LegitPoker: Real-Time Verifiably Fair and Secret Shuffle and Deal

Daniel Rubin, John Wu

October 29, 2025

1 Introduction

Online poker brings in over \$5 billion USD in yearly revenue, with player accounts totaling more than \$100 billion. For an industry that mostly operates in a legal gray zone, it demands an enormous amount of trust from players in the way the game is run, especially in the shuffle process. In a live card game, the physical act of shuffling and the card identities being hidden by shuffling face-down ensure a fair and secret reordering of the deck. But in standard online card games today, players do not get to see how the server runs the game; they don't know how the cards are placed in order, and the order is visible to the server that runs the game. This means that online poker has serious vulnerabilities to unfair play. A malicious poker site could arrange decks to favor more betting or to favor certain players. Even if the shuffles are fair, the operator has a full view of the deck and can use that information to its advantage, and even if the operator is honest, the security of his server could be compromised by malicious players.

Modern cryptography makes it possible to ensure that an electronic card shuffle is fair (the order of the deck is chosen uniformly at random from all possible permutations) and secret (no one knows anything about the order of the cards), and that the cards are dealt secretly and correctly (only the player intended to receive the cards learns the card identities, and when the cards are revealed publicly, the player cannot lie about which cards they are). The challenge in an online card game is to perform a Verifiably Fair and Secret Shuffle and Deal (VFSSD) to the appropriate number of players in the time that players expect from an electronic game (no more than about 3 seconds).

2 Background: Mental Poker

The problem of how players could play poker electronically and keep their cards secret was first addressed by Rivest, Shamir, and Adleman (of RSA) in their paper titled "Mental Poker" [6]. To our knowledge, it is only recently, in the context of blockchain-based online poker, that the Mental Poker scheme has been implemented. ZkHoldem [4] employs the Mental Poker re-encryption shuffle among the players, and augments the shuffle-and-encrypt algorithm of each participant with the generation of a Zero Knowledge Proof (ZKP) that the shuffle and encryption was done correctly, thus securing the protocol even against malicious players. So long as at least one player is honest (meaning that not all players collude, a situation against which players are highly incentivized in a competitive poker game), no player will have any knowledge about the order of the cards at the end of the shuffling process, while the proofs ensure that each played his role correctly. To deal a card, each player other than the player to whom the card is intended must participate in a round of decryption, so that when the card is presented to its owner, his final decryption will privately reveal the card to him. These decryption steps have also been proven to ensure integrity.

The problem is that the shuffle and deal process in zkHoldem takes longer than most online poker players are willing to wait, 38 seconds to shuffle and deal to 3 players.

3 Related Work: InstaPoker

Another team has developed a project called InstaPoker [5], with the aim of improving the shuffle and deal process. InstaPoker improves the speed of the shuffle over zkHoldem, and also reduces the dependence of the live dealing on the time of the shuffle, by designing a protocol where the shuffling can take place ahead of time in an "offline" phase. InstaPoker implements a Multi-Party Computation (MPC) version of a shuffle between committee members, in which the secrecy of the resulting permutation is enforced information-theoretically by the MPC algorithm (each committee member ends up with a secret share of the permutation) rather than reencryption. InstaPoker uses an Identity Based Encryption (IBE) scheme, with keys managed by a "keyper" committee. InstaPoker accomplishes its proof of correct secret shuffling with a sophisticated collaborative zkSNARK generated by the committee as they IBE encrypt their shares of the deck. At the time of writing this document, the InstaPoker team has not launched an app or demo, but the paper claims offline shuffles by an 8-person committee in under 3 seconds, and online decryptions in under 1 second.

4 LegitPoker Fair Shuffle and Deal

The LegitPoker VFSSD algorithm is a hybrid of the zkHoldem and InstaPoker protocols. LegitPoker employs an offline phase shuffle committee and an online dealing phase. Our shuffle committee employs a sequential re-encryption shuffle, as in zkHoldem, instead of an MPC version of the RS algorithm. So long as a single shuffler is honest (doesn't collude with all other shufflers), no party can learn anything about the order of the deck. Notably, our proof establishes that the shuffle is fair in that it is the result of applying the RS algorithm. To our knowledge this is not done in zkHoldem or InstaPoker; while they do have shufflers generate the permutation by a specific algorithm, the proof does not attest to it. What this amounts to is that if the shuffling is compromised by a committee all of whose members collude, with our system the committee would learn the order of the deck, but would still not be able to manipulate it, whereas manipulation would be possible in the zkHoldem and InstaPoker protocols.

