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

Rohann Bella, Xavier Bultel, Céline Chevalier, Pascal Lafourcade, Charles Olivier-Anclin

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

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

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

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 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 [15], 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 [6], 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 [6] 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 15^{th} 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 [6] 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 [6] 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 [6] to cover the French Tarot (see Section 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 [6], 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 [6] (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, 13, 17]. This allows us to instantiate our protocols much more efficiently than in [6], and to propose practical yet secure trick-taking protocols. Details are given in Section 5 and proofs are presented in Appendix E. We also give a protocol for Tarot, with similar complexity (see Section 6, and Appendix D and F).

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 Appendix 7, 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 Section 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 [6] (5.64 s for the proof and 5.72 s for the verification), or with the efficient generic proof proposed by Groth in [17] (234.70 ms for the proof and 175.23 ms for the verification), which is unapplicable to [6]. 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, 10–12, 15, 19, 21, 23], 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 [6]. It is also possible to use generic tools to obtain similar properties such as multiparty computation [9] or proofs of circuits [14], 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 [22]. 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 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 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 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 Definition 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 Figure 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 *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

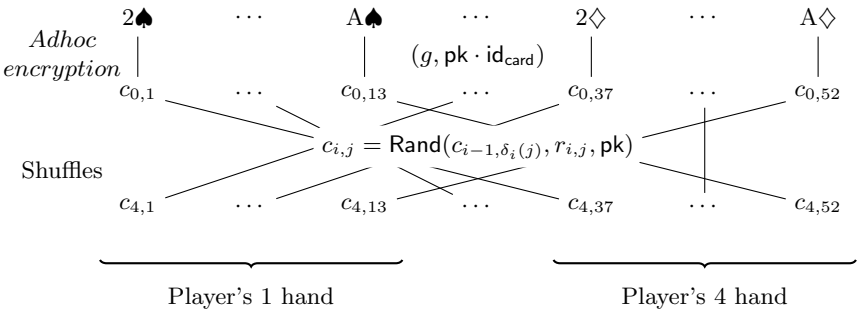

Fig. 1: Dealing cards in our trick-taking protocol. id : cards, pk : a public key, $r_{i,j}$: random numbers, permutations $\delta_i(j) \in [\![1, 52]\!]$ for all $i \in [\![0, 3]\!]$, $j \in [\![1, 52]\!]$.

randomisation algorithm of ElGamal presented in Section 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 $\theta_{i,j}$ (alongside 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 Section 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 [20], 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) [16]. 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 [7], *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 [6], 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 [6]. 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 $W = (\mathsf{Init}, \mathsf{KeyGen}, \mathsf{VerifKey}, \mathsf{DeckGen}, \mathsf{GetHand}, \mathsf{Play}, \mathsf{Verif}, \mathsf{GetSuit})$ *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* $\perp$.

$\mathsf{GetHand}(n, \mathsf{sk}, \mathsf{pk}, \mathsf{PK})$: *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* $\mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$ *and broadcasts* setup*. The players set* $\mathrm{st} = \perp$*. Each player* P_i *runs* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathit{setup})$ *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 [6]. We give a high-level description of its properties, the full formalism is given in Appendix B. Note that we adjusted some parts to make them more generic to cover both the protocol of [6] and our Spades protocol (the model proposed in [6] 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 Appendix C.

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 [6]. 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* $\{\mathsf{pk}_l\}_{1 \leq l \leq 4}$*. This protocol returns a game public key* PK*, or* $\perp$.

Phase 1:

– *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* $c_0 \leftarrow (c_{0,j})_{1 \leq j \leq 52}$.
– *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* $(r_{i,j})_{1 \leq j \leq 52} \xleftarrow{\$} (\mathbb{Z}_q^*)^{52}$. 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*

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

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

Phase 2:
– *For all* $i \in [\![1, 4]\!]$*, player* P_i *parses* $c_4 = (c_{4,j})_{1 \leq j \leq 52}$ *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* $\theta_{(i,j)} = x_j^{\mathsf{sk}_i}$,

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

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* $i \in [\![1, 4]\!]$*, for all* $l \in [\![1, 4]\!]$*, for all* $j \in [\![13 \cdot (l-1) + 1, 13 \cdot l]\!]$*, 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* $(\pi_{\gamma,2})_{\gamma \in [\![1,4]\!] \backslash \{i\}}$.
– *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* $H \leftarrow \{\mathsf{Dec}_{\mathsf{sk}}(c_j^*)\}_{j \in [\![13\cdot(n-1)+1, 13\cdot n]\!]}$.

$\mathsf{Play}(n, \mathsf{id}, \mathsf{sk}, \mathsf{pk}, \mathrm{st}, \mathsf{PK})$: *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* $t \in [\![13 \cdot (n-1) + 1, 13 \cdot n]\!]$ *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* $(c_j^*)_{j \in [\![13\cdot(n-1)+1, 13\cdot n]\!]}$ *that have already been used by player* n *to play a card. For all* $i \in [\![1, 4]\!] \backslash \{n\}$*, 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* $L \subset [\![1, 52]\!]$ *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*

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

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* $t \in [\![13 \cdot (n-1) + 1, 13 \cdot n]\!]$*, if not return 0. If* $\mathrm{st} = \perp$*, it sets four empty sets* U_1*,* U_2*,* U_3 *and* U_4*. Let* $L \in [\![1, 52]\!]$ *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* $i \in [\![1,4]\!] \setminus \{n\}$ *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* $\mathsf{suit}' \neq \mathsf{id.suit}$*, 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 Appendix E whereas a discussion on the arguments used to show each of the properties follows.

Theft-resistant. An adversary trying to play someone else's card falls short due to the required ZKP. While trying to play player P_i's card, its ignorance of the secret key sk_i means that it needs to produce such a proof for someone else's card, *i.e.,* a valid ZKP for a false statement, which is prevented by the soundness of the ZKP. Such a move is only possible with negligible probability since the proof systems are sound.

Cheating-resistant. We start by the same arguments as above: we invoke the soundness of the ZKP to show that an adversary cannot influence the shuffle of the game. To play a card that is not of the leading suit, assuming it has some cards of the leading suit at the time of playing the card, an adversary should forge a proof of a false statement. Again, the soundness of the ZKP implies the inability of the adversary to proceed.

Unpredictable. During the shuffle each action requires a ZKP to prove honesty in the execution. Soundness of these proofs imply that our adversary must follow the shuffle protocol. It still has an influence via the permutations and the randomisation it inputs. Two cases need to be considered, when the challenger is the last to shuffle and when it is not. In the first case the random values outputted by the challenger will vanish any attempt to cheat. However, in the second case, the adversary plays last. Based on the DDH we show that in such a case the adversary does not learn anything about the shuffle cards before it proceeds and cannot decide which value to pick. Hence, the rightfully executed shuffle yields the unpredictability of its outcomes.

Hand-private. Turn by turn, using a random permutation and randomising the ciphertexts, the players shuffle the deck during the execution of DeckGen. Assuming DDH hardness and thus the indistinguishability of the ElGamal encryption, we can conclude that the ciphertexts are completely unlinkable to the card values. An adversary would be unable to learn any information from what it sees and in the meantime ZKP ensure that it follows the protocol.

Game-private. We provide a simulator algorithm for our game. It shows random values instead of ciphertexts and simulats its ZKP in order to link these values to some cards. Due to extractability of the ZKP, we can build an adversary that would have high chances of breaking DDH if it were able to distinguish this simulation from a real game.

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 Section 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 Appendix D for details).*

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 $c_4 = (c_{4,j})_{1 \le j \le 78}$ 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 $\mathsf{id}_j^* \leftarrow \left(\frac{y_j}{\prod_{1 \le \gamma \le 4} \theta_{(\gamma,j)}}\right)$, 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 $c^* \leftarrow (c_{5,j})_{1 \le j \le 78}$.*
- *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 $j_1, \ldots, j_l \in [\![73, 78]\!]$ 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 $1 \le k \le l$. 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 $j \in [\![73, 78]\!] \setminus \{j_1, \ldots, j_l\}$. Player P_n then broadcasts (c^*, π_5, Π_n) and $(\Pi'_n, \{\theta_{j_k}, \pi_{j_k}\}_{1 \le k \le l})$.
- *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 Section 5. The proof is detailed in Appendix F.

7 Efficiency Analysis

Users of online card games want efficient and reliable tools. Waiting in between plays would inevitably lead to a degraded game experience. We compare our protocol to the only other secure Spades protocol [6], and we are able to reduce this waiting to a point where cryptographic operations would no longer cause

delays during the game. To show these claims, we have implemented our protocol in Rust [18], based on the well-known *Curve25519* [5] elliptic curve as it is known for its efficiency. The benchmarks below have been obtained using an Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz processor without using parallelisation. No other Tarot scheme exists in the literature, this one has a very similar execution time as the Spades protocol as it keeps the same structure and uses the same type of proofs. Let n be the number of cards used during a game.

Most of the computations are due to the numerous zero-knowledge (ZK) proofs. A first improvement is that we can implement two designs for Proof 1 of the shuffle.

- Design (1): proof *1-out-of-n* [8] based on Schnorr's proof, complexity $\mathcal{O}(n^2)$.
- Design (2): Groth proof of shuffle from [17], complexity $\mathcal{O}(n)$.

Since the second design proves shuffle of homomorphically encrypted elements, it cannot be used in [6]. In Figure 2, the time execution of these shuffles from 10 to 100 cards is presented.

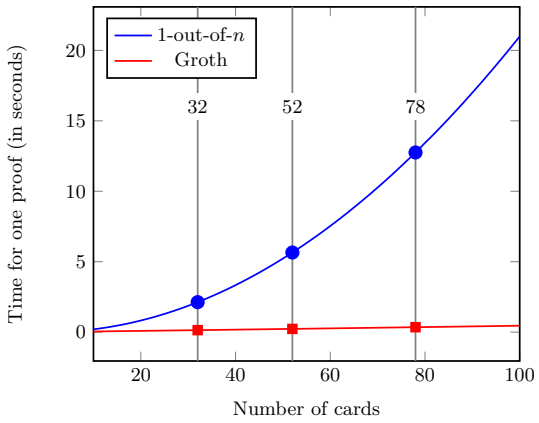

Fig. 2: Complexity Evaluation of the Zero-Knowledge Proofs for Proof 1 of our protocol varying from 10 to 100 cards.

We can refer to Table 1 for a benchmark of the ZK proofs. Instantiated with Design (1), we directly see that our total execution time is going to be unpractical as it takes more than 5 seconds to execute the proof of shuffle and that one proof needs to be executed for each of the players. Provided with a linear execution time, usage of Design (2) 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. The number of players does not influence the complexity of the ZK proofs and the execution time of the shuffle depends linearly on it.

Numerous multiplications by a scalar and zero-knowledge proofs (leading to more multiplications) are used in our protocol. We provide a theoretical comparison of our protocol with the Spades protocol of [6] in Table 2, which records the number of elliptic curve point multiplications at each step of Spades protocols. Note that totals for Play and Verif and a full execution of the game are well over-estimated as we consider the worst possible plays. It is important to note that the ZK proofs in DeckGen dominates the computation cost: The proofs of shuffle used in both [6] and Design (1) are the same *1-out-of-n* proofs as above, and lead to 43 160 multiplications by a scalar. With Design (2), we are thus able to drastically diminish the cost of the DeckGen procedure.

On the whole, a full execution of DeckGen can be performed with Design (1) in about 20 seconds of processor time for all 4 players, against 3 seconds with Design (2) at equivalent security level. It takes 178 milliseconds (ms) to recover each hand. A play takes about 270 ms of time and its verifications only 185 ms. This analysis shows that this protocol is the most efficient secure Spades protocol existing to date. Our Tarot protocol has a computation overhead of the same magnitude. Furthermore, these timings being of the same

	Design (1)		Design (2)	
	Prove	Verify	Prove	Verify
Execution of Proof 1	5.64 s	5.72 s	234.70 ms	175.23 ms
	Prove		Verify	
Execution of Proof 2	586.38μs		1.14 ms	
Execution of Proof 3	1.191 ms		2.69 ms	
Execution of Proof 4	104.06 ms		103.99 ms	

Table 1: Average execution time of the four ZKP for 52 cards.
Proofs' number refers to formulas' number of Section 5.
Note that (1) has similar execution time as previous work [6]

	KeyGen	DeckGen	GetHand	Play	Verif	GetSuit	Total for a full game
[6]	13	43 862	598	1802	1828	0	79 139
Design (1)	2	43 680	13	1802	1828	0	61 209
Design (2)	2	4 458	13	1802	1828	0	21 806

Table 2: Elliptic curve point multiplications in each of the Spades algorithms (for one player).

order of magnitude than 1 RTT (Round-Trip Time), the overhead brought by securing the protocol seems acceptable. Although the communication time has been ignored in this benchmark, some pre-computation during communication time, mostly of the proofs, may compensate this limitation.

8 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. Adam Barnett and Nigel P. Smart. Mental poker revisited. In *Cryptography and Coding, 9th IMA International Conference*. Springer, 2003.
2. Stephanie Bayer and Jens Groth. Efficient zero-knowledge argument for correctness of a shuffle. In *Advances in Cryptology – EUROCRYPT*. Springer, 2012.
3. Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. Public-key encryption in a multi-user setting: Security proofs and improvements. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000.
4. Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. Instantaneous decentralized poker. In *Advances in Cryptology – ASIACRYPT*. Springer, 2017.
5. Daniel J Bernstein. Curve25519: new diffie-hellman speed records. In *International Workshop on Public Key Cryptography*. Springer, 2006.
6. Xavier Bultel and Pascal Lafourcade. Secure trick-taking game protocols - how to play online spades with cheaters. In Ian Goldberg and Tyler Moore, editors, *FC 2019*. Springer, Heidelberg, 2019. `https://eprint.iacr.org/2019/375`.
7. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups. In *Advances in Cryptology — CRYPTO*. Springer, 1997.
8. Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology — CRYPTO*. Springer, 1994.

9. Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO*. Springer, 2012.
10. Bernardo David, Rafael Dowsley, and Mario Larangeira. 21-bringing down the complexity: fast composable protocols for card games without secret state. In *Australasian Conference on Information Security and Privacy*. Springer, 2018.
11. Bernardo David, Rafael Dowsley, and Mario Larangeira. Kaleidoscope: An efficient poker protocol with payment distribution and penalty enforcement. In *21st International Conference, FC*, 2018.
12. Bernardo David, Rafael Dowsley, and Mario Larangeira. Royale: A framework for universally composable card games with financial rewards and penalties enforcement. In *Financial Cryptography and Data Security*. Springer, 2019.
13. Jun Furukawa and Kazue Sako. An efficient scheme for proving a shuffle. In *Advances in Cryptology — CRYPTO*. Springer, 2001.
14. Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. Zkboo: Faster zero-knowledge for boolean circuits. In *Proceedings of the 25th USENIX Conference on Security Symposium*. USENIX Association, 2016.
15. Shafi Goldwasser and Silvio Micali. 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.
16. Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 1989.
17. Jens Groth. A verifiable secret shuffle of homomorphic encryptions. In *Journal of Cryptology*. Springer, 2010.
18. Nicholas D Matsakis and Felix S Klock II. The rust language. In *ACM SIGAda Ada Letters*. ACM, 2014.
19. Heiko Stamer. Bibliography on mental poker. `https://www.nongnu.org/libtmcg/MentalPoker.pdf`.
20. Yiannis Tsiounis and Moti Yung. On the security of elgamal based encryption. In *International Workshop on Public Key Cryptography*. Springer, 1998.
21. Tzer-jen Wei. Secure and practical constant round mental poker. In *Information Sciences*, 2014.
22. Jeff Yan. Collusion detection in online bridge. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI*. AAAI Press, 2010.
23. Weiliang Zhao, Vijay Varadharajan, and Yi Mu. A secure mental poker protocol over the internet. In *ACSW frontiers*, Conferences in research and practice in information technology. Australian Computer Society, 2003.

A Cryptographic Background

In this section, we give more details on the definitions, the DDH assumption and zero-knowledge proofs.

An asymmetric encryption scheme is a mechanism allowing a user to send a plaintext p to a receiver without revealing them to the public or any malicious actor. A sanitary recall is given in the upcoming definition.

Definition 5 (Asymmetric Encryption Scheme). *An encryption algorithm $\mathcal{E}$ is given by four algorithms:*

$\mathsf{Setup}(1^{\mathfrak{K}})$**.** *This algorithm outputs* params*, the global parameters of the scheme.*
$\mathsf{KeyGen}(\mathsf{params})$**.** *This algorithm outputs the keys* $(\mathsf{dk}, \mathsf{ek})$*.*
$\mathsf{Enc}_{\mathsf{ek}}(p)$**.** *For a given plaintext p, a ciphertext c is returned.*
$\mathsf{Dec}_{\mathsf{dk}}(s)$**.** *For a given ciphertext c, a plaintext p is returned.*

An asymmetric encryption scheme has the following properties:

Correctness. *For all $\mathfrak{K} \in \mathbb{N}$, for all possible outputs of* $\mathsf{Setup}(1^{\mathfrak{K}})$*, for all possible pair of keys* $(\mathsf{dk}, \mathsf{ek})$*,* $\mathsf{Dec}_{\mathsf{dk}}(\mathsf{Enc}_{\mathsf{ek}}(p)) = p$

Indistinguishability under Chosen Plaintext Attacks. *(IND-CPA)*
This property ensure that ciphertext leaks no information on the messages. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme and $\mathfrak{K}$ its security parameter. Take $\mathcal{A}$ a probabilistic polynomial time algorithm. Experiment $\mathsf{Exp}^{\mathit{IND\text{-}CPA}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$ is described in Figure 3. The property holds if for any polynomial time adversary $\mathcal{A}$, $\mathsf{Adv}^{\mathit{IND\text{-}CPA}}_{\mathcal{A}}(\mathfrak{K}) = |Pr[\mathsf{Exp}^{\mathit{IND\text{-}CPA}}_{\epsilon,\mathcal{A}}(\mathfrak{K}) = 1] - 1/2| \leq negl(\mathfrak{K})$.

$\mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{E},\mathcal{A}}(\mathfrak{K})$

$\mathsf{params} \leftarrow \mathsf{Setup}(1^{\mathfrak{K}})$
$(\mathsf{dk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$
$(m_0, m_1) \leftarrow \mathcal{A}(\mathsf{params}, \mathsf{ek})$
$b \xleftarrow{\$} \{0, 1\}$
$c \leftarrow \mathsf{Enc}_{\mathsf{ek}}(m_b)$
$b_* \leftarrow \mathcal{A}(c)$
Return $b = b_*$

Fig. 3: Experiment for IND-CPA.

The IND-CPA security of the ElGamal encryption scheme relies on the DDH problem. In the multi-users settings a reduction to the same problem exists. Let $\mathsf{Exp}^{n-\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\mathfrak{K})$ be the IND-CPA experiment with access to a polynomial amount $n(\mathfrak{K})$ of independent oracles all providing ElGamal ciphertexts for different keys or random elements depending on the value of an element b.

Property 1 (From [3]). Let $\mathcal{EG} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be the ElGamal public-key encryption scheme as described above. Then for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{D}$ such that for any $\mathfrak{K} \in \mathbb{N}$ and any polynomial $n(\mathfrak{K})$,

$$\mathsf{Adv}^{n-\mathsf{IND\text{-}CPA}}_{\mathcal{EG},\mathcal{A}}(\mathfrak{K}) \leq 2 \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{D}}(\mathfrak{K}) + \frac{1}{2^{\mathfrak{K}-1}}.$$

For $q_e(\mathfrak{K})$ the number of total queries to each ElGamal encryption's oracle, the running time of $\mathcal{D}$ is that of $\mathcal{A}$ plus $O(q_e(\mathfrak{K})n(\mathfrak{K}) \cdot T^{\mathsf{exp}}_{\mathcal{G}}(\mathfrak{K}))$. Where $T^{\mathsf{exp}}_{\mathcal{G}}(\mathfrak{K})$ represent the time for a single exponentiation in $\mathbb{G}$.

This property gives guarantees on $n-$ IND-CPA security of the ElGamal encryption used in our protocols, we latter extend this property to prove security of our protocols.

B Formal Security Model of Trick-Taking Games

Security of trick-taking protocols requires the simulation of the honest players based on their knowledge of the game (previous plays and their hand). This simulation is achieved using an algorithm called trick-taking strategy, which decides which card to play based on the game situation.

Definition 6. *A* trick-taking strategy *is a polynomial-time algorithm* Strat *that takes as input a tuple of cards* played *(which represents all cards played at some point in a trick-taking game) and a set of cards* hand *(which represents all cards of a player at the same point), a first player index p_*, a player index p, and that returns a card* $\mathsf{id} \in \mathsf{Hand}$ *which is valid according to the rules of the considered game.*

Based on [6] we define an experiment where a challenger simulates a Trick-taking protocol to an adversary. This experiment is used and refined to define the security properties of Trick-taking schemes: *theft-resistant*, *cheating-resistant*, *unpredictability*, *hand-privacy* and *game-privacy*. These properties define the attacks that should not be feasible by a polynomial-time adversary, regardless of the strategies used to simulate the other players.

