they are the small and/or big blinds. $\mathcal{S}$ follows the same recovery triggers of the real world protocol, proceeding to the recovery phase if necessary. If the small and big blinds phase is successful, $\mathcal{S}$ answers with PAYSB, resp. PAYBB, to the query (PAYSB, psb), resp. (PAYBB, pbb), from $\mathcal{F}_{\mathsf{poker}}$ and proceeds.

Hand Execution - Drawing Cards and Private Cards Distribution: Upon receiving HAND together with the values $\widetilde{\mathsf{pc}}_{c,1}, \widetilde{\mathsf{pc}}_{c,2}$ of the corrupted parties' private cards from $\mathcal{F}_{\mathsf{poker}}$, $\mathcal{S}$ proceeds as follows. $\mathcal{S}$ follows the procedures of an honest party to compute the corresponding decryption shares from all parties $\mathcal{P}_h \in \mathcal{H}$ but for one party $\mathcal{P}_s \in \mathcal{H}$, whose decryption share will

be used to open the private cards of the internal corrupted parties to arbitrary values received from $\mathcal{F}_{\mathsf{poker}}$. For every $\mathcal{P}_c \in \mathcal{C}$ and $k \in \{1,2\}$, parse $\mathsf{pc}_{c,k}$ from the internal simulation as $(c_{c,k,1}, c_{c,k,2})$. Let m be the value that is encrypted in the card $\mathsf{pc}_{c,k}$ (that is known to $\mathcal{S}$ since it traces the known cards through the shuffling process), and $\widetilde{m}$ the value for the card $\widetilde{\mathsf{pc}}_{c,k}$ that is received from $\mathcal{F}_{\mathsf{poker}}$. Then $\mathcal{S}$ computes $\mathcal{P}_s$'s original decryption share $d_{c,k,s}$ and the modified decryption share

$$\widetilde{d}_{c,k,s} = \frac{d_{c,k,s} \cdot m}{\widetilde{m}}.$$

$\mathcal{S}$ then uses the DLEQ NIZK simulator to generate a valid proof $\widetilde{\mathsf{DLEQ}}(g, h_s, c_{c,k,1}, \widetilde{d}_{c,k,s})$ and sends $\widetilde{d}_{c,k,s}$ to $\mathcal{P}_c$ along with this proof. Note that the decryption process will give as result

$$\frac{c_{j,k,2}}{\widetilde{d}_{c,k,s} \prod_{i \neq s} d_{c,k,i}} = \frac{c_{j,k,2} \cdot \widetilde{m}}{m \prod_i d_{c,k,i}} = \frac{m \cdot \widetilde{m}}{m} = \widetilde{m}.$$

If the private cards distribution phase succeeds and all private cards are opened to all players, $\mathcal{S}$ answers HAND to $\mathcal{F}_{\mathsf{poker}}$. $\mathcal{S}$ follows the same recovery triggers of the real world protocol, proceeding to the recovery phase if necessary.

Hand Execution - Main Flow: $\mathcal{S}$ simulates the actions of the honest parties by executing the community cards opening and betting round procedures as described below. If a call to the community card opening procedure succeed, $\mathcal{S}$ answers $\mathcal{F}_{\mathsf{poker}}$'s message requesting permission to open the cards involved in this call and proceeds to the next procedure in the main flow. If during either a betting round or a community card opening procedure a recovery trigger happens in the real protocol, then $\mathcal{S}$ would proceed to the recovery phase.

Betting Round: During a betting round, $\mathcal{S}$ receives the actions of the honest parties from $\mathcal{F}_{\mathsf{poker}}$ and simulates the respective actions of the honest parties in the internal simulation. Whenever a corrupted party performs an action in the internal simulation, $\mathcal{S}$ forwards that action to $\mathcal{F}_{\mathsf{poker}}$. $\mathcal{S}$ follows the same recovery triggers as the real protocol to activate the recovery phase.

