Practical Construction for Secure Trick-Taking Games Even with Cards Set Aside

Rohann Bella[1], Xavier Bultel[1], Céline Chevalier[2](✉), Pascal Lafourcade[3], and Charles Olivier-Anclin[1,3,4]

[1] INSA Centre Val de Loire, Laboratoire d'informatique fondamental d'Orléans, Bourges, France
[2] CRED, Université Paris-Panthéon-Assas and DIENS, École normale supérieure, PSL Université, CNRS, INRIA, Paris, France
celine.chevalier@ens.fr
[3] Université Clermont-Auvergne, CNRS, Clermont-Auvergne-INP, LIMOS, Clermont-Ferrand, France
pascal.lafourcade@uca.fr
[4] be ys Pay, Paris, France

Abstract. Trick-taking games are traditional card games played all over the world. There are many such games, and most of them can be played online through dedicated applications, either for fun or for betting money. However, these games have an intrinsic drawback: each player plays its cards according to several secret constraints (unknown to the other players), and if a player does not respect these constraints, the other players will not realize it until much later in the game.

In 2019, X. Bultel and P. Lafourcade proposed a cryptographic protocol for Spades in the random oracle model allowing peer-to-peer trick-taking games to be played securely without the possibility of cheating, even by playing a card that does not respect the secret constraints. However, to simulate card shuffling, this protocol requires a custom proof of shuffle with quadratic complexity in the number of cards, which makes the protocol inefficient in practice. In this paper, we improve their work in several ways. First, we extend their model to cover a broader range of games, such as those implying a set of cards set aside during the deal (for instance Triomphe or French Tarot). Then, we propose a new efficient construction for Spades in the standard model (without random oracles), where cards are represented by partially homomorphic ciphertexts. It can be instantiated by any standard generic proof of shuffle, which significantly improves the efficiency. We demonstrate the feasibility of our approach by giving an implementation of our protocol, and we compare the performances of the new shuffle protocol with the previous

This study was partially supported by the French ANR project ANR-18-CE39-0019 (MobiS5). Other programs also fund to write this paper, namely the French government research program "Investissements d'Avenir" through the IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25) and the IMobS3 Laboratory of Excellence (ANR-10-LABX-16-01). Finally, the French ANR project DECRYPT (ANR-18-CE39-0007) and SEVERITAS (ANR-20-CE39-0009) also subsidize this work.

F. Baldimtsi and C. Cachin (Eds.): FC 2023, LNCS 13950, pp. 166–181, 2024.
https://doi.org/10.1007/978-3-031-47754-6_10

one. Finally, we give a similar protocol for French Tarot, with comparable efficiency.

1 Introduction

Trick-Taking Games. With the development of computers, many traditional games have been adapted into electronic versions. The emergence of the Internet has naturally made it possible to play these games online with opponents from all over the world. This is particularly the case for card games, and it is now possible to play Poker, Bridge, Blackjack, Ramis, Triomphe, Écarté, Euchre or Tarot with human opponents at any time and any place, thanks to the use of dedicated applications on computers or smartphones. While these applications allow users to play for fun, many of them offer to play for money. In this case, there are several security issues to consider, since an application that allows players to cheat would illegitimately make honest players lose money. For this reason, several works, initiated in the seminal paper of Goldwasser and Micali [13], have proposed cryptographic protocols allowing to play cards securely.

Trick-taking games are a family of card games that all have the same structure: the cards are dealt to the players, then the game is divided into several rounds; in each round, players take turns playing a card, and the player with the highest value card wins the round. However, players cannot play any card from their hand and must follow several constraints defined by the rules. For example, in Whist and its variant Spades (which appeared in the 40's), players must play a card of the same suit as the first card of the round if they can. There are many popular trick-taking games around the world such as Belote, Bridge, Tarot, Skat or Whist. Some of them are gambling, and can be played in online casinos, such as Spades, Bourré or Oh Hell Stackpot (a gambling version of Oh Hell).

Unlike other card games, trick-taking games allow players to cheat without it being immediately detectable: since the players' cards are hidden, it is not possible to know if a player respects the rules at the time it plays its card. The cheating is detected later in the game, when the cheater plays a card it is not supposed to have. In this case, the game is cancelled at the detriment of the other players which have lost time and energy. In addition, trick-taking games are often played in teams, and the cheater's teammates must then take responsibility of the cheater's behavior. While this may be embarrassing in the presence of the other players, it is much easier to deal with online when players are anonymous. To avoid this situation, online trick-taking game applications prevent illegal plays. However, to do this control, the application must have access to the cards of all players, which must therefore trust the application by assuming that it is not rigging the games.

Since such cheating is possible with a physical deck of cards, the classical cryptographic card game protocols do not prevent it. In [5], Bultel and Lafourcade introduce the secure trick-taking game protocols, which allows to detect when a player does not respect the rules of the game, without learning anything from its cards. Such protocols have the following properties:

Unpredictability: the cards are dealt at random.

Theft and cheating resistance: a player cannot play a card that is not in its hand, and cannot play a card that does not follow the rules of the game.

Hand and game privacy: players do not know the hidden cards of their adversaries at the beginning of the game, then at each step of the game, the protocol does not reveal anything else than the cards that have been played.

Unfortunately, the security model from [5] cannot be applied to games in which not all cards are used by the players, because the challenger deduces the opponent's hand from the knowledge of the honest players' hands, which is not possible if cards are discarded. This excludes some very famous games, such as the well-known French Tarot, the Skat game, considered as the national card game of Germany, as well as one of the oldest trick-taking games, Triomphe, which dates back to the 15th century and is at the origin of both the word *trump* and many other games, like Écarté and Euchre. As with Spades, for sake of clarity, we choose to focus here on Tarot, but our approach is easily generalized.

Furthermore, the card distribution mechanism of the protocol in [5] suffers from two drawbacks inherent to its design. In a nutshell, each player chooses a secret key sk and computes the corresponding public key pk for each of its cards. It then alters its public key (and other parameters) using a random value, and shuffles the generator/key pairs (with a proof of correctness). At the end of this step, each generator/key pair is assigned a random card thanks to a random value the players need to agree on. The first issue is that this approach is highly dependent on the random oracle model, the second is that the shuffle proof proposed in [5] is not efficient since its complexity is in $\mathcal{O}(n^2)$ in the number of cards, which is 32, 54, 78 or even 104 cards depending on the game.

Contributions. In this paper, we first extend the security model from [5] to cover the French Tarot (see Sect. 4). French Tarot being the most complex of the games with Cards Set Aside, it is easy to simplify our model to adapt it to other games having this property.