Only the algorithms were modified from [6] for this experiment, the purpose of this game remains the same: simulating a Trick-taking scheme with an adversary. The adversary is allowed to take part of the game as one of the four players and to take an accomplice. Note that in this experiment the adversary has access to the private key of all players. The challenger cannot use the hand generation algorithm for the corrupted player, because he does not know its secret key; however, the challenger can deduce the adversary's hand because it contains the 13 cards that are not in the hand of the three other users. After generating a shuffled deck, the challenger and the adversary run the game phase, such that the adversary plays the role of the corrupted user and its accomplice.

Trick-taking Experiment:

Definition 7. *Let* $W = (\mathsf{Init}, \mathsf{KeyGen}, \mathsf{DeckGen}, \mathsf{GetHand}, \mathsf{Play}, \mathsf{Verif}, \mathsf{GetSuit})$ *be a Trick-taking scheme,* $S = (\mathsf{Strat}_1, \mathsf{Strat}_2, \mathsf{Strat}_3, \mathsf{Strat}_4)$ *be a tuple of strategies, and* $\mathfrak{K} \in \mathbb{N}$ *be a security parameter. Let* $\mathcal{A}$ *and* $\mathcal{C}$ *be two polynomial-time algorithms. The* Trick-taking experiment $\mathit{Exp}^{\mathsf{TrickT}}_{W,S,\mathcal{A}}(\mathfrak{K})$ *instantiated by* W *and* S *between the adversary* $\mathcal{A}$ *and the challenger* $\mathcal{C}$ *is defined as follows:*

Keys generation phase: $\mathcal{C}$ *runs* $\mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$*, sets* $\mathrm{st} = \perp$*, and sends the pair* $(\mathit{setup}, \mathrm{st})$ *to* $\mathcal{A}$*, which returns a corrupted user index* $i_c \in [\![1,4]\!]$*. For all* $i \in [\![1,4]\!] \setminus \{i_c\}$*,* $\mathcal{C}$ *runs* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathit{setup})$ *and sends* $(\mathsf{sk}_i, \mathsf{pk}_i)$ *to* $\mathcal{A}$*, which returns the public key* pk_{i_c} *and an accomplice index* i_a*.*

Shuffle phase: $\mathcal{C}$ *and* $\mathcal{A}$ *generate* PK *by running the algorithm* $\mathsf{DeckGen}$ *together, such that* $\mathcal{A}$ *plays the role of the players* P_{i_c} *and* P_{i_a}*, and* $\mathcal{C}$ *plays the role of the other players. If* $\mathsf{PK} = \perp$*, then* $\mathcal{C}$ *aborts and returns* 0*. Following this step* $\mathcal{C}$ *execute* $H_i \leftarrow \mathsf{GetHand}(i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{PK})$ *for* $i \in [\![1,4]\!] \setminus \{i_c\}$*, and set* $H_{i_c} = \{\mathsf{id}_i\}_{1 \le i \le 52} \setminus (\cup_{i=1; i \neq i_c}^{4} H_i)$*.*

Game phase: $\mathcal{C}$ *initializes the first player index* $p_* = 1$*, the current player index* $p = p_*$*, the corrupted play index* $\gamma = 0$*, and* $\mathsf{played} = \perp$*. For* $j \in [\![1,52]\!]$*:*

- **If** $p \neq i_c$ **and** $p \neq i_a$**:** $\mathcal{C}$ *runs* $\mathsf{id} \leftarrow \mathsf{Strat}_p(\mathsf{played}, H_p, p_*, p)$*, then* $\mathcal{C}$ *runs* $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK})$*.* $\mathcal{C}$ *sends* $(\mathsf{id}, \Pi, \mathrm{st}')$ *to* $\mathcal{A}$ *and updates* $\mathrm{st} := \mathrm{st}'$*.*
- **If** $p = i_a$**:** $\mathcal{C}$ *receives* $(\mathsf{id}, \Pi, \mathrm{st}')$ *from* $\mathcal{A}$*. If* $\mathsf{Verif}(i_a, \mathsf{id}, \Pi, \mathsf{pk}_{i_a}, \mathrm{st}, \mathrm{st}', \mathsf{PK}) = 0$*, then* $\mathcal{C}$ *aborts and the experiment returns* 0*. Else,* $\mathcal{C}$ *updates* $\mathrm{st} := \mathrm{st}'$*.*
- **If** $p = i_c$**:** $\mathcal{C}$ *increments* $\gamma := \gamma + 1$*, then receives* $(\mathsf{id}, \Pi, \mathrm{st}')$ *from* $\mathcal{A}$ *and sets* $(\mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}) = (\mathsf{id}, \Pi)$*.* $\mathcal{C}$ *sets* $\mathrm{st}_\gamma = \mathrm{st}$ *and* $\mathrm{st}'_\gamma = \mathrm{st}'$*.* $\mathcal{C}$ *sets* $\mathsf{suit}_{i_c,\gamma} = \mathsf{GetSuit}(\mathrm{st})$*. If* $\mathsf{Verif}(i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK}) = 0$*, then* $\mathcal{C}$ *aborts and the experiment returns* 0*. Else,* $\mathcal{C}$ *updates* $\mathrm{st} := \mathrm{st}'$*.*

$\mathcal{C}$ *then updates the index* p *that points to the next player according to the rule of the Trick-taking game, parses* played *as* $(\mathsf{pl}_1, \ldots, \mathsf{pl}_n)$ *(where* $n = |\mathsf{played}|$*) and updates* $\mathsf{played} := (\mathsf{pl}_1, \ldots, \mathsf{pl}_n, \mathsf{id})$*.*

Final phase: *The experiment returns* 1*.*

Theft-resistant

Definition 8. *A Trick-taking scheme* W *is said to be* theft-resistant *if for any tuple of strategies* $S = (\mathsf{Strat}_1, \mathsf{Strat}_2, \mathsf{Strat}_3, \mathsf{Strat}_4)$ *and any polynomial-time adversary* $\mathcal{A}$ *which plays the Trick-taking experiment instantiated by* W *and* S*, the probability that there exists* $\gamma \in [\![1,13]\!]$ *verifying the two following conditions is negligible.*

- $\mathsf{Verif}(i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK}) = 1$*, i.e., the* γ^{th} *play of the adversary is accepted for the card* $\mathsf{id}_{i_c,\gamma}$ *and*
- $\mathsf{id}_{i_c,\gamma} \notin H_{i_c}$*, i.e., the card* $\mathsf{id}_{i_c,\gamma}$ *is not in the adversary's hand.*

Cheating-resistant

Definition 9. *A Trick-taking scheme* W *is said to be* cheating-resistant *if for any tuple of strategies* $S = (\mathsf{Strat}_1, \mathsf{Strat}_2, \mathsf{Strat}_3, \mathsf{Strat}_4)$ *and any polynomial-time adversary* $\mathcal{A}$ *which plays the Trick-taking experiment instantiated by* W *and* S*, the probability that there exists* $\gamma \in [\![1,13]\!]$ *verifying the following is negligible.*

- $\mathsf{Verif}(i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK}) = 1$*, i.e., the* γ^{th} *play of the adversary is accepted for the card* $\mathsf{id}_{i_c,\gamma}$*;*
- $\mathsf{id}_{i_c,\gamma}.\mathsf{suit} \neq \mathsf{suit}_{i_c,\gamma}$ *and* $\mathsf{suit}_{i_c,\gamma} \neq \perp$ *i.e., the suit of the card* $\mathsf{id}_{i_c,\gamma}$ *is not the leading suit;*
- *There exists* $\mathsf{id} \in H_{i_c}$ *such that:* $\forall\ l \le \gamma, \mathsf{id}_{i_c,l} \neq \mathsf{id}$ *and* $\mathsf{id}.\mathsf{suit} = \mathsf{suit}_{i_c,\gamma}$*. i.e., the adversary has a card of the leading suit in its hand that was not already played before the* γ^{th} *play.*

Unpredictable

Definition 10. *Let* W *be a Trick-taking scheme and* $\mathfrak{K} \in \mathbb{N}$ *be a security parameter. Let* $\mathcal{A}$ *and* $\mathcal{C}$ *be two polynomial time algorithms. The* unpredictable experiment $\mathit{Exp}^{unp}_{W,\mathcal{A}}(\mathfrak{K})$ *instantiated by* W *between the adversary* $\mathcal{A}$ *and the challenger* $\mathcal{C}$ *is defined as follows:*

Keys generation phase: $\mathcal{C}$ *runs* $\mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$, *sets* $\mathrm{st} = \perp$, *and sends the pair* $(\mathit{setup}, \mathrm{st})$ *to* $\mathcal{A}$, *who returns a corrupted user index* $i_c \in [\![1,4]\!]$. *For all* $i \in [\![1,4]\!] \setminus \{i_c\}$, $\mathcal{C}$ *runs* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathit{setup})$. *The challenger sends* $\{\mathsf{pk}_i\}_{i \in [\![1,4]\!] \setminus \{i_c\}}$ *to* $\mathcal{A}$, *who returns its public key* pk_{i_c} *and an accomplice index* i_a. $\mathcal{C}$ *finally sends* sk_{i_a} *to* $\mathcal{A}$ *who answer is a card* id *and a player index* $i_{\mathit{guess}} \in \{1,2,3,4\}$.

Shuffle phase: $\mathcal{C}$ *and* $\mathcal{A}$ *generate* PK *by running the algorithm* $\mathsf{DeckGen}$ *together, such that* $\mathcal{A}$ *plays the role of the players* P_{i_c} *and* P_{i_a}, *and* $\mathcal{C}$ *plays the role of the other players. If* $\mathsf{PK} = \perp$, *then* $\mathcal{C}$ *aborts and returns a random bit following a Bernoulli distribution with parameter* $1/4$. *Following this step* $\mathcal{C}$ *execute* $H_i \leftarrow \mathsf{GetHand}(i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{PK})$ *for* $i \in [\![1,4]\!] \setminus \{i_c\}$, *and set* $H_{i_c} = \{\mathsf{id}_i\}_{1 \le i \le 52} \setminus (\cup_{i=1; i \neq i_c}^{4} H_i)$.

Final phase: *If* id *is in* $H_{i_{\mathit{guess}}}$, *then* $\mathcal{C}$ *returns* 1, *else it returns* 0.

Definition 11. *The Trick-taking scheme* W *is said to be* unpredictable *if for any adversary playing* $\mathit{Exp}^{unp}_{W,\mathcal{A}}(\mathfrak{K})$, *the probability that* $\mathit{Exp}^{unp}_{W,\mathcal{A}}(\mathfrak{K})$ *returns 1 is negligibly close to* $1/4$.

Hand-Privacy

Definition 12. *Let* $W = (\mathsf{Init}, \mathsf{KeyGen}, \mathsf{DeckGen}, \mathsf{GetHand}, \mathsf{Play}, \mathsf{Verif}, \mathsf{GetSuit})$ *be a Trick-taking scheme and* $k \in \mathbb{N}$ *be a security parameter. Let* $\mathcal{A}$ *and* $\mathcal{C}$ *be two polynomial time algorithms. The* hand experiment $\mathit{Exp}^{\mathsf{Hand}}_{W,\mathcal{A}}(\mathfrak{K})$ *instantiated by* W *between the adversary* $\mathcal{A}$ *and the challenger* $\mathcal{C}$ *is defined by:*

Key generation phase: $\mathcal{C}$ *runs* $\mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$. *It sets* $\mathrm{st} = \perp$. *It sends the pair* $(\mathit{setup}, \mathrm{st})$ *to* $\mathcal{A}$, *who returns* $i_c \in [\![1,4]\!]$. *For all* $i \in [\![1,4]\!] \setminus \{i_c\}$, $\mathcal{C}$ *runs* $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathit{setup})$ *and sends* pk_i *to* $\mathcal{A}$, *who returns* pk_{i_c}.

Shuffle phase: $\mathcal{C}$ *and* $\mathcal{A}$ *generate* PK *by running the algorithm* $\mathsf{DeckGen}$ *together, such that* $\mathcal{A}$ *plays the role of* P_{i_c}, *and* $\mathcal{C}$ *plays the role of the three other players. If* $\mathsf{PK} = \perp$, *then* $\mathcal{C}$ *aborts and returns a random bit with uniform distribution. For all* $i \in [\![1,4]\!] \setminus \{i_c\}$, $\mathcal{C}$ *runs* $H_i \leftarrow \mathsf{GetHand}(i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{PK})$, *and sets* $H_{i_c} = \{\mathsf{id}_i\}_{1 \le i \le 52} \setminus (\cup_{i=1; i \neq i_c}^{4} H_i)$.

Challenge phase: $\mathcal{C}$ *picks* θ_0 *and* θ_1 *in* $[\![1,4]\!] \setminus \{i_c\}$ *such that* $\theta_0 \neq \theta_1$. $\mathcal{C}$ *picks* $b \xleftarrow{\$} \{0,1\}$ *and* $\mathsf{id} \xleftarrow{\$} H_{\theta_b}$, *and sends* (id, θ_0, θ_1) *to* $\mathcal{A}$, *who returns* b_*.

Final phase: *If* $b = b_*$, *then* $\mathcal{C}$ *returns* 1, *else it returns* 0.

Definition 13. *A Trick-taking scheme* W *is* hand-private *if for any polynomial-time adversary* $\mathcal{A}$ *which plays* $\mathit{Exp}^{\mathsf{Hand}}_{W,\mathcal{A}}(\mathfrak{K})$, *the probability that the experiment returns* 1 *is negligibly closed to* $1/2$.

Game-privacy

Definition 14. *For any* $\mathfrak{K} \in \mathbb{N}$, *any Trick-taking scheme* W, *any quadruplet of strategies* S, *any adversary* $\mathcal{D}$ *and any element* $K = \left(\mathit{setup}, \{\mathsf{sk}_i, \mathsf{pk}_i\}_{1 \le i \le 4; i \neq i_c}, \mathsf{PK}\right)$, $Exp^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}}(\mathfrak{K})$ *denotes the same experiment as* $Exp^{\mathsf{TrickT}}_{W,S,\mathcal{D}}(\mathfrak{K})$ *except:*

1. *The challenger and the adversary use the setup and the keys in* K *instead of generating fresh setup and keys during the experiment.*
2. $\mathcal{A}$ *has no accomplice.*

A Trick-taking scheme W *is said to be* game-private *if there exists a polynomial-time simulator* Sim *such that for any tuple of strategies* S *and any polynomial-time* 5*-party algorithm* $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5)$, $|\mathsf{P}_{\mathsf{real}}(\mathcal{D}, k) - \mathsf{P}_{\mathsf{sim}}(\mathcal{D}, k)|$ *is negligible, where*

$$\mathsf{P}_{\mathsf{real}}(\mathcal{D}, k) =$$

$$Pr\left[\begin{array}{l} \mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K}); i_c \leftarrow \mathcal{D}_1(\mathit{setup}); \\ \forall i \in [\![1,4]\!] \setminus \{i_c\}, (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathit{setup}); \\ \mathsf{pk}_{i_c} \leftarrow \mathcal{D}_2(\mathit{setup}, \{\mathsf{pk}_i\}_{i \in [\![1,4]\!] \setminus \{i_c\}}, \mathsf{vw}); \\ 1 \leftarrow \mathcal{D}_5(\mathsf{vw}) : \mathsf{PK} \leftarrow \mathsf{DeckGen}_{\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_4} \mathit{where}\ \mathsf{P}_{i_c} = \mathcal{D}_3(\mathsf{vw}); \\ K := (\mathit{setup}, \mathsf{pk}_{i_c}, \{(\mathsf{sk}_i, \mathsf{pk}_i)\}_{1 \le i \le 4; i \neq i_c}, \mathsf{PK}); \\ \mathit{If}\ \mathsf{PK} = \perp, \mathsf{vw} := \perp; \\ \mathit{Else}\ b \leftarrow \mathit{Exp}^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}_4(\mathsf{vw})}(\mathfrak{K}); \end{array}\right]$$

$$\mathsf{P}_{\mathsf{sim}}(\mathcal{D},k) = \\ Pr\left[1 \leftarrow \mathcal{D}_5(\mathsf{vw}) : \begin{array}{l} \mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K}); i_c \leftarrow \mathcal{D}_1(\mathit{setup}); \\ \forall i \in [\![1,4]\!] \setminus \{i_c\}, (\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{KeyGen}(\mathit{setup}); \\ \mathsf{pk}_{i_c} \leftarrow \mathcal{D}_2(\mathit{setup}, \{\mathsf{pk}_i\}_{i\in[\![1,4]\!]\setminus\{i_c\}}, \mathsf{vw}); \\ \mathsf{PK} \leftarrow \mathsf{DeckGen}_{\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_4} \textit{ where } \mathsf{P}_{i_c} = \mathcal{D}_3(\mathsf{vw}); \\ \textit{If } \mathsf{PK} = \perp, \mathsf{vw} := \perp; \\ \textit{Else } b \leftarrow \mathsf{Sim}^{\mathsf{TrickT}}_{W,S,\mathcal{D}_4(\mathsf{vw})}(k, \mathit{setup}, i_c, \{\mathsf{pk}_i,\}_{1\le i\le 4}, \mathsf{PK}, \mathsf{vw}); \end{array}\right]$$

and where vw *denotes the view of* $\mathcal{D}$*, i.e., all the values sent and received by each algorithm of* $\mathcal{D}$ *during its interaction with the experiment.*

C Adaptation of the Security Model to the French Tarot

In this section, a detail version of the model highlighted in Section 4.2 is given. This means giving a formal definition of the experiments associated to each of the following properties: *theft-resistant*, *cheating-resistant*, *Unpredictability hand-privacy*, *game-privacy* and *dog security*. For this purpose we uses Honest players simulations is defined in Section B, these algorithms are now considered in the context of French Tarot. Also, bets are needed in Tarot, in our model, the taker and its bet is set up to the adversary's choice.