While we ultimately plan to employ a keyper committee similar to InstaPoker, for the purpose of our demo, the shufflers themselves play the roles of the keypers, and participation from each shuffler is required for decryption of every card.

5 Fair Shuffles and the Rao–Sandelius (RS) Algorithm

We require an efficient algorithm to sample a permutation of a 52-element set uniformly at random. The two most widely used algorithms, originally developed for use by statisticians, are the Fisher–Yates and Rao–Sandelius algorithms [1]. Both are designed to output a permutation using an asymptotically optimal (least) number of random bits as input. We use the RS algorithm in our shuffle as we found it more ZK-compatible.

RS algorithm (pseudocode)

Adapted from Algorithm 3 in Bacher, Bodini, Hwang, and Tsai (2017).

Algorithm 1 Rao–Sandelius (RS) shuffle (binary coin)

```
Input: sequence $c = (c_1, \dots, c_n)$ of distinct elements
Output: a uniformly random permutation of $c$
function RS($c$)
    if $|c| \le 1$ then
        return $c$
    else if $|c| = 2$ then
        if RAND_BIT() = 1 then
            return $(c_2, c_1)$
        else
            return $(c_1, c_2)$
        end if
    end if
    $A_0 \leftarrow [\,]$; $A_1 \leftarrow [\,]$
    for $i \leftarrow 1$ to $|c|$ do
        $b \leftarrow$ RAND_BIT()
        append $c_i$ to $A_b$
    end for
    return CONCAT(RS($A_0$), RS($A_1$))
end function
```

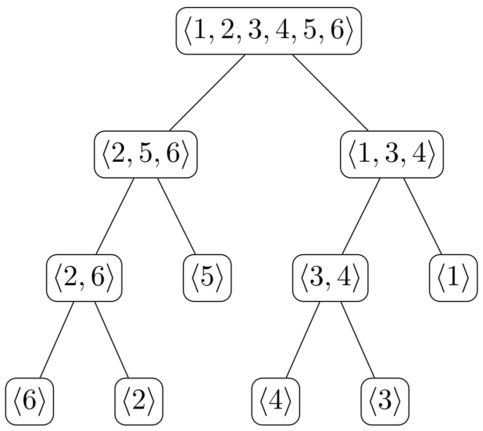
Worked example ($n = 6$)

We shuffle the list $c = \langle 1, 2, 3, 4, 5, 6 \rangle$ using the RS algorithm with the bitstring consumed in order by RAND_BIT():

`10110001011001` (14 bits).

At the root, the first six bits partition c into $A_0 = \langle 2, 5, 6 \rangle$ and $A_1 = \langle 1, 3, 4 \rangle$. We then recurse on each subset; on size-2 leaves we use one bit to decide whether to swap.

The binary tree of arrays for this specific run is:



Concatenating the leaves left-to-right gives the output permutation:

$$\pi = \langle 6, 2, 5, 4, 3, 1 \rangle.$$

5.1 Proving the execution of RS

Part of the work of each shuffler in a Shuffle committee is to prove that the permutation π_S they applied to the incoming encrypted card order was the result of applying the RS algorithm, with bitstring input also verifiably random. To achieve this, we take the following steps:

- The RS algorithm itself must be modified to be more *stable*. We must eliminate the conditional base cases, as well as the logic that depends on the length of lists at every node, since these would reveal the bitstring and therefore also reveal the permutation. To make the make the shuffle algorithm SNARK friendly, we instead perform the first level of the RS algorithm multiple times in succession. This first level of RS in which each card is assigned to the 0 or 1 group and placed in order within its group is called an *inverse riffle shuffle.*

- The uniformity of the repeated riffle shuffle is discussed in the famous paper of Bayer and Diaconis [2], in which they prove that the total variation of the distribution of random repeated riffle shuffles to the uniform measure on permutations of size 52 rapidly approaches 0 after 7 shuffles.

- **Verifiable randomness.** The random bitstring input to the stable RS algorithm is generated by each shuffler using a Cryptographically Secure Pseudorandom Number Generator (CSPRNG) H. (For our protocol, we take H to be the Poseidon hash.) The transcript the shuffler S_j absorbs into H consists of a nonce, the shuffler's ElGamal secret key $sk_j = x_j$, and the corresponding public key $pk_j = x_jG$, or

$$\vec{b} = H(\text{nonce}|sk_j|pk_j). \tag{1}$$

  In addition to the zkSNARK proof of the shuffle, S_j also reveals the nonce and gives a proof of the relation between the secret key and public key variables of the transcript. This way S_j is committed to a transcript which is secret from all other parties.