Then, we propose two new secure Trick-taking protocols based on a common idea (as in [5], for the sake of clarity, we base one of our protocols on Spades, but it can be adapted to any game having the same structure, such as Whist, Bridge, *etc.*, the other is based on Tarot for similar reasons). Their card representations differ from [5] (and is closer to classical cryptographic card game protocols), which allows us to address both of the above drawbacks. Each card is encrypted by a key shared by all players using a partially homomorphic public key encryption scheme, such that all shares are needed to decrypt a card. To shuffle the deck, the players randomise and shuffle these encrypted cards in turn, then each player is given its encrypted cards, and each player uses its key share to partially decrypt the other players' cards. Thus, at the end of this process, the cards are only encrypted by their owner's key share. This method has the advantage of shuffling the cards directly instead of shuffling keys associated with cards assigned *a posteriori*, so it is no longer necessary to use a random oracle to assign the cards randomly. Moreover, the shuffle is done on a partially homomorphic encryption scheme, and there are many efficient generic zero-knowledge proofs to prove the correctness of such a shuffle in the literature with linear complexity in the number of ciphertexts [2,11,15]. This allows us to instantiate our protocols much more efficiently than in [5], and to propose practical yet secure trick-taking protocols. Details are given in Sect. 5 and proofs are presented in the full version [3]. We also give a protocol for Tarot, with similar complexity (see Sect. 6 and the full version [3]).

The goal is to reduce this additional cost to a point where cryptographic operations would no longer cause delays during the game. The efficiency of our Trick-taking protocols is assessed in [3], along with an implementation in Rust to demonstrate their practicality. Most of the complexity cost comes from the proofs (that everything was done correctly), and especially in the shuffle phase (Proof 1 in Sect. 5). A first improvement is that we can implement two designs for this proof. In order to show the advantage of our approach, we evaluate the performance of our protocols when instantiated either with a specific proof built from the same method (and a similar execution time) as [5] (5.64 s for the proof and 5.72 s for the verification), or with the efficient generic proof proposed by Groth in [15] (234.70 ms for the proof and 175.23 ms for the verification), which is unapplicable to [5]. Provided with a linear execution time, usage of this design makes our protocol practical even if used with more cards and/or more players as its overall complexity is linear in the number of cards and in the number of players.

Related Work. There are several cryptographic protocols in the literature for securing online card games [1,4,8–10,13,16,18,20], but most of them do not prevent illegal moves in trick-taking games. To the best of our knowledge, the only protocol with this property is [5]. It is also possible to use generic tools to obtain similar properties such as multiparty computation [7] or proofs of circuits [12], but these approaches are too generic and inefficient. Finally, another line of research, complementary to ours, studies ways to detect cheating in trick-taking games by analysing the behavior of players [19]. The idea is to determine if a player knows its opponent's cards by analysing its playing style.

2 Technical Overview

2.1 Rules of Trick-Taking Games: The Example of Spades

The traditional version of Spades is played by 4 players divided into two teams of 2 players, but the rules can be adapted for more players. It uses the traditional deck of 52 cards divided into the 4 Latin suits, which are swords (spades ♠), cups (hearts ♡), coins (diamonds ♢) and clubs (♣) and its rules are as follows:

Draw. All 52 cards are handed out equally to each player for a total of 13 cards each. Each player then bids on the number of tricks it plans to win.

A round. The first player of a new game is chosen randomly, the others following in a determined order. The game consists of a sequence of rounds, requiring all 4 players to play a card in turn. In each round, the suit of the first card played is called the *leading suit* and the player that plays the highest card wins the tricks (the 4 cards played), and starts the next round.

Rank of cards. The cards of the same suit are ranked from highest to lowest as follows: Ace, King, Queen, Jack, 10, 9, 8, 7, 6, 5, 4, 3, 2. The cards of the spade suit have a higher value than the cards of the leading suit.

Priority of cards. A player *must play a card from the leading suit if it can.* Otherwise, it can play any card it wants. Note that since the players' cards are hidden, the other players cannot check if a player is following this rule at the moment it plays the card. We address this limitation (among others) with our secure trick-taking game protocol.

Objective. If the number of tricks exceeds a team's bet, its players win 10 points per trick, plus 1 point for each additional trick, otherwise 0 points.

Most trick-taking games, including Bridge, Whist, Belotte, Bourré, Coinche, Pinochle, Ho Hell and many others follow the same structure as Spades. The differences are in the number of players or cards, the way scores are calculated, the ranking and the priority of the cards. The rules of priority can be complex, requiring cards of higher and higher values for a given suit, or requiring a particular suit when a player does not have a card of the leading suit. However, as a general rule, at the time the card is played, it is always possible to determine which cards should have been played first if the player had them. Our protocol is based only on this property, so it can be easily generalized.

2.2 The Particularity of French Tarot

By describing Spades, we have given a quite general framework, powerful enough to be adapted to almost any trick-taking game. But one particular case has never been addressed: the case where a set of cards is set aside during the deal, such as the dog (*chien*) in French Tarot. The dealing of this game generates another hand: While played with 4 players, 6 cards are put aside in a fifth hand until the bets are over. Once the cards are dealt, the *bids* start. The *taker* (the player that bets the highest) then plays against the 3 other players and needs to obtain a certain amount of points in its tricks to win. A player that does not bid *passes.* If all players pass, new cards are dealt. Presented below in increasing importance, the bids implies various dealing procedures for the dog:

Petite ("small"): the "dog" is revealed to all players and added to the hand of the taker. The latter confidentially sets aside the same number of cards from its hand and puts them aside to form the beginning of its score pile.

Garde ("guard"): same as *petite*, and points earned by the taker are double.

Garde sans ("guard without" the dog): the dog goes directly into the taker's score pile, no one gets to see it. The point multiplier is set to four.

Garde contre ("guard against" the dog): the dog goes directly into the opposing score pile. The score is worth six times the base score.

The deck in Tarot consists of 78 cards of 3 types: 52+4 normal cards (Ace, King, Queen, *Knight*, Jack, 10 down to 2, nearly as in Spades) and 22 *trumps* (from 1 to 21, and an *Excuse*). Excuse, 1 (*Petit*) and 21 of trumps are special

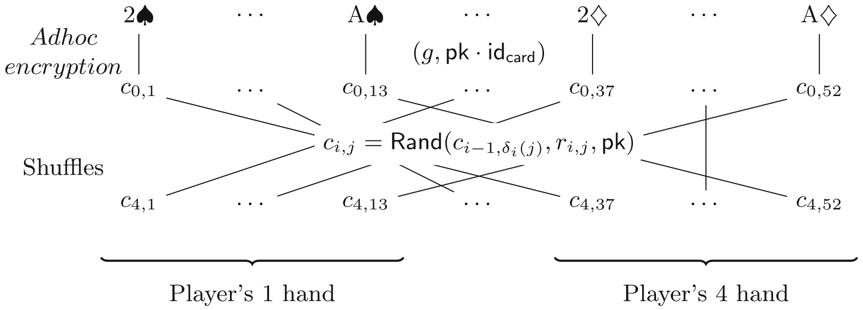

Fig. 1. Dealing cards in our trick-taking protocol. id : cards, pk : a public key, $r_{i,j}$: random numbers, permutations $\delta_i(j) \in [\![1, 52]\!]$ for all $i \in [\![0, 3]\!]$, $j \in [\![1, 52]\!]$.

cards and called the *oudlers*. On a *petite* or *garde*, the taker may not set aside in the dog a king or a trump, except if it cannot discard anything else; In this case, the trumps put in the dog must be displayed. In any case, it is forbidden to discard oudler trumps. Without entering into details of the game, Tarot follows the general rule that at the time the card is played, it is always possible to determine which cards should have been played first if the player had them.

