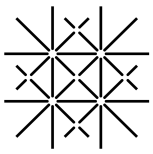


Universität Basel

Analysis of a Dropout-Tolerant Mental Poker Protocol and Implementation of a Modified Variant

Bachelor Thesis

University of Basel - Faculty of Science
Department of Mathematics and Computer Science
Computer Networks Group

Examiner: Prof. Dr. Christian Tschudin
Supervisor: Dr. Erick Lavoie

Stanislav Bondar
bondar.stanislav1@gmail.com

July 15, 2026

Abstract

Mental poker protocols allow mutually distrusting players to play card games over a network without relying on a trusted dealer. This thesis analyzes the dropout-tolerant protocol without a trusted third party proposed by Castellà-Roca and develops a peer-to-peer proof-of-concept implementation.

During the implementation, an incompatibility was identified in the mathematical setting of the protocol and the requirements of Stadler's zero-knowledge proof. To resolve this issue, a second cyclic subgroup is introduced. Since this modification is not compatible with the original vetoing mechanism, a simplified version of the protocol without vetoing is implemented.

The implementation includes distributed key generation, collaborative deck generation, ElGamal encryption, shuffling, re-masking, and private card drawing. Correct execution is verified using different zero-knowledge proofs: Stadler's proof, shuffling proof, and Chaum-Pedersen proof. The implementation is evaluated with respect to the influence of the selected key length, proof-security parameters, and number of players on the computational cost.

Acknowledgments

I would like to express my gratitude to Prof. Dr. Christian Tschudin and Dr. Erick Lavoie for giving me the opportunity to work on this thesis. In particular, I want to thank my thesis supervisor, Dr. Erick Lavoie, for his mentorship, support and motivation. The frequent discussions we had helped me a lot in shaping the thesis, defining its direction, and achieving satisfactory results.
Additionally, I would like to thank Dr. Osman Biçer for sharing his expertise in cryptography. With his help, I acquired a better understanding of the underlying protocol and resolved a mathematical inconsistency within it.
Finally, I want to thank my family and friends for supporting me and creating an encouraging working environment.
Regarding the tools used, ChatGPT was used throughout the writing process to assist with paraphrasing and grammar correction. It was, however, not used to generate any results or data.

Table of Contents

1 Introduction

Traditional online card games rely on a trusted third party, who is responsible for shuffling and distributing the deck of cards. Usually this role is performed by a centralized entity - a game server. While this approach is simple and practical, it requires all participating players to trust that the server is fully honest and does not manipulate the game in any way. Without this trust, however, the players should be worried about the state of the game, as the server is usually able to see the complete state of the game and may potentially influence the outcome by modifying the order of cards or revealing hidden information.

Mental poker protocols try to solve this problem by replacing the trusted dealer with various cryptographic techniques [1–3]. The term mental poker refers to a class of cryptographic protocols which allow players to play card games over a network in a mutually distrusting setting, which refers to a setting where participants do not trust each other and will attempt to cheat whenever the protocol allows it. A correct protocol must ensure the secrecy of cards, that no player can duplicate or modify cards, and that the deck is shuffled fairly without giving any player any advantage or revealing any information about the shuffled deck [4, 5]. Such protocols are difficult to design because of multiple security properties which must hold simultaneously. Players need to collaboratively generate, shuffle and deal a deck of cards while preventing all the participants from manipulating the game in a dishonest way. Additionally, protocols should remain verifiable, meaning that cheating attempts can and will be detected by all of the participants. To achieve these goals, mental poker protocols commonly rely on public-key cryptography, threshold cryptosystems, and zero-knowledge proofs.

However mental poker protocols are still not flawless. One important practical problem of mental poker protocols is player dropout. Many protocols only work under the assumption that all players participate until the end of the game without disconnecting. If a player disconnects or intentionally leaves, the game can become impossible to continue because some cryptographic keys required for decryption are tied to the player who left, therefore they are no longer available. This significantly limits the practicality of such systems in real-world networks.

This thesis focuses on the dropout-tolerant mental poker protocol proposed by Castellà-Roca [3]. The protocol combines threshold ElGamal encryption, collaborative shuffling, veto

mechanisms, and zero-knowledge proofs in order to allow the game to continue even when players leave mid-game.

Originally, the goal of this thesis was to implement and analyze the protocol proposed by Castellà-Roca in Python. However, during the implementation process, several issues and ambiguities were identified, mostly challenges related to the implementation of the Stadler's proof [6] and the use of two different cyclic groups. To the best of our knowledge, these issues cannot be resolved within the assumptions and description of the original protocol. We analyze those issues and then introduce, implement, and analyze a modified working version which retains most of the original design but foregoes the vetoing stage that proved problematic, and the chained deck generation that was no longer necessary without vetoing. The protocol is considered secure against collusion as long as at least one participating player remains honest and does not share their secret information with the colluding players. The players are considered mutually distrusting, and the protocol should remain secure on the cryptographic level if some participants try to cheat within the rules of the protocol. For example, if a player sends an incorrect shuffle, an invalid partial decryption, or incorrectly generated card values, this should be detected by the corresponding zero-knowledge proofs. Our implementation covers core aspects presented by the original protocol (except vetoing and chained deck generation) including creation and verification of all zero-knowledge proofs. We left out complementary mechanisms such as message authentication and proper error reporting to the user, that are orthogonal to the protocol description, and can easily be added with well-known techniques.

2

Cryptographic Background

2.1 Mathematical Background

To understand how the protocol works we first need to understand a few mathematical and cryptographic concepts, from which the protocol is built.

2.1.1 Cyclic Groups

A cyclic group is a group where all elements can be generated from one element, called the generator, by exponentiating it and applying modulo afterwards. Let G be such a group, let p be the modulus and let $g \in G$ be a generator. For every integer n, it holds that:

$$g^n \bmod p \in G$$

In our case, we work with finite groups, so after a finite number of exponentiations the values "wrap around", which means they eventually repeat. The number of different values in a group is called the order of the group. In practice, that means that the number n effectively behaves modulo order of the group.

The protocol needs two such subgroups to work. One part of it works inside $\mathbb{Z}_p^*$, which is the multiplicative group of integers modulo a prime p. It contains all non-zero integers modulo p:

$$\mathbb{Z}_p^* = \{1, 2, 3, \ldots, p-1\}$$

with multiplication performed modulo p. Since p is prime, every element in $\mathbb{Z}_p^*$ has a multiplicative inverse.

The protocol needs a smaller subgroup inside $\mathbb{Z}_p^*$, which is generated by choosing an element $g \in \mathbb{Z}_p^*$. Depending on the chosen value, the subgroup can have a different order. Here we need a value g_q which generates a subgroup of order q. The powers of this element generate the subgroup:

$$G_q = \{g_q^0, g_q^1, g_q^2, \ldots, g_q^{q-1}\}$$

This subgroup has exactly q elements, which means that the exponent of g_q is always reduced mod q. All the protocol encryption and decryption related operations are performed inside this subgroup, using modulo p. The group operation is multiplication modulo p, while exponentiation is used to repeatedly apply this operation.

2.1.2 Group Structure

The protocol uses cyclic groups derived from safe primes. A safe prime is a prime number p such that $\frac{(p-1)}{2}$ is also prime. First, a prime q is generated and used to construct

$$p = 2 \cdot q + 1,$$

where p is also prime. The protocol then needs an element of order q in $\mathbb{Z}_p^*$. This is obtained by choosing a random value $h \in \mathbb{Z}_p^*$ and computing $g_q = h_q^2 \bmod p$. The value h_q itself is only a temporary random value. Squaring it ensures that g_q lies in the subgroup of quadratic residues modulo p. Since this subgroup has order q, and q is prime, every non-identity element in it has order q. Therefore, if $g_q \neq 1$, it can be used as a generator of this subgroup. This group is used for encryption, decryption and zero-knowledge proofs.
For the protocol to work, a second safe-prime chain is required. It is constructed as $o = 2 \cdot p + 1$, with generator $g_p = h_p^2 \bmod o$. The deck values are computed modulo o, while the ciphertext operations are performed modulo p. This is not a part of the original protocol, however this is needed later for the Stadler's proof to work.
This type of prime chain is common enough for it to have its own name: a Cunningham chain of the first kind of length three.

2.1.3 Discrete Logarithm Problem and Diffie-Hellman Assumption

The powers in a cyclic group are easy to compute. This means that given a generator g and a secret exponent a, the value $y = g^a$ can be computed efficiently. However, the inverse problem is assumed to be hard. Given only g and y, it is computationally infeasible to recover the exponent a from y. This inverse problem is called the discrete logarithm problem. The Diffie-Hellman assumption uses this idea and states that, given g^a and g^b, it is hard to compute the shared value g^{ab} without knowing either a or b [7]. The protocol heavily relies on these assumptions, as it will be shown in the following subsections.

2.2 ElGamal

ElGamal is a public-key encryption scheme based on the difficulty of the discrete logarithm problem [8]. It uses public values such as the primes p and q, the generator g, and the public key y. These values are the parameters of a cyclic group, and the fact that they are public means anyone can encrypt a message.
The corresponding private value is the secret key a. This value must remain secret, because it is what makes decryption possible. The important idea is that the public key is computed using the private key, but computing the private key from the public key is infeasible, because this is equivalent to solving the discrete logarithm problem.

2.2.1 N-out-of-n multiplicative ElGamal

ElGamal encryption is typically presented as a two-party protocol but it can be generalized, so that decryption requires the participation of all participants. This version is called

n-out-of-n multiplicative ElGamal and splits the secret key between n participants [9, 10], so $a = a_1 + a_2 + \cdots + a_n$. This means that all n participants are needed to decrypt a ciphertext. This is considered safe, because of the Diffie-Hellman assumption. ElGamal is still used with a multiplicative group of integers modulo p, so every operation is calculated using this modulo.
We now describe the four key ElGamal-based operations in turn, we will use in the protocol later: key generation, encryption, decryption and re-masking.