Trick-taking Experiment:

Definition 15. *Consider a Trick-Taking scheme* $W = (\mathsf{Init}, \mathsf{KeyGen}, \mathsf{DeckGen}, \mathsf{GetHand}, \mathsf{Play}, \mathsf{Verif}, \mathsf{GetSuit})$*, a tuple of strategies* $S = (\mathsf{Strat}_i)_{i=1}^4$*, and a security parameter* $\mathfrak{K} \in \mathbb{N}$*. Let* $\mathcal{A}$ *and* $\mathcal{C}$ *be two polynomial-time algorithms. The* Trick-Taking experiment $\mathit{Exp}^{\mathsf{TrickT}}_{W,S,\mathcal{A}}(\mathfrak{K})$ *instantiated by* W *and* S *between the adversary* $\mathcal{A}$ *and the challenger* $\mathcal{C}$ *is defined as follows:*

Keys generation phase: $\mathcal{C}$ *runs* $\mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$*, sets* $\mathrm{st} = \perp$*, and sends the pair* $(\mathit{setup}, \mathrm{st})$ *to* $\mathcal{A}$*, which returns a corrupted user index* $i_c \in [\![1,4]\!]$*. For all* $i \in [\![1,4]\!] \setminus \{i_c\}$*,* $\mathcal{C}$ *runs* $(\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}) \leftarrow \mathsf{KeyGen}(\mathit{setup})$ *and sends* $\{\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{i\in[\![1,4]\!]\setminus\{i_c\}}$ *to* $\mathcal{A}$*, which returns the public key* $(\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}})$ *and two accomplice indies* i_a^1 *and* i_a^2*.* $\mathcal{C}$ *verifies the latest proof and extracts the secret key* sk_{i_c} *from* $\mathcal{A}$*'s proof* $\Pi_{\mathsf{pk}_{i_c}}$*.* $\mathcal{C}$ *finally sends* $\mathsf{sk}_{i_a^1}$ *and* $\mathsf{sk}_{i_a^2}$ *to* $\mathcal{A}$*.*

Shuffle phase: $\mathcal{C}$ *and* $\mathcal{A}$ *generate* PK *by running the algorithm* DeckGen *together, such that* $\mathcal{A}$ *plays the role of the players* P_{i_c} *and the two accomplice indies* i_a^1 *and* i_a^2 $\mathcal{C}$ *plays the role of the other players. If* $\mathsf{PK} = \perp$*, then* $\mathcal{C}$ *aborts and returns* 0*. Following this step* $\mathcal{C}$ *execute* $H_i \leftarrow \mathsf{GetHand}(i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{PK})$ *for* $i \in [\![1,4]\!]$*.* $\mathcal{A}$ *return an index* i_{Dog} *and a binary value* $d \in \{0,1\}$*. Protocol* MakeDog *is executed with* $\mathcal{C}$ *based on this index if* $d = 1$*. If* $\mathcal{C}$ *is the taker, then it draws random permutation until it attains a valid setup. After this execution, it updates the hand of player* $\mathsf{P}_{i_{Dog}}$ *with* GetHand*.*

Game phase: $\mathcal{C}$ *initializes the first player index* $p_* = 1$*, the current player index* $p = p_*$*, the corrupted play index* $\gamma = 0$*, and* $\mathsf{played} = \emptyset$*. For* $j \in [\![1, n_D]\!]$*:*

- **If** $p \neq i_c, i_a^1, i_a^2$**:** $\mathcal{C}$ *runs* $\mathsf{id} \leftarrow \mathsf{Strat}_p(\mathsf{played}, H_p, p_*, p)$*, then* $\mathcal{C}$ *runs* $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK})$*. Challenger* $\mathcal{C}$ *sends* $(\mathsf{id}, \Pi, \mathrm{st}')$ *to* $\mathcal{A}$ *and updates* $\mathrm{st} := \mathrm{st}'$*.*
- **If** $p = i_a^1$ **or** $p = i_a^2$**:** $\mathcal{C}$ *receives* $(\mathsf{id}, \Pi, \mathrm{st}')$ *from* $\mathcal{A}$*. If* $\mathsf{Verif}(i_a, \mathsf{id}, \Pi, \mathsf{pk}_{i_a}, \mathrm{st}, \mathrm{st}', \mathsf{PK}) = 0$*, then* $\mathcal{C}$ *aborts and the experiment returns* 0*. Else,* $\mathcal{C}$ *updates* $\mathrm{st} := \mathrm{st}'$*.*
- **If** $p = i_c$**:** $\mathcal{C}$ *increments* $\gamma := \gamma + 1$*, then receives* $(\mathsf{id}, \Pi, \mathrm{st}')$ *from* $\mathcal{A}$ *and sets* $(\mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}) = (\mathsf{id}, \Pi)$*.* $\mathcal{C}$ *sets* $\mathrm{st}_\gamma = \mathrm{st}$ *and* $\mathrm{st}'_\gamma = \mathrm{st}'$*.* $\mathcal{C}$ *sets* $\mathsf{suit}_{i_c,\gamma} = \mathsf{GetSuit}(\mathrm{st})$*. If* $\mathsf{Verif}(i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK}) = 0$*, then* $\mathcal{C}$ *aborts and the experiment returns* 0*. Else,* $\mathcal{C}$ *updates* $\mathrm{st} := \mathrm{st}'$*.*

$\mathcal{C}$ *then updates the index* p *that points to the next player according to the rule of Trick-taking game, parses* played *as* $(\mathsf{pl}_1, \ldots, \mathsf{pl}_n)$ *(where* $n = |\mathsf{played}|$*) and updates* $\mathsf{played} := (\mathsf{pl}_1, \ldots, \mathsf{pl}_n, \mathsf{id})$*.*

Final phase: *Then return* 1*.*

For the winning condition of *theft-resistant*, *cheating-resistant* refer to Definitions 8 and 9.

Unpredictability:

Definition 16. *Let W be a Trick-Taking scheme based on a security parameter $\mathfrak{K} \in \mathbb{N}$. Let $\mathcal{A}$ and $\mathcal{C}$ be two polynomial time algorithms, let $i_{guess} \in \{1, \ldots, 5\}$ referring to the four players and index 5 to the dog. The* unpredictable experiment *$Exp^{unp}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$ instantiated by W between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ is defined as follows:*

Keys generation phase: *$\mathcal{C}$ runs $setup \leftarrow \mathsf{Init}(\mathfrak{K})$, sets $\mathrm{st} = \perp$, and sends the pair $(setup, \mathrm{st})$ to $\mathcal{A}$, which returns a corrupted user index $i_c \in [\![1, 4]\!]$. For all $i \in [\![1, 4]\!] \setminus \{i_c\}$, $\mathcal{C}$ runs $(\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}) \leftarrow \mathsf{KeyGen}(setup)$. The challenger sends $\{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{i \in [\![1,4]\!] \setminus \{i_c\}}$ to $\mathcal{A}$, which returns its public information $(\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}})$ and two accomplice indies i^1_a and i^2_a. $\mathcal{C}$ verifies the latest proof and extracts the secret key sk_{i_c} from $\mathcal{A}$'s proof $\Pi_{\mathsf{pk}_{i_c}}$. $\mathcal{C}$ finally sends $\mathsf{sk}_{i^1_a}$ and $\mathsf{sk}_{i^2_a}$ to $\mathcal{A}$, the latest answers a card id.*

Shuffle phase: *$\mathcal{C}$ and $\mathcal{A}$ generate PK by running the algorithm $\mathsf{DeckGen}$ together, such that $\mathcal{A}$ plays the role of the players P_{i_c}, $\mathsf{P}_{i^1_a}$ and $\mathsf{P}_{i^2_a}$, and $\mathcal{C}$ plays the role of the other players. If $\mathsf{PK} = \perp$, then $\mathcal{C}$ aborts and returns 0. Following this step $\mathcal{C}$ execute $H_{i_{guess}} \leftarrow \mathsf{GetHand}(i_{guess}, \mathsf{sk}_{i_{guess}}, \mathsf{pk}_{i_{guess}}, \mathsf{PK})$. On aborts, $\mathcal{C}$ returns a random bit following a Bernoulli distribution with parameter $n_h^{i_{guess}}/n_D$.*

Final phase: *If id is in $H_{i_{guess}}$, then $\mathcal{C}$ returns 1, else it returns 0.*

Definition 17. *A Trick-Taking scheme W is said to be* unpredictable *if for any adversary playing $Exp^{unp}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$ for all indices $i_{guess} \in \{1, \ldots, 5\}$, the sum $\sum_{i_{guess}=1}^{5} \left(\Pr[Exp^{unp}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})] - n_h^{i_{guess}}/n_D \right)$ is negligible.*

Hand-privacy:

Definition 18. *Let W be a Trick-Taking scheme and $\mathfrak{K} \in \mathbb{N}$ be a security parameter. Let $\mathcal{A}$ and $\mathcal{C}$ be two polynomial time algorithms. The* hand-privacy experiment *$Exp^{\mathsf{Hand}}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$ instantiated by W between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ and for an index $i_{guess} \in \{1, \ldots, 5\}$, is defined by:*

Key generation phase: *$\mathcal{C}$ runs $setup \leftarrow \mathsf{Init}(\mathfrak{K})$. It sets $\mathrm{st} = \perp$. It sends the pair $(setup, \mathrm{st}, i_{guess})$ to $\mathcal{A}$, which returns $i_c \in [\![1, 4]\!] \setminus \{i_{guess}\}$. For all $i \in [\![1, 4]\!] \setminus \{i_c\}$, $\mathcal{C}$ runs $(\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}) \leftarrow \mathsf{KeyGen}(setup)$ and sends the pairs $\{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{i \in [\![1,4]\!] \setminus \{i_c\}}$ to $\mathcal{A}$, which returns $(\mathsf{pk}_i, \Pi_{\mathsf{pk}_i})$. $\mathcal{C}$ verifies the latest proof and extracts P_{i_c}'s secret key if $i_{guess} = 5$.*

Shuffle phase: *$\mathcal{C}$ and $\mathcal{A}$ generate PK by running the algorithm $\mathsf{DeckGen}$ together, such that $\mathcal{A}$ plays the role of P_{i_c}, and $\mathcal{C}$ plays the role of the three other players. If $\mathsf{PK} = \perp$, then $\mathcal{C}$ aborts and returns a random bit following a Bernoulli distribution with parameter $n_h^{i_{guess}}/(n_D - n_h^{i_c})$. $\mathcal{C}$ runs $H_i \leftarrow \mathsf{GetHand}(i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{PK})$ for $i = i_{guess}$ if $i_{guess} \neq 5$, otherwise for all $i \in [\![1, 4]\!]$ and thus recovers $H_{i_{guess}} = D \setminus \cup_{i=1}^{4} H_i$.*

Challenge phase: *$\mathcal{C}$ picks $b \xleftarrow{\$} \{0, 1\}$ and $\mathsf{id} \xleftarrow{\$} H_{i_{guess}}$ if $b = 0$ or $\mathsf{id} \xleftarrow{\$} D \setminus H_{i_{guess}}$ and sends id to $\mathcal{A}$, which returns b_*.*

Final phase: *$\mathcal{C}$ returns $b = b_*$.*

Based on the described game, we put forward the following definition for the security of *hand-private* in the context of Tarot schemes.

Definition 19. *A Trick-Taking scheme W is* hand-private *if for any polynomial-time adversary $\mathcal{A}$ which plays $Exp^{\mathsf{Hand}}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$ for all indices $i_{guess} \in \{1, \ldots, 5\}$, the sum $\sum_{i_{guess}=1}^{5} \left(\Pr[Exp^{\mathsf{Hand}}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})] - n_h^{i_{guess}}/(n_D - n_h^{i_c}) \right)$ is negligible.*

Game-privacy: The modifications on this property allows complying with a model fitting the Tarot scheme. In this definition, we will use $K = \left(\mathsf{setup}, \left\{\mathsf{pk}_i, \mathsf{sk}_i, \Pi_{\mathsf{pk}_i}\right\}_{1 \leq i \leq 4; i \neq i_c}, (\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}}), \mathsf{PK}\right)$ and add up the necessary algorithms into the sequence of execution.

Definition 20. *For any $\mathfrak{K} \in \mathbb{N}$, any Trick-Taking scheme W, any quadruplet of strategies S, any adversary $\mathcal{D}$ and any element $K = \left(setup, (\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}}), \left\{\mathsf{pk}_i, \mathsf{sk}_i, \Pi_{\mathsf{pk}_i}\right\}_{1 \leq i \leq 4; i \neq i_c}, \mathsf{PK}\right)$, $Exp^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}}(\mathfrak{K})$ denotes the same experiment as $Exp^{\mathsf{TrickT}}_{W,S,\mathcal{D}}(\mathfrak{K})$ except:*

1. *The challenger and the adversary use the setup and the keys in K instead of generating fresh setup and keys during the experiment.*
2. *$\mathcal{A}$ has no accomplice.*

A Trick-Taking scheme W is said to be game-private *if there exists a polynomial time simulator* Sim *such that for any tuple of strategies S and any polynomial time 5-party algorithm $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6)$, $|\mathsf{P}_{\mathsf{real}}(\mathcal{D}, k) - \mathsf{P}_{\mathsf{sim}}(\mathcal{D}, k)|$ is negligible, where*

$$\mathsf{P}_{\mathsf{real}}(\mathcal{D}, k) =$$
$$Pr\left[1 \leftarrow \mathcal{D}_6(\mathsf{vw}) : \begin{array}{l} \mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K}); i_c \leftarrow \mathcal{D}_1(\mathit{setup}); \\ \forall i \in [\![1,4]\!] \setminus \{i_c\}, (\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}) \leftarrow \mathsf{KeyGen}(\mathit{setup}); \\ (\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}}) \leftarrow \mathcal{D}_2(\mathit{setup}, \{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{i \in [\![1,4]\!] \setminus \{i_c\}}, \mathsf{vw}); \\ \mathsf{PK} \leftarrow \mathsf{DeckGen}_{\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_4} \textit{where } \mathsf{P}_{i_c} \textit{ is } \mathcal{D}_3(\mathsf{vw}); 0/1, n \leftarrow \mathcal{D}_3(\mathsf{vw}); \\ (a = 0/1, n) \leftarrow \mathcal{D}_3(\mathsf{vw}); \\ if\, a = 1\text{: } \mathsf{PK} \leftarrow \mathsf{MakeDog}_{\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_4}(n, \mathsf{PK}) \textit{where } \mathsf{P}_{i_c} \textit{ is } \mathcal{D}_4(\mathsf{vw}); \\ K := (\mathit{setup}, (\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}}), \{(\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i})\}_{1 \leq i \leq 4; i \neq i_c}, \mathsf{PK}); \\ \textit{If } \mathsf{PK} = \perp, \mathsf{vw} := \perp; \\ \textit{Else } b \leftarrow \mathit{Exp}^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}_5(\mathsf{vw})}(\mathfrak{K}); \end{array}\right]$$

$$\mathsf{P}_{\mathsf{sim}}(\mathcal{D}, k) =$$
$$Pr\left[1 \leftarrow \mathcal{D}_6(\mathsf{vw}) : \begin{array}{l} \mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K}); i_c \leftarrow \mathcal{D}_1(\mathit{setup}); \\ \forall i \in [\![1,4]\!] \setminus \{i_c\}, (\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}) \leftarrow \mathsf{KeyGen}(\mathit{setup}); \\ (\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}}) \leftarrow \mathcal{D}_2(\mathit{setup}, \{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{i \in [\![1,4]\!] \setminus \{i_c\}}, \mathsf{vw}); \\ \mathsf{PK} \leftarrow \mathsf{DeckGen}_{\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_4} \textit{where } \mathsf{P}_{i_c} \textit{ is } \mathcal{D}_3(\mathsf{vw}); (a = 0/1, n) \leftarrow \mathcal{D}_3(\mathsf{vw}); \\ if\, a = 1\text{: } \mathsf{PK} \leftarrow \mathsf{MakeDog}_{\mathsf{P}_1,\mathsf{P}_2,\mathsf{P}_3,\mathsf{P}_4}(n, \mathsf{PK}) \textit{where } \mathsf{P}_{i_c} \textit{ is } \mathcal{D}_4(\mathsf{vw}); \\ \textit{If } \mathsf{PK} = \perp, \mathsf{vw} := \perp; \\ \textit{Else } b \leftarrow \mathsf{Sim}^{\mathsf{TrickT}}_{W,S,\mathcal{D}_5(\mathsf{vw})}(k, \mathit{setup}, i_c, \{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{1 \leq i \leq 4}, \mathsf{PK}, \mathsf{vw}); \end{array}\right]$$

and where vw *denotes the view of $\mathcal{D}$, i.e., all the values sent and received by each algorithm of $\mathcal{D}$ during his interaction with the experiment.*

Dog Security:

Definition 21. *Let W be a trick-taking scheme with a dog parametrised by a security parameter $\mathfrak{K} \in \mathbb{N}$. Let $\mathcal{A}$ and $\mathcal{C}$ be two polynomial time algorithms. The* dog experiment $\mathit{Exp}^{Dog}_{W,\mathcal{A}}(\mathfrak{K})$ *instantiated by W between the adversary $\mathcal{A}$ and the challenger $\mathcal{C}$ is defined as follows:*

Keys generation phase: *$\mathcal{C}$ runs $\mathit{setup} \leftarrow \mathsf{Init}(\mathfrak{K})$, sets $\mathrm{st} = \perp$, and sends the pair $(\mathit{setup}, \mathrm{st})$ to $\mathcal{A}$, who returns a corrupted user index $i_c \in [\![1,4]\!]$. For all $i \in [\![1,4]\!] \setminus \{i_c\}$, $\mathcal{C}$ runs $(\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{pk}_i}) \leftarrow \mathsf{KeyGen}(\mathit{setup})$. The challenger sends $\{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{i \in [\![1,4]\!] \setminus \{i_c\}}$ to $\mathcal{A}$, who returns $(\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}})$ and two accomplice indices i^1_a, i^2_a. $\mathcal{C}$ verifies the latest proof and extracts the secret key sk_{i_c} from $\mathcal{A}$'s proof $\Pi_{\mathsf{pk}_{i_c}}$. If both operations succeeded $\mathcal{C}$ finally sends $\mathsf{sk}_{i^1_a}$ and $\mathsf{sk}_{i^2_a}$ to $\mathcal{A}$.*

Shuffle phase: *$\mathcal{C}$ and $\mathcal{A}$ generate PK by running the algorithm $\mathsf{DeckGen}$ together, such that $\mathcal{A}$ plays the role of the players P_{i_c} ,$\mathsf{P}_{i^1_a}$ and $\mathsf{P}_{i^2_a}$, and $\mathcal{C}$ plays the role of the remaining player. If $\mathsf{PK} = \perp$ then $\mathcal{C}$ aborts and returns 0. Then $\mathcal{A}$ outputs n and both executes $\mathsf{MakeDog}(n, \mathsf{PK})$ to output PK'. Following this step $\mathcal{C}$ executes $H_i \leftarrow \mathsf{GetHand}(i, \mathsf{sk}_i, \mathsf{pk}_i, \mathsf{PK}')$ for $i \in [\![1,4]\!]$.*

Final phase: *If for all $\mathsf{id} \in D \setminus \cup_{i=1}^4 H_i$, $\mathsf{id} \notin \mathsf{O}$, $\mathcal{C}$ returns 1, else it returns 0.*

We say that a tarot protocol is dog-secure *if a polynomial adversary has only negligible advantage to win the above experiment.*

D Detailed Tarot Scheme

In this section we formally describe all algorithms of the French Tarot scheme of Definition 4, this in order to prove the security the protocol in Section F.

Definition 22. *Let $\mathfrak{K} \in \mathbb{N}$ be a security parameter. The Tarot protocol is a tulpe of algorithms composed by the following algorithms:*

$\mathsf{Init}(\mathfrak{K})$: *Same as in Definition 3.*

$\mathsf{KeyGen}(\mathsf{setup})$: *It picks $\mathsf{dk} \xleftarrow{\$} \mathbb{Z}_q^*$ and computes $\mathsf{ek} = g^{\mathsf{dk}}$. Then a proof of knowledge $\Pi_{\mathsf{ek}} = \mathsf{ZK}\{\mathsf{dk} : \mathsf{ek} = g^{\mathsf{dk}}\}$ 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 its secret keys and the public keys of all the players $(\mathsf{pk}_i, \{\mathsf{pk}_l\}_{1\leq l\leq 4})$. This protocol returns a game public key PK, or $\perp$.*

Phase 1:

- *The canonical deck $D \in \mathsf{Decks}$ is initialized by each player.*
- *Each user parses $D = (\mathsf{id}_1, \ldots, \mathsf{id}_{78})$ and computes $\mathsf{pk} = \prod_{i=1}^{4} \mathsf{pk}_i$, then for all $j \in [\![1,78]\!]$ each player computes $c_{0,j} \leftarrow (g, \mathsf{pk} \cdot \mathsf{id}_j)$ and set $c_0 \leftarrow (c_{0,j})_{1\leq j\leq 78}$.*
- *For each $i \in \{1,2,3,4\}$, each P_i does in turn: it picks at random a permutation $\delta_i \in [\![1,78]\!]^{78}$, and $(r_{i,j})_{1\leq j\leq 78} \xleftarrow{\$} (\mathbb{Z}_q^*)^{78}$. 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*

$$\pi_{i,1} \leftarrow \mathsf{ZK}\left\{(\delta_i, (r_{i,j})_{1\leq j\leq 78}) : c_{i,j} = \mathsf{Rand}(c_{i-1,\delta_i(j)}, r_{i,j}, \mathsf{pk})\right\}.$$

  Finally, P_i sets $c_i \leftarrow (c_{i,j})_{1\leq j\leq 78}$ and broadcasts $(c_i, \pi_{i,1})$.
- *Each player verifies the proofs $(\pi_{i,1})_{1\leq i\leq 4}$.*

Phase 2:

- *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 $j \in [\![1,72]\!]\backslash[\![18\cdot(i-1)+1, 18\cdot i]\!]$, each P_i computes $\theta_{(i,j)} = x_j^{\mathsf{sk}_i}$, $\pi_{i,2} \leftarrow \mathsf{ZK}\left\{\mathsf{sk}_i : \bigwedge_{j\in[\![1,72]\!]\backslash[\![18\cdot(i-1)+1,18\cdot i]\!]} \theta_{(i,j)} =\right.$ then P_i broadcasts $(\theta_{(i,j)})_{j\in[\![1,72]\!]\backslash[\![18\cdot(i-1)+1,18\cdot i]\!]}$ and $\pi_{i,2}$.*
- *For all $i \in [\![1,4]\!]$, for all $l \in [\![1,4]\!]$, for all $j \in [\![18\cdot(l-1)+1, 18\cdot l]\!]$, 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 $(\pi_{\gamma,2})_{\gamma\in[\![1,4]\!]\backslash\{i\}}$. For $j \in [\![73,78]\!]$ let $c_j^* = c_{4,j}$.*

$\mathsf{GetHand}(n, \mathsf{sk}, \mathsf{pk}, \mathsf{PK})$: *The algorithm parses PK as $(c_j^*)_{1\leq j\leq 78}$ and returns a hand $H \leftarrow \{\mathsf{Dec}_{\mathsf{sk}}(c_j^*)\}_{j\in[\![18\cdot(n-1)+1,18\cdot n]\!]}$.*

$\mathsf{MakeDog}$: *as specified in Definition 4.*

$\mathsf{Play}(n, \mathsf{id}, \mathsf{sk}, \mathsf{pk}, \mathrm{st}, \mathsf{PK})$: *It parses $\mathsf{PK} = (c_j^*)_{1\leq j\leq 78}$ 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 $t \in [\![18\cdot(n-1)+1, 18\cdot n]\!]$ 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 $(c_j^*)_{j\in[\![18\cdot(n-1)+1,18\cdot n]\!]}$ that have already been used by player n to play a card. For all $i \in [\![1,4]\!]\backslash\{n\}$, 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

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

which proves that the played card id matches one of the ciphertexts in PK attributed to the player n. Let $L \subset [\![1,78]\!]$ 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 $|U_n \cup \{t\}| = 18$, 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* $\mathsf{suit}' \neq \mathsf{id.suit}$ *and* $|U_n \cup \{t\}| < 18$*, it generates*

$$\Pi_1 = \mathsf{ZK}\left\{\mathsf{sk} : \bigwedge_{\substack{j \in [\![18\cdot(n-1)+1, 18\cdot n]\!] \\ j \notin U_n \cup \{t\}}} \bigvee_{l \in L} \mathsf{id}_l = \mathsf{Dec}_{\mathsf{sk}}(c_j^*)\right\}.$$

This proof ensures that the cards that are encrypted in the c_j^* *are not of the leading suit, 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* $(c_j^*)_{1 \leq j \leq 78}$*, and* Π *as* (t, Π_0, Π_1)*. First checks if* $t \in [\![18 \cdot (n-1) + 1, 18 \cdot n]\!]$*, if not return 0. If* $\mathrm{st} = \perp$*, it sets four empty sets* U_1*,* U_2*,* U_3 *and* U_4*. Let* $L \in [\![1, 78]\!]$ *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 verifies that the state* st *is correctly updated in* st' *according to the* Play *algorithm:*

– *If there exists* $i \in [\![1, 4]\!] \setminus \{n\}$ *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* $\mathsf{suit}' \neq \mathsf{id.suit}$*, 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* $j \in [\![1, 18]\!]$ *such that* $(18 \cdot (n-1) + j) \notin U_n$ *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*.*

E Proofs for the Spades Scheme

The proof of Theorem 1 follows from the lemmas of this section. We need to introduce a few notations in order to prove our main result. Let G_0 and G_1 be two probability distributions and $\mathcal{A}$ a polynomial time algorithm, we define the advantage of $\mathcal{A}$ in distinguishing in between two experiments as:

$$\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}) = |\Pr[\mathcal{A}(k, G_0)] - \Pr[\mathcal{A}(k, G_1)]|.$$

For the shake of simplicity, in the following proofs we will consider $\epsilon_{\mathsf{sound}} = \mathsf{max}(\epsilon^{\pi_*}_{\mathsf{sound}}, \epsilon^{\Pi_*}_{\mathsf{sound}})$, the maximum of the advantage that any probabilistic polynomial time adversary has against the soundness of the proofs of knowledge used in the protocol. As all are considered negligible so is their maximum.