Community Card Opening: $\mathcal{S}$ follows a similar strategy as in the private cards distribution phase, that is, generating a decryption share that forces the ciphertext representing the card to be decrypted to an arbitrary card value obtained from $\mathcal{F}_{\mathsf{poker}}$. It simulates honestly the protocol procedures of all parties $\mathcal{P}_h \in \mathcal{H}$ but for one party $\mathcal{P}_s \in \mathcal{H}$ whose decryption share will be used to open the community card cc to the value received from $\mathcal{F}_{\mathsf{poker}}$. Parse cc from the internal simulation as (cc_1, cc_2). Let m be the value that is encrypted in the card cc (that is known to $\mathcal{S}$ since it traces the known cards through the shuffling process), and $\widetilde{m}$ the value for the card $\widetilde{\mathsf{cc}}$ that is received from $\mathcal{F}_{\mathsf{poker}}$. Then $\mathcal{S}$ computes $\mathcal{P}_s$'s original decryption share d_s and the modified decryption share

$$\widetilde{d}_s = \frac{d_s \cdot m}{\widetilde{m}}.$$

$\mathcal{S}$ then uses the DLEQ NIZK simulator to generate a valid proof $\widetilde{\mathsf{DLEQ}}(g, h_s, cc_1, \widetilde{d}_s)$ and sends $\widetilde{d}_s$ along with this proof to the parties. As above, it is easy to verify that the result of the decryption process will be the desired value. If the community card(s) opening procedure succeeds, $\mathcal{S}$ answers $\mathcal{F}_{\mathsf{poker}}$ confirming that the community cards involved in the procedure can be opened. $\mathcal{S}$ follows the same recovery triggers as the real protocol to activate the recovery phase.

Showdown: During the showdown, if a corrupted party $\mathcal{P}_s$ tries to reveal its cards, $\mathcal{S}$ simply simulates the actions of the honest parties. If a honest party $\mathcal{P}_s$ tries to reveal its cards, $\mathcal{S}$ follows a similar strategy as in the private cards distribution phase, that is, generating a decryption share that forces the ciphertexts representing the cards in the hand that is being opened to be decrypted to an arbitrary card value obtained from $\mathcal{F}_{\mathsf{poker}}$. If $\mathcal{S}$ receives $(\text{OPEN}, s, \widetilde{\mathsf{pc}}_{s,1}, \widetilde{\mathsf{pc}}_{s,2})$ containing the cards of a party $\mathcal{P}_s \in \mathcal{H}$ that is opening during showdown, it broadcasts to the internal corrupted parties $\mathcal{P}_c \in \mathcal{C}$ the decryption shares $d_{s,k,i}$ along with their corresponding DLEQ proofs for all $\mathcal{P}_i \in \mathcal{H} \setminus \{\mathcal{P}_s\}$. For $k \in \{1, 2\}$, parse $\mathsf{pc}_{s,k}$ from the internal simulation as $(c_{s,k,1}, c_{s,k,2})$. Let m be the value that is encrypted in the card $\mathsf{pc}_{s,k}$ (that is known to $\mathcal{S}$ since it traces the known cards through the shuffling process), and $\widetilde{m}$ the value for the card $\widetilde{\mathsf{pc}}_{s,k}$ that is received from $\mathcal{F}_{\mathsf{poker}}$. Then $\mathcal{S}$ computes $\mathcal{P}_s$'s original decryption share $d_{s,k,s}$ and the modified decryption share

$$\widetilde{d}_{s,k,s} = \frac{d_{s,k,s} \cdot m}{\widetilde{m}}.$$

$\mathcal{S}$ then uses the DLEQ NIZK simulator to generate a valid proof $\widetilde{\mathsf{DLEQ}}(g, h_s, c_{s,k,1}, \widetilde{d}_{s,k,s})$ and sends $\widetilde{d}_{s,k,s}$ along with this proof to the corrupted parties. Note that the decryption process will give the desired result as in the case of drawing cards.

If the opening succeeds, $\mathcal{S}$ answers (OPEN, s) to $\mathcal{F}_{\mathsf{poker}}$. $\mathcal{S}$ follows the same recovery triggers as the real protocol to activate the recovery phase.

Pot Distribution: $\mathcal{S}$ simulates the behavior of the honest parties and wait to see if the pot distribution succeeds. If this phase succeeds, $\mathcal{S}$ answers $\mathcal{F}_{\mathsf{poker}}$ with DISTRIBUTE to the query $(\text{DISTRIBUTE}, \mathsf{pot})$. $\mathcal{S}$ gets from $\mathcal{F}_{\mathsf{poker}}$ the information about which honest parties want to continue for the next hand and which parties want to leave the game and simulates the behavior of the honest parties in the internal simulated execution; if some party obtains all the signatures necessary to check-out, $\mathcal{S}$ perform the check-out for that party. $\mathcal{S}$ follows the same recovery triggers as the real protocol to activate the recovery phase.