2.2.2 Key generation

First, a cyclic group with a group generator g is chosen. Then each participant P_i independently and secretly chooses a random value $a_i \in \mathbb{Z}_q$, called secret key share, and computes the corresponding public key share:

$$y_i = g^{a_i}$$

The joint public key y is computed by multiplying all public key shares together:

$$y = \prod_{i=1}^{n} y_i$$

Substituting $y_i = g^{a_i}$ gives:

$$y = \prod_{i=1}^{n} g^{a_i} = g^{a_1 + a_2 + \cdots + a_n}$$

So the full secret key is never stored by one participant, but it is equal to the sum of the private key shares:

$$a = a_1 + a_2 + \cdots + a_n$$

The public key shares y_i can be shared without the fear, that the secret a_i can be recovered from y_i. Retrieving the private key share a_i from the public key share y_i is equivalent to solving the discrete logarithm problem. It is therefore computationally infeasible for an adversary, which may be one of the participants, to retrieve the full secret key only by observing the public key shares exchanged.

2.2.3 Encryption

To encrypt a message m, the sender chooses a random value r and uses the joint public key y to compute the encryption tuple $E_y(m)$:

$$E_y(m) = (c_1, c_2) = (g^r,\ m \cdot y^r)$$

This is the same as a normal ElGamal encryption, but the public key y now belongs to all participants together.

2.2.4 Decryption

To decrypt, every participant has to compute their own partial decryption using their secret key share a_i:

$$c_1^{a_i} = (g^r)^{a_i} = g^{r a_i}$$

All partial decryptions are then multiplied together:

$$\prod_{i=1}^{n} c_1^{a_i} = c_1^{a_1+a_2+\cdots+a_n} = c_1^a$$

Now the message can be recovered by dividing c_2 by the product of all partial decryptions:

$$m = \frac{c_2}{c_1^a} = \frac{m \cdot y^r}{c_1^a}$$

Now remember:

$$y = g^a$$

$$y^r = (g^a)^r = g^{ra}$$

$$c_1^a = (g^r)^a = g^{ra}$$

Therefore:

$$\frac{m \cdot y^r}{c_1^a} = \frac{m \cdot g^{ra}}{g^{ra}} = m$$

2.2.5 Secrecy

Because each share is chosen independently and randomly, no share can be derived from the others. Moreover, under the Diffie-Hellman assumption, with $g^a = g^r$, $g^b = g^{a_i}$, $g^{ab} = g^{ra_i}$, computing $c_1^{a_i}$ is hard for anyone other than P_i. Therefore, decryption requires the explicit collaboration of all participants.

2.2.6 Re-masking

Assume a ciphertext:

$$(c_1, c_2) = (g^r,\ m \cdot y^r)$$

If someone chooses a fresh random value r' and multiplies the ciphertext by a fresh encryption of 1, which is just a tuple $(g^{r'}, y^{r'})$, they get:

$$(c_1', c_2') = (c_1 \cdot g^{r'},\ c_2 \cdot y^{r'})$$

Substituting c_1 and c_2 gives:

$$(c_1', c_2') = (g^r \cdot g^{r'},\ m \cdot y^r \cdot y^{r'}) = (g^{r+r'},\ m \cdot y^{r+r'})$$

So the new ciphertext has exactly the same form and looks the same as a fresh ElGamal encryption of m, but now with randomness $r + r'$. Therefore, the encrypted message does not change, but the ciphertext looks different.

Re-masking allows a ciphertext to be refreshed without changing the encrypted message. This hides the connection between an old ciphertext and its updated version. This can be used, for example, to preserve the secret value used to represent a card while randomizing its public value.

2.2.7 Summary

N-out-of-n ElGamal encryption works as follows:

$$\text{Joint public key: } y = \prod_{i=1}^{n} y_i$$

$$\text{Encrypt: } (c_1, c_2) = (g^r,\ m \cdot y^r)$$

$$\text{Partial decryptions: } c_1^{a_i}$$

$$\text{Decrypt: } m = \frac{c_2}{\prod_{i=1}^{n} c_1^{a_i}}$$

3

The Two-Deck Mental Poker Protocol

This chapter provides a detailed explanation of the dropout-tolerant mental poker protocol. However, as mentioned in the introduction, this thesis does not focus on the original protocol proposed by Castellà-Roca [3], but on a modified version of it. The modifications were introduced to resolve inconsistencies in the original mathematical setting, but this will be discussed in chapter 5. In the remainder of this thesis, this protocol version will be referred to as the *two-deck protocol*.

3.1 Protocol Flow

This subchapter provides an overview of one dealing round of the protocol. In each round, the players first generate a new face-up deck D, then construct its encrypted counterpart C, thus obtaining a shuffled face-down deck from which cards can be drawn. These decks are only used for one round, so they get discarded after the round is complete. In the following explanations, the subscript j will be used to denote an individual card, while the subscript i will denote the player identifier.

3.1.1 Setup

The first step of the protocol is to choose the core parameters:

1. Primes: We need to choose primes o, p, q such that $o = 2 \cdot p + 1$ and $p = 2 \cdot q + 1$, as previously described in chapter 2.1.2. These primes will be used to create the Cyclic groups and do the modulo operations.

2. Generators: $g_q \in (\mathbb{Z}/p\mathbb{Z})^*$ of order q and $g_p \in (\mathbb{Z}/o\mathbb{Z})^*$ of order p. The first subgroup is needed for the encryption and decryption. Additionally, the zero-knowledge proofs also need this subgroup, which will be discussed in section 4. The second subgroup is needed for the generation of the face-up deck of cards and the encryption proofs.

3. Joint public key: $y = \prod_{i}^{n} g_q^{a_i} = g_q^{a_1+a_2+...+a_n}$ where the shares $a_1, a_2, ..., a_n$ are the private key shares of each player.

3.1.2 Generation of the Face-Up Deck D

For each playing card position $j \in \{1, \ldots, 52\}$, every player P_i chooses a new fresh secret exponent $m_{i,j} \in \mathbb{Z}_p$, where $2 < m_{i,j} < p$. Using these exponents player P_i computes an individual card:

$$d_{i,j} = g_p^{m_{i,j}}.$$

These values get arranged into an ordered tuple representing the cards:

$$D_i = (d_{i,1}, d_{i,2}, \ldots, d_{i,52})$$

Each player shares their tuple with the other participants. After receiving the contributions from all the players, each player computes the new face-up deck component-wise as:

$$d_j = \prod_{i=1}^{n} d_{i,j} = \prod_{i=1}^{n} g_p^{m_{i,j}} = g_p^{\sum_{i=1}^{n} m_{i,j}},$$

After doing this, every player gets the new shared face-up deck:

$$D = (d_1, d_2, \ldots, d_{52}).$$

3.1.3 Construction of the Encrypted Face-Down Deck C

Now each player encrypts the exponents used in the generation of D. More precisely, each player P_i encrypts their own secret exponents $m_{i,j}$ under the joint public key y using ElGamal encryption:

$$c_{i,j} = E_y(m_{i,j}^{-1}) = \left(g_q^{r_{i,j}},\ m_{i,j}^{-1} \cdot y^{r_{i,j}}\right),$$

where $m_{i,j}^{-1}$ is the modular inverse of exponent $m_{i,j}$ taken modulo p, and $r_{i,j} \in \mathbb{Z}_q$ (with $1 < r_{i,j} < q$) is fresh encryption randomness. The inverting of the exponent before the encryption is needed to produce encryption proofs, which will be discussed later. For each card position j, the ciphertexts of all players are grouped into one tuple:

$$c_j = (c_{1,j}, c_{2,j}, \ldots, c_{n,j}),$$

Now these tuples are combined into another tuple, which is the initial encrypted deck:

$$C_0 = (c_1, c_2, \ldots, c_{52}).$$

Thus, C_0 is the face-down representation of the cards. Each card is just a tuple of encrypted exponents which were used to generate the face-up deck D. Here the cards ordering of the deck is still preserved, i.e. it corresponds to the ordering of the deck D. In the next steps Shuffling and Re-masking will be applied to break the ordering of the deck, having the same effect as shuffling a real-life deck of cards.

Although we now have obtained a valid face-down deck of cards, this step is not completed yet, as the encrypting player also needs to prove to other players that the encryption was done properly, i.e. that they used the same exponents $m_{i,j}$ as in the face-up deck generation step and that the resulting values are valid ElGamal encryptions. The prover must do this without revealing the values such as $m_{i,j}$ or the re-masking values $r_{i,j}$, as they should be kept secret. Castellà-Roca's protocol suggests using zero-knowledge proofs to solve this problem. We will discuss how to construct such a proof in section 4.3.

3.1.4 Shuffling and Re-masking

Starting from C_0, each player P_ℓ performs two operations sequentially:

1. A secret permutation σ_i of the 52 card positions,
2. A fresh re-masking of every ElGamal ciphertext in the deck with $r'_{\ell,i,j}$ where $r'_{\ell,i,j} \in \mathbb{Z}_q$ (with $1 < r'_{\ell,i,j} < q$).

The re-masking step does not change the encrypted plaintexts but refreshes the randomness of the ciphertexts. Since all players apply these operations one after another, no player can track the permutations and re-masking choices of the others. After every player has shuffled and re-masked the deck, we get the final shuffled face-down deck, which is denoted by C_n, or simply C.
Shuffling and re-masking also involves using secret values, so it should be verified by other players, that it was done correctly, without giving away the secrets used. For this, another zero-knowledge proof is used, which will be discussed in section 4.2.

3.1.5 Card Drawing and Decryption

To draw a card, a player selects one ciphertext tuple:

$$c_j = (c_{1,j}, c_{2,j}, \ldots, c_{n,j}) \in C.$$

Each component $c_{i,j}$ is an ElGamal ciphertext of the form:

$$c_{i,j} = (u_{i,j}, v_{i,j}).$$

Each player P_ℓ uses their private key share a_ℓ to compute a partial decryption of the first component:

$$u_{i,j}^{a_\ell}.$$

The drawing player combines all partial decryptions as:

$$u'_{i,j} = \prod_{\ell=1}^{n} u_{i,j}^{a_\ell},$$

and then recovers the plaintext inverse exponent by doing the ElGamal decryption:

$$m_{i,j}^{-1} = v_{i,j} \cdot (u'_{i,j})^{-1} \bmod p.$$

Repeating this for all $i \in \{1, \ldots, n\}$ gives the tuple of decrypted exponents corresponding to the selected card.
Once again, each player needs to provide a zero-knowledge proof of the correct partial decryption, which will be discussed in section 4.1.