Most of the modifications in our game hope are done in the subpart of the game where $\mathcal{C}$ and $\mathcal{A}$ will interact to execute the DeckGen algorithm, hence we will only highlight the modification in this algorithm when nothing is changed in the rest of the protocol.

Lemma 1. *If our Spades protocol is instantiated by proofs of knowledge that are sound and extractable then it is* theft-resistant*. The adversary's advantage against theft-resistance is bounded by* $\mathsf{Adv}^{th-res}_{W,S,\mathcal{A}}(\mathfrak{K}) \leq (4 + 13) \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})$*.*

Proof. We use the following game hops to show theft-resistant security. We first recall the conditions to break this property. There exists a play $\gamma \in [\![1, 13]\!]$ such that:

$$1 = \mathsf{Verif}(i_c, \mathsf{id}_{i,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK}), \tag{7}$$

$$\text{and} \quad \mathsf{id}_{i,\gamma} \notin H_{i_c}. \tag{8}$$

Game 0: This is the Trick-Taking experiment $\mathsf{Exp}^{TrickT}_{W,S,\mathcal{A}}(\mathfrak{K})$ with the winning conditions (7) and (8) as describe in the definition of the property.

Game 1: In this game, at the end of phase 1 of the $\mathsf{DeckGen}$ algorithm, $\mathcal{C}$ will use the extractor $\mathsf{Ext}^{\mathcal{A}}(\pi_{i_c,1})$ in order to recover P_{i_c}'s secret random values $(\delta_{i_c}, (r_{i_c,j})_{1\leq j\leq 52})$. If this extraction fails the game is aborted, otherwise $\mathcal{C}$ reproduce the protocol to checks if the result is the same. Game is also aborted if this is not true. The difference of outputs between G_0 and G_1 only occur if P_{i_c} has outputted a valid zero-knowledge proof of a false statement or if the extraction of the secret by the extractor fails. This has probability $\epsilon_{\mathsf{extract}}(\mathfrak{K})$ of happening and a reduction breaking the soundness of the zero-knowledge proof is straight forward. As it is assumed that no polynomial time adversary has non-negligible advantage against this property of the NIZKP, the difference in between the games is negligible. Finally, $\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 2: The same modification can be applied for P_{i_a}.
Game 3: This step introduces changes in phase 2 of the $\mathsf{DeckGen}$ algorithm. For the proof provided by the entity P_{i_c}, the challenger has extracts the secret $\mathsf{sk}_{i_c} = \mathsf{dk}_{i_c}$. It can now check whether for all $j \in [\![1, 52]\!] \setminus [\![13 \cdot (i_c - 1) + 1, 13 \cdot i_c]\!]$, the equality $\theta_{(i_c,j)} = x_j^{\mathsf{dk}_{i_c}}$ holds, if not, $\mathcal{C}$ abort the game. The rest of the game remains unchanged. Soundness of the proof implies that the game has been modified by the negligible chance that $\mathcal{A}$ has to forge a valid zero-knowledge proof $\pi_{i_c,2}$ for a false statement. We directly obtain the following inequality $\mathsf{Adv}^{indist}_{G_3,G_4}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 4: This is the same modifications as game G_3 but this time for P_{i_a}. We reach the same negligible changes as from *Game 2* to *Game 3*.

In Game 4 any action of $\mathcal{A}$ during the shuffle phase is verified by the challenger. If the challenger computation does not match the adversary's ones, $\mathcal{A}$ losses. The values returned by the adversary throughout the algorithm $\mathsf{DeckGen}$ are then consistent with the protocol and the cards are shuffled correctly. From condition (8), $\mathsf{id}_{i,\gamma} \notin H_{i_c}$, we can deduce that in case $\mathcal{A}$ wins their should exist $i \in [\![1, 4]\!] \setminus \{i_c\}$ such that $\mathsf{Dec}_{\mathsf{dk}_i}(c^*_t) = \mathsf{id}_{i,\gamma}$ for some $t \in [\![1, 52]\!] \setminus [\![13 \cdot (i_c - 1) + 1, 13 \cdot i_c]\!]$. But as the shuffle has been conducted consistently, it is impossible to have a second ciphertext $c^*_{t'}$ verifying $\mathsf{Dec}_{\mathsf{dk}_i}(c^*_t) = \mathsf{Dec}_{\mathsf{dk}_{i_c}}(c^*_{t'}) = \mathsf{id}_{i,\gamma}$ with $t \neq t'$. Hence, condition 7 and 8 can only be reached by outputting a valid proof of a false statement during the game phase. This can only be done with a negligible probability $\epsilon_{sound}(\mathfrak{K})$ by a polynomial time algorithm.
Game 5: When $\mathcal{A}$ plays a card $(\mathsf{id}, \Pi = (\Pi_0, \Pi_1, t))$ associated to a cipher c_j as P_{i_c}, $\mathcal{C}$ now decrypt the value behind c_j with sk_{i_c} and checks if this is coherent with the given proof Π_0. The adversary can try to cheat during any of the 13 turns it plays, hence by a hybrid argument we have $\mathsf{Adv}^{indist}_{G_5,G_6}(\mathcal{A}) \leq 13 \cdot \epsilon_{sound}(\mathfrak{K})$ and as $\mathcal{A}$ has no way of winning this concludes the proof. □

Lemma 2. *Instantiated by proofs of knowledge that are sound and extractable, our Spades protocol is* cheating-resistant. *The adversary's advantage against this property is at most* $\mathsf{Adv}^{cheat}_{W,S,\mathcal{A}}(\mathfrak{K}) \leq 30 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

Proof. Let $\mathcal{A}$ be an adversary against the cheating resistance of the protocol. In order to break this property it must output $i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}'_\gamma$, for some $\gamma \in [\![1, 13]\!]$, verifying the conditions:

1. $1 = \mathsf{Verif}(i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK})$,
2. $\mathsf{id}_{i_c,\gamma}.\mathsf{suit} \neq \mathsf{suit}_{i_c,\gamma}$ and $\mathsf{suit}_{i_c,\gamma} \neq \perp$
3. $\exists\ \mathsf{id} \in H_{i_c}$ such that:
   (a) $\forall\ l \leq \gamma, \mathsf{id}_{i_c,l} \neq \mathsf{id}$
   (b) $\mathsf{id}.\mathsf{suit} = \mathsf{suit}_{i_c,\gamma}$

Condition 1 must always hold, otherwise the Trick-Taking experiment would abort during the game phase when this play was received by the challenger. As we have just proven that our Spades protocol is theft-resistant, we can assume that $\mathcal{A}$ will always play cards form the hand H_{i_c} on P_{i_c}'s turn. We show that $\mathsf{Adv}^{cheat}_{\mathcal{A}}(\mathfrak{K})$ is negligible for any probabilistic polynomial time adversary $\mathcal{A}$ by using the following sequence of games.
Game 0: This is the Trick-Taking experiment with the winning conditions of the cheating-resistance stated above.
Game 1: *Game 0* is append with the modification up to *Game 4* of the proof of Lemma 1. As argued previously $\mathsf{Adv}^{indist}_{cheat,G_0}(\mathcal{A}) \leq 4 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

Game 2: After the shuffle, $\mathcal{C}$ is now decrypting P_{i_c}'s hand by running $\mathsf{GetHand}$ with sk_{i_c}. Each time $\mathcal{A}$ returns a play $(\mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma} = (\Pi_0, \Pi_1, t))$ during a turn $\gamma \in [\![1, 13]\!]$, the challenger check that given the cards it knows, $\mathcal{A}$ was allowed to play this card. If it is not, $\mathcal{C}$ aborts the game and returns 0. In order to justify the negligible gap in between *Game 1* and *Game 2*, we make use of a sequence of 14 games. Set $G_0 = H_0, \ldots, H_{13} = G_1$ and for all $\gamma \in [\![0, 13]\!]$, $H_{\gamma+1}$ is a copy of H_γ with an additional action of $\mathcal{C}$. In $H_{\gamma+1}$, once the condition $1 = \mathsf{Verif}(i_c, \mathsf{id}_{i_c,\gamma}, \Pi_{i_c,\gamma}, \mathsf{pk}_{i_c}, \mathrm{st}_\gamma, \mathrm{st}'_\gamma, \mathsf{PK})$ is passed on round γ, $\mathcal{C}$ executes the above described verification by checking the hand of the player. In the case where it cannot find the cards that was played, $\mathcal{C}$ abort the game.

In $H_{13} = G_1$, the game is aborted if $\mathcal{A}$ forged a valid zero-knowledge proof of an incoherent play. For each step $\mathsf{Adv}^{indist}_{H_\gamma, H_{\gamma+1}}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$, hence

$$\mathsf{Adv}^{indist}_{H_\gamma, H_{\gamma+1}}(\mathcal{A}) \leq 2 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K}).$$

Thus, we obtain an upper bound on the distinguishability of games G_1 and G_2:

$$\mathsf{Adv}^{indist}_{G_1, G_2}(\mathcal{A}) \leq 13 \cdot (2 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})).$$

Only valid proof of valid statements were produced during the experiment. Thus, the verification at each step provides that for all $t \in U_{i_c}$, for all $j \in [\![1, 13]\!]$ we have a valid proof Π_j reproduced by $\mathcal{C}$. This implies that all cards in P_{i_c}'s hand none are of the leading suit *i.e.,* for all $\mathsf{id} \in H_{i_c}$ such that $\forall\, l \leq \gamma, \mathsf{id}_{i_c,l} \neq \mathsf{id}$ we know that $\mathsf{id.suit} \neq \mathsf{suit}_{i_c,\gamma}$.

Hence, conditions 2 and 3 can no longer be verified if condition 1 holds in game 2. This concludes the proof as no adversary would be able to win this game. Adding up the advantages, we obtain $\mathsf{Adv}^{cheat}_{W,S,\mathcal{A}}(\mathfrak{K}) \leq 30 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

□

Lemma 3. *If our Spades protocol is instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, and assuming hardness of the DDH problem, then the protocol is* unpredictable.

Proof. These proof uses the previous arguments invoked in the demonstration of indistinguishability. We show that for each of the four indices $i_{\mathsf{guess}} \in \{1, 2, 3, 4\}$, the adversary has negligible chance make a better guess than by picking randomly.
Game 0: This is the unpredictability experiment $\mathsf{Exp}^{unp}_{W,S,\mathcal{A}}(\mathfrak{K}, i_{\mathsf{guess}})$.
Game 1: In this game, at the end of phase 1 of the $\mathsf{DeckGen}$ algorithm, $\mathcal{C}$ will use the extractor in order to recover P_{i_c}'s secret random values $(\delta_{i_c}, (r_{i_c,j})_{1 \leq j \leq 52})$. From these values, $\mathcal{C}$ reproduce the protocol to checks if the result is the same. Game is aborted if the extraction fails or if the results are not consistent with the received values. As justified above this leads to $\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}, i_{\mathsf{guess}}) = \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 2: We are making the same modification but this time for P_{i_a}. Just like in the previous step, $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}, i_{\mathsf{guess}}) = \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 3: This step introduces changes in phase 2 of the $\mathsf{DeckGen}$ algorithm. For the proof provided by the entity P_{i_c} at the beginning of the experiment, $\mathcal{C}$ has extracted the secret $\mathsf{sk}_{i_c} = \mathsf{dk}_{i_c}$. It can now check whether for all $j \in [\![1, 52]\!] \setminus [\![13 \cdot (i_c - 1) + 1, 13 \cdot i_c]\!]$, the equality $\theta_{(i_c,j)} = x_j^{\mathsf{dk}_{i_c}}$ holds, if some of these equalities do not match, $\mathcal{C}$ abort the game. The rest of the experiment remains unchanged. Soundness of the proof implies that the game has been modified by the negligible chance that $\mathcal{A}$ has to forge a valid zero-knowledge proof $\pi_{i_c,2}$ for a false statement. We directly obtain the following inequality $\mathsf{Adv}^{indist}_{G_2,G_3}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 4: The same modification as in *Game 3* is made regarding player P_{i_a}.

In order to finish this proof two scenario need to be investigated. The first one will assume that the last player to shuffle is simulated by $\mathcal{C}$, the second consider the case where this last player is controlled by the adversary.

Case $i_c, i_a \in \{1, 2, 3\}$***:*** In this case the entity P_4 is run honestly by the challenger. This means that the last shuffle, using δ_4 is performed following the prescribed algorithm and δ_4 picked at random. This completely randomise the previous (potentially maliciously chosen) shuffles used by $\mathcal{A}$. We can directly conclude the

proof for this case as the last permutation is always random and will send each ciphertext $c_{4,j}$ (respectively each card id_j) uniformly to one of the 52 possible positions of our ordered set $(c_{4,j})_{j\in[\![1,52]\!]}$ (resp. $(\mathsf{id}_j)_{j\in[\![1,52]\!]}$). From the previous steps we know that $\mathcal{A}$ has followed the protocol, hence was not able to duplicate any card. This concludes our case as it is impossible for $\mathcal{A}$ to predict the outputted values after the last random permutation.

Case $i_c = 4$ ***or*** $i_a = 4$***:*** We follow the following logic: given that the ElGamal encryption is IND-CPA, $\mathcal{A}$ learns nothing on the cards underlying the ciphertexts, hence, $\mathcal{A}$'s permutations cannot be chosen better than random. For that we replace the values that $\mathcal{A}$ sees after $\mathcal{C}$'s shuffle by random values, we latter remove them in order for $\mathcal{A}$ to recover his cards. We are showing that sending random elements lead to a negligible change due to the DDH hypothesis. Our scheme does not use the ElGamal encryption as a black box, thus reduce ourselves to the DDH problem instead of using its IND-CPA security.

Game 5: During the key generation phase, $\mathcal{C}$ randomly chooses two index $i_1, i_2 \in \{1,2,3\}$. If $\mathcal{A}$ claims one of these two role for i_a or i_c the game is aborted. This has probability 7/12 of happening, hence, $7/12 \cdot \mathsf{Adv}^{unp}_{G_4,\mathcal{A}}(\mathfrak{K}, i_{\mathsf{guess}}) = \mathsf{Adv}^{unp}_{G_5,\mathcal{A}}(\mathfrak{K}, i_{\mathsf{guess}})$.

For the rest of this proof we assume that $i_c = 4$ or $i_a = 4$.

Game 6: Let $i_1, i_2 \in \{1,2,3,4\} \setminus \{i_a, i_c\}$, with $i_1 \leq i_2$. We are now making some changes in the behaviours of the challenger while acting as the honest player P_{i_1}. During part 1 of the DeckGen algorithm, on receiving c_{i_1-1}, it parses it as $(c_{i_1-1,j})_{1\leq j\leq 52} = (x_j, y_j)_{1\leq j\leq 52}$. $\mathcal{C}$ will then draw a random vector $(Z_{i_1,j})_{1\leq j\leq 52} \xleftarrow{\$} \mathbb{G}^{52}$ and the usual $\delta_{i_1}, (r_{i_1,j})_{1\leq j\leq 52}$. Then $c_{i_1,j} \leftarrow \mathsf{Rand}((x_{\delta_{i_1}(j)}, Z_{\delta_{i_1}(j)} \cdot y_{\delta_{i_1}(j)}), r_{i_1,j}, \mathsf{pk})$ is computed. Then $\mathcal{C}$ uses the simulator in order to obtain a proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, we return $(c_{i_1}, \pi_{i_1,1})$ to the adversary. The value $Z_{i_1,j}$ completely hide the underlying message of the ciphertext as $Z_{i_1,j} \cdot y_{\delta_{i_1}(j)} = (Z \cdot \mathsf{id}_{\delta_{i_1}\circ\ldots\circ\delta_1(j)}) \cdot \mathsf{pk}^{\sum_{l=1}^{i_1} r_l}$, for some integers r_l that are introduced through the randomness of the previous players.

During the second phase of DeckGen we are making the following changes: in order to let the adversary recover the values for its cards we compute $\theta_{i_1,j} = Z_{\delta_4\circ\ldots\circ\delta_{i_1}(j)} \cdot \mathsf{dk}_{i_1}^{\sum_{l=1}^{4} r_{\delta_l\circ\ldots\circ\delta_1(j)}}$ for $j \in [\![1,52]\!] \setminus [\![13\cdot(i_1-1)+1, 13\cdot i_1]\!]$. $\mathcal{C}$ parses c_4 as $(c_{4,j})_{j\in[\![1,52]\!]}$ and $c_{4,j}$ as (x_j, y_j), using the simulator the challenger produces $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{\theta_{i_1,j}, x_j\}_j, \mathsf{ek}_{i_1})$, finally, sends $(\{\theta_{i_1,j}\}_{j\in[\![1,52]\!]\setminus[\![13\cdot(i_1-1)+1,13\cdot i_1]\!]}, \pi_{i_1,2})$ to $\mathcal{A}$. The remaining part of the game is unchanged.

Reduction: we define a sequence of games $G_4 = H_0, \ldots, H_{52} = G_5$ where G_4 denotes the *Game 4* and G_5 the *Game 5*. For all $j \in [\![1,52]\!]$, let H_j be a challenge to $\mathcal{A}$, where only the j first elements of c_{i_1-1} are modified using the random vector $Z \xleftarrow{\$} \mathbb{G}^{52}$. The elements out of this range stay still. Latter, knowing the permutations δ_{i_a} and δ_{i_c} we will only modify the $\theta_{i_1,l}$, for which there exist $k \leq j$ such that $l = \delta_4\circ\ldots\circ\delta_{i_1}(\mathfrak{K})$, *i.e.,* the elements associated by the permutations to the previous ones. This definition is coherent with the equalities $G_4 = H_0$ and $G_5 = H_{52}$. In the first equality, no element is actually modified in H_0 as $j = 0$, thus it still the *Game 4*. The second one holds as for $j = 52$ we have applied the random value Z on the full length of the ciphertext c_{i_1-1}, like in G_5. Latter the challenger uses the values Z to compute the values $\theta_{i_1,l}$, for $1 \leq l \leq 52$. Hence, all actions conducted in G_5 matches the one we are doing in H_{52}.

