TrueShuffle: A Verifiably Fair Poker Protocol Using Commutative Elliptic Curve Encryption and On-Chain Settlement

Technical Whitepaper v1.0

TrueShuffle Protocol Team

February 2026

Contents

1 Abstract

Online poker constitutes a global market exceeding $100 billion in annual gross gaming revenue, yet the industry remains fundamentally compromised by a structural trust deficit. Players must entrust platform operators with complete authority over card dealing, hand evaluation, and fund custody — a concentration of power that has repeatedly invited catastrophic abuse. The Full Tilt Poker insolvency of 2011, in which executives misappropriated approximately $440 million in player deposits [1], and the Ultimate Bet superuser scandal of 2008, in which insiders exploited privileged access to observe opponents' hole cards [2], represent only the most prominent instances of systemic fraud in an industry where the incentive to cheat scales directly with the stakes.

Existing blockchain-based poker platforms have attempted to address these trust failures, but most merely transfer wagering to a smart contract while retaining a centralized server as the trusted dealer. The server still observes the entire deck, selects permutations, and distributes cards — preserving precisely the attack surface that enabled historical scandals. So-called "provably fair" systems that publish server-side random seeds post hoc provide audit trails but not prevention: a compromised server can still deal selectively in real time.

This paper presents TrueShuffle, a mental poker protocol that eliminates the trusted dealer entirely. TrueShuffle employs commutative elliptic curve encryption over the Secp256k1 curve [3] to enable players to collectively encrypt, shuffle, and deal a deck such that no single party — including the coordinating server — ever observes any card before its intended recipient decrypts it. The protocol proceeds through formally defined phases: `KEY_COMMIT`, `KEY_REVEAL`, `SHUFFLE`, `LOCK`, `DEAL`, `BETTING`, `SHOWDOWN`, and `COMPLETE`, with each transition governed by on-chain state recorded on the MegaETH Layer 2 network [4]. Per-card encryption keys are exchanged via X25519-authenticated end-to-end channels [5], ensuring that even the relay infrastructure cannot access card data. All financial settlements — buy-ins, pot distributions, and dispute resolutions — execute as atomic smart contract operations, yielding trustless payouts with sub-cent gas costs and approximately 10-millisecond confirmation latency.

We make the following contributions: (1) the first production-grade implementation of mental poker on an EVM-compatible Layer 2, with complete Solidity contracts for on-chain shuffle verification, deck hash chaining, card key commitment, and showdown card verification using native Secp256k1 elliptic curve arithmetic; (2) a formal security analysis demonstrating that no coalition of fewer than m players can determine any unrevealed card's identity, under the hardness of the Elliptic Curve Discrete Logarithm Problem; (3) a complete on-chain dispute resolution mechanism with cryptographic proof-of-cheating and escrow slashing; and (4) an autonomous bot framework supporting DNA-based strategy parameterization and adversarial learning, enabling continuous competitive play without human intervention.

2 Introduction — The Trust Crisis in Online Poker

2.1 The $100 Billion Trust Problem

Online poker is among the largest segments of the global online gambling industry, generating tens of billions of dollars in annual revenue and serving hundreds of millions of registered accounts across dozens of jurisdictions. The game's appeal is structural: poker is a game of incomplete information in which skill demonstrably dominates luck over statistically significant sample sizes, distinguishing it from purely stochastic casino games. This strategic depth attracts a player population that ranges from casual recreational participants to professional grinders whose livelihoods depend on edge extraction over millions of hands.

Yet the architecture of every major online poker platform rests on a single, fragile assumption: that the operator is honest. In the traditional client-server model, the platform operator functions as the omniscient dealer. The server generates the deck permutation, observes every card in every hand, assigns hole cards to players, controls the sequence of community card reveals, evaluates hands at showdown, and ultimately custodies all deposited funds. This architectural choice is not merely an implementation convenience — it is the foundational design pattern of the industry, inherited from the earliest platforms of the late 1990s and never fundamentally reconsidered.

The consequences of this trust concentration have been severe, repeated, and structurally predictable.