3.1.6 Card Identification

Let the recovered exponents for the selected ciphertext tuple be $m_{1,j}^{-1}, \ldots, m_{n,j}^{-1}$. First of all, we need to take the modular inverse of each exponent to get $m_{1,j}, \ldots, m_{n,j}$. Their sum:

$$m_j = \sum_{i=1}^{n} m_{i,j} \bmod p$$

identifies the exponent that links the selected ciphertext tuple to one card of the current face-up deck. This is the exponent which was used to produce face-up deck card d_j. To determine which logical card has been drawn, the player checks, for each position $t \in \{1, \ldots, 52\}$, whether:

$$g_p^{m_j} \stackrel{?}{=} d_t.$$

If this holds for some t, then the selected ciphertext tuple corresponds to card position t in the face-up deck D. This position encodes the value of the card, e.g. the first card on the deck D is the Ace of Spades, the second is the Two of Spades, etc. If no such t exists, some of the steps were done incorrectly, which means a player, intentionally or not, has disrupted the game. To prove that all the steps involving player's secrets were done correctly, each player will need to prove and verify encryption, shuffling, re-masking, and the decryption steps.

4

Relevant Zero-Knowledge Proofs

Zero-knowledge proofs are used in the poker protocol to prove that certain computations were done correctly, without revealing the secret values hidden in these computations. In general, a zero-knowledge proof is a probabilistic proof between a prover and a verifier. The prover wants to convince the verifier that some statement is true, for example that they know a secret exponent. The verifier challenges the prover with random values, and the prover has to answer these challenges correctly. If the prover does not know the secret, they could still guess a correct answer. However, by using random challenges, or by repeating the proof several times, the probability that a cheating prover successfully fakes the proofs becomes very small. A more detailed explanation of zero-knowledge proofs and their formal properties can be found in [11].

A zero-knowledge proof has three main properties: completeness, soundness, and zero-knowledge.
Completeness means that an honest prover can convince an honest verifier. If the proof statement is true, and the prover knows the required secret, then the verifier accepts the proof. In this thesis, this means for example that a player who computed a partial decryption correctly can create a valid Chaum-Pedersen proof, or that a player who correctly encrypted the exponent used for creation of a face-up deck card can create a valid Stadler's proof.
Soundness means that a cheating prover cannot convince the verifier of a fake statement, except with very small probability. For example, a player should not be able to prove that a ciphertext contains the correct exponent if this is not actually the case. Similarly, a player should not be able to prove that a partial decryption was computed correctly if they used a wrong secret key share. The security of these proofs is based on the hardness of the discrete logarithm problem. This means that, without knowing the required secret exponent, it is computationally hard to create fake answers that pass the random challenges presented by verifier.
Zero-knowledge means that the proof does not reveal the secret itself, or any new information about the secret. The verifier only learns that the statement is true, but does not learn anything else, that makes it true. For example, in the Chaum-Pedersen proof, the verifier learns that two discrete logarithms are equal, but nothing else. In the Stadler-style proof used in this thesis, the verifier learns that the encrypted exponent corresponds to the published

deck value, but nothing else.
These properties are crucial for the protocol. Completeness ensures that honest players are not rejected. Soundness prevents dishonest players from faking the knowledge of secret values and producing fake proofs. Zero-knowledge protects the private values of the players, such as secret key shares, random exponents, and card information from leaking.

4.1 Chaum-Pedersen Proof

A decrypting player uses the Chaum-Pedersen proof [12] to prove to other players, that given the encrypted card $c_{i,j} = (u_{i,j}, v_{i,j})$ their partial decryption $u'_{i,j} = u_{i,j}^{a_\ell}$ was done correctly, i.e. that they used their real secret key share a_ℓ, without revealing a_ℓ itself. For this, they can use another public value, which was computed using their private key share - their public key share: $y_\ell = g_q^{a_\ell}$. Now the decrypting player can prove, that the same a_ℓ was used to compute both $u'_{i,j}$ and y_ℓ, without revealing the value a_ℓ itself. The process goes as follows:

1. The prover picks a commitment, which is a random value w and computes:

   $$t_1 = g_q^w, t_2 = u_{i,j}^w.$$

   They send (t_1, t_2) to the verifier

2. The verifier sends a random challenge b

3. Prover computes: $s = w + b \cdot a_\ell \pmod q$ and sends the s back

4. Verifier checks:

   $$g_q^s \stackrel{?}{=} t_1 \cdot y_\ell^b$$
   $$u_{i,j}^s \stackrel{?}{=} t_2 \cdot u_{i,j}^{'b}$$

   which is:

   $$g_q^s \stackrel{?}{=} g_q^w \cdot g_q^{a_\ell b}$$
   $$u_{i,j}^s \stackrel{?}{=} u_{i,j}^w \cdot u_{i,j}^{a_\ell b}$$

   which is:

   $$g_q^s \stackrel{?}{=} g_q^{a_\ell b + w}$$
   $$u_{i,j}^s \stackrel{?}{=} u_{i,j}^{a_\ell b + w}$$

   So the prover must know the secret value a_ℓ in order to be able to compute $s = w + b \cdot a_\ell$ and satisfy both equations.

For this protocol, the proof can be made non-interactive using the Fiat-Shamir transformation [13]. Instead of receiving a random challenge from a verifier, the prover computes

$$b = H(g_q, u_{i,j}, y_\ell, u'_{i,j}, t_1, t_2) \bmod q,$$

where H is a cryptographic hash function. The proof is a tuple consisting of values (t_1, t_2, s). Each verifier independently recomputes b from the same public values and checks the two verification equations above. Although the challenge is deterministic, it has the same relevant properties as a randomly chosen challenge.

4.2 Mask-Shuffling Proof

Barnett and Smart present a zero-knowledge proof [2] which was used in Castellà-Roca's poker protocol. This proof is needed to show that a player has shuffled and re-masked a deck correctly. More precisely, it proves that no cards were added, removed or replaced during the shuffle. Let C_{i-1} be the face-down deck before player P_i shuffles it, and let C_i be the shuffled and re-masked deck produced by P_i. The goal of the proof is to show that C_i is only a permutation and re-masking of C_{i-1}, without revealing the secret permutation itself. The proof also needs a security parameter s, which is agreed upon during the protocol setup phase. Following steps are performed:

1. The prover chooses a secret permutation σ_i together with a set of re-masking values R_i and computes
$$C_i = \mathcal{E}_y(\sigma_i(C_{i-1}); R_i),$$
where $\mathcal{E}_y$ denotes the re-masking of ciphertexts under the joint public key y.

2. The prover publishes the shuffled and re-masked deck C_i.

3. The prover generates s auxiliary shuffled decks
$$(T_1, \ldots, T_s),$$
where each deck is computed as
$$T_t = \mathcal{E}_y\big(\sigma'_{i,t}(C_i); R_{i,t}\big).$$

4. The verifier chooses s random challenge bits
$$(b_1, \ldots, b_s), \qquad b_t \in \{0, 1\},$$
and sends them to the prover.

5. For every auxiliary deck T_t for which $b_t = 0$, the prover reveals the permutation $\sigma'_{i,t}$ together with the re-masking values $R_{i,t}$. This allows the verifier to check that
$$T_t = \mathcal{E}_y\big(\sigma'_{i,t}(C_i); R_{i,t}\big).$$

6. For every auxiliary deck T_t for which $b_t = 1$, the prover reveals the composed permutation

$$\sigma_i \circ \sigma'_{i,t}$$

together with re-masking values $R'_{i,t}$ which transform the original deck C_{i-1} directly into T_t:

$$T_t = \mathcal{E}_y\big((\sigma_i \circ \sigma'_{i,t})(C_{i-1}); R'_{i,t}\big).$$

The values $R'_{i,t}$ can be computed by combining the original re-masking values R_i with the auxiliary re-masking values $R_{i,t}$.

7. The verifier checks every revealed opening. If all checks succeed, the verifier accepts the shuffle as valid.

The proof works because the prover must generate all auxiliary decks before knowing how they will be opened. If C_i is a valid permutation and re-masking of C_{i-1}, each auxiliary deck can be opened in either of the two required ways. However, if the shuffle is invalid, a cheating prover can generally prepare an auxiliary deck for only one of the two possible challenge values. The probability that an invalid shuffle passes all s independent checks is therefore equal to 2^{-s}.
The shuffle proof can also be made non-interactive using the Fiat-Shamir transformation. The prover first generates all auxiliary decks $T_1, \ldots, T_s$. The public transcript is then hashed to obtain the challenge bits:

$$(b_1, \ldots, b_s) = H(C_{i-1}, C_i, T_1, \ldots, T_s), \qquad b_t \in \{0, 1\}.$$

Each challenge bit determines how the corresponding auxiliary deck must be opened. If $b_t = 0$, the prover opens T_t relative to C_i. If $b_t = 1$, the prover opens T_t directly relative to C_{i-1}.
The non-interactive proof consists of the auxiliary decks together with the corresponding openings. Each verifier recomputes the challenge bits from the public transcript and checks all openings. Since the auxiliary decks are included in the hash input, they must be fixed before the challenge bits are determined.
This is the most computationally expensive proof used in the protocol because it requires the generation and verification of s auxiliary decks. A larger value of s decreases the probability that an invalid shuffle is accepted, but also increases the computational costs of the proof.

4.3 Stadler's Proof

Stadler's proof [6] is used to prove a correct encryption. The goal is to prove that a ciphertext contains the discrete logarithm of a public value. More precisely, it is used to show that the encrypted secret $m_{i,j}$ encrypted as inverse modulo p in $c_{i,j} = (c_{1,i,j}, c_{2,i,j}) = (g_q^{r_{i,j}}, m_{i,j}^{-1} \cdot y^{r_{i,j}})$ is the same value as the value used in the exponentiation of $d_{i,j} = g_p^{m_{i,j}}$. This is done without revealing any secret information, such as the exponent $m_{i,j}$ and the encryption randomness

$r_{i,j}$. Same as the shuffling proof, this proof also needs a security parameter, which we will call l

The proof relies on the following relation:

$$\begin{aligned} d_{i,j}^{c_{2,i,j}} &= \left(g_p^{m_{i,j}}\right)^{c_{2,i,j}} \\ &= g_p^{m_{i,j}c_{2,i,j}} \\ &= g_p^{y^{r_{i,j}}}. \end{aligned}$$

The final equality holds because

$$c_{2,i,j} \equiv m_{i,j}^{-1} y^{r_{i,j}} \pmod{p},$$

$$m_{i,j}c_{2,i,j} \equiv y^{r_{i,j}} \pmod{p}.$$

Since g_p belongs to a group of order p, its exponents are considered modulo p.
The prover must therefore show that the same encryption randomness $r_{i,j}$ satisfies both

$$c_{1,i,j} = g_q^{r_{i,j}}$$

$$d_{i,j}^{c_{2,i,j}} = g_p^{y^{r_{i,j}}}.$$

In other words, the discrete logarithm of $c_{1,i,j}$ to the base g_q must be the same value that appears in the double exponentiation relation.