We will provide a reduction in between games H_j and H_{j+1} for all $j \in [\![0,51]\!]$. Consider the DDH triple (X, Y, Z), where $X = g^a$, $Y = g^b$ and Z is either g^{ab} or a random element of $\mathbb{G}$. We will build an algorithm $\mathcal{B}$ breaking the DDH problem based on a distinguisher $\mathcal{D}$ distinguishing in between games H_j and H_{j+1}. Our adversary start by receiving a challenge triple (X, Y, Z) from the challenger $\mathcal{C}_{\mathsf{DDH}}$. Then, it does the following:

Key generation phase: First $\mathcal{B}$ receives the corrupted player index i_c. Then $\mathcal{B}$ claims the DDH challenge (X, Y, Z), it uses the first value to set its encryption key $\mathsf{ek}_{i_1} = X$ and generates the keys $(\mathsf{sk}_{i_2}, \mathsf{pk}_{i_2})$, $(\mathsf{sk}_{i_a}, \mathsf{pk}_{i_a}) \leftarrow \mathsf{KeyGen}(\mathfrak{K})$ for i_2 and i_a using the key generation algorithm. A zero-knowledge proof for the encryption key of P_{i_1} is generated using the simulator, $\Pi_{\mathsf{ek}_{i_1}} \leftarrow \mathsf{Sim}(\mathsf{ek}_{i_1})$ and we set $\mathsf{pk}_{i_1} = (\mathsf{ek}_{i_1}, \Pi_{\mathsf{ek}_{i_1}})$. The key $\{\mathsf{pk}_i\}_{i=i_1,i_2,i_a}$ and i_{guess} are sent to $\mathcal{D}$. The distinguisher's response should be the index i_a with the elements $\mathsf{pk}_{i_c}, \Pi_{\mathsf{ek}_{i_c}}$. The latest proof is checked, then dk_{i_a} is sent by $\mathcal{B}$ and id and are received from $\mathcal{D}$.

Shuffle phase: $\mathcal{B}$ is first checking the proof $\Pi_{\mathsf{ek}_{i_c}}$ outputted by $\mathcal{A}$, initialize the canonical deck $D = (\mathsf{id}_1, \ldots, \mathsf{id}_{52})$, computes $\mathsf{pk} = \mathsf{ek}_1 \cdot \mathsf{ek}_2 \cdot \mathsf{ek}_3 \cdot \mathsf{ek}_4$ and $c_{0,j} \leftarrow (g, \mathsf{pk} \cdot \mathsf{id}_j)$. And set $c_0 \leftarrow (c_{0,j})_{1 \le j \le 52}$.

Phase 1: For i in $[\![1, 4]\!]$:

- If $i \in \{i_c, i_a\}$, $\mathcal{D}$ returns $(c_i, \pi_{i,1})$, then $\mathcal{B}$ checks the proof $\pi_{i,1}$ that was outputted by the adversary. Then it executes $\mathsf{Ext}^{\mathcal{A}}(\pi_{i,1}) \rightarrow (\delta_i, (r_{i,})_{1 \le l \le 52})$. $\mathcal{B}$ executes $c'_{i,l} \leftarrow \mathsf{Rand}(c_{i-1,\delta_i(l)}, r_{i,l}, \mathsf{pk})$ and checks if $c_{i,l} = c'_{i,l}$ for all $l \in [\![1, 52]\!]$. If the proof does not verify or if the equality does not hold then return a random bit from a Bernoulli distribution of parameter $p = 1/4$.
- If $i = i_1$, on receiving $c_{i_1-1} = (c_{i_1-1,l})_{1 \le l \le 52} = (x_l, y_l)_{1 \le l \le 52}$. $\mathcal{B}$ draws randomly the elements $(Z_{i_1,l})_{1 \le l \le 52} \xleftarrow{\$} \mathbb{G}^j$, $\delta_{i_1}, (r_{i_1,l})_{1 \le l \le 52}$. Then $c_{i_1,l} \leftarrow \mathsf{Rand}((x_{\delta_{i_1}(l)}, Z_{i_1,\delta_{i_1}(l)} \cdot y_{\delta_{i_1}(l)}), r_{i_1,l}, \mathsf{pk})$ is computed for all $1 \le l < j$. We also compute $c_{i_1,j} \leftarrow \mathsf{Rand}((x_{\delta_{i_1}(j)}, Z \cdot y_{\delta_{i_1}(j)}), r_{i_1,j}, \mathsf{pk})$ and $c_{i_1,l} \leftarrow \mathsf{Rand}(c_{i_1-1,\delta_{i_1}(l)}, r_{i_1,l}, \mathsf{pk})$ for all $j < l \le 52$. Set $c_{i_1} = (c_{i_1,l})_{1 \le l \le 52}$. Then $\mathcal{B}$ uses the simulator in order to produce a zero-knowledge proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, $(c_{i_1}, \pi_{i_1,1})$ is returned to $\mathcal{D}$.
- If $i = i_2$, $\mathcal{D}$ executes the protocol as usual.

Phase 2: On receiving c_4, $\mathcal{B}$ parses it as $(c_{4,l})_{1 \le l \le 52}$ and each $c_{4,l}$ as (x_l, y_l). First we define $\delta = \delta_4 \circ \ldots \circ \delta_{i_1}$. We set, $\theta_{i_1,\delta(l)} = Z_{i_1,l}$ for all $1 \le l < j$, $\theta_{i_1,\delta(j)} = Z$ and $\theta_{i_1,\delta(l)} = \mathsf{ek}_{i_1}^{\sum_{i=1}^{4} r_{\delta_i \circ \ldots \circ \delta_1(i)}} (= x_{\delta(l)}^{\mathsf{dk}_{i_1}})$ for all $j < l \le 52$ and given that dk_{i_1} is unknown but all the randomness have been extracted in phase 1. $\mathcal{B}$ uses the simulator to obtain the proof $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{x_l, \theta_{(i_1,l)}\}_{l \in [\![1,52]\!] \setminus [\![13 \cdot (i_1-1)+1, 13 \cdot i_1]\!]}, \mathsf{ek}_{i_1})$. Also, the elements $\theta_{(i_2,l)} = x_l^{\mathsf{dk}_{i_2}}$ for $l \in [\![1, 52]\!] \setminus [\![13 \cdot (i_2 - 1) + 1, 13 \cdot i_2]\!]$ are computed with their associated zero-knowledge proof

$$\pi_{i_2,2} \leftarrow \mathsf{ZK}\left\{ \mathsf{dk}_{i_2} : \bigwedge_{l \in [\![1,52]\!] \setminus [\![13 \cdot (i_2-1)+1, 13 \cdot i_2]\!]} \theta_{(i_2,l)} = x_l^{\mathsf{dk}_{i_2}} \right\}.$$

Values $(\theta_{(i_1,l)})_{l \in [\![1,52]\!] \setminus [\![13 \cdot (i_1-1)+1, 13 \cdot i_1]\!]}$, $(\theta_{(i_2,l)})_{l \in [\![1,52]\!] \setminus [\![13 \cdot (i_2-1)+1, 13 \cdot i_2]\!]}$, $\pi_{i_1,2}$ and $\pi_{i_2,2}$ are broadcasted to $\mathcal{D}$. Then, the adversary outputs the values $(\theta_{(i_c,l)})_{l \in [\![1,52]\!] \setminus [\![13 \cdot (i_c-1)+1, 13 \cdot i_c]\!]}, (\theta_{(i_a,l)})_{l \in [\![1,52]\!] \setminus [\![13 \cdot (i_a-1)+1, 13 \cdot i_a]\!]}$ and $\pi_{i_c,2}, \pi_{i_a,2}$. For $i = i_c, i_a$, $\mathcal{B}$ verifies the proof $\pi_{i,2}$, executes $\mathsf{Ext}^{\mathcal{A}}(\pi_{i,2}) \rightarrow \mathsf{dk}_i$, and computes $\theta'_{(i,l)} = x_l^{\mathsf{dk}_i}$ for all $l \in [\![1, 52]\!] \setminus [\![13 \cdot (i_c - 1) + 1, 13 \cdot i_c]\!]$. It checks if $\theta'_{(i,l)} = \theta_{(i,l)}$. If these equalities do not hold then $\mathcal{B}$ returns a random bit. Then, for all $l \in [\![1, 4]\!]$, for all $l \in [\![13 \cdot (l-1) + 1, 13 \cdot l]\!]$, $\mathcal{B}$ computes $c_l^* \leftarrow \left(x_l, \frac{y_l}{\prod_{1 \le \gamma \le 4; \gamma \ne l} \theta_{(\gamma,l)}} \right)$. $\mathcal{B}$ send $\mathsf{PK} \leftarrow (c_l^*)_{1 \le l \le 52}$ to $\mathcal{D}$ as the result of this algorithm. If the result obtain by $\mathcal{D}$ and $\mathcal{B}$ for PK is different or equal to $\perp$, $\mathcal{B}$ returns a random bit.

Final phase: $\mathcal{D}$ returns a bit after these interactions; we forward this bit to the challenger of the DDH problem.

Introducing the DDH challenge (X, Y, Z) in particularly Z in the j^{th} element of c_{i_1-1}, we simulate game H_{j-1} if this element corresponds to the real Diffie-Hellman of X and Y. If this value was random we will obtain a simulation of game H_j as $c_{i_1,\delta_{i_1}(j)}$ is given back to the adversary as a random value. Hence, this reduction provides that if there exist an adversary distinguishing in between games H_{j-1} and H_j, $\mathcal{B}$ is breaking the DDH problem with same probability. This gives:$\mathsf{Adv}^{indist}_{G_4,G_5}(\mathcal{A}, i_{\mathsf{guess}}) \le 52 \cdot \mathsf{Adv}^{\mathsf{DDH}}(\mathfrak{K})$.

Conclusion: Since we replay all actions of $\mathcal{A}$, it has to follow the shuffle protocol, its capability are reduced to apply chosen randomization values and permutation that could influence the final order of the cards. This was already enough to conclude for the case where $i_c, i_a \in \{1, 2, 3\}$. We proceeded to further reduction in the other cases. Indeed, the adversary could still gain an advantage if it was playing last during the first phase of shuffle as it determines the final order of the cards. In *Game 6*, after P_{i_1}'s turn, $\mathcal{A}$ sees only random values when it shuffles. Hence, the adversary learns no information about the final dealing given that cards it has to shuffle are in an unknown random order, thus, it cannot influence the outputs with some wise choice on $\delta_{i_c}, \delta_{i_a}$ and $(r_{i_c,j})_{1 \le j \le 52}$, $(r_{i_a,j})_{1 \le j \le 52}$. Here $\mathsf{Adv}^{unp}_{G_6,\mathcal{A}}(\mathfrak{K}) = 0$. We have provided reductions that lead to

Game 6 with each time a negligible upper bound in the adversary's advantage against the indistinguishability with the original game, this allows us to conclude that:

$$\mathsf{Adv}^{unp}_{W,\mathcal{A}}(\mathfrak{K}, i_{\mathsf{guess}}) \leq 4 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K}) + 12/7 \cdot 52 \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{A}}(\mathfrak{K})$$

for each of the possible index. Hence, the sum for all index is also negligible. □

Lemma 4. *Assuming hardness of the DDH problem. If our Spades protocol is instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, then it is* hand-private.

Proof. The aborting condition $\mathsf{PK} = \perp$ of the shuffle phase always generates a random bit, hence this gives no advantage to our adversary. An adversary making the game abort each time will always lose, hence $\mathcal{A}$ should allow derivation of the public information $\mathsf{PK} = \{c^*_j\}_{j \in [\![1,52]\!]}$ with non-negligible probability. Unlink in the previous experiments, our adversary $\mathcal{A}$ does not have an accomplice. Hence, $\mathcal{A}$ plays P_{i_c} and $\mathcal{C}$ the rest of the players. Let us define a sequence of games in order to prove the hand-privacy of our Spades protocol.
Game 0: This is the hand-privacy experiment of definition 12.
Game 1: We want to make sure that the outputted values of $\mathcal{A}$ are following the requirement of the protocol. This is the same arguments as stated before in proof of Lemma 3. $\mathcal{A}$ has no accomplice hence only the proofs $\pi_{i_c,1}$ and $\pi_{i_c,2}$ need to be checked by $\mathcal{C}$. This leads to $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}) \leq 2 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

The remaining of the proof will focuses on hiding the underlying messages of the ciphertext such that $\mathcal{A}$ cannot guess the order of the cards from the ciphertexts it sees. Let $i_1, i_2, i_3 \in \{1,2,3,4\} \setminus \{i_c\}$, with $i_1 < i_2 < i_3$.
Game 2: We first make changes in the behaviours of the challenger while acting for the honest player P_{i_1}. During part 1 of the $\mathsf{DeckGen}$ algorithm, on obtaining c_{i_1-1}, it parses it as $(c_{i_1-1,j})_{1 \leq j \leq 52} = (x_j, y_j)_{1 \leq j \leq 52}$. $\mathcal{C}$ will then draw a random vector $(Z_{i_1,j})_{1 \leq j \leq 52} \xleftarrow{\$} \mathbb{G}^{52}$ and the usual $\delta_{i_1}, (r_{i_1,j})_{1 \leq j \leq 52}$. Then $c_{i_1,j} \leftarrow \mathsf{Rand}((x_{\delta_{i_1}(j)}, Z_{\delta_{i_1}(j)} \cdot y_{\delta_{i_1}(j)}), r_{i_1,j}, \mathsf{pk})$ is computed. Then $\mathcal{C}$ uses the simulator in order to obtain a proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, $\mathcal{C}$ returns $(c_{i_1}, \pi_{i_1,1})$ to the adversary.

Then during the second phase of $\mathsf{DeckGen}$, in order to let the adversary recover the values for cards in its hand, $\mathcal{C}$ computes the $\theta_{i_1,j} = Z_{\delta_4 \circ \ldots \circ \delta_{i_1}(j)} \cdot \mathsf{ek}_{i_1}^{\sum_{l=1}^{4} r_{\delta_l \circ \ldots \circ \delta_1(j)}}$ for $j \in [\![13 \cdot (i_c - 1) + 1, 13 \cdot i_c]\!]$, for all $j \in [\![1,52]\!] \setminus (\cup_{i=i_1,i_c} [\![13 \cdot (i-1)+1, 13 \cdot i]\!])$ the $\theta_{i_1,j}$ are draw randomly. We parse c_4 as $(c_{4,j})_{j \in [\![1,52]\!]}$ and $c_{4,j}$ as (x_j, y_j). Then using the simulator the challenger produces $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{\theta_{i_1,j}, x_j\}_j, \mathsf{ek}_{i_1})$. Finally, $\mathcal{C}$ sends $(\{\theta_{i_1,j}\}_{j \in [\![1,52]\!] \setminus [\![13 \cdot (i_1-1)+1, 13 \cdot i_1]\!]}, \pi_{i_1,2})$ to $\mathcal{A}$. The remaining part of the game is conducted as previously.

Doing so, we completely hide the cards in the challenger's hands to the adversary's as $\mathcal{C}$ have outputted random values $\theta_{i_1,j}$ for $j \in [\![1,52]\!] \setminus (\cup_{i=i_1,i_c} [\![13 \cdot (i-1)+1, 13 \cdot i]\!])$ and that the $c^*_{4,j}$ with $j \in [\![13 \cdot (i_1-1)+1, 13 \cdot i_1]\!]$ are hidden by some random values.

Reduction: This reduction is similar to one that was described previously explained in proof of Lemma 3.

Game 4: We proceed to the same modification for P_{i_2}.
Game 5: We proceed to the same modification for P_{i_3}.

Conclusion: Finally, as $\mathcal{A}$ sees only random values for the challenger's hands, it has no other option than returning values following the scheme specifications, doing better than randomness is impossible unless it is able to decrypt the player's cards. We can conclude to the total advantage of any adversary is less than:

$$\mathsf{Adv}^{h-priv}_{\mathcal{A}}(\mathfrak{K}, i_{\mathsf{guess}}) \leq 3 \cdot \epsilon_{\mathsf{sound}}(\mathfrak{K}) + 78 \cdot \mathsf{Adv}^{\mathsf{DDH}}(\mathfrak{K}).$$

□

Definition 23 (n-IND-CPA). *Let k be a security parameter. Let $\mathbb{G}$ be a multiplicative group of prime order q and $g \in \mathbb{G}$ be a generator and $(\mathsf{dk}, \mathsf{ek}) \xleftarrow{\$} \mathsf{KeyGen}(\mathfrak{K})$ an instance of keys. Given an instance of n messages $m_1, \ldots, m_n \in \mathbb{G}$, the* n-indistinguishability under chosen plaintext attack *(n-IND-CPA) problem is to distinguish in between $(\mathsf{Enc}_{\mathsf{ek}}(m_i))_{i \in [\![1,n]\!]}$ and $(r_i) \xleftarrow{\$} (\mathbb{G} \times \mathbb{G})^n$. The n-IND-CPA* assumption *states that there exists no polynomial time algorithm that solves the n-IND-CPA problem with a non-negligible advantage even if allowed to choose the input messages.*

Property 2. For any $n \in \mathbb{N}$, n-IND-CPA holds under the IND-CPA assumption (and then also under the DDH assumption).

Proof. We use a hybrid argument. Consider the following problem:

(j,n)-IND-CPA problem: Let k be a security parameter. Let $\mathbb{G}$ be a multiplicative group of prime order q and $g \in \mathbb{G}$ be a generator and $(\mathsf{dk}, \mathsf{ek}) \xleftarrow{\$} \mathsf{KeyGen}(\mathfrak{K})$ an instance of keys. Let $j \in \mathbb{N}$ be such that $0 \leq j \leq n$. Given an instance of n messages $\{m_i, h_{i,b}\}_{1 \leq i \leq n}$ such that for all $i \in [\![1, n]\!]$, such that:

- if $i \leq j$, $h_{i,0} \xleftarrow{\$} \mathbb{G}^2$ and $h_{i,1} \leftarrow \mathsf{Enc}_{\mathsf{ek}}(m_i)$,
- else, $h_{i,1} \xleftarrow{\$} \mathbb{G}^2$ and $h_{i,0} \leftarrow \mathsf{Enc}_{\mathsf{ek}}(m_i)$.

Let $\mathsf{Adv}^{(j,n)\text{-IND-CPA}}(\mathfrak{K})$ (resp. $\mathsf{Adv}^{n\text{-IND-CPA}}(\mathfrak{K})$, $\mathsf{Adv}^{\text{IND-CPA}}(\mathfrak{K})$) be the advantage of the best algorithm that solves the (j,n)-IND-CPA (resp. n-IND-CPA, IND-CPA) problem. Let (j,n) be a couple of positive integers such that $0 \leq j \leq n-1$. For any adversary that solves the (j,n)-IND-CPA problem with advantage $\mathsf{Adv}^{(j,n)\text{-IND-CPA}}_{\mathcal{A}}(\mathfrak{K})$, we build the algorithm $\mathcal{B}$ that tries to solve the IND-CPA problem.

Algorithm $\mathcal{B}(\mathbb{G}, g, \mathsf{ek}, m, h)$: This algorithm picks $b' \xleftarrow{\$} \{0,1\}$, then for all $i \in [\![1, n]\!] \backslash \{j+1\}$ it picks $m_i \xleftarrow{\$} \mathbb{G}$ and sets:

- if $i \leq j$, it picks $h_{i,0} \xleftarrow{\$} \mathbb{G}^2$ and sets $h_{i,1} = \mathsf{Enc}_{\mathsf{ek}}(m_i)$
- else, $h_{i,1} \xleftarrow{\$} \mathbb{G}$ and $h_{i,0} = g^{a_i \cdot b_i}$.

It sets $m_{j+1} = m$, and $h_{j+1,1} = h_{j+1,0} = h$. Then it runs the adversary: $b_* \xleftarrow{\$} \mathcal{A}(\{(g_{i,1}, g_{i,2}, h_{i,b})\}_{1 \leq i \leq n})$ and returns b_*.

We then deduce that:

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B}}(\mathfrak{K}) = \left| \mathsf{Adv}^{(j,n)\text{-IND-CPA}}_{\mathcal{A}}(\mathfrak{K}) - \mathsf{Adv}^{(j+1,n)\text{-IND-CPA}}_{\mathcal{A}}(\mathfrak{K}) \right|.$$

Hence,

$$\mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{B}}(\mathfrak{K}) \geq \left| \mathsf{Adv}^{(j,n)\text{-IND-CPA}}_{\mathcal{A}}(\mathfrak{K}) - \mathsf{Adv}^{(j+1,n)\text{-IND-CPA}}_{\mathcal{A}}(\mathfrak{K}) \right|$$

which implies

$$n \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathcal{A}}(\mathfrak{K}) \geq \mathsf{Adv}^{n\text{-IND-CPA}}_{\mathcal{A}}(\mathfrak{K}).$$

This concludes the proof. □

Lemma 5. *Instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, our Spades protocol is* Game-private *under the DDH assumption.*

Proof. We first recall that $K = (\mathsf{setup}, \mathsf{pk}_{i_c}, \{\mathsf{sk}_i, \mathsf{pk}_i, \Pi_{\mathsf{ek}}\}_{1 \leq i \leq 4; i \neq i_c}, \mathsf{PK})$ and consider the experiment $Exp^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}}(\mathfrak{K})$ modified according to the definition. This game will also be denoted as *Game 0*.