The Ultimate Bet Superuser Scandal (2008). Beginning as early as 2004, insiders at Ultimate Bet and its sister site Absolute Poker exploited a software feature that granted certain accounts the ability to observe opponents' hole cards in real time — a capability referred to internally as "God Mode" [2]. The fraud persisted for years, extracting an estimated $22 million from unsuspecting players before sta-

tistical anomalies in hand histories drew the attention of the poker community. Crucially, the cheating was not detected by the platform's internal controls; it was discovered by players who applied forensic statistical analysis to their own hand history databases. The platform's architecture had made such abuse not merely possible but trivially easy for any insider with database access.

The Full Tilt Poker Ponzi Scheme (2011). On April 15, 2011 — a date the poker community refers to as "Black Friday" — the United States Department of Justice unsealed indictments against the operators of the three largest online poker sites serving U.S. players: PokerStars, Full Tilt Poker, and Cereus (the parent company of Ultimate Bet) [1]. The subsequent investigation revealed that Full Tilt Poker had been operating as a de facto Ponzi scheme: the company had distributed approximately $440 million in player deposits to its board members and professional endorsers as dividend payments, leaving the site unable to honor withdrawal requests. Players' funds existed only as database entries with no corresponding reserves. The platform's complete custody over player balances had enabled a misappropriation of historic proportions.

Systematic Bot Rings. Beyond these headline scandals, the industry contends with an endemic problem of unauthorized bot operations. Sophisticated bot rings — networks of algorithmically controlled accounts — have been documented on virtually every major platform, from PokerStars to GGPoker to Winamax. These bots share hand information in real time, coordinate play across multiple seats at the same table, and exploit the platform's inability to verify that each account corresponds to an independent human decision-maker. The platforms' response has been reactive and opaque: accounts are occasionally banned, funds sometimes confiscated, but the adjudication process is entirely internal, with no mechanism for independent verification.

The common thread across these failures is architectural, not moral. The operators were not uniquely corrupt; they operated within a system that concentrated extraordinary power in a single entity with minimal accountability. The question is not whether such systems will be abused, but when.

2.2 Why Existing Solutions Fail

The blockchain revolution has prompted numerous attempts to bring transparency and trustlessness to online poker. These efforts fall into several categories, none of which adequately addresses the core problem.

"Provably Fair" Server-Side RNG. Platforms such as Stake.com and similar crypto-native gambling sites employ a commit-reveal scheme in which the server commits to a random seed before the hand, and reveals it afterward for verification. While this creates an audit trail, it does not prevent real-time cheating. The server still generates the deck permutation, still observes all cards, and still could selectively deal favorable hands to colluding accounts. The "provably fair" label refers only to the generation of the random seed, not to the fairness of the entire dealing pipeline. A compromised server can choose from among many possible seeds to find one that produces a desired outcome, particularly if the commitment scheme admits precomputation.

On-Chain Betting with Off-Chain Dealing. A more common approach places the wagering logic on a blockchain — typically Ethereum or a sidechain — while retaining a centralized server for card dealing and game orchestration. Projects in this category use smart contracts to manage buy-ins, pot calculations, and payouts, but the server still functions as the trusted dealer. The blockchain provides fund custody guarantees but does nothing to address the information asymmetry at the heart of the

trust problem. If the server is compromised, it can deal cards to favor specific players, and the on-chain settlement layer will faithfully execute the fraudulent outcome.

Existing Mental Poker Implementations. A small number of projects, most notably Virtue Poker, have attempted to implement mental poker protocols on blockchain. However, Virtue Poker targeted Ethereum Layer 1, where 12-second block times and gas costs exceeding $50 per transaction render real-time poker play economically impractical. The fundamental tension is between cryptographic rigor and user experience: a protocol that requires multiple on-chain transactions per hand cannot sustain the pace of play that poker demands. This has confined mental poker to the realm of academic curiosity rather than practical deployment.

2.3 The TrueShuffle Approach

TrueShuffle resolves this tension by eliminating the trusted dealer entirely while deploying on infrastructure capable of supporting real-time play. The protocol's central insight derives from a property of elliptic curve scalar multiplication that has been known since the foundational work of Shamir, Rivest, and Adleman on mental poker [6], later refined by Barnett and Smart using elliptic curve cryptography [7]:

> If each player encrypts a card by multiplying the corresponding curve point by their secret scalar, then the resulting ciphertext is independent of the order of encryption. That is, $s_1 \cdot (s_2 \cdot C) = s_2 \cdot (s_1 \cdot C)$ for any card point C and secret scalars s_1, s_2.