1. The prover chooses a random value w and computes:

$$\begin{aligned} t_1 &= g_q^w, \\ t_2 &= g_p^{y^w}. \end{aligned}$$

2. The verifier sends a random challenge bit:

$$b \in \{0, 1\}.$$

3. The prover computes the response:

$$s = w - b \cdot r_{i,j} \pmod{q}.$$

4. The verifier checks:

$$t_1 \stackrel{?}{=} g_q^s c_{1,i,j}^b$$

$$t_2 \stackrel{?}{=} \begin{cases} g_p^{y^s}, & \text{if } b = 0, \\ d_{i,j}^{c_{2,i,j} \cdot y^s}, & \text{if } b = 1. \end{cases}$$

Because the prover does not know the challenge in advance, both of the cases only hold if the provided ciphertext encrypts the discrete logarithm of $d_{i,j}$ to the base g_p.
The proof can also be made non-interactive using the Fiat-Shamir heuristic. Since one execution of Stadler's proof uses only a one-bit challenge, it has a soundness error of $1/2$.

Therefore, to decrease the likelihood of a dishonest player trying to guess the correct proof, the proof is repeated l times. For every repetition k, the prover chooses an independent random value w_k and computes

$$t_{1,k} = g_q^{w_k}, \qquad t_{2,k} = g_p^{y^{w_k}}.$$

For the non-interactive version, the prover computes the challenge vector

$$(b_1, \ldots, b_l) = H(d_{i,j}, c_{1,i,j}, c_{2,i,j}, t_{1,1}, t_{2,1}, \ldots, t_{1,l}, t_{2,l}),$$

with length l, where every $b_k \in \{0, 1\}$. For each repetition, the prover computes

$$s_k = w_k - b_k r_{i,j} \pmod{q}.$$

Each verifier recomputes the commitments and the challenge bits, and then checks the verification equation for every repetition. Since the prover cannot predict the challenge bits before fixing the commitments, a false proof is accepted with probability at most 2^{-l}.

5
Problems and Adaptations

5.1 Protocol Modifications

As mentioned in the Introduction, this protocol is a modified version of Castellà-Roca's protocol. The biggest differences between the two versions are the mathematical setting and the vetoing mechanism.

While trying to implement the protocol, we discovered an algebraic mismatch between its setting and the setting of Stadler's proof. The first difference is that Stadler's proof works with encryptions of m^{-1}, which is the modular inverse of $m \pmod{p}$, while Castellà-Roca's protocol encrypts m directly. However, this is not a big problem, as encrypting the inverse does not require substantial changes to other parts of the protocol.

The more significant problem concerns the groups in which the protocol operations are performed. Castellà-Roca's poker protocol only defines primes p and q, and later uses only one group $\mathbb{Z}_p^*$ and its subgroup of order q. All of the protocol operations are carried out in these groups, including the encryption, decryption and creation of the face-up deck D. So effectively instead of two generators g_q of order q and g_p of order p, only one generator g_q of order q is used.

Stadler's construction requires a different setting. The proof performs double exponentiations such as:

$$t_2 = g_p^{y^w}$$

inside this separate group G_p of order p. Stadler uses the fact that the inverse of m is encrypted. Let:

$$E(m^{-1}) = (A, B) = \left(g_q^{\alpha}, m^{-1} y^{\alpha}\right) \pmod{p}$$

and:

$$V = g_p^m.$$

Since:

$$B \equiv m^{-1} y^{\alpha} \pmod{p},$$

it follows that:

$$mB \equiv y^{\alpha} \pmod{p}.$$

Because g_p has order p, its exponents are interpreted modulo p. Therefore:

$$V^B = g_p^{mB} = g_p^{m \cdot m^{-1} y^{\alpha}} = g_p^{y^{\alpha}}.$$

Stadler explicitly defines this setting with $g_q \in \mathbb{Z}_p^*$ of order q, and with a separate group G_p of order p, used to define double exponentiation as $x \mapsto g_p^{g_q^x}$.
If no separate group of order p is used and the face-up deck values are constructed in the subgroup of order q, this argument breaks. In that setting, a face-up deck value has the form:

$$d_j = g_q^m.$$

Therefore:

$$d_j^B = (g_q^m)^B = g_q^{mB}.$$

For this value to equal $g_q^{y^{\alpha}}$, the exponents would have to satisfy:

$$mB \equiv y^{\alpha} \pmod{q},$$

because g_q has order q. However, the ciphertext relation only guarantees:

$$B \equiv m^{-1} y^{\alpha} \pmod{p},$$

and consequently:

$$mB \equiv y^{\alpha} \pmod{p}.$$

Congruence modulo p does not imply congruence modulo q. Therefore, the equality:

$$g_q^{mB} = g_q^{y^{\alpha}}$$

is not guaranteed. The proof consequently does not work when the outer exponentiation is performed in the group generated by g_q. The ciphertext relation establishes:

$$mB \equiv y^{\alpha} \pmod{p},$$

while the equality of powers of g_q, which has order q, depends on the exponents modulo q. Since the required congruence modulo q does not follow from the ciphertext relation modulo p, Stadler's equation cannot be derived. As a result, a separate group of order p is needed for the outer exponentiation.
This gives a mismatch between the algebraic settings of Stadler's proof and Castellà-Roca's poker protocol. To the best of our knowledge, there is no easy way to use Stadler's proof without introducing two separate subgroups of orders p and q, which also forces the introduction of another prime $o = 2 \cdot p + 1$.

But introducing a subgroup of order p and using it to compute the face-up deck values introduces a new problem. The vetoing mechanism described in Castellà-Roca's paper stops working, but to understand this, we will first explain how the vetoing works.

5.2 Vetoing Mechanism

To understand vetoing, we first need to understand the setting in which it is used. As mentioned above, the protocol of Castellà-Roca only uses one subgroup of order q for most of the operations. This means that only one generator value g is used, which is equivalent to the value g_q from the two-deck protocol.

The vetoing mechanism was introduced as a solution to the dropout problem and to keep track of the cards which were discarded. Each player vetoes the cards that they have previously drawn. This means that the player re-masks such a card in a special way, which makes the corresponding face-down card undecryptable. At the same time, the player can prove in zero-knowledge that the veto was applied to a card that they have indeed drawn before, without revealing the identity of the card.

Another important difference between the two protocols is the chained deck generation. The protocol of Castellà-Roca reuses the old deck D_k to generate a new deck D_{k+1}. Instead of starting every round from the generator value, the cards from the previous round are exponentiated with new secret exponents directly. More precisely, let $d_{k,j}$ denote the public value representing card j in round k, then the value of the same card in the next round is computed as:

$$d_{k+1,j} = d_{k,j}^{m_{k+1,j}},$$

where $m_{k+1,j}$ is the sum of the individual player exponents:

$$m_{k+1,j} = \sum_i m_{k+1,i,j} \pmod{q}.$$

Thus, for every round there is a direct discrete-logarithm relation between the old card value and the new card value.

When a player draws a card, the encrypted exponents corresponding to that card are decrypted for that player. Therefore, the drawing player learns the exponent which links the previous face-up representation of the card to the current one. If player P_i draws card j in round l, then P_i learns:

$$m_{l,j} = \log_{d_{l-1,j}}(d_{l,j}).$$

The knowledge of the decrypted exponent allows the drawing player to later veto the same card. Vetoing works by multiplying the encryptions of the secrets $m_{i,j}$, i.e. the face-down cards, with special vetoing tuples. Each player computes one tuple for each card, and later those are combined, analogously to the combination of the individual face-up decks D_i. For cards that are not vetoed, the player computes an ordinary ElGamal re-masking factor. This has the form:

$$(u_j, v_j) = (g^{r_j}, y^{r_j}),$$

for a randomly chosen value r_j. Such a pair is a valid re-masking factor because both components use the same exponent r_j. This can be proven with a Chaum-Pedersen proof of equality of discrete logarithms:

$$\log_g(u_j) = \log_y(v_j).$$

For a vetoed card, however, the player does not compute a valid ElGamal re-masking factor. Instead, the player computes:

$$u_j = d_{k,j}^{m_{l,j}^{-1}}$$

where $m_{l,j}^{-1}$ is the inverse of $m_{l,j}$ modulo the order of the group, and chooses v_j as a random value. As a result, except with negligible probability, the pair (u_j, v_j) does not satisfy the equality:

$$\log_g(u_j) = \log_y(v_j).$$

When this veto pair is multiplied into the face-down card, the corresponding ciphertext is no longer a valid encryption of the secret card exponent. This breaks the encryption, which means if a player later draws this card, the decryption yields an arbitrary value instead of a valid card value. This makes the card effectively unavailable, or vetoed.
In a later round k, the same card has a new public representation $d_{k,j}$. The vetoing player, who has previously drawn this card in the round l, applies their veto, which ensures that:

$$u_j^{m_{l,j}} = d_{k,j}.$$

Therefore, the player can later prove the equality:

$$\log_{d_{l-1,j}}(d_{l,j}) = \log_{u_j}(d_{k,j}).$$

This proves that the player knows the same exponent $m_{l,j}$ that links $d_{l-1,j}$ to $d_{l,j}$, and that this exponent also links u_j to the current card value $d_{k,j}$. In other words, the proof shows that the player is vetoing a card for which they know the drawing exponent from a previous round.
The player also hides which cards they vetoed, because revealing this would reveal which cards the player had previously drawn. For this reason, the player does not separately announce which factors are normal re-masking factors and which factors are vetoes. The protocol uses zero-knowledge proofs to show only that the correct number of cards was vetoed and that the remaining cards were re-masked. This allows the other players to verify that the player behaves correctly, while the identity of the vetoed cards is not revealed.
The vetoing mechanism is what allows the game to continue after a player leaves. If all players remain in the game, then they are present to always veto their previously drawn cards in the following rounds, making the cards undecryptable. If a player drops out, that player is no longer present to veto their previously drawn cards. As a result, the cards held

by the leaving player become available again, while the cards held by the remaining players are still vetoed. This allows the game to continue without requiring a trusted third party and without requiring the leaving player to reveal their secret information.