Game 1: We modify *Game 0* and particularly the behaviours of the first honest player that we will call P_{i_1}. The rest of the game remains the same. During part 1 of the DeckGen algorithm, on obtaining c_{i_1-1}, it parses it as $(c_{i_1-1,j})_{1 \leq j \leq 52} = (x_j, y_j)_{1 \leq j \leq 52}$. $\mathcal{C}$ will then draw a random vector $(Z_{i_1,j})_{1 \leq j \leq 52} \xleftarrow{\$} \mathbb{G}^{52}$ and the usual $\delta_{i_1}, (r_{i_1,j})_{1 \leq j \leq 52}$. Then $c_{i_1,j} \leftarrow \mathsf{Rand}((x_{\delta_{i_1}(j)}, Z_{\delta_{i_1}(j)} \cdot y_{\delta_{i_1}(j)}), r_{i_1,j}, \mathsf{pk})$ is computed. Then $\mathcal{C}$ uses the simulator in order to obtain a proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, we return $(c_{i_1}, \pi_{i_1,1})$ to the adversary.

Then during the second phase of DeckGen, in order to let the adversary recover the values for cards in its hand, we will compute the $\theta_{i_1,j} = Z_{\delta_4 \circ \ldots \circ \delta_{i_1}(j)} \cdot \mathsf{ek}_{i_1}^{\sum_{l=1}^{4} r_{\delta_l \circ \ldots \circ \delta_1(j)}}$ for $j \in [\![13 \cdot (i_c - 1) + 1, 13 \cdot i_c]\!]$, for all $j \in [\![1, 52]\!] \setminus (\cup_{i=i_1,i_c} [\![13 \cdot (i-1)+1, 13 \cdot i]\!])$ the $\theta_{i_1,j}$ are draw randomly. We parse c_4 as $(c_{4,j})_{j \in [\![1,52]\!]}$ and $c_{4,j}$ as (x_j, y_j). Then using the simulator the challenger produces $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{\theta_{i_1,j}, x_j\}_j, \mathsf{ek}_{i_1})$. Finally, we send $(\{\theta_{i_1,j}\}_{j \in [\![1,52]\!] \setminus [\![13 \cdot (i_1-1)+1, 13 \cdot i_1]\!]}, \pi_{i_1,2})$ to $\mathcal{A}$. The remaining part of the game is conducted as previously.

Doing so, we are completely hiding the cards in the challenger's hands to the adversary's as we have outputted random values $\theta_{i_1,j}$ for $j \in [\![1, 52]\!] \setminus (\cup_{i=i_1,i_c} [\![13 \cdot (i-1)+1, 13 \cdot i]\!])$ and that the $c^*_{4,j}$ with $j \in [\![13 \cdot (i_1 - 1) + 1, 13 \cdot i_1]\!]$ are hidden by some random values.

Construction of $\mathsf{Sim}^{\mathsf{TrickT}}_{W,S,\mathcal{A}(\mathsf{vw})}(\mathfrak{K}, \mathit{setup}, i_c, \{\mathsf{pk}_i\}_{1\leq i\leq 4}, \mathsf{PK}, \mathsf{vw})$*:*

Key generation phase: The simulator try to deduce from vw the value dk_{i_c} such that for all $j \in [\![13\cdot(i_c-1)+1, 13\cdot i_c]\!]$, $id_{j'} = \mathsf{Dec}_{\mathsf{dk}_{i_c}}(c_j)$. The simulator does not abort at this point even if it cannot find dk_{i_c} correctly. It sets $\mathrm{st} = \perp$. It sends $(\mathsf{setup}, \mathrm{st})$ to $\mathcal{D}_1$ and for all $i \in [\![1,4]\!] \setminus \{i_c\}$ latter sends $(\mathsf{pk}_i, \Pi_{\mathsf{ek}_i})$ to $\mathcal{D}_2$ where Π_{ek_i} was produced using the simulator of the zero-knowledge proof.

Shuffle phase: $\mathcal{C}$ sends PK to the adversary $\mathcal{D}_3$. It computes $H_{i_c} \leftarrow \mathsf{GetHand}(i_c, \mathsf{sk}_{i_c}, \mathsf{pk}_{i_c}, \mathsf{PK})$ after extracting $\mathsf{dk}_{i_c} \leftarrow \mathsf{Ext}(\Pi_{\mathsf{ek}_{i_c}})$ if it does not already know it. For all $i \in \{1,2,3,4\}\setminus\{i_c\}$, the simulator picks H_i at random such that $|H_i| = 13$ and $H_i \subset \{\mathsf{id}_l\}_{1\leq l\leq 52} \setminus (H_{i_c} \cup (\cup_{l=1,l\neq i_c}^{i-1} H_l))$.

Game phase: The simulator sets $\gamma = 0$ and $\mathsf{played} = \perp$. The simulator defines the first player index $p_* = 1$ and set $p = p_*$. Then for all $j \in [\![1,52]\!]$:

If $p \neq i_c$**:** The simulator runs $\mathsf{id} \leftarrow \mathsf{Strat}_p(\mathsf{played}, H_p, p_*, p)$, it parses st as $(\alpha, \mathsf{suit}, U_1, U_2, U_3, U_4)$, then it processes as the algorithm $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK})$ except that:

- It picks t at random in $[\![13\cdot(p-1)+1, 13\cdot p]\!]$.
- It computes Π_0 using the simulator $\Pi_0 \leftarrow \mathsf{Sim}(c^*_t, \mathsf{id}, \mathsf{ek}_p)$.
- If $\mathsf{suit}' \neq \mathsf{id}.\mathsf{suit}$ and $|U_i \cup \{t\}| \neq 13$, it computes Π_1 using the simulator Sim to produce $\Pi_1 \leftarrow \mathsf{Sim}(\{c^*_j, \mathsf{Dec}_{\mathsf{dk}_p}(c^*_j)\}_{j\notin U_n\cup\{t\}}, \mathsf{ek}_p)$. Otherwise, it set the proof $\Pi_1 = \perp$.

Set $\Pi = (t, \Pi_0, \Pi_1)$ and send $(\mathsf{id}, \Pi, \mathrm{st}')$ to $\mathcal{A}$ and updates $\mathrm{st} := \mathrm{st}'$. Finally, it updates the index p that points the next player according to the rule of Spades. It then parses played as $(\mathsf{pl}_1, \ldots, \mathsf{pl}_n)$ (where $n = |\mathsf{played}|$) and updates $\mathsf{played} := (\mathsf{pl}_1, \ldots, \mathsf{pl}_n, \mathsf{id})$.

If $p = i_c$**:** The simulator processes as in *Game 1*.

Final phase: The simulator returns 0.

Assume that there exists a distinguisher $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5)$ such that $|\mathsf{P}_{\mathsf{Game}\ 1}(\mathcal{D}, k) - \mathsf{P}_{\mathsf{sim}}(\mathcal{D}, k)| = \lambda(\mathfrak{K})$ where λ is non-negligible. We show how to build an algorithm $\mathcal{B}$ that solves the 52-IND-CPA problem with non-negligible advantage.

First notice that the differences in between *Game 1* and above defined simulator is the generation of zero-knowledge proof and the random ciphertexts played as another card. The proof Π_0 produced at each turn of the challenger creates a association in between a random ciphertext c^*_j with $j \in [\![13\cdot(i-1)+1, 13\cdot i]\!]$ and a card $\mathsf{id}_v \in \{\mathsf{id}_l\}_{1\leq l\leq 52} \setminus H_{i_c}$ that would be played according to the strategy.

Construction of $\mathcal{B}$*:* It will interact an 52-IND-CPA challenger $\mathcal{C}_{52-\mathsf{IND\text{-}CPA}}$ and a tuple of algorithm $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5)$ that is capable in distinguishing in between *Game 1* and our simulator. We show that building on $\mathcal{D}$'s answer we can construct an adversary winning with non-negligible probability against the indistinguishability of $52-\mathsf{IND\text{-}CPA}$. We denote the index of the honest player controlled by $\mathcal{B}$ by $i_1, i_2, i_3 \in \{1,2,3,4\}\setminus\{i_c\}$ such that $i_1 < i_2 < i_3$.

Key generation phase: This algorithm sets $\mathrm{st} = \perp$. It sends setup to $\mathcal{D}_1$ which returns the corrupted index i_c. Then $\mathcal{B}$ draw random values s_i for all $i \in [\![1,4]\!]\setminus\{i_c\}$ and sets $\mathsf{ek}_i = \mathsf{ek}^{s_i}$, where ek is the key sent by the challenger of the 52-IND-CPA game. It simulates $\Pi_{\mathsf{ek}_i} \leftarrow \mathsf{Sim}(\mathsf{ek}_i, g)$ for all $i \in [\![1,4]\!]\setminus\{i_c\}$ and sends $(\mathsf{setup}, \{\mathsf{pk}_i, \Pi_{\mathsf{ek}_i}\}_{i\neq i_c}, \mathsf{vw})$ to $\mathcal{D}_2$ and obtains $\mathsf{pk}_{i_c} = (\mathsf{ek}_{i_c}, \Pi_{\mathsf{ek}_{i_c}})$ as its answer. $\mathcal{B}$ checks the latest proof $\Pi_{\mathsf{ek}_{i_c}}$ and extracts the decryption key dk_{i_c}.

Shuffle phase: $\mathcal{B}$ initialize the canonical deck $D = (\mathsf{id}_1, \ldots, \mathsf{id}_{52})$, computes $\mathsf{pk} = \mathsf{ek}_1 \cdot \mathsf{ek}_2 \cdot \mathsf{ek}_3 \cdot \mathsf{ek}_4$ and $c_{0,j} \leftarrow (g, \mathsf{pk}\cdot\mathsf{id}_j)$. It sets $c_0 \leftarrow (c_{0,j})_{1\leq j\leq 52}$.

Phase 1: For all $i \in [\![1,4]\!]$:

- If $i = i_c$, $\mathcal{D}$ returns $(c_{i_c}, \pi_{i_c,1})$, then $\mathcal{B}$ checks the proof $\pi_{i_c,1}$ that was outputted by the adversary. Then it executes $\mathsf{Ext}^{\mathcal{A}}(\pi_{i_c,1}) \to (\delta_i, (r_{i_c,})_{1\leq j\leq 52})$. $\mathcal{B}$ executes $c'_{i_c,j} \leftarrow \mathsf{Rand}(c_{i_c-1,\delta_{i_c}(j)}, r_{i_c,j}, \mathsf{pk})$ and checks if $c_{i_c,j} = c'_{i_c,j}$ for all $j \in [\![1,52]\!]$. If the proof does not verify or if the equality does not hold then $\mathcal{B}$ returns a random bit.
- If $i = i_1$, when the adversary played before, $\mathcal{B}$ uses the previously extracted permutation to set $\delta = \delta_{i_c}$ else it sets δ to the identity if $i_1 = 1$ and sends $(\mathsf{id}_{\delta j})_{j\in[\![1,52]\!]})$ to $\mathcal{C}_{52-\mathsf{IND\text{-}CPA}}$. The challenger returns a vector $(c'_j)_{j\in[\![1,52]\!]}$, $\mathcal{B}$ parses each $c'_j = (x'_j, y'_j)$ and sets $c'_{i_1} = (x'_j, y'_j = y_j^{b_{i_1}+b_{i_2}+b_{i_3}}$.

$x_j^{'sk_{i_c}} \cdot \mathsf{id}_{\delta(j)}^{-b_1-b_2-b_3+1}$). $\mathcal{B}$ draws random elements δ_{i_1} and $(r_{i_1,j})_{1\leq j\leq 52}$, for all $j \in [\![1,52]\!]$, set $c_{i_1,j} \leftarrow \mathsf{Rand}(c'_{i_1}, r_{i_1,j}, \mathsf{pk})$, set $c_{i_1} = (c_{i_1,j})_{1\leq j\leq 52}$. Then $\mathcal{B}$ uses the simulator in order to produce a zero-knowledge proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, $(c_{i_1}, \pi_{i_1,1})$ is returned to $\mathcal{D}$.

- If $i = i_2$, $\mathcal{B}$ executes the protocol as usual.

Phase 2: On receiving c_4, $\mathcal{B}$ parses it as $(c_{4,j})_{1\leq j\leq 52}$ and each $c_{4,j}$ as (x_j, y_j). For player P_{i_1}, $\mathcal{B}$ sets $\delta = \delta_4 \circ \ldots \circ \delta_1$, sets $\theta_{i_1,j} \xleftarrow{\$} \mathbb{G}^*$ for all $j \in [\![1,52]\!] \setminus (\cup_{i=i_1,i_c} [\![13\cdot(i-1)+1, 13\cdot i]\!])$, and $\theta_{i_1,j} = y_j \cdot \left(\mathsf{id}_{\delta(j)} \cdot x_j^{(\mathsf{dk}_{i_2}+\mathsf{dk}_{i_3}+\mathsf{dk}_{i_c})}\right)^{-1}$ for all $j \in [\![13\cdot(i_c-1)+1, 13\cdot i_c]\!]$. Then algorithm $\mathcal{B}$ uses the simulator to produce the proof $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{x_j, \theta_{(i_1,j)}\}_{j\in[\![1,52]\!]\setminus[\![13\cdot(i_1-1)+1,13\cdot i_1]\!]}, \mathsf{ek}_{i_1})$.

For $i = i_2, i_3$, $\mathcal{B}$ computes $\theta_{(i,j)} = x_j^{\mathsf{dk}_i}$ for $j \in [\![1,52]\!] \setminus [\![13\cdot(i-1)+1, 13\cdot i]\!]$ and their proofs

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

The values $(\theta_{(i,j)})_{j\in[\![1,52]\!]\setminus[\![13\cdot(i-1)+1,13\cdot i]\!]}$ and $\pi_{i,2}$ are broadcasted for all $i = i_1, i_2, i_3$. $\mathcal{D}$ returns $(\theta_{(i_c,j)})_{j\in[\![1,52]\!]\setminus[\![13\cdot(i_c-1)+1,13\cdot i_c]\!]}$ and $\pi_{i_c,2}$.

$\mathcal{B}$ verifies the proof $\pi_{i_c,2}$, executes $\mathsf{Ext}^{\mathcal{A}}(\pi_{i_c,2}) \rightarrow \mathsf{dk}_{i_c}$, and computes $\theta'_{(i_c,j)} = x_j^{\mathsf{dk}_{i_c}}$ for all $j \in [\![1,52]\!] \setminus [\![13\cdot(i_c-1)+1, 13\cdot i_c]\!]$. It checks if $\theta'_{(i,j)} = \theta_{(i,j)}$, if false $\mathcal{B}$ returns a random bit.

Then, for all $j \in [\![1,4]\!]$, for all $j \in [\![13\cdot(j-1)+1, 13\cdot j]\!]$, $\mathcal{B}$ computes $c_j^* \leftarrow \left(x_j, \frac{y_j}{\prod_{1\leq\gamma\leq 4;\gamma\neq l} \theta_{(\gamma,j)}}\right)$. $\mathcal{B}$ send $\mathsf{PK} \leftarrow (c_j^*)_{1\leq j\leq 52}$ to $\mathcal{D}$ as the result of this algorithm. If the result obtain by $\mathcal{D}$ and $\mathcal{B}$ for PK is different or equal to $\perp$, $\mathcal{B}$ returns a random bit.

Game phase: $\mathcal{B}$ sets p equal to the first player index $p_* = 1$, $\gamma = 0$ and $\mathsf{played} = \perp$. For $j \in [\![1,52]\!]$:

If $p \neq i_c$: $\mathcal{B}$ runs $\mathsf{id} \leftarrow \mathsf{Strat}_p(\mathsf{played}, H_p, p_*, p)$, then it processes as in the algorithm $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK}, D)$ except that:

- It chooses t at random instead than setting t to the coherent value.
- It computes proofs Π_0 and Π_1 as in the above defined simulator $\mathsf{Sim}^{\mathsf{TrickT}}_{W,S,\mathcal{A}(\mathsf{vw})}(\mathfrak{K}, \mathsf{setup}, s, \{\mathsf{pk}_i\}_{1\leq i\leq 4}, \mathsf{PK}, \mathsf{vw})$.

It sets $\Pi = (t, \Pi_0, \Pi_1)$ and sends (Π, st') to $\mathcal{D}_4$. Then it updates $\mathrm{st} := \mathrm{st}'$ and the index p that points the next player according to the rule of Spades. It then parses played as $(\mathsf{pl}_1, \ldots, \mathsf{pl}_n)$ (where $n = |\mathsf{played}|$) and updates $\mathsf{played} := (\mathsf{pl}_1, \ldots, \mathsf{pl}_n, \mathsf{id})$.

If $p = i_c$: $\mathcal{B}$ processes as in $\mathsf{Exp}^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}}(\mathfrak{K})$.

Final phase The simulated experiment returns 1.

Finally, $\mathcal{B}$ runs $b_* \leftarrow \mathcal{D}_5(\mathsf{vw})$, where vw denotes all the values send and received by $\mathcal{D}$ during its interaction with the simulated experiment, then $\mathcal{B}$ returns b_*.

Note that an adversary solving the 39-IND-CPA problem can solve the 52-IND-CPA problem.

Analysis: We distinguish two cases:

- The adversary forges a proof of a false statement during the DeckGen protocol. In this case, if $\mathcal{D}$ does not produce a valid proof for the false statement, then $\mathsf{PK} = \perp$ so the experiment aborts, hence the advantage of $\mathcal{B}$ is lower than $2 \cdot \epsilon_{\mathsf{sound}}(\mathfrak{K})$.
- The adversary does not forge a proof of a false statement during the DeckGen protocol. In this case, if $\mathsf{PK} = \perp$ then $\mathcal{D}_5$ has no information hence cannot do better than answering randomly. Now, assuming that $\mathsf{PK} \neq \perp$ and $b = 1$, for all $v \in [\![1,52]\!]$, there exist $i \in [\![1,4]\!]$ and $j \in [\![13\cdot(i-1)+1, 13\cdot i]\!]$ such that $\mathsf{id}_v = \mathsf{Dec}_{\mathsf{dk}_i}(c^*_{4,j})$.

  In the following, we show that in this case, the advantage of $\mathcal{B}$ is lower than $2 \cdot \epsilon_{39-\mathsf{IND}-\mathsf{CPA}}(\mathfrak{K})$. If $b = 1$, then the experiment is perfectly simulated, else, the simulator is perfectly simulated. We observe that:

$$\begin{aligned}\mathsf{Pr}[\mathcal{B} \text{ wins}] &= \frac{1}{2}\left(\mathsf{Pr}[1 \leftarrow \mathcal{D}_5(\mathsf{vw})|b=1] + \mathsf{Pr}[0 \leftarrow \mathcal{D}_5(\mathsf{vw})|b=0]\right)\\ &= \frac{1}{2}\left(\mathsf{Pr}[1 \leftarrow \mathcal{D}_5(\mathsf{vw})|b=1] + 1 - \mathsf{Pr}[1 \leftarrow \mathcal{D}_5(\mathsf{vw})|b=0]\right).\end{aligned}$$

Finally:

$$\left|\mathsf{Pr}[\mathcal{B}\text{ wins}] - \frac{1}{2}\right| = \frac{1}{2}\cdot|\mathsf{P}_{\mathsf{real}}(\mathcal{D},k) - \mathsf{P}_{\mathsf{sim}}(\mathcal{D},k)| = \frac{\lambda(\mathfrak{K})}{2}.$$

The two cases imply that the advantage of $\mathcal{B}$ is lower than $2(\epsilon_{39-\mathsf{IND\text{-}CPA}}(\mathfrak{K}) + \epsilon_{\mathsf{sound}}(\mathfrak{K}))$, which concludes the proof. □

F Proof for the Tarot Scheme

The Tarot protocol relies on the security model highlighted in Section 4.2 and formally described in Appendix C. Here we prove Theorem 2 by proving the lemmas of this section. As our Tarot protocol share a common structure with our Spades protocol arguments are directly adapted form the previous security considerations. Some part of the below proofs follows directly from previously presented reductions.

We assume that the adversary $\mathcal{A}$ determines the taker and its bets. Hence, it is able to simulate all type of scenario at needs regarding the bets. Depending on the announcements two cases have to be distinguished: When *garde sans* or *garde contre* is picked by $\mathcal{A}$ and the second case when *petite* or *garde* is chosen. In the latest case, the shuffle protocol DeckGen continue through Part 4. Also, additional consideration need to be done: the taker could be simulated by the adversary or by the challenger.