This commutativity property enables a protocol in which players collectively encrypt and shuffle a deck such that (a) the deck is a uniformly random permutation from the perspective of any proper subset of players, and (b) individual cards can be selectively decrypted for their intended recipients without revealing information to other parties. The server functions as a message relay and state coordinator but never possesses the cryptographic keys required to decrypt any card. Even a fully compromised server learns nothing about the deck ordering or any player's hole cards.

TrueShuffle deploys this protocol on MegaETH [4], a high-performance EVM-compatible Layer 2 network offering approximately 10-millisecond transaction finality and sub-cent gas costs. This combination — cryptographic trustlessness on fast, cheap infrastructure — makes real-time mental poker economically and experientially viable for the first time.

2.4 Contributions

This paper makes the following contributions:

1. **First production-grade mental poker on an EVM L2.** We present a complete system implementation comprising TypeScript client libraries (`@noble/curves` Secp256k1 arithmetic), Solidity smart contracts (on-chain shuffle verification via `ecMul`, deck hash chaining, card key commitment and verification, showdown card verification), and a WebSocket-based coordination layer. The system supports 2–10 player Texas Hold'em with full hand evaluation.
2. **Commutative EC encryption on Secp256k1.** We employ the Ethereum-native Secp256k1 curve for all mental poker cryptographic operations, enabling direct on-chain verification of shuffle correctness, lock-step integrity, and card key commitments using native elliptic curve

arithmetic in Solidity. Card points are generated via deterministic hash-to-curve, ensuring that discrete logarithm relationships between card points are computationally unknowable.

3. **Complete on-chain settlement with dispute resolution.** The smart contract system includes escrow management, atomic pot distribution, deck hash chaining for tamper detection, card key commitment-reveal verification, and a cryptographic proof-of-cheating mechanism (`abortAndSlash`) that enables any player to trigger escrow forfeiture of a cheating participant by referencing on-chain verification failures.
4. **Autonomous bot framework.** We provide a complete SDK for building autonomous poker agents that participate in the mental poker protocol, including cryptographic key generation, shuffle and lock operations, E2E-encrypted card key exchange, and strategy execution. Bots are parameterized by DNA-based strategy configurations supporting adversarial learning and continuous competitive play.
5. **Formal security analysis.** We prove that the protocol satisfies fairness (no coalition of fewer than m players can determine any unrevealed card), completeness (honest execution always succeeds), and verifiability (any observer can audit correctness from the on-chain transcript), under the Elliptic Curve Discrete Logarithm assumption on Secp256k1.

2.5 Paper Organization

The remainder of this paper is organized as follows. Chapter 2 surveys the background and related work, covering the history of mental poker protocols, the relevant elliptic curve cryptography foundations, existing blockchain poker platforms, and the smart contract infrastructure requirements for real-time applications. Chapter 3 presents the formal protocol specification, including the state machine, all cryptographic phases, and proofs of the core security properties. Chapter 4 provides the security analysis and threat model. Subsequent chapters address anti-collusion mechanisms, system architecture, on-chain settlement, game-theoretic considerations, the autonomous bot framework, scalability analysis, token economics, governance, regulatory compliance, and future research directions.

3 Background and Related Work

3.1 Mental Poker: A Half-Century of Theory

The problem of playing a fair card game without a trusted dealer was first posed by Shamir, Rivest, and Adleman (1979), later published in [6]. Their seminal formulation — universally known as the SRA protocol — established the theoretical foundations upon which all subsequent mental poker research builds. The insight was elegant: if the players employ a commutative encryption scheme, they can collectively encrypt, shuffle, and selectively decrypt a deck of cards without any single party gaining unauthorized information about the deck ordering.

The SRA protocol operated as follows. Let $E_k(\cdot)$ denote encryption with key k, and let the commutativity property hold: $E_{k_1}(E_{k_2}(m)) = E_{k_2}(E_{k_1}(m))$ for all messages m and keys k_1, k_2. The original construction used modular exponentiation as the commutative cipher: $E_k(m) = m^k \bmod p$ for a large prime p. Each player encrypts every card in the deck with their secret key, permutes the encrypted deck, and passes it to the next player. After all players have encrypted and shuffled, each card is encrypted under all players' keys simultaneously, and the deck's ordering is a composition of all players'

secret permutations. To deal a card to a specific player, all other players decrypt their layer from that card, leaving only the target player's encryption, which they then remove to read the card.