5.3 Problem of Two Groups and Vetoing

As discussed above, the protocol needs two different subgroups for its mathematical setting to match that of Stadler's. The face-up deck values are calculated in a subgroup generated by g_p, while the ElGamal encryption and re-masking operations are performed in a different subgroup generated by g_q. This separation makes the protocol consistent with Stadler's proof, but it also breaks the vetoing mechanism.

The reason is that the veto factor u_j has two different roles in the original protocol. First, it must be connected to the public face-up deck value $d_{k,j}$. For a card drawn in round l, the player knows:

$$m_{l,j} = \log_{d_{l-1,j}}(d_{l,j}).$$

To veto the same card in a later round k, the player computes:

$$u_j = d_{k,j}^{m_{l,j}^{-1}},$$

which satisfies:

$$u_j^{m_{l,j}} = d_{k,j}.$$

Therefore, u_j must live in the same group as the face-up deck values $d_{k,j}$, because the zero-knowledge proof compares the discrete-logarithm relations $d_{l,j} = d_{l-1,j}^{m_{l,j}}$ and $d_{k,j} = u_j^{m_{l,j}}$. Second, the same value u_j must also be used as part of an ElGamal re-masking factor. It is multiplied into the face-down card together with another component v_j. For a normal, non-vetoing re-masking factor, this pair has the form:

$$(u_j, v_j) = (g_q^{r_j}, y^{r_j}).$$

Thus, from the point of view of ElGamal encryption, u_j must be an element of the subgroup generated by g_q.

This is the point where the two-subgroup setting breaks the construction. The veto proof requires u_j to be an element of the face-up deck group, because it must satisfy $u_j^{m_{l,j}} = d_{k,j}$. However, the ElGamal re-masking operation requires u_j to be an element of the encryption group, because it must be multiplied into a ciphertext of the form $(g_q^r, m_{i,j}^{-1} \cdot y^r)$. With two different subgroups, these are no longer the same requirement. The value u_j cannot simultaneously be a face-up deck element in the subgroup generated by g_p and an ElGamal re-masking component in the subgroup generated by g_q.

This also creates a problem with the exponent inverse. In the face-up deck group, the exponent $m_{l,j}$ is interpreted modulo the order of that group. In the ElGamal group, exponents are interpreted modulo the order of the encryption group. If these groups have different orders, then the inverse $m_{l,j}^{-1}$ is not the same in both groups. An inverse computed modulo the

order of the ElGamal group does not generally satisfy the relation needed in the face-up deck group, and an inverse computed modulo the order of the face-up deck group is not the correct exponent for an ElGamal re-masking factor.
Therefore, the central assumption of the vetoing mechanism no longer holds in a well-defined way:

$$u_j = d_{k,j}^{m_{l,j}^{-1}}.$$

If u_j is computed in the face-up deck group, then it cannot be used directly as an ElGamal re-masking factor. If it is instead computed in the ElGamal group, then it no longer satisfies the relation:

$$u_j^{m_{l,j}} = d_{k,j}$$

with the face-up deck value. In both cases, one of the two necessary parts of the vetoing mechanism fails.
For this reason, we concluded, that the original vetoing mechanism cannot be directly transferred to the modified two-subgroup protocol. The two-deck protocol removes vetoing and treats the generation of a fresh deck independently from the old deck. This avoids the algebraic mismatch, but it also means that the properties of the original protocol, like dropout tolerance do not apply in the same way anymore.

5.4 Properties

As expected, by removing vetoing we change some of the properties of Castellà-Roca's poker protocol. The main advantage of vetoing is the dropout tolerance: when a player drops out mid-round (intentionally or unintentionally), the public key and the decks get recomputed, and the cards of the player who has left return to the decks automatically, because the player is not participating in the game anymore to veto their cards when the deck gets recomputed. The other players, however, veto the cards they have previously drawn, which means they get to keep the drawn cards, while ensuring that previously drawn and discarded cards remain unavailable.

For some poker variants (e.g. the classical poker) this mechanism is crucial, as the players are allowed to discard some of their cards one time per round, and those cards need to become undrawable, which is guaranteed by vetoing. But in other popular versions of poker, like Texas hold'em, there is only one set of cards drawn per round, and there is no possibility to discard the cards during the round, which means there is no need to keep track of the previously drawn cards. Texas hold'em is thus somewhat inherently dropout-tolerant due to its game flow. The only case where the vetoing would be advantageous is when all of the players have already drawn their cards and one of them leaves before the cards in the middle get decrypted. In this case, the players cannot proceed normally, so they need to recompute the whole deck, thus losing their drawn hand. But as long as no cards, or all cards are decrypted, the vetoing does not contribute any value.

This thesis makes three main contributions: it identifies an incompatibility between the original protocol setting and Stadler's proof; it derives a consistent two-group variant; and it implements and evaluates the resulting modified protocol.

6 Security Considerations

6.1 Choosing Primes

The main security parameter is the key length, which determines the length of the prime o, and consequently the lengths of p and q. The key length is defined in bits. This parameter directly influences performance and security of the protocol. Various attack types become possible if the chosen primes are too small. On the other hand, choosing larger primes means that almost every protocol operation is now done using larger numbers, which worsens the performance greatly.

Another important security consideration is the way the primes are chosen. Allowing a single player to generate them could be dangerous. The other players can easily verify whether o, p, and q are prime, whether $o = 2 \cdot p + 1$ and $p = 2 \cdot q + 1$, and whether the generators have the required subgroup orders, however, these checks do not guarantee that the parameters were not generated in a malicious way. A malicious player could intentionally choose weak parameters which make the discrete logarithm problem easier to solve.

The possible solution for this would be hardcoding the secure primes into the application. This, however, is also not a perfect solution, because if the attacker knows the primes in advance, they could perform an expensive precomputation using the Number Field Sieve [14]. After this precomputation, individual discrete logarithms for the same modulus can be computed considerably faster, which breaks the encryption. Adrian et al. [15] demonstrated such an attack for commonly used 512-bit groups, where the precomputation took approximately one week. They also argued that 768-bit groups are within the reach of academic-level adversaries, while attacks against 1024-bit groups possibly require nation-state-level computational power. This means that precomputed primes is a viable option for our implementation, as long as the chosen key length is large enough.

Alternatively, the public parameters could be generated jointly using a commit-reveal randomness procedure. First, every player P_i chooses a random value ρ_i and publishes a commitment:

$$c_i = H(\rho_i)$$

After all commitments have been published, the players reveal their values ρ_i, and the

commitments are verified. The shared seed is then computed as

$$H(\rho_1|\cdots|\rho_n).$$

Starting from this seed, all players deterministically search for primes of the required form. Since the search is deterministic, all players obtain the same parameters and can verify them independently. As long as at least one honest player contributes fresh randomness, no single player can fully control the resulting parameters.

6.2 Security Parameter for Shuffling and Stadler's Proofs

Another important parameter which affects security and performance is s, which is the number of auxiliary decks generated for the shuffling proof. This parameter is needed to prevent dishonest players from causing an invalid shuffle to be accepted.
Older cryptographic systems often considered 80-bit security sufficient, while modern systems usually aim for 128-bit security. For this proof, these values would correspond to cheating probabilities of:

$$2^{-80} \approx 8.27 \cdot 10^{-25}$$

$$2^{-128} \approx 2.94 \cdot 10^{-39}.$$

Such values are mainly needed in very serious cryptographic systems which protect very valuable information for long periods of time, where state-level adversaries are considered. The requirements of this poker protocol are humbler. A dishonest player gets only one attempt per round to submit a valid shuffling proof. If the proof fails, the invalid shuffle is detected and the game is stopped. So if the attacker would just randomly guess the proof instead of computing it, much smaller values of s may already be sufficient.
For example, choosing:

$$s = 20$$

gives a cheating probability of:

$$2^{-20} \approx 9.54 \cdot 10^{-7},$$

which is approximately one successful attempt in one million. Choosing:

$$s = 30$$

reduces this probability to:

$$2^{-30} \approx 9.31 \cdot 10^{-10},$$

which is approximately one successful attempt in one billion.

This, however, only holds under the assumption, that the attacker only tries to guess the proof once. There exists another, more dangerous kind of attack. Because the zero-knowledge proofs are implemented non-interactively, the prover must generate the challenge bitstream themself by hashing the public values together with the proof commitments. This approach has one weakness: the prover can generate multiple candidate commitments locally and then publish the values whose resulting challenge bits are favorable.

For the shuffling proof, the auxiliary decks $T_1, \ldots, T_s$ act as the proof commitments. A dishonest player who produced an invalid shuffled deck may still be able to open every auxiliary deck for one of the two possible challenge branches. The faking prover can therefore repeatedly generate new auxiliary decks and compute

$$H(C_{\text{old}}, C_{\text{new}}, T_1, \ldots, T_s)$$

until the resulting challenge bitstream matches the branches that they are able to open. The attacker does not need to break the hash function or find a full SHA-256 collision. They only need to find a hash where the first s challenge bits have the required values. For one candidate transcript, this succeeds with probability 2^{-s}, so approximately 2^s attempts are needed on average.
In our implementation, this attack is limited by the protocol timeout. After the shuffling phase begins, the other participants expect the shuffled deck and its proof within a fixed amount of time. If the dishonest player spends too long (e.g. because they are trying to fake the shuffling proof), the remaining players close the connection and remove them from the lobby. This prevents the attacker from searching for long periods of time during one protocol execution.
Since every additional repetition of the shuffling proof requires another complete auxiliary deck to be generated, sent over the network and verified, large values of s have a significant effect on performance. For the intended use of this protocol, values around 20 provide a more reasonable balance between security and usability than the standard 80 or 128.

Similar reasoning as to the parameter s can be applied to the parameter l used in the Stadler's proof. In this proof however, increasing the number l has a smaller effect on the performance, because Stadler's proof is computationally cheaper than the shuffling proof, so more secure higher values can be used.

7 Implementation

In this chapter, some relevant details regarding the implementation of the two-deck protocol will be presented. The source code is publicly available on GitHub at https://github.com/S-L-Bondar/Two-Deck-Mental-Poker-Protocol.