Lemma 6. *If our Tarot protocol is instantiated by proofs of knowledge that are sound and extractable then it is* theft-resistant. *The adversary's advantage against theft-resistance is bounded by* $\mathsf{Adv}^{th-res}_{W,S,\mathcal{A}}(\mathfrak{K}) \leq (11+18)\cdot\epsilon_{\mathsf{extract}}(\mathfrak{K})$.

Proof. This proof uses similar arguments as in the proof given for Lemma 1. Now players have 18 cards in their hand, hence the hybrid arguments need to be adapted. Let $i_{guess} \in \{1,\ldots,5\}$ and the associated experiment $\mathsf{Exp}^{\mathsf{Exp}^{TrickT}}_{W,S,\mathcal{A}}(\mathfrak{K}, i_{guess})$.

Game 0: This is the Trick-Taking experiment $\mathsf{Exp}^{\mathsf{Exp}^{TrickT}}_{W,S,\mathcal{A}}(\mathfrak{K}, i_{guess})$.
Game 1: $\mathcal{A}$ has to output a valid zero knowledge proof for P_{i_c}'s key. Under soundness of the NIZKP used for $\Pi_{\mathsf{pk}_{i_c}}$, we have that the extracted key sk_{i_c} verifies $g^{\mathsf{sk}_{i_c}} = \mathsf{pk}_{i_c}$. $\mathcal{C}$ check this equality and aborts the game if this does not hold. Thus, $\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
The three following games modify the behaviours of the challenger during the first phase of KeyGen.
Game 2: In this game we decrypt the elements $(c_{i_c,j})_{1\leq j\leq 78}$ using sk_{i_c} obtained previously. The challenger aborts if $\{\mathsf{Dec}_{\mathsf{sk}_{i_c}}(c_{i_c,j})\}_{1\leq j\leq 78} \neq D$. Under soudness of the NIZKP, we obtain $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 3: The same is done for $\mathsf{P}_{i^1_a}$, its key have been produced by the challenger, hence we do not rely on the soudness of any proof before this.
Game 4: Same for $\mathsf{P}_{i^1_a}$. The three following games modify the behaviours of the challenger during the second phase of KeyGen.
Game 5: P_{i_c} secret key sk_{i_c} underlying P_{i_c}'s proof of phase 2 is extracted. $\mathcal{C}$ checks whether for all $j \in [\![1,72]\!] \setminus [\![18\cdot(i_c-1)+1, 18\cdot i_c]\!]$, the equality $\theta_{(i_c,j)} = x_j^{\mathsf{sk}_{i_c}}$ holds, if not, $\mathcal{C}$ abort the game. We obtain $\mathsf{Adv}^{indist}_{G_4,G_5}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 6: Same for $\mathsf{P}_{i^1_a}$.
Game 7: Same for $\mathsf{P}_{i^2_a}$.
Game 8: After the shuffle, $\mathcal{C}$ receive a player index i_{Dog}, and executes MakeDog with the adversary. During this protocol, we can assess the right execution of the protocol by the various players controlled by $\mathcal{A}$. On receiving $\pi'_{i_c,2}$, we extract pk_i and reproduce the computations made by $\mathcal{A}$ and abort if it does not match.
Game 9: Same for $\mathsf{P}_{i^1_a}$.
Game 10: Same for $\mathsf{P}_{i^2_a}$.
The latest games guaranty that all the three maliciously controlled players have to behave according to the protocol up to the reveal of the cards in the dog. Depending on the taker index n, if $n \in \{i_c, i^1_a, i^2_a\}$ we need

the following game: Game 11: $\mathcal{C}$ extract the secret value underlying proof Π_n or $\Pi_{n,2}$ and reproduce the computation. If this is not consistent with the return value it aborts.

In Game 11, $\mathcal{C}$ has reproduced all the actions of $\mathcal{A}$ through the three players it controls, hence the shuffle have been produced according to the protocol unless the adversary was able to break soudness of one of the ZKP.
Game 12: When $\mathcal{A}$ plays a card $(\mathsf{id}, \Pi = (\Pi_0, \Pi_1, t))$ associated to a cipher c_j as P_{i_c}, $\mathcal{C}$ now decrypt the value behind c_j with sk_{i_c} and checks if this is coherent with the given proof Π_0. The adversary can try to cheat during any of the 18 turns it plays, hence by a hybrid argument we have $\mathsf{Adv}^{indist}_{G_{11},G_{12}}(\mathcal{A}) \leq 18 \cdot \epsilon_{sound}(\mathfrak{K})$ and as $\mathcal{A}$ has no way of winning this concludes the proof. □

Lemma 7. *Instantiated by proofs of knowledge that are sound and extractable, our Tarot protocol is* cheating-resistant. *The adversary's advantage against this property is at most* $\mathsf{Adv}^{cheat}_{W,S,\mathcal{A}}(\mathfrak{K}) \leq 47 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

Proof. Game 0: This is the cheating-resistance experiment (*i.e.,* trick-taking experiment with the winning condition associated to the cheating resistance).
Game 1: $\mathcal{C}$ copies the behaviours of the challenger in Game 11 in the proof of lemma 6, the winning condition remains the one associated to the cheating resistance. As justified before, we obtain $\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}) \leq 11 \cdot \epsilon_{sound}(\mathfrak{K})$.
Game 2: During P_{i_c}'s plays, $\mathcal{C}$ that knows its hand verifies that the played card is indeed in the corrupted player's hand, instead of validating the ZKP. This implies a negligible modification ϵ_π for each of the played cards. The hybrid argument associated has been given previously. We achieve $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}) \leq 18 \cdot \epsilon_{sound}(\mathfrak{K})$.
Game 3: During P_{i_c}'s plays, $\mathcal{C}$ checks that the played card follows the Tarot's rules validating the ZKP using the knowledge of P_{i_c}'s hand instead of verifying the ZKP $\pi_{i_c,2}$. This implies a negligible modification ϵ_π for each of the played cards. The hybrid argument associated has been given previously. We achieve $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}) \leq 18 \cdot \epsilon_{sound}(\mathfrak{K})$.

At this stage of the game, it becomes impossible for $\mathcal{A}$ to plays invalid cards on P_{i_c}'s turn as the challenger always verifies if it has the card in its hand before accepting the play. This concludes the proof. We conclude to $\mathsf{Adv}^{indist}_{G_0,G_3}(\mathcal{A}) \leq 47 \cdot \epsilon_{sound}(\mathfrak{K})$. □

Lemma 8. *If our Tarot protocol is instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, and assuming hardness of the DDH problem, then the protocol is* unpredictable *and* $\mathsf{Adv}^{Unp}_{W,\mathcal{A}}(\mathfrak{K}, i_{\mathsf{guess}}) \leq 7 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K}) + 78 \cdot \mathsf{Adv}^{DDH}_{\mathcal{A}}(\mathfrak{K})$.

Proof. Let $i_{guess} \in \{1, \ldots, 5\}$ and the associated experiment $\mathsf{Exp}^{Unpre}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$. We provide a game hope independent of the parameter i_{guess}, this allows us to provide the same reduction for all cases.
Game 0: Let Game 0 be the is unpredicatability experiment $\mathsf{Exp}^{Unpre}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$.
Game 1: $\mathcal{A}$ has to output a valid zero knowledge proof for P_{i_c}'s key. Under soundness of the NIZKP used for $\Pi_{\mathsf{pk}_{i_c}}$, we have that the extracted key sk_{i_c} verifies $g^{\mathsf{sk}_{i_c}} = \mathsf{pk}_{i_c}$. $\mathcal{C}$ check this equality and aborts the game if this does not hold. Thus, $\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
The three following games modify the behaviours of the challenger during the first phase of KeyGen.
Game 2: In this game we decrypt the elements $(c_{i_c,j})_{1 \leq j \leq 78}$ using sk_{i_c} obtained previously. The challenger aborts if $\{\mathsf{Dec}_{\mathsf{sk}_{i_c}}(c_{i_c,j})\}_{1 \leq j \leq 78} \neq D$. Under soudness of the NIZKP, we obtain $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 3: The same is done for $\mathsf{P}_{i_a^1}$, its key have been produced by the challenger, hence we do not rely on the soudness of any proof before this.
Game 4: Same for $\mathsf{P}_{i_a^1}$. The three following games modify the behaviours of the challenger during the second phase of KeyGen.
Game 5: P_{i_c} secret key sk_{i_c} underlying P_{i_c}'s proof of phase 2 is extracted. $\mathcal{C}$ checks whether for all $j \in [\![1, 72]\!] \setminus [\![18 \cdot (i_c - 1) + 1, 18 \cdot i_c]\!]$, the equality $\theta_{(i_c,j)} = x_j^{\mathsf{sk}_{i_c}}$ holds, if not, $\mathcal{C}$ abort the game. We obtain $\mathsf{Adv}^{indist}_{G_4,G_5}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.
Game 6: Same for $\mathsf{P}_{i_a^1}$.
Game 7: Same for $\mathsf{P}_{i_a^2}$.

The latest games guaranty that all the three maliciously controlled players have to behave according to the protocol up to the end of the KeyGen protocol. We now show that it is impossible for an adversary following the prescripted operations to influence the outcome of the shuffle even when it controls all but one player. We investigated two scenarios. The first one will assume that the last player to shuffle is the challenger, while the second case is to consider the case where this last player is controlled by the adversary.

Case $\{i_c, i_a^1, i_a^2\} = \{1,2,3\}$**:** P_4 is run honestly executed, hence the last step of the shuffle maps the cards randomly and distribute them uniformly to all players. This concludes our case as it is impossible for $\mathcal{A}$ to predict the outputted values after the last random permutation nor to influence the position of the cards afterwards.

Case $4 \in \{i_c, i_a^1, i_a^2\}$**:** The ElGamal encryption is an IND-CPA cipher, this means that under encryption, $\mathcal{A}$ does not learn the cards underlying a ciphertext. We have shown that under this consideration, $\mathcal{A}$'s permutations cannot be chosen better than random as it cannot infer the value of the cards after an honest shuffle. We prove this statement using the following game.
Game 8: Let $i_{\mathcal{C}} \in \{1,2,3\}$ be the challenger's index. We are now making some changes in the behaviours of the challenger while acting as the honest player $\mathsf{P}_{i_{\mathcal{C}}}$. During part 1 of the DeckGen algorithm, on receiving $c_{i_{\mathcal{C}}-1}$, it parses it as $(c_{i_{\mathcal{C}}-1,j})_{1\leq j\leq 78} = (x_j, y_j)_{1\leq j\leq 78}$. $\mathcal{C}$ will then draw a random vector $(Z_{i_{\mathcal{C}},j})_{1\leq j\leq 78} \xleftarrow{\$} \mathbb{G}^{78}$ and the usual $\delta_{i_{\mathcal{C}}}, (r_{i_{\mathcal{C}},j})_{1\leq j\leq 78}$. Then $c_{i_{\mathcal{C}},j} \leftarrow \mathsf{Rand}((x_{\delta_{i_{\mathcal{C}}}(j)}, Z_{\delta_{i_{\mathcal{C}}}(j)} \cdot y_{\delta_{i_{\mathcal{C}}}(j)}), r_{i_{\mathcal{C}},j}, \mathsf{pk})$ is computed. Then $\mathcal{C}$ uses the simulator in order to obtain a proof $\pi_{i_{\mathcal{C}},1} \leftarrow \mathsf{Sim}(c_{i_{\mathcal{C}}-1}, c_{i_{\mathcal{C}}}, \mathsf{pk})$. Finally, we return $(c_{i_{\mathcal{C}}}, \pi_{i_{\mathcal{C}},1})$ to the adversary. The value $Z_{i_{\mathcal{C}},j}$ completely hide the underlying message of the ciphertext as $Z_{i_{\mathcal{C}},j} \cdot y_{\delta_{i_{\mathcal{C}}}(j)} = (Z \cdot \mathsf{id}_{\delta_{i_{\mathcal{C}}} \circ \ldots \circ \delta_1(j)}) \cdot \mathsf{pk}^{\sum_{l=1}^{i_{\mathcal{C}}} r_l}$, for some integers r_l that are introduced through the randomness of the previous players.

During the second phase of DeckGen we are making the following changes: in order to let the adversary recover the values for its cards we compute $\theta_{i_{\mathcal{C}},j} = Z_{\delta_4 \circ \ldots \circ \delta_{i_{\mathcal{C}}}(j)} \cdot \mathsf{sk}_{i_{\mathcal{C}}}^{\sum_{l=1}^{4} r_{\delta_l \circ \ldots \circ \delta_1(j)}}$ for $j \in [\![1,78]\!] \setminus [\![18 \cdot (i_{\mathcal{C}}-1)+1, 18 \cdot i_{\mathcal{C}}]\!]$. $\mathcal{C}$ parses c_4 as $(c_{4,j})_{j\in[\![1,78]\!]}$ and $c_{4,j}$ as (x_j, y_j). Then using the simulator it produces $\pi_{i_{\mathcal{C}},2} \leftarrow \mathsf{Sim}(\{\theta_{i_{\mathcal{C}},j}, x_j\}_j, \mathsf{pk}_{i_{\mathcal{C}}})$ and sends $(\{\theta_{i_{\mathcal{C}},j}\}_{j\in[\![1,78]\!]\setminus[\![18\cdot(i_{\mathcal{C}}-1)+1,18\cdot i_{\mathcal{C}}]\!]}, \pi_{i_{\mathcal{C}},2})$ to $\mathcal{A}$. The remaining part of the game is unchanged.

Reduction: A similar reduction has been presented to prove Lemma 3.

Due to the verification of the ZKP $\mathcal{A}$ has to follow the protocol and pick values to influence its outcome. With the latest modifications, once $\mathcal{C}$ proceeds to its shuffle, it distributes the ciphertext uniformly and hide them using random values. In *Game 8*, $\mathcal{A}$ is unable to do better that trying random values as it only sees random values during the entire shuffle. We lead to the conclusion that $\mathsf{Adv}_{W,\mathcal{A}}^{Unp}(\mathfrak{K}, i_{\mathsf{guess}}) \leq 7 \cdot \epsilon_{\mathsf{extract}}(\mathfrak{K}) + 78 \cdot \mathsf{Adv}_{\mathcal{A}}^{\mathsf{DDH}}(\mathfrak{K})$ for each of the possible index. Hence, the sum on $i_{guess} \in \{1,\ldots,5\}$ is also negligible. □

Lemma 9. *Assuming hardness of the DDH problem. If our Tarot protocol is instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, then it is* hand-private.

Proof. This proof the same arguments as in the proof of Lemma 4, only the cardinal of the deck is modified, and the index value is set in parameter of the experiment and not obtained from the adversary. The final advantage for this lemma is:

$$\mathsf{Adv}_{\mathcal{A}}^{h-priv}(\mathfrak{K}, i_{\mathsf{guess}}) \leq 3 \cdot \epsilon_{\mathsf{sound}}(\mathfrak{K}) + 3 \cdot 78 \cdot \mathsf{Adv}^{\mathsf{DDH}}(\mathfrak{K}).$$

Lemma 10. *Instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, our Tarot protocol is* Game-private *under the DDH assumption.*

Proof. Let $K = \left(\mathsf{setup}, (\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}}), \left\{\mathsf{pk}_i, \mathsf{sk}_i, \Pi_{\mathsf{pk}_i}\right\}_{1\leq i\leq 4; i\neq i_c}, \mathsf{PK}\right)$, we consider the experiment $Exp_{W,S,K,\mathcal{D}}^{\mathsf{TrickT}}(\mathfrak{K})$ modified according to Definition 20. This game will be denoted as *Game 0*. In this experiment $\mathcal{A}$ controls P_{i_c} and $\mathcal{C}$ controls the three other players that we index by $i_1 < i_2 < i_3$.
Game 1: We modify the behaviours of the first honest player P_{i_1}. During part 1 of the DeckGen algorithm, on obtaining c_{i_1-1}, it parses it as $(c_{i_1-1,j})_{1\leq j\leq 78} = (x_j, y_j)_{1\leq j\leq 78}$. $\mathcal{C}$ will then draw a random vector

$(Z_{i_1,j})_{1\leq j\leq 78} \xleftarrow{\$} \mathbb{G}^{78}$ and the usual $\delta_{i_1}, (r_{i_1,j})_{1\leq j\leq 78}$. Then $c_{i_1,j} \leftarrow \mathsf{Rand}((x_{\delta_{i_1}(j)}, Z_{\delta_{i_1}(j)} \cdot y_{\delta_{i_1}(j)}), r_{i_1,j}, \mathsf{pk})$ is computed. Then $\mathcal{C}$ uses the simulator in order to obtain a proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, we return $(c_{i_1}, \pi_{i_1,1})$ to the adversary.

Then during the second phase of DeckGen, in order to let the adversary recover the values of the cards in its hand, we will compute the $\theta_{i_1,j} = Z_{\delta_4\circ\ldots\circ\delta_{i_1}(j)} \cdot \mathsf{pk}_{i_1}^{\sum_{l=1}^{4} r_{\delta_l\circ\ldots\circ\delta_1(j)}}$ for $j \in [\![18\cdot(i_c-1)+1, 18\cdot i_c]\!]$, for all $j \in [\![1,78]\!] \setminus (\cup_{i=i_1,i_c}[\![18\cdot(i-1)+1, 18\cdot i]\!])$ the $\theta_{i_1,j}$ are draw randomly. We parse c_4 as $(c_{4,j})_{j\in[\![1,78]\!]}$ and $c_{4,j}$ as (x_j, y_j). Then using the simulator the challenger produces $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{\theta_{i_1,j}, x_j\}_j, \mathsf{pk}_{i_1})$. Finally, we send $(\{\theta_{i_1,j}\}_{j\in[\![1,78]\!]\setminus[\![18\cdot(i_1-1)+1,18\cdot i_1]\!]}, \pi_{i_1,2})$ to $\mathcal{A}$. The remaining part of the game is conducted as previously.

Doing so, we are completely hiding the cards in the challenger's hands to the adversary's as we have outputted random values $\theta_{i_1,j}$ for $j \in [\![1,78]\!] \setminus (\cup_{i=i_1,i_c}[\![18\cdot(i-1)+1, 18\cdot i]\!])$ and that the $c^*_{4,j}$ with $j \in [\![18\cdot(i_1-1)+1, 18\cdot i_1]\!]$ are hidden by some random values.

With this we also need to modify the values that we return during the MakeDog algorithm when player P_{i_c} takes. In the same way as before when $\mathcal{C}$ acts as P_{i_1} it produces values $\theta_{i_1,j} = Z_{\delta_4\circ\ldots\circ\delta_{i_1}(j)} \cdot \mathsf{pk}_{i_1}^{\sum_{l=1}^{4} r_{\delta_l\circ\ldots\circ\delta_1(j)}}$ for $j \in [\![73,78]\!]$. This value works just like in the shuffle DeckGen and uncover the random values that $\mathcal{A}$ sees in the dog.

Construction of $\mathsf{Sim}^{\mathsf{TrickT}}_{W,S,\mathcal{A}(\mathsf{vw})}\left(\mathfrak{K}, \mathit{setup}, i_c, \{\mathsf{pk}_i, \Pi_{\mathsf{pk}_i}\}_{1\leq i\leq 4}, \mathsf{PK}, \mathsf{vw}\right)$*:*

Key generation phase: The simulator starts by setting st $=\perp$ and sends $(\mathsf{setup}, \mathrm{st})$ to $\mathcal{D}_1$, for all $i \in [\![1,4]\!] \setminus \{i_c\}$, it sends $(\mathsf{pk}_i, \Pi_{\mathsf{pk}_i})$ to $\mathcal{D}_2$ where Π_{pk_i} was produced using the simulator of the zero-knowledge proof. On receiving the pair $(\mathsf{pk}_{i_c}, \Pi_{\mathsf{pk}_{i_c}})$ it extracts the secret value sk_{i_c} from $\Pi_{\mathsf{pk}_{i_c}}$.

Shuffle phase: $\mathcal{C}$ sends PK to $\mathcal{D}_3$. It computes $H_{i_c} \leftarrow \mathsf{GetHand}(i_c, \mathsf{sk}_{i_c}, \mathsf{pk}_{i_c}, \mathsf{PK})$. If $(1, n)$ was received from $\mathcal{D}_3$, the simulator takes random cards for the dog, reveals them and produces the necessary proofs π_5 for all $i \in [\![1,4]\!] \setminus \{i_c\}$. When $n \neq i_c$, first it draws hands for the three players it controls: For all $i \in \{1,2,3,4\} \setminus \{i_c\}$, the simulator picks H_i at random such that $|H_i| = 18$ and $H_i \subset \{\mathsf{id}_l\}_{1\leq l\leq 78} \setminus (H_{i_c} \cup (\cup_{l=1, l\neq i_c}^{i-1} H_l) \cup H_5)$. Then it keeps on picking random cards within $H_n \cup H_5$, where H_5 represent the dog, in order to produce a valid dog regarding the rules of Tarot. It then simulates the needed proofs.