- **Prefix sums to determine new card position.** The circuit structure of the Shuffler's zkSNARK proof for each level/inverse riffle shuffle consists of equations

$$\begin{aligned} b[i](1-b[i]) &= 0 \\ z[i+1] &= z[i] + (1-b[i]) \\ o[i+1] &= o[i] + b[i] \\ n_0 &= \sum_{i=0}^{51}(1-b[i]) \end{aligned}$$

  where $\vec{b}$ is the bitstring, $\vec{z}, \vec{o}$ are prefix sums counting the number of zeros and ones respectively, and n_0 is the total number of zeros. To constrain the position of a card in the next level, we use a selector

$$\text{newpos}[i] = (1-b[i])z[i] + b[i](n_0 + o[i]). \tag{2}$$

- **Product-based set permutation argument to enforce assignments.** Two additional witness vectors, idxcurr$[i]$ and idxnext$[i]$, identify the card identity number at position i in the current and next levels. We cannot use standard circuit constraints to specify the values of idxnext since the position depends on the secret witness. Instead we set key variables

$$\begin{aligned} T[i] &= \text{newpos}[i]K + \text{idxcurr}[i] \\ T'[j] &= jK + \text{idxnext}[j] \end{aligned}$$

with $K = 64$ a fixed constant. The Shuffler enforces equality over the sets $\{T[i]\}, \{T'[j]\}$ by generating a Fiat-Shamir challenge α and proving

$$\prod_{i=0}^{51}(\alpha - T[i]) = \prod_{j=0}^{51}(\alpha - T'[j]). \tag{3}$$

- The R1CS constraints and permutation argument are implemented in a Groth16 proof.

6 RS zkSNARK + Bayer–Groth

Each member of our shuffle committee must permute the deck, re-encrypt, and prove that he shuffled and encrypted correctly. Permuting by RS makes the shuffle fair, having multiple parties each put on their own encryption keeps the shuffle secret, and the proofs enforce that all parties perform their tasks correctly.

The main requirement of the encryption scheme for keeping the shuffle secret is that it is additive, in the sense that

$$E_{\mathrm{sk}_2}\big(E_{\mathrm{sk}_1}(m)\big) = E_{\mathrm{sk}_1+\mathrm{sk}_2}(m).$$

For this we use ElGamal encryption over the curve BN254.

The ideal tool for proving correctness of a secret shuffle is the Bayer–Groth (BG) proof [3]. BG proves that a public output list of ElGamal-encrypted messages is a permutation and re-encryption of a public input list. Generating a BG proof is dramatically faster than employing a general purpose SNARK, such as Groth16, Plonk, or Spartan, that encodes the encryption in each entry as a circuit.

First recall the Chaum–Pedersen (CP) proof of equality of discrete log. CP is a protocol by which a Prover can convince a Verifier that two elliptic curve points P', Q' are multiples of given points P, Q by the same scalar multiple x.

Algorithm 2 Chaum–Pedersen (CP) proof of equality of discrete log

```
1: Prover:
2:     Input: $P, Q \in \mathbf{G}$, secret $x \in \mathbf{F}_q$, scalar field.
3:     Let $P' = xP$, $Q' = xQ$.
4:     Sample $r$ uniformly at random from $\mathbf{F}_q$.
5:     Let $S = rP$, $T = rQ$.
6:     Simulate challenge $c = H(S, T) \in \mathbf{F}_q$ ($H$ a cryptographic hash function).
7:     Let $e = r + cx$.
8:     Output: Send to Verifier $(P, Q, P', Q', S, T, e)$.
9: Verifier:
10:    Recompute $c = H(S, T)$.
11:    Check if $eP = S + cP'$ and $eQ = T + cQ'$.
12:    Output: Accept if both true, otherwise Reject.
```

BG is essentially a souped-up version of a CP proof that can be applied to a list of encrypted elements even when the order is permuted. We will have use of the CP proof itself in the dealing process and when a player reveals his cards at showdown, see below.

Bayer–Groth shuffle argument (high-level)

We summarize the core BG protocol used to prove that a public output list of ElGamal ciphertexts is a permutation and re-encryption of a public input list, without revealing the permutation.