Note that unlike Tryomphe or Euchre, this game has very specific rules giving rise to several particular cases. We treat the case of the French Tarot because its model and protocol can be adapted easily to other games with cards set aside.

2.3 An Overview of Our Protocols

To ensure that honest users can play online while no cheater can proceed for more than one round, our trick-taking protocols (formally presented in Definitions 3 and 4) require the following properties: First, at each step of the game, the previous plays should have been valid for the rounds to continue. Secondly, no player or central authority must have been trusted to reach the first requirement. Finally, maybe the most important of the conditions, the algorithm has to be practical, since a significant computational overhead would prevent any attempt of a player to play the game. To achieve this level of security, we choose a model in which at each round, for each of the played cards, the players must provide a proof for each of their actions, that their fellows verify before proceeding. These proofs have to be *zero-knowledge*, *i.e.,* reveal nothing about the players' hands.

Card Dealing. Before playing, the cards must have been shuffled and drawn (proofs ensuring each player that everything was executed correctly). We use randomisable encryption (that allows to randomise the ciphertext). A first phase (graphically represented in Fig. 1, for a standard set of cards) allows to give each player its (encrypted) hand. A second phase allows it to recover its hand.

Setup. Each player P_i starts the game by (1.i) generating a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ from which a global public key pk is generated. The canonical deck (with predefined order) is denoted as $D = (\mathsf{id}_1, \ldots, \mathsf{id}_{52})$. Proofs ensure that the keys were generated correctly.

Generation of the Ciphertexts. Each player (1.ii) computes on his side *ad hoc* randomisable (ElGamal) ciphertexts $(c_{0,j})_{j=1,\ldots,52}$ of all cards in D with the common public key pk.

Shuffle. To shuffle this set of encrypted cards, each player P_i in turn (1.iii) sequentially applies a random permutation $(\delta_{i,j})$ to the ciphertexts and randomise them using a secret random vector $(r_{i,j})$ and the randomisation algorithm of ElGamal presented in Sect. 2.3. Each of these steps is associated with a proof. Cards are now shuffled and distributed in between the players. For $i \in \{1, 2, 3, 4\}$, player P_i receives the ciphertexts of indices in $\{13 \cdot (i-1) + 1, 13 \cdot i\}$.

Hand Recovery. All players (2.i) broadcast some values $\theta_{i,j}$ (beside a proof) for the 39 ciphertexts they have not been attributed. This allows each player P_i to (2.ii) remove the randomness on the other players' keys on the ciphertexts to recover a vector of ciphertexts only encrypted by pk_i. Its cards remain oblivious to the other players as they are still encrypted with its key. It can finally obtain its cards by decrypting these values using sk_i.

Dog Generation. The rules of a trick-taking game may require some cards to be set aside during the shuffle. To keep these cards secret, some ciphertext indices are associated to the dog and the matching $\theta_{i,j}$ may not be revealed by the players. Unrevealed cards form the dog, based on the rules, they can later be revealed (through a similar process as part 2 of the shuffle), permuted or shuffled with some other cards (as in 1.iii). All outputs of these operations are produced alongside the associated proofs. As highlighted in Sect. 2.2, in French Tarot, kings and trumps may not be placed in the dog unless it is impossible to proceed otherwise. For later use, we define a set $\mathsf{O} \subset D \in \mathsf{Deck}$ composed of the cards id that may not be discarded. To guaranty that rules are followed, one has to prove that none of the cards placed in the dog do belong to O.

Card Playing. How a card is picked is not specified in our protocol, but it ensures that it follows the rules of the game. When player P_i picks one of its cards to be played, it first proves that the played card is indeed in its hand (by showing it matches one of its ciphertexts). Then it shows that the played card follows the rules of the game: if it does not follow the leading suit, it has to prove that none of its remaining ciphertexts encrypt cards that could have followed this suit. Immediate verification of the proofs by the other players remove all potential doubts on the validity of the new play.

3 Cryptographic Tools

First we recall the Decision Diffie-Hellman hypothesis (DDH): Let $\mathbb{G}$ be a group. The DDH assumption states that given $(g, g^a, g^b, g^z) \in \mathbb{G}^4$, there exists no polynomial-time algorithm able to decide whether $z = a \cdot b$ or not. Our schemes uses the ElGamal encryption scheme defined by the following algorithms:

$\mathsf{KeyGen}(\mathfrak{K})$: Picks $\mathsf{dk} \xleftarrow{\$} \mathbb{Z}_q^*$ (draw uniformly in the specified set) and computes $\mathsf{ek} = g^{\mathsf{dk}}$. Returns $(\mathsf{ek}, \mathsf{dk})$.

$\mathsf{Enc}(m, \mathsf{ek})$: Draws $y \xleftarrow{\$} \mathbb{Z}_q^*$, returns $c = (c_1 = g^y, c_2 = m \cdot \mathsf{ek}^y)$.

$\mathsf{Dec}(c, \mathsf{dk})$: Parses c as (c_1, c_2) and returns $m = c_2 \cdot c_1^{-\mathsf{dk}}$.

ElGamal is IND-CPA secure (indistinguishable under chosen plaintext attack) under DDH [17], moreover it is *partially homomorphic* and *randomizable*, which means that there exists an algorithm Rand that changes a ciphertext c into a new ciphertext c' of the same plaintext:

$\mathsf{Rand}(c, r, \mathsf{ek})$: Parses c as (c_1, c_2) and returns $c' = (c_1' = c_1 \cdot g^r, c_2' = c_2 \cdot \mathsf{ek}^r)$.