Game phase: The simulator sets $\gamma = 0$ and played $=\perp$. The simulator defines the first player index $p_* = 1$ and set $p = p_*$. Then for all $j \in [\![1,78]\!]$:

If $p \neq i_c$**:** The simulator runs $\mathsf{id} \leftarrow \mathsf{Strat}_p(\mathsf{played}, H_p, p_*, p)$, it parses st as $(\alpha, \mathsf{suit}, U_1, U_2, U_3, U_4)$, then it processes as the algorithm $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK})$ except that:

- It picks t at random in $[\![18\cdot(p-1)+1, 18\cdot p]\!]$.
- It computes Π_0 using the simulator $\Pi_0 \leftarrow \mathsf{Sim}(c^*_t, \mathsf{id}, \mathsf{pk}_p)$.
- If $\mathsf{suit}' \neq \mathsf{id.suit}$ and $|U_i \cup \{t\}| \neq 18$, it computes Π_1 using the simulator Sim to produce $\Pi_1 \leftarrow \mathsf{Sim}(\{c^*_j, \mathsf{Dec}_{\mathsf{sk}_p}(c^*_j)\}_{j\notin U_n\cup\{t\}}, \mathsf{pk}_p)$. Otherwise, it set the proof $\Pi_1 = \perp$.

Set $\Pi = (t, \Pi_0, \Pi_1)$ and send $(\mathsf{id}, \Pi, \mathrm{st}')$ to $\mathcal{A}$ and updates st $:=$ st$'$. Finally, it updates the index p that points the next player according to the rule of the French Tarot. It then parses played as $(\mathsf{pl}_1, \ldots, \mathsf{pl}_n)$ (where $n = |\mathsf{played}|$) and updates $\mathsf{played} := (\mathsf{pl}_1, \ldots, \mathsf{pl}_n, \mathsf{id})$.

If $p = i_c$**:** The simulator processes as in *Game 1*.

Final phase: The simulator returns 0.

Assume that there exists a distinguisher $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6)$ such that $|\mathsf{P}_{\mathsf{Game}\ 1}(\mathcal{D}, k) - \mathsf{P}_{\mathsf{sim}}(\mathcal{D}, k)| = \lambda(\mathfrak{K})$ where λ is non-negligible. We show how to build an algorithm $\mathcal{B}$ that solves the 78-IND-CPA problem with non-negligible advantage.

First notice that the differences in between *Game 1* and above defined simulator is the generation of zero-knowledge proof and the random ciphertexts played as another card. The proof Π_0 produced at each turn of the challenger creates an association in between a random ciphertext c^*_j with $j \in [\![18\cdot(i-1)+1, 18\cdot i]\!]$ and a card $\mathsf{id}_v \in \{\mathsf{id}_l\}_{1\leq l\leq 78} \setminus H_{i_c}$ that would be played according to the strategy.

Construction of $\mathcal{B}$*:* It will interact an 78-IND-CPA challenger $\mathcal{C}_{78-\mathsf{IND\text{-}CPA}}$ and a tuple of algorithm $\mathcal{D} = (\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4, \mathcal{D}_5, \mathcal{D}_6)$ that is capable in distinguishing in between *Game 1* and our simulator. We show that building on $\mathcal{D}$'s answer we can construct an adversary winning with non-negligible probability against

the indistinguishably of $78-\mathsf{IND\text{-}CPA}$. We denote the index of the honest player controlled by $\mathcal{B}$ by $i_1, i_2, i_3 \in \{1,2,3,4\} \setminus \{i_c\}$ such that $i_1 < i_2 < i_3$.

Key generation phase: This algorithm sets $\mathsf{st} = \perp$. It sends setup to $\mathcal{D}_1$ which returns the corrupted index i_c. Then $\mathcal{B}$ draw random values s_i for all $i \in [\![1,4]\!] \setminus \{i_c\}$ and sets $\mathsf{ek}_i = \mathsf{ek}^{s_i}$, where ek is the key sent by the challenger of the 78-IND-CPA game. It simulates $\Pi_{\mathsf{ek}_i} \leftarrow \mathsf{Sim}(\mathsf{ek}_i, g)$ for all $i \in [\![1,4]\!] \setminus \{i_c\}$ and sends $(\mathsf{setup}, \{\mathsf{pk}_i, \Pi_{\mathsf{ek}_i}\}_{i \neq i_c}, \mathsf{vw})$ to $\mathcal{D}_2$ and obtains $\mathsf{pk}_{i_c} = (\mathsf{ek}_{i_c}, \Pi_{\mathsf{ek}_{i_c}})$ as its answer. $\mathcal{B}$ checks the latest proof $\Pi_{\mathsf{ek}_{i_c}}$ and extracts the decryption key dk_{i_c}.

Shuffle phase: $\mathcal{B}$ initialize the canonical deck $D = (\mathsf{id}_1, \ldots, \mathsf{id}_{78})$, computes $\mathsf{pk} = \mathsf{ek}_1 \cdot \mathsf{ek}_2 \cdot \mathsf{ek}_3 \cdot \mathsf{ek}_4$ and $c_{0,j} \leftarrow (g, \mathsf{pk} \cdot \mathsf{id}_j)$. It sets $c_0 \leftarrow (c_{0,j})_{1 \leq j \leq 78}$.

Phase 1: For all $i \in [\![1,4]\!]$:

- If $i = i_c$, $\mathcal{D}$ returns $(c_{i_c}, \pi_{i_c,1})$, then $\mathcal{B}$ checks the proof $\pi_{i_c,1}$ that was outputted by the adversary. Then it executes $\mathsf{Ext}^{\mathcal{A}}(\pi_{i_c,1}) \rightarrow (\delta_i, (r_{i_c,})_{1 \leq j \leq 78})$. $\mathcal{B}$ executes $c'_{i_c,j} \leftarrow \mathsf{Rand}(c_{i_c-1,\delta_{i_c}(j)}, r_{i_c,j}, \mathsf{pk})$ and checks if $c_{i_c,j} = c'_{i_c,j}$ for all $j \in [\![1,78]\!]$. If the proof does not verify or if the equality does not hold then $\mathcal{B}$ returns a random bit.
- If $i = i_1$, when the adversary played before, $\mathcal{B}$ uses the previously extracted permutation to set $\delta = \delta_{i_c}$ else it sets δ to the identity if $i_1 = 1$ and sends $(\mathsf{id}_{\delta j})_{j \in [\![1,78]\!]})$ to $\mathcal{C}_{78-\mathsf{IND\text{-}CPA}}$. The challenger returns a vector $(c'_j)_{j \in [\![1,78]\!]}$, $\mathcal{B}$ parses each $c'_j = (x'_j, y'_j)$ and sets $c'_{i_1} = (x'_j, y'_j = y_j^{b_{i_1}+b_{i_2}+b_{i_3}} \cdot x_j^{'sk_{i_c}} \cdot \mathsf{id}_{\delta(j)}^{-b_1-b_2-b_3+1})$. $\mathcal{B}$ draws random elements δ_{i_1} and $(r_{i_1,j})_{1 \leq j \leq 78}$, for all $j \in [\![1,78]\!]$, set $c_{i_1,j} \leftarrow \mathsf{Rand}(c'_{i_1}, r_{i_1,j}, \mathsf{pk})$, set $c_{i_1} = (c_{i_1,j})_{1 \leq j \leq 78}$. Then $\mathcal{B}$ uses the simulator in order to produce a zero-knowledge proof $\pi_{i_1,1} \leftarrow \mathsf{Sim}(c_{i_1-1}, c_{i_1}, \mathsf{pk})$. Finally, $(c_{i_1}, \pi_{i_1,1})$ is returned to $\mathcal{D}$.
- If $i = i_2$, $\mathcal{B}$ executes the protocol as usual.

Phase 2: On receiving c_4, $\mathcal{B}$ parses it as $(c_{4,j})_{1 \leq j \leq 78}$ and each $c_{4,j}$ as (x_j, y_j). For player P_{i_1}, $\mathcal{B}$ sets $\delta = \delta_4 \circ \ldots \circ \delta_1$, sets $\theta_{i_1,j} \xleftarrow{\$} \mathbb{G}^*$ for all $j \in [\![1,78]\!] \setminus (\cup_{i=i_1,i_c} [\![18 \cdot (i-1)+1, 18 \cdot i]\!])$, and $\theta_{i_1,j} = y_j \cdot \left(\mathsf{id}_{\delta(j)} \cdot x_j^{(\mathsf{dk}_{i_2}+\mathsf{dk}_{i_3}+\mathsf{dk}_{i_c})}\right)^{-1}$ for all $j \in [\![18 \cdot (i_c-1)+1, 18 \cdot i_c]\!]$. Then algorithm $\mathcal{B}$ uses the simulator to produce the proof $\pi_{i_1,2} \leftarrow \mathsf{Sim}(\{x_j, \theta_{(i_1,j)}\}_{j \in [\![1,78]\!] \setminus [\![18 \cdot (i_1-1)+1, 18 \cdot i_1]\!]}, \mathsf{ek}_{i_1})$.

For $i = i_2, i_3$, $\mathcal{B}$ computes $\theta_{(i,j)} = x_j^{\mathsf{dk}_i}$ for $j \in [\![1,78]\!] \setminus [\![18 \cdot (i-1)+1, 18 \cdot i]\!]$ and their proofs $\pi_{i,2} \leftarrow \mathsf{ZK}\left\{\mathsf{dk}_i : \bigwedge_{j \in [\![1,78]\!] \setminus [\![18 \cdot (i-1)+1, 18 \cdot i]\!]} \theta_{(i,j)} = x_j^{\mathsf{dk}_i} \wedge \mathsf{pk}_i = g^{\mathsf{sk}_i}\right\}$.

The values $(\theta_{(i,j)})_{j \in [\![1,78]\!] \setminus [\![18 \cdot (i-1)+1, 18 \cdot i]\!]}$ and $\pi_{i,2}$ are broadcasted for all $i = i_1, i_2, i_3$. $\mathcal{D}$ returns $(\theta_{(i_c,j)})_{j \in [\![1,78]\!] \setminus [\![18 \cdot (i_c-1)+1, 18 \cdot i_c]\!]}$ and $\pi_{i_c,2}$.

$\mathcal{B}$ verifies the proof $\pi_{i_c,2}$, executes $\mathsf{Ext}^{\mathcal{A}}(\pi_{i_c,2}) \rightarrow \mathsf{dk}_{i_c}$, and computes $\theta'_{(i_c,j)} = x_j^{\mathsf{dk}_{i_c}}$ for all $j \in [\![1,78]\!] \setminus [\![18 \cdot (i_c-1)+1, 18 \cdot i_c]\!]$. It checks if $\theta'_{(i,j)} = \theta_{(i,j)}$, if false $\mathcal{B}$ returns a random bit.

Then, for all $j \in [\![1,4]\!]$, for all $j \in [\![18 \cdot (j-1)+1, 18 \cdot j]\!]$, $\mathcal{B}$ computes $c^*_j \leftarrow \left(x_j, \frac{y_j}{\prod_{1 \leq \gamma \leq 4; \gamma \neq l} \theta_{(\gamma,j)}}\right)$. $\mathcal{B}$ send $\mathsf{PK} \leftarrow (c^*_j)_{1 \leq j \leq 78}$ to $\mathcal{D}$ as the result of this algorithm. If the result obtain by $\mathcal{D}$ and $\mathcal{B}$ for PK is different or equal to $\perp$, $\mathcal{B}$ returns a random bit.

Once the $\mathsf{DeckGen}$ Algorithm has been executed, we proceed to $\mathsf{MakeDog}$ if needed. $\mathcal{B}$ starts as $\mathsf{P}_{\mathsf{i}_1}$, it produces $\theta_{i_1,j} = y_j \cdot \left(\mathsf{id}_{\delta(j)} \cdot x_j^{(\mathsf{dk}_{i_2}+\mathsf{dk}_{i_3}+\mathsf{dk}_{i_c})}\right)^{-1}$ for all $j \in [\![73,78]\!]$. For players P_i for $i = i_2, i_3$, it produces $\theta_{(i,j)} = x_j^{\mathsf{dk}_i}$ for $j \in [\![73,78]\!]$. Like in the previous step, it outputs alongside those values the zero-knowledge proof $\pi_{i,2} \leftarrow \mathsf{Sim}(\{x_j, \theta_{(i_1,j)}\}_{j \in [\![1,78]\!] \setminus [\![18 \cdot (i_1-1)+1, 18 \cdot i_1]\!]}, \mathsf{ek}_{i_1})$ for $i \in \{i_1, i_2, i_3\}$. Then verifies the zero-knowledge proof produced by the adversary, extract them and replay them and returns a random bit if it is not valid. If one of the players have been designated to take, it picks randomly a valid dog from cards in $H_i \cup H_5$, randomize the set of cipher associated from these cards as before and then simulate both proofs π_5 and Π_n or $\Pi_{n,2}$ depending on the case.

Game phase: $\mathcal{B}$ sets p equal to the first player index $p_* = 1$, $\gamma = 0$ and $\mathsf{played} = \perp$. For $j \in [\![1,78]\!]$:

If $p \neq i_c$: $\mathcal{B}$ runs $\mathsf{id} \leftarrow \mathsf{Strat}_p(\mathsf{played}, H_p, p_*, p)$, then it processes as in the algorithm $(\Pi, \mathrm{st}') \leftarrow \mathsf{Play}(p, \mathsf{id}, \mathsf{sk}_p, \mathsf{pk}_p, \mathrm{st}, \mathsf{PK}, D)$ except that:

- It chooses t at random instead than setting t to the coherent value.

– It computes proofs Π_0 and Π_1 as in the above defined simulator $\mathsf{Sim}^{\mathsf{TrickT}}_{W,S,\mathcal{A}(\mathsf{vw})}(\mathfrak{K}, \mathsf{setup}, s, \{\mathsf{pk}_i\}_{1\leq i\leq 4}, \mathsf{PK}, \mathsf{vw})$.

It sets $\Pi = (t, \Pi_0, \Pi_1)$ and sends (Π, st') to $\mathcal{D}_4$. Then it updates $\mathrm{st} := \mathrm{st}'$ and the index p that points the next player according to the rule of the French Tarot. It then parses played as $(\mathsf{pl}_1, \ldots, \mathsf{pl}_n)$ (where $n = |\mathsf{played}|$) and updates $\mathsf{played} := (\mathsf{pl}_1, \ldots, \mathsf{pl}_n, \mathsf{id})$.

If $p = i_c$: $\mathcal{B}$ processes as in $\mathsf{Exp}^{\mathsf{TrickT}}_{W,S,K,\mathcal{D}}(\mathfrak{K})$.

Final phase The simulated experiment returns 1.

Finally, $\mathcal{B}$ runs $b_* \leftarrow \mathcal{D}_5(\mathsf{vw})$, where vw denotes all the values send and received by $\mathcal{D}$ during its interaction with the simulated experiment, then $\mathcal{B}$ returns b_*.

Note that an adversary solving the 72-IND-CPA problem can solve the 78-IND-CPA problem.

Analysis: We distinguish two cases:

– The adversary forges a proof of a false statement during the DeckGen protocol. In this case, if $\mathcal{D}$ does not produce a valid proof for the false statement, then $\mathsf{PK} = \perp$ so the experiment aborts, hence the advantage of $\mathcal{B}$ is lower than $2 \cdot \epsilon_{\mathsf{sound}}(\mathfrak{K})$.

– The adversary does not forge a proof of a false statement during the DeckGen protocol. In this case, if $\mathsf{PK} = \perp$ then $\mathcal{D}_5$ has no information hence cannot do better than answering randomly. Now, assuming that $\mathsf{PK} \neq \perp$ and $b = 1$, for all $v \in [\![1, 78]\!]$, there exist $i \in [\![1, 4]\!]$ and $j \in [\![18 \cdot (i-1) + 1, 18 \cdot i]\!]$ such that $\mathsf{id}_v = \mathsf{Dec}_{\mathsf{dk}_i}(c^*_{4,j})$.

In the following, we show that in this case, the advantage of $\mathcal{B}$ is lower than $2 \cdot \epsilon_{78-\mathsf{IND-CPA}}(\mathfrak{K})$. If $b = 1$, then the experiment is perfectly simulated, else, the simulator is perfectly simulated. We observe that:

$$\begin{aligned}\mathsf{Pr}[\mathcal{B} \text{ wins}] &= \frac{1}{2}\left(\mathsf{Pr}[1 \leftarrow \mathcal{D}_5(\mathsf{vw}) | b = 1] + \mathsf{Pr}[0 \leftarrow \mathcal{D}_5(\mathsf{vw}) | b = 0]\right) \\ &= \frac{1}{2}\left(\mathsf{Pr}[1 \leftarrow \mathcal{D}_5(\mathsf{vw}) | b = 1] + 1 - \mathsf{Pr}[1 \leftarrow \mathcal{D}_5(\mathsf{vw}) | b = 0]\right).\end{aligned}$$

Finally:

$$\left|\mathsf{Pr}[\mathcal{B} \text{ wins}] - \frac{1}{2}\right| = \frac{1}{2} \cdot |\mathsf{P}_{\mathsf{real}}(\mathcal{D}, k) - \mathsf{P}_{\mathsf{sim}}(\mathcal{D}, k)| = \frac{\lambda(\mathfrak{K})}{2}.$$

The two cases imply that the advantage of $\mathcal{B}$ is lower than $2(\epsilon_{39-\mathsf{IND-CPA}}(\mathfrak{K}) + \epsilon_{\mathsf{sound}}(\mathfrak{K}))$, which concludes the proof. □

Lemma 11. *Instantiated by proofs of knowledge that are sound, extractable and zero-knowledge, our Tarot protocol is* Dog-private *under the DDH assumption.*

Proof. As previously the main intuition is based on the fact that $\mathcal{A}$ has to outputs valid proof for each of its actions.

Game 0: Let Game 0 be the is dog experiment $\mathsf{Exp}^{dog}_{W,\mathcal{A}}(\mathfrak{K}, i_{guess})$.

Game 1: $\mathcal{A}$ has to output a valid zero knowledge proof for P_{i_c}'s key. Under soundness of the NIZKP used for $\Pi_{\mathsf{pk}_{i_c}}$, we have that the extracted key sk_{i_c} verifies $g^{\mathsf{sk}_{i_c}} = \mathsf{pk}_{i_c}$. $\mathcal{C}$ check this equality and aborts the game if this does not hold. Thus, $\mathsf{Adv}^{indist}_{G_0,G_1}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

The three following games modify the behaviours of the challenger during the first phase of KeyGen.

Game 2: In this game we decrypt the elements $(c_{i_c,j})_{1\leq j\leq 78}$ using sk_{i_c} obtained previously. The challenger aborts if $\{\mathsf{Dec}_{\mathsf{sk}_{i_c}}(c_{i_c,j})\}_{1\leq j\leq 78} \neq D$. Under soudness of the NIZKP, we obtain $\mathsf{Adv}^{indist}_{G_1,G_2}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$.

Game 3,4: The same is done for $\mathsf{P}_{i^1_a}$, its key have been produced by the challenger, hence we do not rely on the soudness of any proof before this. Game 4: is the same for $\mathsf{P}_{i^1_a}$.

The three following games modify the behaviours of the challenger during the second phase of KeyGen.

Game 5,6,7: P_{i_c} secret key sk_{i_c} underlying P_{i_c}'s proof of phase 2 is extracted. $\mathcal{C}$ checks whether for all $j \in [\![1, 72]\!] \setminus [\![18 \cdot (i_c - 1) + 1, 18 \cdot i_c]\!]$, the equality $\theta_{(i_c,j)} = x_j^{\mathsf{sk}_{i_c}}$ holds, if not, $\mathcal{C}$ abort the game. We obtain

$\mathsf{Adv}^{indist}_{G_4,G_5}(\mathcal{A}) \leq \epsilon_{\mathsf{extract}}(\mathfrak{K})$. *Game 6* comprise the same modification for $\mathsf{P}_{i^1_a}$. *Game 7* is the same modification for $\mathsf{P}_{i^2_a}$.

We now modify the behaviours of the corrupted players during the MakeDog protocol.

Game 8,9,10: The first step of the MakeDog protocol requires the players to output $\theta_{(i,j)}$ values alongside a zero-knowledge proof. We proceed to the same modification as above for the cards in the dog for each of the three corrupted players.

These two additional games are only required if one of the corrupted player takes.

Game 11: If the taker is corrupted it has to randomise its cards with the dog. First we extract δ_5 and the $\{r_j\}_{73\leq j\leq 78}$ and replay the randomisation, if it does not match the outputted values the game is aborted.

Game 12: In addition to the previous modifications, we bypass proof Π_n or $\Pi_{n,2}$ (depending on the case) and directly executes GetHand and verify that for all $\mathsf{id} \in d \setminus \cup_{i=1}^{4} H_i$, $\mathsf{id} \notin \mathsf{O}$. If not we abort the games and return a random bit. □