Player Check-out: $\mathcal{S}$ simulates $\mathcal{F}_{\mathsf{SC}}$ internally as well as the honest parties. If a corrupted party $\mathcal{P}_c$ performs a check-out in the internal execution, $\mathcal{S}$ performs $\mathcal{P}_c$'s check-out on $\mathcal{F}_{\mathsf{poker}}$ and use the received coins to pay $\mathcal{P}_c$. If an honest player $\mathcal{P}_h$ is able to check-out in the internal execution, then $\mathcal{S}$ allows $\mathcal{P}_h$'s check-out from $\mathcal{F}_{\mathsf{poker}}$ to proceed. $\mathcal{S}$ follows the same recovery triggers as the real protocol to activate the recovery phase.

Recovery: $\mathcal{S}$ emulates $\mathcal{F}_{\mathsf{SC}}$ and simulates the behavior of the honest parties according to the procedures described above for the respective part of the protocol. If a timeout occurs or a misbehavior is detected $\mathcal{S}$ performs the compensation: $\mathcal{S}$ aborts the execution in $\mathcal{F}_{\mathsf{poker}}$, thus activating the compensation phase of $\mathcal{F}_{\mathsf{poker}}$ and getting the coins that $\mathcal{S}$ uses to compensate the corrupted parties that are supposed to get some. Otherwise, $\mathcal{S}$ returns to the normal execution when the next checkpoint is achieved.

Simulator Analysis: Notice that the simulator $\mathcal{S}$ conducts a simulation with internal copies of the corrupted parties by emulating $\mathcal{F}_{\mathsf{SC}}$ and executing the protocol exactly as an honest party would do for most of the protocol, except for phases where cards are opened. In these phases, $\mathcal{S}$ waits for information on the values that specific cards are supposed to have from $\mathcal{F}_{\mathsf{poker}}$ and then produces bogus decryption shares that result into the threshold ElGamal ciphertext that represents the card being decrypted to the value obtained from $\mathcal{F}_{\mathsf{poker}}$. $\mathcal{S}$ is able to do this because it can use the simulators for the NIZKs used in π_{Poker} to produce a valid NIZK showing

that the bogus decryption share is valid even without knowing a witness. As the shares of the secret key are never revealed, it is clear that the decryption shares are indistinguishable from a random element of the same group. Hence, the execution with $\mathcal{S}$ could only be distinguished from the real execution if the NIZKs generated without witnesses by their respective simulators are distinguishable from an actual real world NIZK generated with a witness. This is clearly not the case, since distinguishing the NIZKs generated by their simulator from NIZKs generated by real world parties who know the witnesses would break those NIZKs' zero-knowledge property. □

5 Concrete Complexity Analysis

In this section, we analyze the concrete communication and computational complexities of π_{Poker}.

We estimate (off-chain) communication and computational complexities for the case where no user cheats (thus never triggering the recovery phase). Notice that, during recovery, every party has to post on the public ledger both their most recent checkpoint witness along with protocol messages generated after that witness has been generated and then execute the protocol by posting each of the next messages in the public ledger. Thus, in case the recovery phase is triggered, the cost of performing recovery will depend on the exact point of the protocol where the recovery request happened. Nevertheless, we discuss why our on-chain space complexity is generally low given that we explicitly define compact witnesses for intermediate step of the protocol (even inside poker rounds). On the other hand, previous works in [35] and [9] only mention (but not define) intermediate witnesses for each round of the poker game.

Moreover, we exclude the cost of generating and sending the messages between the parties and $\mathcal{F}_{\mathsf{SC}}$. We remark that these messages are basically transactions being posted in the blockchain and their size and generation cost may vary depending on the concrete implementation.