Our construction also uses Non-Interactive Zero-Knowledge Proofs of Knowledge (NIZKP) [14]. Let $\mathcal{R}$ a binary relation and s, w two elements verifying $(s, w) \in \mathcal{R}$. A (NIZKP) is a cryptographic primitive allowing a prover knowing a witness w to show that w and s verify the relation $\mathcal{R}$ leaking no information on w. Throughout this paper, we use the Camenisch and Stadler notation [6], *i.e.,* $\mathsf{ZK}\{w : (w, s) \in \mathcal{R}\}$ denotes the proof of knowledge of w for the statement s and the relation $\mathcal{R}$, and $\mathsf{Ver}(s, \pi)$ returns 1 if the proof π is correct, 0 otherwise.

Let $\mathcal{L}$ be a language such that $s \in \mathcal{L} \Leftrightarrow (\exists w, (s, w) \in \mathcal{R})$. A NIZKP is said to be *sound* when there is no polynomial-time adversary $\mathcal{A}$ such that $\mathcal{A}(\mathcal{L})$ outputs (s, π) such that $\mathsf{Ver}(s, \pi) = 1$ and $s \notin \mathcal{L}$ with non-negligible probability. It is said to be *extractable* when there exist a polynomial-time knowledge *extractor* Ext and a negligible function ϵ_{SoK} such that, for any algorithm $\mathcal{A}^{\mathsf{Sim}(\cdot,\cdot)}$ that outputs a fresh statement (s, π) with $\mathsf{Ver}(s, \pi) = 1$ such that $\mathcal{A}$ has access to a simulator that forges proofs for chosen statements, $\mathsf{Ext}^{\mathcal{A}}$ outputs w such that $(s, w) \in \mathcal{R}$ having access to $\mathcal{A}$ with probability $1 - \epsilon_{\mathsf{extract}}$. It is said to be *Zero-knowledge* when a proof leaks no information, *i.e.,* there exists a polynomial-time algorithm Sim called the *simulator* such that $\mathsf{ZK}\{w : (s, w) \in \mathcal{R}\}$ and $\mathsf{Sim}(s)$ follow the same probability distribution.

4 Models for Trick-Taking Game Revisited

4.1 Formal Definitions of Trick-Taking Scheme and Protocol

Trick-taking schemes and protocols were formalised in [5], but their definitions miss the French Tarot. Here we extend them to cover this additional game while staying consistent with the existing. We introduce a new definition covering both the existing and our work, for that we merge algorithms $\mathsf{DeckGen}$ and $\mathsf{GKeyGen}$ as it could have been in [5]. Only $\mathsf{DeckGen}$ is kept for the shuffle. In order to cover the dog in French Tarot, we also add up an algorithm named $\mathsf{MakeDog}$.

Trick-Taking Game Scheme. In trick-taking games, a card is defined based on two attributes: a suit and a number, such that $\mathsf{id} = (\mathsf{suit}, \mathsf{val}) \in \mathsf{Suits} \times \mathsf{Values}$ is a card. A *deck* of k *cards* is modeled by a k-tuple $D = (\mathsf{id}_1, \ldots, \mathsf{id}_k)$, where $\forall i, j \in [\![1, k]\!]$, $\mathsf{id}_i \neq \mathsf{id}_j$. The set of all possible decks is denoted by Decks. A deck D might contains a subset O of cards that may not be discarded in the dog.

We first define trick-taking schemes, which contain all the algorithms that are used by the players. KeyGen allows each player to generate its public/secret key. $\mathsf{DeckGen}$ is a protocol that distributes the cards. $\mathsf{MakeDog}$ allows to manipulate a dog. $\mathsf{GetHand}$ determines the hand of a given player from its secret key and the game key. Play allows a player to play a card, and to prove that it follows the rules of the game. Verif allows the other players to check this proof. Finally, $\mathsf{GetSuit}$ returns the leading suit of the current round. Formally:

Definition 1. *A* trick-taking scheme *W, definied as a tuples composed of algorithms* $(\mathsf{Init}, \mathsf{KeyGen}, \mathsf{VerifKey}, \mathsf{DeckGen}, \mathsf{GetHand}, \mathsf{Play}, \mathsf{Verif}, \mathsf{GetSuit})$ *executed between m participants is defined as follows:*

$\mathsf{Init}(\mathfrak{K})$*: It returns a setup parameter* setup*.*

$\mathsf{KeyGen}(\mathsf{setup})$*: It returns a key pair* $(\mathsf{pk}, \mathsf{sk})$*.*

$\mathsf{DeckGen}$*: It is a m-party protocol, where for all $i \in [\![1, m]\!]$ the i^{th} party, denoted as P_i, takes as input $(\mathsf{sk}_i, \{\mathsf{pk}_l\}_{1 \leq l \leq m})$. This protocol returns a deck D and a game public key* PK*, or the bottom symbol $\bot$.*

$\mathsf{GetHand}(n, \mathsf{sk}, \mathsf{pk}, \mathsf{PK})$*: It returns a set of cards $H \subset D$ called* a hand *if the player index n matches the keys.*

$\mathsf{Play}(n, \mathsf{id}, \mathsf{sk}, \mathsf{pk}, \mathrm{st}, \mathsf{PK})$*: It takes as input a player index $n \in [\![1, m]\!]$, a card* id*, a pair of secret/public key, a global state* st *that stores the relevant information about the previous plays, the game public key* PK *and returns a proof Π, and the updated global state* st'*.*

$\mathsf{Verif}(n, \mathsf{id}, \Pi, \mathsf{pk}, \mathrm{st}, \mathrm{st}', \mathsf{PK})$*: It takes as input a player index $n \in [\![1, m]\!]$, a card identity* id*, a proof Π generated by the algorithm* Play*, the global state* st *and the updated global state* st'*, the game public key* PK *and returns a bit b. If $b = 1$, we say that Π is* valid*.*

$\mathsf{GetSuit}(\mathrm{st})$*: It returns a suit* $\mathsf{suit} \in \mathsf{Suits}$ *from the current global state of the game* st*, where* suit *is the leading suit for the current turn.*

An additional algorithm can be added to trick-taking schemes to support a dog:

$\mathsf{MakeDog}(n, \mathsf{PK})$*: This is an m-party protocol outputting an updated game public key* PK *based on the previously derived key and a player index n.*

Trick-Taking Protocol. We now present the trick-taking protocol, which defines the order of execution of the above algorithms. It is divided into three phases: keys generation, shuffle and splitting of the card, and finally the game phase.