7.1 Overview

The implementation was done using Python. This choice was made because Python is simple to work with, especially for networking and socket programming. It also provides useful libraries for working with large integers, cryptographic randomness and hashing, which are all needed by the protocol. The `gmpy2` library was used for modular arithmetic and efficient exponentiation with large numbers. Random values were generated by `secrets` module, and the `hashlib` library was used for SHA-256 function, to generate the challenges for the non-interactive zero-knowledge proofs.

The application has a peer-to-peer structure, so every player runs their own instance of the program, which means that the card generation, shuffling and dealing is not done by a trusted server, but jointly by the players themselves. One player creates the lobby and manages the connection between the participants, but this player does not receive any additional cryptographic information or privileges.

The implementation can be divided into two main parts. The networking part is responsible for peer discovery, lobby management and the exchange of messages between the players. The lobby is only a lightweight class used to store and manage data about the participating players. It does not perform any cryptographic operations and does not give the player who created it any privileges. The cryptographic part implements the generation of the parameters and key shares, deck generation, encryption, shuffling, re-masking and card decryption. It also contains the implementations of Stadler's proof, the shuffling proof and the Chaum-Pedersen proof.

Each player stores their private information locally. This includes their private key share, random exponents, encryption randomness, permutations and re-masking values. Public values, such as public key shares, encrypted decks and zero-knowledge proofs, are sent to the other players. Before a received result is used in the next step of the protocol, the

corresponding proof is verified.
The implementation is mainly intended as a prototype. Its purpose is to show that the modified protocol can be implemented and executed between several independent players, and to evaluate the two-deck protocol's computational costs.

7.2 Parameters

Now we will present some parameter choices which are hardcoded into the implementation, which means clients cannot change them without modifying the code. These choices were made based on the speed and security considerations, as explained in the chapters 6.2 and 8.
The key length was set to `1024` bits, which is a reasonable trade-off between the speed and security for our goals, as it makes the precomputation attacks very expensive while not slowing down the protocol too much.
For our implementation, we decided to use precomputed values for o, p, and q, thus skipping the safe-prime-chain generation during the execution of the protocol. This was done for performance reasons, since generating a chain of large primes is expensive and can take several hours on a common computer, which is really suboptimal for a poker protocol. The parameters are hardcoded into the implementation, while the generators are chosen by the lobby leader. Before using them, every client verifies the subgroup orders of the generators.
The primes o, p and q were generated locally and are provided in the appendix A. This is a valid Cunningham chain of the first kind of length three, with the required key length.
As for the zero-knowledge proofs security parameters s and l, they were set to `24` and `40` respectively. After trying out different values and considering the resulting performance and security, we concluded that this is a good trade-off between those two criteria.

7.3 Game Flow

The implementation uses the rules of Texas Hold'em as its basis. Various details, such as betting and changing the order of the players, were not considered. As mentioned above, the purpose of the implementation is to demonstrate a working cryptographic protocol rather than to implement a fully playable poker game.
One simulated round follows the complete protocol flow described in section 3.1. First, each player draws two cards from the encrypted deck and requests partial decryptions from all other participants. The drawing order is the order of the player identifiers sorted from lowest to highest. The implementation assumes that the identifiers are unique. The drawing player then applies their own partial decryption locally without publishing it. Therefore, the own drawn cards are initially known only to that player.
After every player has received their private cards, the community cards are drawn. These consist of the flop, which contains three cards, followed by the turn and the river, which contain one card each. Since the community cards must be visible to all players, every participant publishes their partial decryption, allowing the cards to be fully decrypted.
In a normal game of Texas Hold'em, betting rounds take place before and between the flop, turn and river. Since betting was omitted from the implementation, all of the cards are

drawn and decrypted directly without any breaks between the different stages.
Finally, the players reveal their private cards. For this, each player publishes the partial decryptions that were previously kept private. The remaining players can then complete the decryption and verify the revealed hands. This stage represents the showdown. At this point, all cards used during the round are known, and a new round can begin.

7.4 Networking

The networking part of the implementation uses both UDP and TCP, which are used in the different communication stages. UDP broadcasts are used for lobby discovery. A player who creates a lobby regularly broadcasts basic information about it, such as its name, identifier, number of players and address. Other players can listen for these broadcasts and use the received information to join the lobby.
After joining, the players communicate using TCP connections. TCP was chosen for the exchange of lobby information and cryptographic protocol messages because it is very reliable and provides an ordered byte stream, which is an important property for a cryptographic protocol. The messages are represented as JSON objects and separated by newline characters, which allows multiple messages received through the same connection to be processed individually.
Each player establishes direct connections to the other participants in the lobby. A full-mesh topology may seem expensive, but it is fine for the poker protocol, as the number of players in a normal poker round usually does not exceed 9. A short initial message is also exchanged when a connection is created, allowing the receiver to associate it with the correct player. Once again, the protocol assumes that the players choose unique identifiers.
Heartbeat messages are sent regularly between connected players. If no valid message is received from a player within a fixed timeout, the connection is considered lost and the player is removed from the lobby. The remaining players are then informed about the dropout so that a new key can be computed and the game can continue. Heartbeats are handled by the networking layer and are not forwarded to the cryptographic protocol.

8 Evaluation

8.1 Speed measurements

In this chapter we will demonstrate the time it takes for one round of poker protocol to complete with different security parameters and number of players. All performance measurements were performed on the same computer. The system used an Intel(R) Core™ i7-8700 processor, 16 GB of RAM, and ran Windows 10. The implementation was executed using Python 3.12, together with gmpy2 version 2.3.0 and PyCryptodome version 3.23.0. During the measurements, no other computationally intensive applications were running. The tests were done on one machine, so the network latency is not considered. As a baseline we have chosen the values used in the implementation, with 6 clients playing, which is a usual number of players in poker.

8.1.1 Performance for Baseline Parameters

The baseline measurements were performed using the baseline values defined above. Ten complete poker rounds were executed. For each round, the execution time of every protocol phase was recorded separately, with additional measurement of the total time per round. Since the first player performs a different amount of work during the shuffling phase (they don't need to verify their own shuffling proof), its shuffling-proof time is reported separately. For players 2-6, the arithmetic mean of their shuffling-proof times per round is treated as a single observation. All values presented in Table 8.1 are based on ten rounds.

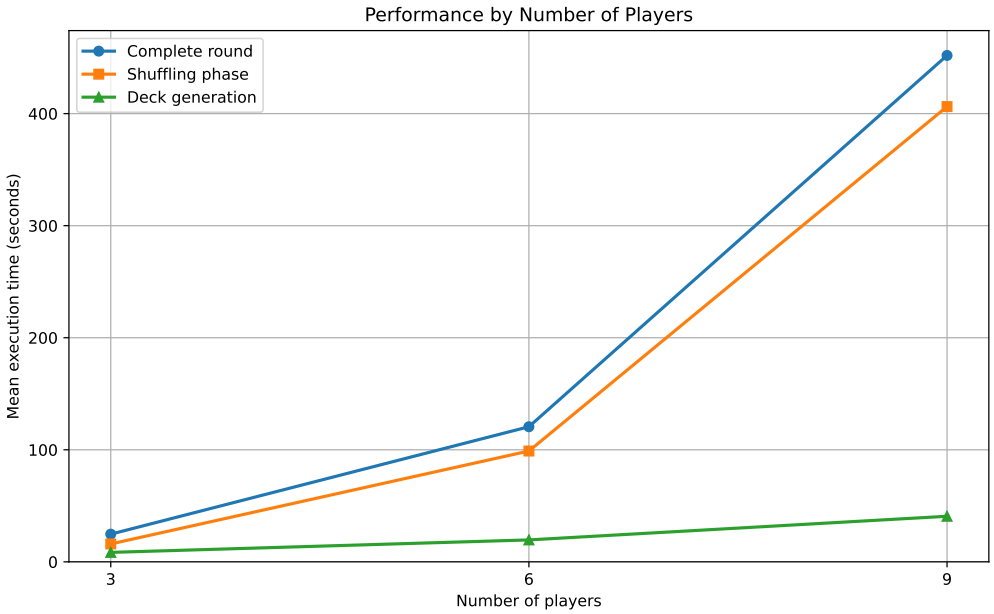
Table 8.1: Execution times for the baseline values

Phase	Mean	SD	Min	Max
Deck generation	19.572	0.572	18.526	20.392
Shuffling proof, first player	16.336	0.065	16.262	16.379
Shuffling proof, players 2-6	98.899	0.703	98.047	99.958
Drawing	0.761	0.005	0.758	0.769
Flop	0.186	0.011	0.173	0.206
Turn	0.070	0.010	0.055	0.086
River	0.062	0.003	0.056	0.069
Showdown	0.096	0.020	0.051	0.135
Complete round	120.526	1.015	119.560	121.584

8.1.2 Performance with Varying Parameters

To evaluate the influence of the protocol parameters, each parameter was varied separately while all remaining parameters were kept at their baseline values. The resulting execution times for the entire round were then compared with the baseline measurements. The detailed measurements for each phase can be found in the appendix B.

8.1.2.1 Varying Number of Players



	3 players	6 players	9 players
Complete round	24.7	120.6	451.9
Shuffling phase	16.0	98.9	406.3
Deck generation	8.4	19.6	40.7

Figure 8.1: Mean execution times for different numbers of players using the baseline parameters

As shown in Figure 8.1, the execution time grows strongly as the number of players increases. The large increase for nine players is mainly caused by the shuffling phase. This is expected, as the shuffling proof is the most computationally heavy part of the protocol and its cost increases considerably with the increasing number of players. In comparison to the shuffling time, the deck-generation time increases much more slowly. Since all clients were executed on the same computer, the experiment with an increasing number of players also includes the effects of CPU load and process scheduling. Therefore, the measured times represent the performance of the local prototype and do not directly show how the protocol would perform if every player used a separate computer. Such a distributed setup would also be affected by network latency and differences in the hardware of the players.