Estimating Complexity: We estimate computational complexity in terms of the number of exponentiations that each party has to perform in each phase of the protocol. On the other hand, we estimate communication complexity in terms of the total number of group (*i.e.* $\mathbb{G}$) elements and ring (*i.e.* $\mathbb{Z}_p$) elements transferred by all parties in each phase of the protocol. Most of the messages exchanged in the protocol are broadcast to all parties[2]. However, during private cards distribution, decryption shares for each card are sent directly to its owner through a private channel. We denote messages transmitted through private channels by [private] and messages broadcast through public channels by [broadcast]. Messages that are not explicitly marked are assumed to be broadcast by public channels.

Both the Betting Round and Showdown phases have complexities that fully depend on the behavior of each player in the game of poker and other conditions such as the stake of the game. For example, a user can choose to keep raising his bet in a Betting Round and users can choose whether to show their cards or muck in Showdown. Those choices are perfectly honest and permitted in the game but they result in different final complexities for these phases of π_{Poker}. In the case of the Betting Round phase, we estimate the complexity for the case where all players speak once, which can be easily used to compute the complexity in cases where each player speaks multiple times. In the case of the Showdown phase, we estimate the complexity for the worst case (in terms of complexity), where all players choose to show their cards.

[2]We remark that, in our scenario, broadcasts can achieved by having parties communicate directly with each other due to the low number of parties (typically $n \leq 10$).

Instantiating the Building Blocks: In this analysis we instantiate ZKSH (NIZK of correctness of a shuffle) with parameters $k = 4$ and $l = 13$, which results in 208 exponentiations for the prover and 208 exponentiations for the verifier, with a proof size of 44 elements of $\mathbb{G}$ and 65 elements of $\mathbb{Z}_p$. Notice that this estimation is actually an upper bound for concrete communication complexity, since it pertains to the interactive version of ZKSH, which is significantly improved in terms of concrete communication complexity after applying the Fiat-Shamir heuristic.

We instantiate the signature scheme SIG with the ECDSA scheme [33]. This scheme is chosen because it does not require any extra computational assumptions (apart from the ones π_{Poker} is already based on) and because it can be efficiently implemented [10]. In the ECDSA scheme, a public key consists of a elliptic curve point (that we count as an element of $\mathbb{G}$) and a signature consists of two scalars (that we count as elements of $\mathbb{Z}_p$). The ECDSA scheme requires one elliptic curve point multiplication by a scalar for generating a key pair, one for signing and two for signature verification (without optimizations). Similarly, the exponentiations required by π_{Poker} can be efficiently implemented over elliptic curves as point multiplications by a scalar. However, we count the operations of elliptic curve point multiplications by scalars as exponentiations since π_{Poker} is written in terms of groups with multiplicative notation.

We present the concrete communication and computational complexities of π_{Poker} in Table 1.

Phase	Exponentiations (Per Player)	Communication (Total)	
		$\mathbb{G}$	$\mathbb{Z}_p$
Players Check-in	$2n + 1$	$2n$	$2n$
Hand Execution - Shuffle	$209n + 104$	$148n$	$67n$
Hand Execution - Blinds	$12n$	0	$24n$
Hand Execution - Drawing/ Private Cards Distribution	$16n - 13$	$2(n^2 - n)$ [private]	$4(n^2 - n)$ [private], $2n$ [broadcast]
Hand Execution - Main Flow	$52n - 20$	$5n$	$28n$
Showdown (Worst Case)	$8(n-1)^2$	$2n^2$	$4n^2$
Pot Distribution	$2n$	0	$4n$
Total	$8n^2 + 271n + 82$	$2n^2 + 155n$ [broadcast], $2(n^2 - n)$ [private]	$4n^2 + 127n$ [broadcast], $4(n^2 - n)$ [private]

Table 1: Concrete communication and computational complexities of π_{Poker} in terms of number of exponentiations executed per player and numer of elements of $\mathbb{G}$ and $\mathbb{Z}_p$ transmitted by all players in total for each phase with n players. During private cards distribution, some messages are sent through a private channel, which we denote by [private]. All the other messages in the protocol are broadcast through public channels, which we denote by [broadcast]. Messages that are not explicitly marked are assumed to be broadcast by public channels.

On-chain Space Complexity: Considering that players act honestly throughout the protocol, information only stored in the blockchain when a player wishes to redeem its rewards. In this case, the player must post a witness showing that all players agree that the protocol was correctly executed. This witness consists of a simple digital signature.