Definition 2. *Let W be a trick-taking scheme potentially with a* $\mathsf{MakeDog}$ *algorithm and $\mathfrak{K} \in \mathbb{N}$ be a security parameter. Let $\mathsf{P}_1, \ldots, \mathsf{P}_m$ be m polynomial-time algorithms. The* trick-taking protocol *instantiated by W between $\mathsf{P}_1, \ldots, \mathsf{P}_m$ is the following protocol:*

Keys generation phase: P_1 *runs* $\mathsf{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$ *and broadcasts* setup*. The players set* $\mathrm{st} = \bot$*. Each player P_i runs* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{setup})$ *and broadcasts* pk_i*.*

Shuffle phase: *All the players start by checking the other players' proofs. Then* P_1 *generates a deck* $D \in \mathsf{Decks}$ *and broadcasts it. The players generate* PK *by running the protocol* $\mathsf{DeckGen}$ *together. For all* $i \in [\![1, m]\!]$*,* P_i *runs* $H_i \leftarrow \mathsf{GetHand}(n, \mathsf{sk}, \mathsf{pk}, \mathsf{PK})$*. Then if instantiated, the players run* $\mathsf{MakeDog}$ *based on the derived game public key* PK *and for a common index* n*.*

Game phase: *This phase is composed of* k *(sequential) steps (corresponding to the number of cards played in a game). The players initialize the current player index* $p = 1$*. At each turn,* P_p *designates the player which plays. Each step proceeds as follows:*

- P_p *chooses* $\mathsf{id} \in H_p$*, then runs* $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK})$*.*
- *For all* $i \in [\![1, m]\!] \setminus \{p\}$*,* P_p *sends* $(\mathsf{id}, \Pi, \mathrm{st}')$ *to* P_i*.*
- *Each* P_i *then checks that* $\mathsf{Verif}(p, \mathsf{id}, \Pi, \mathsf{pk}_p, \mathrm{st}, \mathrm{st}', \mathsf{PK}) = 1$*, otherwise,* P_i *sends* `error` *to* P_p*, which repeats this step.*
- *If* $\mathsf{Verif}(p, \mathsf{id}, \Pi, \mathsf{pk}_p, \mathrm{st}, \mathrm{st}', \mathsf{PK}) = 1$*, all players update the state* $\mathrm{st} := \mathrm{st}'$*, and update the index* p *that points to the next player according to the rule of the game.*

4.2 Security Properties

We now recall the security model of trick-taking protocols introduced in [5]. We give a high-level description of its properties, the full formalism is given in the full version [3]. Note that we adjusted some parts to make them more generic to cover both the protocol of [5] and our Spades protocol (the model proposed in [5] being too specific to the design of the related protocol). To formalise the security of our French Tarot protocol, that does not fall within the general model, an *ad hoc* model is depicted at the end of this section and detailed in [3].

In general, we consider a security experiment where a challenger interacts with an adversary. The adversary simulates the behaviour of a malicious player and its teammate, which we will refer to as an *accomplice* (we therefore consider strong attacks where the adversary colludes with its teammate). The adversary chooses the secret key of the malicious player and shares its public key after the challenger has sent the public keys of the other three players, then the adversary chooses its accomplice, and the challenger reveals the key of the accomplice to the adversary. They then perform the shuffle phase, where the adversary plays the role of the malicious user and its accomplice, and the challenger simulates the behaviour of the other two players. Note that the challenger knows the secret keys of three players, so it can determine their hands, and thus deduce the hand of the malicious user. Finally, the adversary and the challenger simulate the game phase, where the adversary plays the role of the malicious user and its accomplice, and the challenger plays the role of the other two honest players. Of course, the security properties we describe must be proven regardless of the algorithm the challenger uses to simulate the two honest players.

Theft and Cheating Resistance: A protocol is *theft-resistant* when a player cannot play a card that is not in its hand. To attack the theft-resistance, the adversary must make the challenger accept a card that is not in the hand of the malicious player during the experiment with non-negligible probability. A protocol is *cheating-resistant* when a player cannot play a card that does not follow the rules of the game. To attack the cheating-resistance in a trick-taking protocol, the adversary must make the challenger accept a card that is not of the leading suit from the malicious player during the experiment with non-negligible probability, even though it has such cards in its hand.

Unpredictability: The *unpredictability* ensures that the cards are dealt at random. The adversary breaks this property if it can alter the shuffle in such a way that a card chosen at the beginning of the experiment ends up in one chosen hand with a significantly different probability than the usual distribution. Thus, unpredictable holds if no adversary succeeds this attack for any chosen card with a significant advantage. We have slightly modified this property to achieve a stronger version that the one originally presented in [5]. Here, our adversary chooses the card and the hand where it expects the card to be distributed.

Hand-Privacy: The *hand-privacy* ensures that the players do not know the hand of the other players at the beginning of the game. This time, the adversary has no accomplice, and the original experiment is truncated before the game phase. The challenger then chooses two out of the three honest players, and randomly picks one of their cards. To break the hand-privacy, the adversary must guess which player owns this card with a non-negligible advantage.

Game-Privacy: A protocol is *game-private* when at each step of the game phase, the players learn nothing else than the previously played cards. This property is defined by a real/simulated experiment. In the real setting, the adversary plays the real protocol with a challenger as in the experiment described above (again, the adversary has no accomplice). In the ideal one, the protocol is simulated using the public parameters of the honest users only. If there is a simulator such that the adversary cannot distinguish whether it is playing a real or simulated experiment with a non-negligible advantage, then the protocol is game-private. Intuitively, this means that a player could have simulated the protocol itself convincingly, which means that an adversary does not learn anything private during the game. Note that the combination of hand-privacy and game-privacy shows that the players have no information about the other players' hands except for all the cards they have already played.

Particularity of Dog's Security. One would expect a dog (or any set of card set aside in general) to behave as one of the player's hands: it should not be possible to steal (covered by theft resistance), to predict (unpredictability), to influence (theft-resistance) nor learn the cards in the dog (hand and game privacy) at the end of the shuffle. Despite fitting the model in terms of required properties, games with dogs do not allow us to rely completely on what exists. As specified above, the challenger must deduce the adversary's hand from its knowledge of the other three. With the dog, since some cards are not in the players' hands, this is no longer possible. The model must therefore be refined, at the expense of its genericity. Since the hand can no longer be implicitly inferred, we need to add an extractable NIZK of the players' secret keys to the formal definition to allow the challenger to explicitly retrieve the hand of the adversary. A less *ad hoc* model is left as an open problem.

In addition, to empower our adversary we let it decide which player takes and its bet. A second accomplice is also granted. Based on the rules of the Tarot game, the security of the dog should be insured through an additional property. The rules disallow to place some cards in the dog during the $\mathsf{MakeDog}$ algorithm. The latter is ensured through a property that we call *Dog security.*

5 Our Spades Protocol

We first define our new Spades protocol based on the randomisation of ElGamal. Here the deck D contains 52 cards, and each of the 4 players hands 13 cards.