8.1.2.2 Varying Key Length

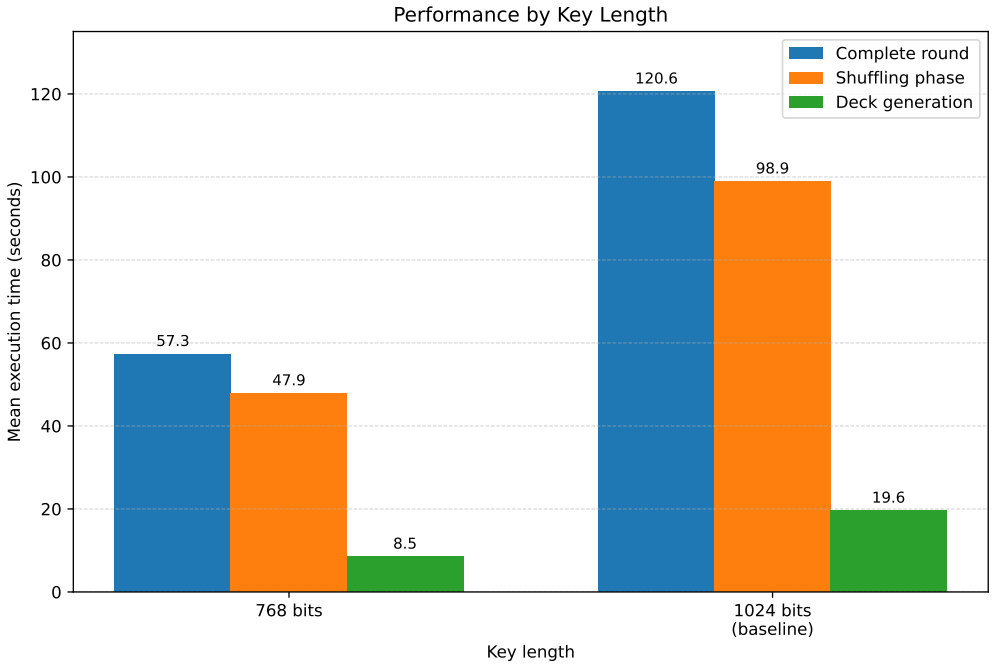


Figure 8.2: Mean execution times with 768-bit and 1024-bit keys

As shown in Figure 8.2, increasing the key length from 768 to 1024 bits increases the execution time greatly. Once again, the total runtime is dominated by the shuffling phase. In this experiment, both deck generation and shuffling become approximately half as fast when the larger key is used. This demonstrates the strong influence of the key length on the practical performance of the protocol.

8.1.2.3 Varying Shuffling Parameter s

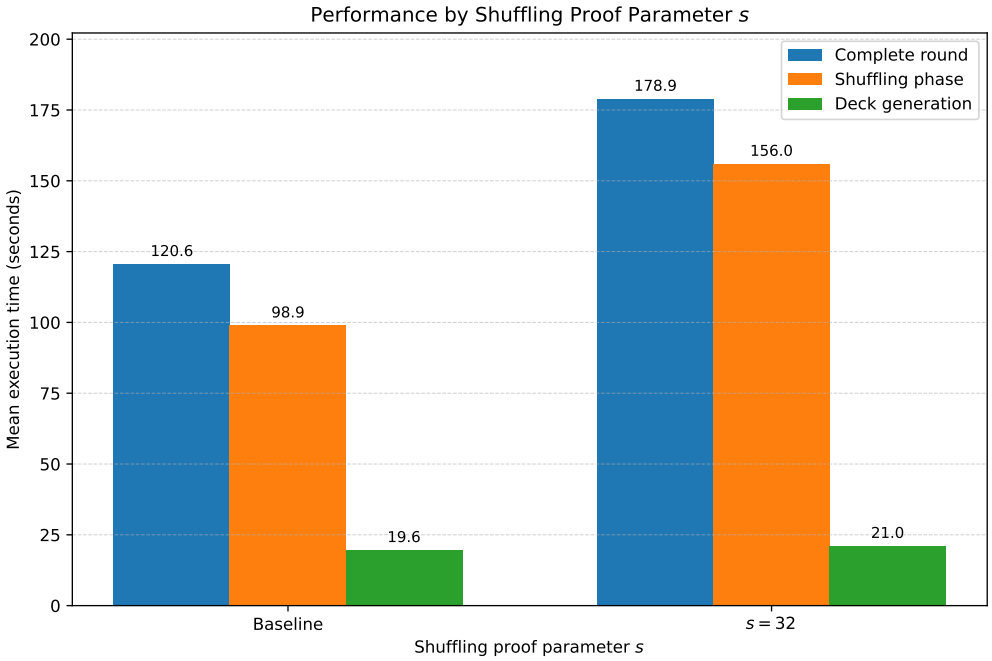


Figure 8.3: Mean execution times for the baseline configuration and $s = 32$.

As shown in Figure 8.3, Increasing the shuffling-proof parameter s increases the time required to generate and verify the proof. This result is expected, however it is interesting, how increasing the proof parameter by 33.3% increases the runtime by roughly 50%.

8.1.2.4 Varying Stadler Parameter l

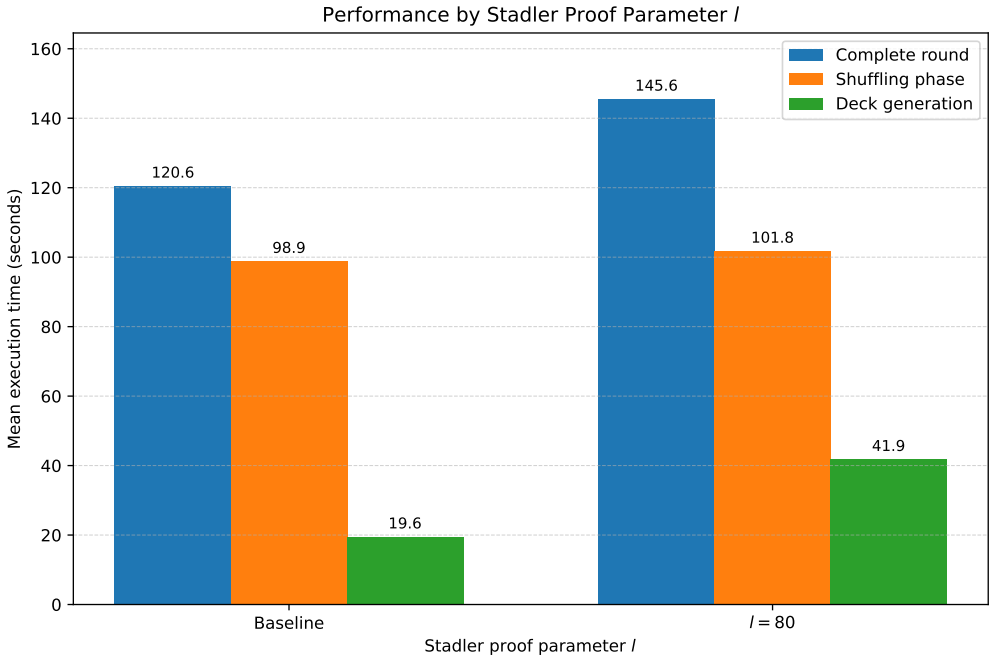


Figure 8.4: Mean execution times for the baseline configuration and $l = 80$.

As shown in Figure 8.4, when increasing the security parameter l, we get the expected results: The runtime only notably increases for the deck generation. We can also conclude that, although Stadler's proof is cheaper than the shuffling proof, its computational cost is still notable. By increasing the parameter l by the factor 2, the deck generation time also roughly doubled.

9
Related Work

The idea of mental poker is not new, so naturally several different approaches have been proposed. This chapter compares the two-deck protocol implemented in this thesis with other related protocols and implementations.

9.1 Early Mental Poker Protocols

The first mental poker protocol was proposed by Shamir, Rivest and Adleman [1]. It is designed for only two players and uses a commutative encryption scheme. One player creates the deck, encrypts every card and applies a secret permutation. The other player can then select cards without knowing their values, because the encryption layers can be removed in a different order. Compared to the two-deck protocol, this construction is much simpler and requires fewer cryptographic operations. However, it does not have the same properties as the more modern poker protocols. It does not use zero-knowledge proofs to verify that the deck was created and shuffled correctly. The encryption is also deterministic and preserves some mathematical properties of the card values, which makes it possible to mark and distinguish certain encrypted cards.

The two-deck protocol uses a more complex structure based on threshold ElGamal encryption. All players contribute to the generation of the face-up deck and encrypt the corresponding exponents to create the face-down deck. Afterwards, every player shuffles and re-masks the encrypted deck. Stadler's proof is used to verify the encrypted exponents, the shuffling proof verifies that no cards were added, removed or replaced, and the Chaum-Pedersen proof verifies the partial decryptions. This allows the protocol to support more than two players and to detect incorrect operations without revealing the private values used by the players, with the disadvantages being the complexity and computational overhead.

9.2 Castellà-Roca's Non-Dropout-Tolerant Protocol

A more comparable mental poker protocol is the protocol presented in Chapter 6 of Castellà-Roca's thesis [16]. The design of this protocol does not consider dropout tolerance. Because of that, it has a much simpler mathematical setup than the two-deck mental poker protocol

presented in this thesis. It works with one mathematical setting and mainly focuses on shuffling, drawing, opening and discarding cards. Instead of generating a new face-up deck with encrypted exponents, the protocol starts with a normal face-up representation of the cards and then turns this deck into a face-down deck. The players then shuffle and re-mask this encrypted deck. Cards then are drawn by selecting one encrypted card and decrypting it with the help of the other players.
The two-deck protocol implemented in this thesis is different. It uses newly generated face-up deck values, encrypted exponents as face-down values, and zero-knowledge proofs that the encrypted exponents were computed correctly. This different architecture adds more complexity and computational overhead, so the protocol presented in chapter 6 of Castellà-Roca's thesis is expected to be faster and easier to implement, while preserving the same important properties as the two-deck protocol implemented here. This includes properties like: no trusted third party, multiple players, correct and verifiable shuffling, private card drawing, cheating detection, resistance against coalitions and confidentiality of player strategies.

9.3 Golle's On-Demand Card Generation

Another relevant protocol is the protocol by Golle [17], which is designed specifically for poker games. In contrast to protocols that shuffle the whole deck before the game starts, Golle's protocol generates cards only when they are needed. A new card is computed as an encrypted random value, and the players then test whether this encrypted card has already been dealt. If a collision is found, the card is discarded and a new one is generated. This makes the protocol well suited for Texas Hold'em, because only a small part of the deck is used during one hand.
Compared to the two-deck protocol implemented in this thesis, Golle's protocol is more application-oriented and is expected to be faster for practical poker games, because it avoids a full initial deck shuffle. Its ElGamal version still requires distributed key generation, homomorphic encryption, encrypted equality tests, and zero-knowledge proofs. However, it does not require the Stadler-style proof used in this thesis, and therefore it does not need the same two-group mathematical setting.