While theoretically sound, the SRA protocol suffered from practical limitations. The modular exponentiation cipher, while commutative, required careful parameter selection to avoid attacks based on the multiplicative structure of $\mathbb{Z}_p^*$. The protocol also demanded $O(mn)$ public-key operations for m players and n cards, which was computationally prohibitive on hardware of the era.

Barnett and Smart revisited the problem in 2003, proposing an elliptic curve-based construction that substantially improved both security and efficiency [7]. Their protocol replaced modular exponentiation with elliptic curve scalar multiplication, gaining the benefits of smaller key sizes (256-bit keys providing 128-bit security versus 3072-bit RSA keys for equivalent strength) and faster group operations. The commutativity property in the elliptic curve setting takes the form:

$$s_1 \cdot (s_2 \cdot P) = s_2 \cdot (s_1 \cdot P) = (s_1 \cdot s_2) \cdot P$$

for any point P on the curve and scalars $s_1, s_2 \in \mathbb{Z}_n^*$, where n is the order of the curve's base point. This property follows directly from the associativity of scalar multiplication in the elliptic curve group, requiring no additional algebraic structure beyond what the group itself provides.

Subsequent work by Castella-Roca et al. [8] extended mental poker to multi-party settings and formalized the security model in terms of standard cryptographic definitions. However, all of these contributions remained academic: they produced proofs of concept, complexity analyses, and security reductions, but never production systems capable of sustaining real-time play at scale. The gap between theoretical feasibility and practical deployment — a gap of computational overhead, network latency, and settlement infrastructure — persisted for over two decades.

TrueShuffle bridges this gap by combining the Barnett-Smart elliptic curve approach with modern high-performance blockchain infrastructure, producing the first mental poker system that operates at the speed and cost required for real poker.

3.2 Elliptic Curve Cryptography Foundations

TrueShuffle's cryptographic operations are performed on the Secp256k1 elliptic curve, chosen for its native compatibility with the Ethereum ecosystem and the availability of on-chain verification primitives.

3.2.1 The Secp256k1 Curve

Secp256k1 is defined by the short Weierstrass equation [3]:

$$y^2 = x^3 + 7 \quad \text{over } \mathbb{F}_p$$

where the field prime is:

$$p = 2^{256} - 2^{32} - 977$$

or equivalently, $p = 2^{256} - 4294968273$. The curve has a prime-order subgroup of order:

$$n = \texttt{0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141}$$

with designated generator (base point) $G = (G_x, G_y)$ specified in the SEC 2 standard. The cofactor is 1, meaning every non-identity point on the curve is a generator of the full group $\mathbb{G}$ of order n.

3.2.2 The Elliptic Curve Discrete Logarithm Problem

The security of TrueShuffle rests on the computational hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP):

> **Definition (ECDLP).** Given points $P, Q \in \mathbb{G}$ where $Q = k \cdot P$ for some unknown scalar $k \in \mathbb{Z}_n^*$, compute k.

Secp256k1 possesses a well-known GLV endomorphism that enables efficient scalar multiplication. While this endomorphism can accelerate Pollard's rho algorithm by a factor of approximately $\sqrt{3}$, reducing the effective security from approximately 128 bits to approximately 127 bits, it does not fundamentally weaken the ECDLP hardness assumption. The protocol retains a security margin well above the 112-bit threshold recommended by NIST for near-term applications. The best known algorithms for ECDLP on Secp256k1 remain generic group algorithms — principally Pollard's rho method — which run in $O(\sqrt{n})$ time.

3.2.3 Scalar Multiplication as Encryption

In TrueShuffle, encryption and decryption of card points are defined as scalar multiplication operations:

$$\text{Enc}(C, s) = s \cdot C$$

$$\text{Dec}(E, s) = s^{-1} \cdot E$$

where s^{-1} denotes the modular multiplicative inverse of s modulo n, computed via the extended Euclidean algorithm. The correctness of decryption follows immediately:

$$\text{Dec}(\text{Enc}(C, s), s) = s^{-1} \cdot (s \cdot C) = (s^{-1} \cdot s) \cdot C = 1 \cdot C = C$$

The critical property enabling the mental poker protocol is commutativity:

$$\text{Enc}(\text{Enc}(C, s_1), s_2) = s_2 \cdot (s_1 \cdot C) = (s_2 \cdot s_1) \cdot C = (s_1 \cdot s_2) \cdot C = s_1 \cdot (s_2 \cdot C) = \text{Enc}(\text{Enc}(C, s_2), s_1)$$

This means that multiple players can encrypt a card in any order, and the resulting ciphertext is the same regardless of the encryption sequence. More importantly, any player can remove their encryption layer without requiring the other players to remove theirs first — the decryption order is unconstrained.