Definition 3. *Algorithms of our Spades scheme are instantiated as follows:*

$\mathsf{Init}(\mathfrak{K})$*: It generates a group* $\mathbb{G}$ *of prime order* q*, a generator* $g \in \mathbb{G}$ *and returns* $\mathsf{setup} = (\mathbb{G}, q, g)$*.*

$\mathsf{KeyGen}(\mathsf{setup})$*: It picks* $\mathsf{dk} \xleftarrow{\$} \mathbb{Z}_q^*$ *and computes* $\mathsf{ek} = g^{\mathsf{dk}}$*. Then a proof of knowledge* $\Pi_{\mathsf{ek}} = \mathsf{ZK}\{\mathsf{dk}\colon \mathsf{ek} = g^{\mathsf{dk}}\}$ *is computed and* $(\mathsf{sk} = \mathsf{dk}, \mathsf{pk} = (\mathsf{ek}, \Pi_{\mathsf{pk}}))$ *is returned.*

$\mathsf{DeckGen}$*: It is a 4-party protocol, where for all* $i \in [\![1,4]\!]$ *the* i^{th} *party is denoted as* P_i*, and takes as input his/her secret keys* sk_i *and the public keys of all the players* $\{\mathsf{pk}_l\}_{1\le l\le 4}$*. This protocol returns a game public key* PK*, or* $\perp$*.*

<u>Phase 1:</u>

- *The canonical deck* $D \in \mathsf{Decks}$ *is initialized by each player.*
- *Each user parses* $D = (\mathsf{id}_1, \ldots, \mathsf{id}_{52})$ *and computes* $\mathsf{pk} = \prod_{i=1}^{4} \mathsf{ek}_i$*, then for all* $j \in [\![1,52]\!]$ *each player computes* $c_{0,j} \leftarrow (g, \mathsf{pk} \cdot \mathsf{id}_j)$ *and set* $c_0 \leftarrow (c_{0,j})_{1\le j\le 52}$*.*
- *For each* $i \in \{1,2,3,4\}$*, each* P_i *does in turn: it picks at random a permutation* $\delta_i \in [\![1,52]\!]^{52}$*, and* $(r_{i,j})_{1\le j\le 52} \xleftarrow{\$} (\mathbb{Z}_q^*)^{52}$*.* P_i *then computes* $c_{i,j} \leftarrow \mathsf{Rand}(c_{i-1,\delta_i(j)}, r_{i,j}, \mathsf{pk})$ *and generates a proof*

$$\pi_{i,1} \leftarrow \mathsf{ZK}\left\{(\delta_i, (r_{i,j})_{1\le j\le 52}) : c_{i,j} = \mathsf{Rand}(c_{i-1,\delta_i(j)}, r_{i,j}, \mathsf{pk})\right\}. \qquad (1)$$

Finally, P_i *sets* $c_i \leftarrow (c_{i,j})_{1\le j\le 52}$ *and broadcasts* $(c_i, \pi_{i,1})$*.*
- *Each player verifies the proofs* $(\pi_{i,1})_{1\le i\le 4}$*.*

<u>Phase 2:</u>

- *For all* $i \in [\![1,4]\!]$*, player* P_i *parses* $c_4 = (c_{4,j})_{1\le j\le 52}$ *and* $c_{4,j} = (x_j, y_j)$*.*
- *For all* $j \in [\![1,52]\!] \backslash [\![13\cdot(i-1)+1, 13\cdot i]\!]$*, each* P_i *computes* $\theta_{(i,j)} = x_j^{\mathsf{sk}_i}$*,*

$$\pi_{i,2} \leftarrow \mathsf{ZK}\left\{\mathsf{sk}_i : \bigwedge_{j\in[\![1,52]\!]\backslash[\![13\cdot(i-1)+1,13\cdot i]\!]} \theta_{(i,j)} = x_j^{\mathsf{sk}_i} \wedge \mathsf{pk}_i = g^{\mathsf{sk}_i}\right\}, \qquad (2)$$

then P_i *broadcasts* $(\theta_{(i,j)})_{j\in[\![1,52]\!]\backslash[\![13\cdot(i-1)+1,13\cdot i]\!]}$ *and* $\pi_{i,2}$*.*

- For all $i \in \llbracket 1,4 \rrbracket$, for all $l \in \llbracket 1,4 \rrbracket$, for all $j \in \llbracket 13 \cdot (l-1)+1, 13 \cdot l \rrbracket$, each P_i computes $c_j^* \leftarrow \left(x_j, \frac{y_j}{\prod_{1 \leq \gamma \leq 4; \gamma \neq l} \theta_{(\gamma,j)}} \right)$, and verifies the proofs $(\pi_{\gamma,2})_{\gamma \in \llbracket 1,4 \rrbracket \setminus \{i\}}$.
- Each player returns $\mathsf{PK} \leftarrow (c_j^*)_{1 \leq j \leq 52}$.

$\mathsf{GetHand}(n, \mathsf{sk}, \mathsf{pk}, \mathsf{PK})$: The algorithm parses PK as $(c_j^*)_{1 \leq j \leq 52}$ and returns a hand $H \leftarrow \{\mathsf{Dec}_{\mathsf{sk}}(c_j^*)\}_{j \in \llbracket 13 \cdot (n-1)+1, 13 \cdot n \rrbracket}$.

$\mathsf{Play}(n, \mathsf{id}, \mathsf{sk}, \mathsf{pk}, \mathrm{st}, \mathsf{PK})$: It parses $\mathsf{PK} = (c_j^*)_{1 \leq j \leq 52}$ and the state element $\mathrm{st} = (\alpha, \mathsf{suit}, U_1, U_2, U_3, U_4)$. If $\mathrm{st} = \perp$ it sets four empty sets U_1, U_2, U_3 and U_4. Let $t \in \llbracket 13 \cdot (n-1)+1, 13 \cdot n \rrbracket$ be the integer such that $\mathsf{id} = \mathsf{Dec}_{\mathsf{sk}}(c_t^*)$. It sets $U_n' = U_n \cup \{t\}$. Note that at each step of the game, the set U_n contains the indices of all the $(c_j^*)_{j \in \llbracket 13 \cdot (n-1)+1, 13 \cdot n \rrbracket}$ that have already been used by player n to play a card. For all $i \in \llbracket 1,4 \rrbracket \setminus \{n\}$, it sets $U_i' = U_i$.