Algorithm 3 Modified Bayer–Groth shuffle argument (high-level sketch)

```
1: Public: ElGamal key pk, commitment key ck; input ciphertexts $C_1, \dots, C_N$ and output ciphertexts $C'_1, \dots, C'_N$.
2: Goal: Prove knowledge of a permutation $\pi$ resulting from RS shuffle and re-encryption randomness $\rho_1, \dots, \rho_N$ such that $C'_i = C_{\pi(i)} \cdot E_{\mathsf{pk}}(1; \rho_i)$ for all $i$.
3: Input the permutation and power-permutation. Take as secret witnesses $(\pi(1), \dots, \pi(N))$ and the "power-encoding" $(x^{\pi(1)}, \dots, x^{\pi(N)})$.
4: Inner product argument linking commitment to output of RS Commit to "power-encoding". Perform folding inner-product with verifier challenges. Link to public folded output of RS.
5: Tie to the ciphertexts (multi-exponentiation check). Prove that there exists $\rho$ such that $\prod_{i=1}^{N} C_i^{x^i} = E_{\mathsf{pk}}(1; \rho) \cdot \prod_{i=1}^{N} (C'_i)^{x^{\pi(i)}}$ via the BG multi-exponentiation sub-argument.
6: Conclude correctness. From ElGamal homomorphism, this implies $\prod M_i^{x^i} = \prod M'^{x^{\pi(i)}}_i$, where $M_i, M'_i$ are the messages corresponding to ciphertexts $C_i, C'_i$. Interpreting both sides as polynomials in $x$ and applying Schwartz–Zippel again, it follows (except with negligible probability) that $(M'_1, \dots, M'_N)$ is a permutation of $(M_1, \dots, M_N)$ under the same $\pi$.
```

Adapted from Bayer and Groth, EUROCRYPT 2012, p. 269.

7 Combining RS With BG

The novel part of our zkSNARK is in combining the proof of the RS shuffle with BG. BG only proves that the list is an encryption of *some* permutation, but does not on its own enforce how the permutation was generated. We wish to prove that the shuffle was performed fairly by proving that the permutation was generated by using the RS algorithm on a truly random bitstring, and then employ a BG proof showing correct encryption on the same permutation. The challenge is to combine the two in such a way that the prover does not need to prove in an R1CS circuit the calculation of a commitment to the permuted indices, which is as large a circuit as a circuit for the re-encryption of every element.

Both the RS and BG proofs must be modified to prove to a verifier that the permutation generated by RS is the permutation committed to in BG. A naive approach would be to perform a SNARK proof of the computation of the Pedersen commitment of the permutation indices $[\pi(1), \dots, \pi(52)]$. But this involves generating a SNARK of 52 scalar multiplications, which is as expensive as a direct SNARK proof of the ElGamal encryption in each index, which is what the efficient BG proof is designed to avoid.

Instead we take the following approach. It suffices to convince the verifier that the Pedersen commitment to the power-permutation vector

$$\begin{aligned} C_b &= \mathrm{com}(b; r), \\ b &= \left[x^{\pi(1)}, \dots, x^{\pi(52)}\right] \end{aligned}$$

opens to the secret vector b with exponents produced by RS. To do this, we perform a Bulletproof-style Inner Product Argument on the commitment C_b, obtaining a folded commitment C_b^F. The verifier receives C_b along with the folding challenges, plus the RS zkSNARK encoding the running of RS as well as the computation of its public output F_b, the folded linear combination of the secret vector b. The verifier checks that F_b commits to C_b^F and C_b folds to C_b^F.

Note that the product-based permutation argument of BG that in a standard application of BG attests that the reordered indices really do constitute a permutation is no longer necessary, since we prove the new indices are produced by RS, which always produces a permutation.

8 Dealing Protocol

With the cards secretly shuffled, the next challenge is to efficiently provide players the information needed to open their cards and only their cards. For our technical demo, we take the following approach:

- The players and shufflers engage in a decryption protocol. The encrypted card at a given position is given in the form
$$c_1 = rG, \qquad c_2 = M + rPK_{comm} \tag{4}$$
where G is the generator, $M = mG$ where $m \in \{0, 1, \dots, 51\}$ identifies the card, and
$$PK_{comm} = \sum_j x_j G \tag{5}$$
with x_j the secret key of shuffler S_j.
- Each player P_α creates a secret–public keypair, $SK_\alpha = s_\alpha$, $PK_\alpha = s_\alpha G$. P_α publishes PK_α to the game ledger.
- Each shuffler re-encrypts a card to be dealt to player P_α with their own blinding factor. That is, each shuffler S_j chooses a random δ_j and publishes $\delta_j G$ and $\delta_j H$ where
$$H = PK_{comm} + PK_\alpha,$$
along with a CP proof that these points come from scalar multiplying G and H by the same factor.
- The game ledger operator verifies the CP proofs from all shufflers.
- Let
$$\Delta = \sum_j \delta_j.$$
P_α forms the *blinded base*
$$A_\alpha = c_1 + \sum_j \delta_j G = (r + \Delta)G, \tag{6}$$
and the *blinded message*
$$B_\alpha = c_2 + \sum_j \delta_j H = M + (r + \Delta)PK_{comm} + \Delta PK_\alpha. \tag{7}$$
- Then each shuffler removes his encryption that was put on during the shuffle process. Each shuffler produces
$$\mu_{\alpha,j} = x_j A_\alpha = x_j(r + \Delta)G, \tag{8}$$
which the player can sum to produce
$$\mu_\alpha = \sum_j \mu_{\alpha,j} = (r + \Delta)\sum_j x_j G = (r + \Delta)PK_{comm}. \tag{9}$$

- Player P_α (and only player P_α) multiplies the public ΔG by the secret s_α to get

$$S = s_\alpha \Delta G = \Delta PK_\alpha. \tag{10}$$

  With this, P_α can recover the card by computing

$$\begin{aligned} B_\alpha - \mu_\alpha - S &= (M + (r+\Delta)PK_{comm} + \Delta PK_\alpha) - (r+\Delta)PK_{comm} - \Delta PK_\alpha \\ &= M. \end{aligned}$$

- P_α decodes the card identity m from M by reference to a lookup table.

This decryption process requires all of the shufflers to participate (n-of-n).

If a player P_α is forced to reveal his cards at showdown, he must produce a CP proof for the tuple $(G, \Delta G, PK_\alpha, S)$ indicating correct use of his secret key s_α without revealing it.

9 Implementation

We implemented this shuffle and deal protocol on a MacBook Pro with an M1 Max chip with 10 cores. The total time of the shuffle and deal with 7 shufflers and 7 players was 1.4 seconds. The average time of proof generation of a shuffle and encrypt proof was 120ms, with 20ms verification time. Card decryptions took 52ms-55ms each, for a total dealing time of 374ms. Note that in a practical deployment, the shuffling phase would be performed ahead of time, and only the dealing would occur live.

10 Future Work

In future work, we will replace this n-of-n shuffler decryption process with a threshold decryption process using an Identity Based Encryption scheme with key shares managed by a separate "keyper" committee (somewhat more like InstaPoker). This is to allow shufflers to spend their time only shuffling during a phase involving no interaction with live players, and not have to be live to participate in dealing. Likewise, the keyper committee spends its effort exclusively responding to player decryption requests.

References

[1] A. Bacher, O. Bodini, H.-K. Hwang, and T.-H. Tsai. Generating random permutations by coin-tossing: Classical algorithms, new analysis, and modern implementation. *ACM Transactions on Algorithms*, 13(2):24:1–24:43, 2017. doi:10.1145/3009909.

[2] D. Bayer and P. Diaconis. Trailing the dovetail shuffle to its lair. *The Annals of Applied Probability*, Vol. 2, No. 2, pp. 294-313. 1992.

[3] S. Bayer and J. Groth. Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In D. Pointcheval and T. Johansson (eds.), *Advances in Cryptology – EUROCRYPT 2012*, LNCS 7237, pp. 263–280. Springer, 2012. PDF.

[4] zkHoldem Team. ZKShuffle: Mental Poker on SNARK for Ethereum (whitepaper). Available at `https://zkholdem.xyz/wp-content/themes/zkholdem-theme/zkshuffle.pdf`.

[5] S. Garg, A. Kate, P. Mukherjee, R. Sinha, and S. Sridhar. Insta-Pok3r: Real-time Poker on Blockchain. *Cryptology ePrint Archive*, Paper 2024/1061, 2024. `https://eprint.iacr.org/2024/1061`.

[6] A. Shamir, R. Rivest, and L. Adleman. Mental Poker. *The Mathematical Gardener*, pp. 37–43. Ed. D. Klarner. Prindle, Weber, and Schmidt, 1981.