3.2.4 Card Point Generation

A standard 52-card deck must be mapped to 52 distinct points on the Secp256k1 curve such that the discrete logarithm relationship between any pair of card points is computationally unknowable. TrueShuffle achieves this via deterministic hash-to-curve: for each card i with canonical name ℓ_i (e.g., `"molt-poker-card-Ah"` for the ace of hearts), the corresponding curve point is:

$$C_i = H_{\text{scalar}}(\ell_i) \cdot G$$

where $H_{\text{scalar}}(\ell_i) = \text{SHA-256}(\ell_i) \bmod n$ and G is the Secp256k1 generator. Since the discrete logarithm of each C_i with respect to G is the output of SHA-256 on a distinct input, and SHA-256 is modeled as a random oracle, the discrete logarithm between any two card points C_i and C_j — i.e., the scalar α such that $C_j = \alpha \cdot C_i$ — is computationally unknowable. This property is essential: if a player could compute α, they could determine which card a ciphertext corresponds to without possessing the decryption keys.

3.3 X25519 Key Exchange for End-to-End Encrypted Communication

During the deal phase of the protocol, players must transmit per-card decryption keys to specific recipients. If these keys were transmitted in cleartext through the coordinating server, the server could observe them and decrypt cards, defeating the purpose of the mental poker protocol.

TrueShuffle addresses this by establishing pairwise end-to-end encrypted channels between all players using X25519 Diffie-Hellman key agreement [5] combined with XSalsa20-Poly1305 authenticated encryption. At session initialization, each player generates an X25519 keypair (distinct from their Secp256k1 mental poker keys) and publishes the public key to all other participants. The public key is committed during the `KEY_COMMIT` phase to prevent substitution attacks.

Curve25519 operates over the prime field $\mathbb{F}_q$ where $q = 2^{255} - 19$, using the Montgomery curve $v^2 = u^3 + 486662u^2 + u$. The X25519 function computes the u-coordinate of the shared secret point from the scalar multiplication of one party's secret key with the other party's public key. The resulting 32-byte shared secret is used as the symmetric key for XSalsa20-Poly1305, which provides both confidentiality (via the XSalsa20 stream cipher with a 192-bit nonce) and integrity (via the Poly1305 MAC). This construction provides approximately 128 bits of security against both passive eavesdroppers and active tampering.

The choice of X25519 over Secp256k1-based ECDH for the communication channel is deliberate: X25519 is purpose-built for key agreement with constant-time implementations that resist side-channel attacks, and its use creates a clean separation between the mental poker encryption keys (Secp256k1 scalars, which are used for on-chain verification) and the communication encryption keys (Curve25519, which are used only for ephemeral peer-to-peer channels).

3.4 Existing Blockchain Poker Platforms

Several blockchain-based poker platforms have been developed, each making different trade-offs between decentralization, performance, and security. Table 1 summarizes the key differences.

Table 1: Comparison of blockchain poker platforms.

Platform	Card Dealing	Settlement	Latency	Cost per Hand	Verification
Virtue Poker	Mental poker (EC)	Ethereum L1	~12s blocks	$50–200+ gas	On-chain shuffle proof
CoinPoker	Centralized RNG	CHP token (ERC-20)	~2s (custom chain)	Low	RNG seed published post hoc
PokerDAO	Centralized server	DAO treasury	Variable	Moderate	Governance vote
Blockchain Poker	Centralized server	Bitcoin (on-chain)	~10min (BTC)	BTC fees	None (trust-based)
TrueShuffle	**Mental poker (Secp256k1 EC)**	**MegaETH L2**	**~10ms**	**<$0.01**	**On-chain EC verification**

Virtue Poker represents the closest prior work to TrueShuffle. Developed from approximately 2016–2020, Virtue Poker implemented a mental poker protocol using elliptic curve cryptography and deployed settlement contracts on Ethereum mainnet. However, the platform was constrained by Ethereum L1's fundamental limitations: 12-second block times made each protocol phase transition agonizingly slow, and gas costs of $50–200 per transaction (at 2021 gas prices) rendered the economic model unsustainable for the small-to-medium stakes games that constitute the majority of online poker volume. The project ultimately failed to achieve meaningful adoption, demonstrating that cryptographic soundness alone is insufficient — the infrastructure must also meet the performance requirements of real-time gameplay.