If $\alpha = 4$ or $\mathrm{st} = \perp$ then it sets $\alpha' = 1$ and $\mathsf{suit}' = \mathsf{id}.\mathsf{suit}$. Else it sets $\alpha' = \alpha + 1$ and $\mathsf{suit}' = \mathsf{suit}$. The index α states how many players have already played this round, so if $\alpha = 4$, players start a new round. Moreover, suit states which suit is the leading suit of the round, given by the first card played in the round. This algorithm sets $\mathrm{st}' = (\alpha', \mathsf{suit}', U_1', U_2', U_3', U_4')$. It generates

$$\Pi_0 = \mathsf{ZK}\left\{ \mathsf{sk} : \mathsf{id} = \mathsf{Dec}_{\mathsf{sk}}(c_t^*) \right\}, \tag{3}$$

which proves that the played card id matches one of the ciphertexts in PK attributed to the player n. Let $L \subset \llbracket 1,52 \rrbracket$ be a set such that for all $l \in L$, $\mathsf{suit}' \neq \mathsf{id}_l.\mathsf{suit}$, i.e., L is the set of the indices of the cards that are not of the leading suit this round. Then it produces:

- If $\mathsf{suit}' = \mathsf{id}.\mathsf{suit}$ or if $|U_n \cup \{t\}| = 13$, it sets $\Pi_1 \leftarrow \perp$ (if the card id is of the leading suit, then the player can play it in any case).
- If $\mathsf{suit}' \neq \mathsf{id}.\mathsf{suit}$ and $|U_n \cup \{t\}| < 13$, it generates

$$\Pi_1 = \mathsf{ZK}\left\{ \mathsf{sk} : \bigwedge_{\substack{j \in \llbracket 13 \cdot (n-1)+1, 13 \cdot n \rrbracket \\ j \notin U_n \cup \{t\}}} \bigvee_{l \in L} \mathsf{id}_l = \mathsf{Dec}_{\mathsf{sk}}(c_j^*) \right\}. \tag{4}$$

Which proves that the player n cannot play a card of the leading suit.

Finally, it returns the proof $\Pi = (t, \Pi_0, \Pi_1)$, and the updated value st'.

$\mathsf{Verif}(n, \mathsf{id}, \Pi, \mathsf{pk}, \mathrm{st}, \mathrm{st}', \mathsf{PK})$: It parses st as $(\alpha, \mathsf{suit}, U_1, U_2, U_3, U_4)$, st' as $(\alpha', \mathsf{suit}', U_1', U_2', U_3', U_4')$, the key PK as $(c_j^*)_{1 \leq j \leq 52}$, and Π as (t, Π_0, Π_1). First checks if $t \in \llbracket 13 \cdot (n-1)+1, 13 \cdot n \rrbracket$, if not return 0. If $\mathrm{st} = \perp$, it sets four empty sets U_1, U_2, U_3 and U_4. Let $L \in \llbracket 1,52 \rrbracket$ be a set such that for all $l \in L$, $\mathsf{suit}' \neq \mathsf{id}_l.\mathsf{suit}$, i.e., L is the set of the indices of the cards that are not of the leading suit. This algorithm first checks that the state st is correctly updated:

- If there exists $i \in \llbracket 1,4 \rrbracket \setminus \{n\}$ such that $U_i' \neq U_i$, then it returns 0.
- If $t \in U_n$ or $U_n \cup \{t\} \neq U_n'$, then it returns 0.
- If $\alpha = 4$ or $\mathrm{st} = \perp$, and $\alpha' \neq 1$ or $\mathsf{suit}' \neq \mathsf{id}.\mathsf{suit}$, then it returns 0.
- If $\alpha \neq 4$ and $\mathsf{suit} \neq \perp$, and $\alpha' \neq \alpha + 1$ or $\mathsf{suit}' \neq \mathsf{suit}$, then it returns 0.

This algorithm then verifies the ZKP to check that the player does not cheat by playing a card it has not, or by playing a card that is not of the leading suit even though it could play a card of the leading suit.

- *If Π_0 is not valid then it returns* 0.
- *If* $\mathsf{suit}' \neq \mathsf{id.suit}$ *and there exists an integer* $j \in [\![1, 13]\!]$ *such that* $(13 \cdot (n - 1) + j) \notin U_n$ *and* Π_1 *is not valid then it returns* 0.

If none of the previous checks fails, then this algorithm returns 1.

$\mathsf{GetSuit}(\mathrm{st})$*: It parses* st *as* $(\alpha, \mathsf{suit}, U_1, U_2, U_3, U_4)$ *and returns* suit.

Security. This Spades protocol relies on the unpredictability of the randomness introduced by the players, security of the ZKP and the DDH hypothesis.

Theorem 1. *Given proofs of knowledge with soundness, extractability and zero-knowledge, our protocol is theft-resistant, cheating-resistant, hand-private, unpredictable, and game-private under the DDH assumption.*

For lack of space, the proof of this theorem is given in the full version [3].

6 Our French Tarot's Protocol

We now show how to achieve a protocol that contains a dog through highlighting an instantiation of a Tarot protocol. Adapted from our previously presented Spades scheme of Sect. 5, we need to address the MakeDog algorithm based on the rules of this game. We present this protocol for 4 players and a regular deck of 78 cards. Based on the rules this leads to 18 cards for each player and a dog composed 6 cards. We assume that cards indexed by $i \in [\![73, 78]\!]$ are reserved for the dog and that O contains the cards that may not be discarded in the dog.

Definition 4. *Our French Tarot protocol is defined similarly to Definition 3 (the few differences are implied trivially by the specificity of the rules) except for the algorithm* MakeDog *defined as follows (see the full version [3] for details).*

$\mathsf{MakeDog}$*: It is a* 4*-party protocol taking as input the index* n *of a player.*

- *For all* $i \in [\![1, 4]\!]$*, player* P_i *parses* $c_4 = (c_{4,j})_{1 \leq j \leq 78}$ *and* $c_{4,j} = (x_j, y_j)$.
- *For all* $j \in [\![73, 78]\!]$*, each* P_i *send* $\theta_{(i,j)} = x_j^{\mathsf{sk}_i}$*, as well as a proof* $\pi'_{i,2} \leftarrow \mathsf{ZK}\left\{\mathsf{sk}_i : \bigwedge_{j \in [\![73,78]\!]} \theta_{(i,j)} = x_j^{\mathsf{sk}_i} \wedge \mathsf{pk}_i = g^{\mathsf{sk}_i}\right\}$.
- *For all* $i \in [\![1, 4]\!]$, $j \in [\![73, 78]\!]$*, each* P_i *recovers* $\mathsf{id}^*_j \leftarrow \left(\frac{y_j}{\prod_{1 \leq \gamma \leq 4} \theta_{(\gamma,j)}}\right)$*, the cards of the dog, and verifies the proofs* $(\pi'_{\gamma,2})_{\gamma \in [\![1,4]\!] \setminus \{i\}}$.
- P_n *shuffles its cards with the dog: first sets* $c^*_j = (g, \mathsf{pk} \cdot \mathsf{id}_j)$ *for* $j \in [\![73, 78]\!]$*, then let* $\mathsf{K} = [\![18 \cdot (n - 1) + 1, 18 \cdot n]\!] \cup [\![73, 78]\!]$*. It picks a permutation* $\delta \in \mathsf{K}^{24}$*, and* $(r_j)_{j \in \mathsf{K}} \xleftarrow{\$} (\mathbb{Z}_q^*)^{24}$*, computes* $c_{5,j} \leftarrow \mathsf{Rand}(c^*_{\delta(j)}, r_j, \mathsf{pk})$ *for* $j \in \mathsf{K}$ *and a proof* $\pi_5 \leftarrow \mathsf{ZK}\left\{(\delta, (r_j)_{j \in \mathsf{K}}) : c^*_{5,j} = \mathsf{Rand}(c^*_{\delta(j)}, r_j, \mathsf{pk})\right\}$*. For all* $j \in [\![1, 78]\!] \setminus \mathsf{K}$*, set* $c_{5,j} \leftarrow c^*_j$*. Player* P_n *sets* $c^* \leftarrow (c_{5,j})_{1 \leq j \leq 78}$.
- P_n *shows that it follows the rules and did not put unauthorized card in the dog by producing the proof:*