In case a malicious player does cheat and an honest player triggers the recovery mechanism, all players are required to post to the blockchain their latest checkpoint witness and the protocol messages generated after that witness. Notice that this checkpoint witness is also a simple digital

signature and that the bulk of the data posted on the blockchain actually depends on which phase of the protocol is currently being executed. Our protocol employs compact witnesses that show that several intermediate protocol phases have been correctly executed during each round of poker. For example, if recovery is triggered during the Main Flow phase of Hand Execution, only the latest checkpoint witness and short messages required in that phase would have to be posted to the blockchain, excluding the long messages previously sent in the Shuffle and Drawing Cards phase. On the other hand, previous protocols in [35] and [9] only mention that intermediate witnesses could be generated after a full round of poker, incurring in a much higher overhead in terms of blockchain storage when recovery happens. Moreover, such witnesses are only mentioned but not explicitly defined in [35] and [9].

Comparison with Previous Protocols: While we present estimated computational and communication complexities for each phase of a *complete poker game*, previous works only focus on individual card operations [47, 46, 42, 53, 52, 13, 5], making it hard to provide direct comparisons to our results. In order to provide evidence that our protocol indeed achieves high efficiency while providing provable security guarantees, we will focus on the card shuffling phase, which is the main bottleneck of poker protocols.

Most previous protocols rely on cut-and-choose techniques to prove that the shuffle phase was correctly executed by all users. On the other hand, we employ efficient zero knowledge proofs of correctness of a shuffle, which results in improvements of the order of (at least) 10 times. Considering a deck of 52 cards (necessary for a poker game) and a security parameter $k = 40$ for the cut-and-choose step (which is the lowest security parameter used for this kind of technique in modern cryptography), the protocol of [47] (used as a building block in [9]) requires $2120n$ exponentiations per player in the Shuffle phase where n is the number of players. With the same parameters, the Shuffle phase of the protocol proposed in [5] requires $6240(n-1)+8320$ exponentiations, where n is the number of players. On the other hand, our protocol only requires $209n + 104$ exponentiations per player as described in Table 1.

6 Conclusion

In this paper, we introduce the first specific purpose protocol for secure poker with payment distribution and penalty enforcement with fully-simulatable security. In order to argue about our protocol's security, we introduce the first formal simulation based security notions for such protocols, which have been overlooked in previous works. Moreover, we identify concrete flaws in previous protocols proposed in [13] and [5], showcasing the need for formal security definitions and proofs in designing poker protocols. Our work improves on previous heuristic approaches for constructing poker protocols and provides a more efficient alternative to general results that provide payment distribution and penalty enforcement for general MPC protocols, where generality comes at the cost of efficiency.

Even though we can formally prove that our protocol securely realizes a poker game with payment distribution and penalty enforcement in a sequentially composable scenario, our current results do not provide any guarantees in scenarios with arbitrary composability. In other words, our protocol retains its security guarantees when several executions are run in sequence but not when executed in parallel with other protocols or instances of itself. Therefore, we leave open the important open problem of designing protocols for the same poker functionality with arbitrary composability guarantees.

Another interesting (and related) open problem lies in constructing efficient protocols for

each of the individual card operations commonly used in card games, in such a way that these operations can be mixed and matched to obtain secure card games with payment distribution and penalty enforcement. Even though we prove that our protocol securely realizes the game of poker and intuitively the same operations could be applied to other games, our security proofs (as well as those of previous works) do not suffice for a setting where these operations are arbitrarily used to construct different games. In order to obtain such a protocol, it will be needed to obtain arbitrarily composable protocols for each individual card operation as opposed to a protocol for the full game of poker.

References

[1] Murad Ahmed. How UK beat the odds to win at online gambling . `https://www.ft.com/content/044a3d9e-7d1a-11e7-9108-edda0bcbc928`, 2017. [Online; accessed 29-August-2017].

[2] 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.

[3] 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.

[4] Imre Bárány and Zlotán Füredi. Mental Poker with Three or More Players. *Information and Control*, 59(1–3):84–93, 1983.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. *To appear in ASIACRYPT*, 2017. `http://eprint.iacr.org/2017/875`.