CoinPoker adopted a different approach, using a proprietary blockchain for payments but retaining a centralized server for card dealing. The platform publishes random number generation seeds after each hand, enabling retrospective verification, but the server possesses complete knowledge of all cards during play. This architecture provides stronger financial transparency than traditional platforms but does not address the fundamental information asymmetry of centralized dealing.

PokerDAO and similar governance-focused projects bring decentralized governance structures (DAOs) to the platform management layer — deciding rake structures, resolving disputes, managing treasury — but retain conventional centralized servers for the actual poker gameplay. The dealing server remains a single point of trust compromise.

The consistent pattern across these platforms is a failure to address the dealing problem. Most blockchain poker projects apply decentralization technology to the financial layer (payments, escrow, governance) while leaving the information layer (who knows which cards) entirely centralized. TrueShuffle is, to our knowledge, the first platform to achieve both financial settlement and card dealing trustlessness on infrastructure fast enough for real-time play.

3.5 Smart Contract Platforms for Real-Time Applications

The feasibility of on-chain mental poker depends critically on the performance characteristics of the underlying smart contract platform. Each phase transition in the mental poker protocol requires at least one on-chain transaction (for commitment storage, hash chaining, or verification), and the overall protocol must complete within a timeframe acceptable for real-time poker — ideally under 10 seconds for all pre-deal cryptographic phases.

Ethereum Mainnet (L1) produces blocks approximately every 12 seconds with gas costs that fluctuate between 10 and 200+ gwei, yielding per-transaction costs of $1–100+ depending on computational complexity [9], [10]. A mental poker protocol requiring 5–10 on-chain transactions per hand would cost $10–1000 per hand and take 1–2 minutes to complete the cryptographic phases alone. This is flatly incompatible with real-time poker, where hands at a 6-player table should complete in 1–3 minutes including all betting rounds.

Optimistic Rollups (Arbitrum, Optimism) reduce gas costs substantially but introduce 7-day challenge periods for withdrawals and effective confirmation latencies of 250ms–2s. While adequate for many DeFi applications, the confirmation latency adds meaningful overhead to a multi-round protocol, and the 7-day withdrawal delay is unacceptable for a poker platform where players expect prompt access to winnings.

ZK-Rollups (zkSync, StarkNet, Scroll) offer faster finality (minutes to hours for hard finality, seconds for soft finality) and lower costs, but at the time of writing, most ZK-rollup platforms impose significant constraints on smart contract complexity and do not natively support the Secp256k1 elliptic curve precompiles required for efficient on-chain shuffle verification.

MegaETH [4] represents a new category of EVM-compatible Layer 2 designed specifically for real-time applications. Key architectural properties include:

- **Block time:** approximately 10 milliseconds, enabling near-instantaneous transaction confirmation.
- **Gas costs:** sub-cent per transaction for typical contract interactions, making multi-transaction protocols economically viable even for micro-stakes games.
- **EVM compatibility:** full support for Solidity smart contracts, including the `ecMul` and `ecAdd` precompiles required for on-chain Secp256k1 arithmetic.
- **Contract size:** support for contracts up to 512KB, accommodating the substantial verification logic required for mental poker (shuffle permutation verification, lock-step verification, card key commitment, showdown card verification).

These properties make MegaETH the first smart contract platform on which a complete mental poker protocol — with full on-chain verification of every cryptographic phase — is economically and temporally feasible. The 10ms block time means that a 10-transaction protocol phase completes in approximately 100ms of on-chain latency, dominated by network round-trip times rather than block production. The sub-cent gas costs mean that the total on-chain cost of a complete hand, including all commitments, hash chain entries, and showdown verification, remains under $0.10 — comparable to the rake charged by conventional poker platforms and negligible relative to the stakes of any meaningful game.

4 Protocol Specification

This chapter presents the complete formal specification of the TrueShuffle mental poker protocol. We define the cryptographic primitives, the protocol state machine, each phase in detail, and prove the core security properties.