$$\Pi_n \leftarrow \mathsf{ZK}\left\{\mathsf{sk}_n : \bigwedge_{j \in [\![73,78]\!]} \bigvee_{l \notin \mathsf{O}} \mathsf{id}_l = \mathsf{Dec}_{\mathsf{sk}_n}(c_{5,j})\right\}, \tag{5}$$

then it sends (c^*, π_5, Π_n)*. If* P_n *has no choice but to put* l *trumps in the dog, then it cannot produce this proof. Let* $j_1, \dots, j_l \in [\![73, 78]\!]$ *be the indices of these cards. In this case,* P_n *produces the tokens* $\theta_{j_k} = x_{j_k}^{\mathsf{sk}_n}$ *and the proofs* $\pi_{j_k} \leftarrow \mathsf{ZK}\left\{\mathsf{sk}_n : \theta_{j_k} = x_{j_k}^{\mathsf{sk}_n} \wedge \mathsf{pk}_i = g^{\mathsf{sk}_i}\right\}$ *for* $1 \leq k \leq l$*. It also proves than it cannot proceed otherwise:*

$$\Pi'_n \leftarrow \mathsf{ZK}\left\{\mathsf{sk}_n : \bigwedge_{j \in [\![18\cdot(j-1)+1, 18\cdot j]\!]} \bigvee_{l \in \mathsf{O}} \mathsf{id}_l = \mathsf{Dec}_{\mathsf{sk}_n}(c_{5,j})\right\}, \tag{6}$$

and then produces proof 5, with $j \in [\![73, 78]\!] \setminus \{j_1, \dots, j_l\}$*. Player* P_n *then broadcasts* (c^*, π_5, Π_n) *and* $(\Pi'_n, \{\theta_{j_k}, \pi_{j_k}\}_{1 \leq k \leq l})$*.*

- *Each* P_i *for* $i \in [\![1, 4]\!] \setminus \{n\}$*, checks all the received proofs and checks that for all* $j \in [\![1, 78]\!] \setminus \mathsf{K}$*,* $c_{5,j} = c_l^*$*. In case* P_n *has revealed a card,* P_i *computes* $\mathsf{id}_{j_k} \leftarrow y_{j_k}/\theta_{j_k}$ *and checks* id_{j_k} *is an authorised oudler.*
- *Each player returns* $\mathsf{PK} \leftarrow c^*$*.*

Theorem 2. *Given proofs of knowledge with soundness, extractability and zero-knowledge, our tarot protocol is theft-resistant, cheating-resistant, hand-private, unpredictable, game-private and dog-secure under the DDH assumption.*

This theorem is based on similar arguments as exposed in Sect. 5. The proof is available in the full version [3].

7 Conclusion

In this paper, we modify and expand the security model for trick-taking games. It encompasses the security for a broader range of protocols and enables to put aside some cards after the shuffle and appoint them to a player later in the game. Two new trick-taking schemes with security in the standard model are proposed. These protocols can be instantiated with any proof of shuffle on partially homomorphic encryption, which makes them efficient and usable.

Future work would consist in implementing them in real conditions, with real and not simulated interactions between the players.

References

1. Barnett, A., Smart, N.P.: Mental poker revisited. In: Paterson, K.G. (ed.) Cryptography and Coding 2003. LNCS, vol. 2898, pp. 370–383. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40974-8_29
2. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_17
3. Bella, R., Bultel, X., Chevalier, C., Lafourcade, P., Olivier-Anclin, C.: Practical construction for secure trick-taking games even with cards set aside. Cryptology ePrint Archive, Paper 2023/309 (2023). https://eprint.iacr.org/2023/309

4. Bentov, I., Kumaresan, R., Miller, A.: Instantaneous decentralized poker. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 410–440. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_15
5. Bultel, X., Lafourcade, P.: Secure trick-taking game protocols - how to play online spades with cheaters. In: Goldberg, I., Moore, T. (eds.) FC 2019. Springer, Heidelberg (2019). https://eprint.iacr.org/2019/375
6. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups. In: Kaliski, B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 410–424. Springer, Heidelberg (1997). https://doi.org/10.1007/BFb0052252
7. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-32009-5_38
8. David, B., Dowsley, R., Larangeira, M.: 21 - bringing down the complexity: fast composable protocols for card games without secret state. In: Susilo, W., Yang, G. (eds.) ACISP 2018. LNCS, vol. 10946, pp. 45–63. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93638-3_4
9. David, B., Dowsley, R., Larangeira, M.: Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In: 21st International Conference, FC (2018)
10. David, B., Dowsley, R., Larangeira, M.: ROYALE: a framework for universally composable card games with financial rewards and penalties enforcement. In: Goldberg, I., Moore, T. (eds.) FC 2019. LNCS, vol. 11598, pp. 282–300. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32101-7_18
11. Furukawa, J., Sako, K.: An efficient scheme for proving a shuffle. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 368–387. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44647-8_22
12. Giacomelli, I., Madsen, J., Orlandi, C.: Zkboo: faster zero-knowledge for boolean circuits. In: Proceedings of the 25th USENIX Conference on Security Symposium. USENIX Association (2016)
13. Goldwasser, S., Micali, S.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing. STOC. ACM (1982)
14. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM J. Comput. (1989)
15. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. In: Journal of Cryptology. Springer (2010)
16. Stamer, H.: Bibliography on mental poker. https://www.nongnu.org/libtmcg/MentalPoker.pdf
17. Tsiounis, Y., Yung, M.: On the security of ElGamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0054019
18. Wei, T.J.: Secure and practical constant round mental poker. In: Information Sciences (2014)
19. Yan, J.: Collusion detection in online bridge. In: Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI. AAAI Press (2010). http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1942
20. Zhao, W., Varadharajan, V., Mu, Y.: A secure mental poker protocol over the internet. In: ACSW frontiers. Conferences in research and practice in information technology, Australian Computer Society (2003)