9.4 Practical Implementations

There also exists a practical implementation called Mental Texas Hold'em [18]. Compared to the two-deck protocol implemented in this thesis, Mental Texas Hold'em is less complex. It is based on SRA commutative encryption, where cards are represented as integers and encryption and decryption are done by modular exponentiation. Each player encrypts and shuffles the deck, and cards are later opened by revealing the corresponding card-specific keys. Therefore, the main implementation difficulty is mostly the management of encrypted decks, keys, and peer-to-peer communication.
As mentioned above, the protocol in this thesis requires a more complex mathematical setting and computes two decks. This makes the implementation more difficult, because the protocol

must not only compute the card operations, but also prove that they were done correctly. Therefore, Mental Texas Hold'em is more usable as a practical game implementation, while this thesis focuses on the implementation challenges of a modification of a zero-knowledge-based TTP-free dropout-tolerant protocol.

10 Conclusion

In this thesis we analyzed the dropout-tolerant mental poker protocol proposed by Castellà-Roca. We discovered some ambiguities in the mathematical setting and implemented and evaluated a working prototype based on the two-deck construction. The resulting implementation demonstrates that the main cryptographic operations of the protocol can be combined with a peer-to-peer system to simulate a round of Texas Hold'em without relying on a trusted third party. The implementation demonstrates the important concepts of mental poker, including distributed card encryption, partial decryption, verifiable cryptographic operations and peer-to-peer communication between the participants. Players can create and discover lobbies, establish direct connections, generate a shared encryption key and create an encrypted deck. Cards are drawn and partially decrypted while remaining hidden to the other participants until they are revealed. The cryptographic proofs exchanged during the protocol are also verified, thus detecting cheating attempts.

Since the vetoing mechanism was omitted, the resulting protocol cannot be considered fully dropout-tolerant. A player can interrupt an ongoing round by disconnecting, refusing to publish a required partial decryption or publishing an invalid proof. In such a case, the cards are no longer decryptable, and the current round has to be restarted. However, the remaining players can remove the participant, compute a new shared key and begin a new round without recreating the complete lobby. The implementation is thus tolerant of dropouts between rounds, but only partially tolerant of dropouts during an active round.

We also tested and analyzed different parameter configurations, such as the key length and security parameters of the zero-knowledge proofs. We measured the execution time and reasoned about an appropriate trade-off between the security and performance.

Future work could focus on redesigning the original vetoing mechanism to fit the two-deck protocol, and analyzing its properties and usability. The application could also be extended, by adding the missing components needed for a normal poker game, such as betting rounds, automatic hand evaluation and GUI. Regarding the performance, parallelization could be used to speed up the computation, possibly even allowing use of stronger security parameters. Another optimization could include finding a more efficient shuffling proof, which would not dominate the runtime.

Bibliography

[1] Shamir, A., Rivest, R. L., and Adleman, L. M. Mental Poker. Technical Report MIT-LCS-TM-125, Massachusetts Institute of Technology, Laboratory for Computer Science (1979).

[2] Barnett, A. and Smart, N. P. Mental Poker Revisited. In *Cryptography and Coding*, volume 2898 of *Lecture Notes in Computer Science*, pages 370–383. Springer (2003).

[3] Castellà-Roca, J., Sebé, F., and Domingo-Ferrer, J. Dropout-Tolerant TTP-Free Mental Poker. In *Trust, Privacy and Security in Digital Business*, volume 3592 of *Lecture Notes in Computer Science*, pages 30–40. Springer (2005).

[4] Crépeau, C. A Secure Poker Protocol That Minimizes the Effect of Player Coalitions. In *Advances in Cryptology - CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 73–86. Springer (1986).

[5] Crépeau, C. A Zero-Knowledge Poker Protocol That Achieves Confidentiality of the Players' Strategy or How to Achieve an Electronic Poker Face. In *Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 239–250. Springer (1987).

[6] Stadler, M. Publicly Verifiable Secret Sharing. In *Advances in Cryptology - EUROCRYPT '96*, volume 1070 of *Lecture Notes in Computer Science*, pages 190–199. Springer (1996).

[7] Diffie, W. and Hellman, M. E. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654 (1976).

[8] ElGamal, T. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472 (1985).

[9] Desmedt, Y. and Frankel, Y. Threshold Cryptosystems. In *Advances in Cryptology - CRYPTO '89*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer (1990).

[10] Pedersen, T. P. A Threshold Cryptosystem without a Trusted Party. In *Advances in Cryptology - EUROCRYPT '91*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer (1991).

[11] Boneh, D. and Shoup, V. *A Graduate Course in Applied Cryptography* (2023). Version 0.6, latest version January 2023. Available at https://toc.cryptobook.us/, accessed 10 July 2026.

[12] Chaum, D. and Pedersen, T. P. Wallet Databases with Observers. In *Advances in Cryptology - CRYPTO '92*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer (1993).

[13] Fiat, A. and Shamir, A. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology - CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer (1987).

[14] Gordon, D. M. Discrete Logarithms in GF(p) Using the Number Field Sieve. *SIAM Journal on Discrete Mathematics*, 6(1):124–138 (1993).

[15] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Béguelin, S., and Zimmermann, P. Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 5–17. ACM (2015).

[16] Castellà-Roca, J. *Contributions to Mental Poker*. Ph.D. thesis, Universitat Autònoma de Barcelona (2005).

[17] Golle, P. Dealing Cards in Poker Games. In *Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC 2005)*, volume 1, pages 506–511. IEEE Computer Society (2005).

[18] PredatorRay. Mental Texas Hold'em. https://github.com/predatorray/mental-texas-holdem. GitHub repository, accessed 30 June 2026.

A

Primes used

o (1024 bits) = 16222179049251211207128416528461969064324828243781812261887680768816209390437644104593006270956628841984569112709600215248411539944811108389068331935228315957489546006487399167640026771085594017323043825975243178183173105588498975632724950500640631340682728762614756742732298275762361500836323039491221628896 7

p (1023 bits) = 81110895246256056035642082642309845321624141218909061309438403844081046952188220522965031354783144209922845563548001076242057699724055541945341659676141579787447730032436995838200133855427970086615219129876215890915865527942494878163624752503203156703413643813073783713661491378811807504181615197456108144483

q (1022 bits) = 40555447623128028017821041321154922660812070609454530654719201922040523476094110261482515677391572104961422781774000538121028849862027770972670829838070789893723865016218497919100066927713985043307609564938107945457932763971247439081812376251601578351706821906536891856830745689405903752090807598728054072241

o (768 bits) = 135848669038672331791579175110297830452635814350988290278606944402231951555004531931372617957849310706384251751462169619728194219127683983678191446527092859121765879251718628297926219304548746002219736832526858554753297556981990208 7

p (767 bits) = 679243345193361658957895875551489152263179071754941451393034722011159757775022659656863089789246553531921258757310848098640971095638419918390957232635464295608829396258593141489631096522743730011098684162634292773766487784909951043

q (766 bits) = 339621672596680829478947937775744576131589535877470725696517361005579878887511329828431544894623276765960629378655424049320485547819209959195478616317732147804414698129296570744815548261371865005549342081317146386883243892454975521

B

Measurements for Varying Parameters

B.1 Baseline Values with Three Players

Table B.1: Execution times with three players, in seconds

Phase	Mean	SD	Min	Max
Deck generation	8.442	0.183	8.307	8.801
Shuffling proof, first player	5.301	0.038	5.267	5.359
Shuffling proof, players 2-3	15.963	0.118	15.818	16.136
Drawing	0.132	0.029	0.112	0.186
Flop	0.051	0.017	0.041	0.084
Turn	0.019	0.006	0.014	0.027
River	0.019	0.006	0.014	0.029
Showdown	0.021	0.007	0.017	0.034
Complete round	24.707	0.332	24.438	25.320

B.2 Baseline Values with Nine Players

Table B.2: Execution times with nine players, in seconds

Phase	Mean	SD	Min	Max
Deck generation	40.669	0.711	39.561	41.328
Shuffling proof, first player	44.472	1.778	43.318	46.519
Shuffling proof, players 2-9	406.264	7.360	401.125	414.696
Drawing	2.932	0.022	2.914	2.960
Flop	0.520	0.010	0.511	0.532
Turn	0.166	0.004	0.162	0.172
River	0.170	0.004	0.162	0.175
Showdown	0.225	0.035	0.180	0.266
Complete round	451.912	7.872	445.946	460.834

B.3 Baseline Values with a 768-Bit Key

Table B.3: Execution times with a 768-bit key, in seconds

Phase	Mean	SD	Min	Max
Deck generation	8.488	0.319	8.086	9.136
Shuffling proof, first player	7.750	0.114	7.585	7.845
Shuffling proof, players 2-6	47.856	0.760	46.929	49.014
Drawing	0.398	0.056	0.355	0.494
Flop	0.094	0.013	0.075	0.113
Turn	0.035	0.007	0.029	0.047
River	0.035	0.004	0.030	0.043
Showdown	0.062	0.011	0.043	0.081
Complete round	57.322	0.860	56.514	58.727

B.4 Baseline Values with $l = 80$

Table B.4: Execution times with $l = 80$, in seconds

Phase	Mean	SD	Min	Max
Deck generation	41.873	2.482	38.960	45.353
Shuffling proof, first player	16.773	0.221	16.610	17.086
Shuffling proof, players 2-6	101.843	0.988	101.258	102.984
Drawing	0.754	0.013	0.738	0.775
Flop	0.176	0.016	0.159	0.199
Turn	0.064	0.004	0.059	0.069
River	0.059	0.003	0.054	0.063
Showdown	0.084	0.029	0.044	0.119
Complete round	145.615	2.938	143.392	149.849

B.5 Baseline Values with $s = 32$

Table B.5: Execution times with $s = 32$, in seconds

Phase	Mean	SD	Min	Max
Deck generation	21.047	0.328	20.598	21.465
Shuffling proof, first player	25.834	0.374	25.596	26.265
Shuffling proof, players 2-6	155.970	0.578	155.336	156.469
Drawing	0.778	0.022	0.754	0.804
Flop	0.178	0.010	0.160	0.186
Turn	0.063	0.004	0.058	0.069
River	0.063	0.003	0.059	0.069
Showdown	0.122	0.027	0.071	0.148
Complete round	178.936	0.415	178.405	179.269