[10] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, volume 6917 of *Lecture Notes in Computer Science*, pages 124–142, Nara, Japan, September 28 – October 1, 2011. Springer, Heidelberg, Germany.

[11] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[12] Jordi Castella-Roca, Josep Domingo-ferrer, Andreu Riera, and Joan Borrell. Practical mental poker without a ttp based on homomorphic encryption. In *In Progress in Cryptology-Indocrypt, volume 2904 of LNCS*, pages 280–294. Springer-Verlag, 2003.

[13] 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.

[14] 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.

[15] Don Coppersmith. Cheating at mental poker, 1985.

[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 Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 239–247, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[18] Robert DellaFave. The Five Most Interesting Online Poker Data Trends Of 2015. `https://www.onlinepokerreport.com/18982/online-poker-2015-looking-back/`, 2015. [Online; accessed 24-August-2017].

[19] Robert DellaFave. New Jersey's Online Poker Sites: A Poker Player's Review. `https://www.onlinepokerreport.com/9981/nj-online-poker-room-reviews/`, 2017. [Online; accessed 24-August-2017].

[20] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 120–127, Santa Barbara, CA, USA, August 16–20, 1988. Springer, Heidelberg, Germany.

[21] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Heidelberg, Germany.

[22] The Economist. A Big Deal. `http://www.economist.com/node/10281315#print`, 2007. [Online; accessed 24-August-2017].

[23] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 10–18, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[24] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology – CRYPTO'86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Heidelberg, Germany.

[25] Faith Fich and Shafi Goldwasser. Sending messages without hidden information and applications to multiperson games, 1984.

[26] Steven Fortune and Michael Merritt. Poker protocols. In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, pages 454–464, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[27] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004.

[28] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, NY, USA, May 25–27, 1987. ACM Press.

[29] Shafi Goldwasser and Silvio Micali. Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing (STOC '82)*, pages 365–377. ACM Press, 1982.

[30] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A "paradoxical" solution to the signature problem (abstract) (impromptu talk). In G. R. Blakley and David Chaum, editors, *Advances in Cryptology – CRYPTO'84*, volume 196 of *Lecture Notes in Computer Science*, page 467, Santa Barbara, CA, USA, August 19–23, 1984. Springer, Heidelberg, Germany.

[31] 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.

[32] IMDb. Kaleidoscope. `http://www.imdb.com/title/tt0060581/`, 2017. [Online; accessed 12-September-2017].

[33] Don Johnson, Alfred Menezes, and Scott Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International Journal of Information Security*, 1(1):36–63, Aug 2001.

[34] 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.

[35] 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.

[36] Kaoru Kurosawa, Yutaka Katayama, Wakaha Ogata, and Shigeo Tsujii. General public key residue cryptosystems and mental poker protocols. In Ivan Damgård, editor, *Advances in Cryptology – EUROCRYPT'90*, volume 473 of *Lecture Notes in Computer Science*, pages 374–388, Aarhus, Denmark, May 21–24, 1991. Springer, Heidelberg, Germany.

[37] Richard Jay Lipton. How to cheat at mental poker, 1981.

[38] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *Advances in Cryptology – EUROCRYPT'91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526, Brighton, UK, April 8–11, 1991. Springer, Heidelberg, Germany.

[39] 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.

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

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

[42] 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.

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

[44] Michael Singh. Cryptocurrencies Provide New Processing Channels for Online Gambling Operators. http://professionalrakeback.com/bitcoin-and-cryptocurrency-provide-payment-processing-solutions-for-online-gaming-sites, 2017. [Online; accessed 24-August-2017].

[45] Heiko Stamer. Efficient electronic gambling: An extended implementation of the toolbox for mental card games. In *WEWoRC 2005, volume P-74 of Lecture Notes in Informatics*, 2005.

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

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

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

[49] Wikipedia. Poker Boom. `https://en.wikipedia.org/wiki/Poker_boom`, 2017. [Online; accessed 24-August-2017].

[50] C. C. Yeh. Secure and verifiable p2p card games. In *2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, volume 2, pages 344–349, Dec 2008.

[51] Mordechai Yung. Cryptoprotocols: Subscription to a public key, the secret blocking and the multi-player mental poker game. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 439–453. Springer, 1984.

[52] 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.

[53] 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.