4.1 Notation and Primitives

Let $\mathbb{G}$ denote the Secp256k1 elliptic curve group of prime order n, with generator point G. We write $\mathbb{Z}_n^*$ for the multiplicative group of integers modulo n (i.e., $\{1, 2, \ldots, n-1\}$). All arithmetic on scalars is performed modulo n unless stated otherwise.

Players. A game involves m players $P_1, P_2, \ldots, P_m$ where $2 \leq m \leq 10$. Each player P_i is identified by an Ethereum address addr_i and possesses a Secp256k1 keypair for the mental poker protocol (distinct from their Ethereum signing keys).

Deck. A standard 52-card deck is represented as an ordered sequence of distinct curve points:

$$\mathscr{D}_0 = (C_1, C_2, \ldots, C_{52}) \in \mathbb{G}^{52}$$

where each C_j is derived via deterministic hash-to-curve:

$$C_j = H_s(\ell_j) \cdot G, \quad H_s(\ell) = \text{SHA-256}(\ell) \bmod n$$

and ℓ_j is the canonical label for card j (e.g., `"molt-poker-card-Ah"`). The mapping from card points back to card identities is public and deterministic: given a decrypted point P, the card identity is recovered by searching the precomputed table $\{C_1, \ldots, C_{52}\}$ for a match.

Encryption and Decryption. For a point $C \in \mathbb{G}$ and scalar $s \in \mathbb{Z}_n^*$:

$$\text{Enc}(C, s) = s \cdot C \in \mathbb{G}$$

$$\text{Dec}(E, s) = s^{-1} \cdot E \in \mathbb{G}$$

where s^{-1} is the modular multiplicative inverse of s modulo n, satisfying $s \cdot s^{-1} \equiv 1 \pmod{n}$.

Commutativity. For any scalars $s_1, s_2 \in \mathbb{Z}_n^*$ and point $C \in \mathbb{G}$:

$$\text{Enc}(\text{Enc}(C, s_1), s_2) = s_2 \cdot (s_1 \cdot C) = (s_2 \cdot s_1) \cdot C = (s_1 \cdot s_2) \cdot C = s_1 \cdot (s_2 \cdot C) = \text{Enc}(\text{Enc}(C, s_2), s_1)$$

This property is the algebraic foundation of the entire protocol.

Hash Functions. We use $H : \{0,1\}^* \to \{0,1\}^{256}$ to denote the Keccak-256 hash function (as used in Ethereum for on-chain operations) and $H_s : \{0,1\}^* \to \mathbb{Z}_n$ to denote SHA-256 reduced modulo n (used for card point generation and shuffle seeding).

Permutations. A permutation π on $\{1, \ldots, 52\}$ is a bijection $\pi : \{1, \ldots, 52\} \to \{1, \ldots, 52\}$. We write $\pi(j)$ for the image of j under π, and $\pi_1 \circ \pi_2$ for the composition of permutations (applying π_2 first, then π_1).

4.2 Formal State Machine

The protocol is modeled as a deterministic finite state machine with the following states and transitions:

```
+------+------+
```

> **Note:** Any state may transition to ABORTED if a phase deadline expires or a verification check fails. See Abort Conditions below.

Each state transition is governed by a **guard condition** that must be satisfied before the transition fires. The on-chain coordinator contract (`MentalPokerCoordinator`) enforces strictly sequential phase advancement — no phase may be skipped — and maintains a **phase deadline** (default 60 seconds) after which any registered player may challenge the coordinator and trigger rotation.

State Invariants:

- *I1 (Commitment Integrity):* Once a commitment hash h_i is recorded on-chain for player P_i, it is immutable for the duration of the game.
- *I2 (Sequential Shuffle):* Players shuffle in a fixed order $(P_1, P_2, \ldots, P_m)$ determined at registration. Each player's shuffle operates on the output of the previous player's shuffle.
- *I3 (Deck Hash Chain):* After each shuffle and lock step, the hash of the resulting deck state is recorded on-chain, forming an immutable chain: $h_{\text{init}} \to h_{\text{shuffle}_1} \to \cdots \to h_{\text{shuffle}_m} \to h_{\text{lock}_1} \to \cdots \to h_{\text{lock}_m}$. The input hash of each step must equal the output hash of the preceding step.
- *I4 (Point Uniqueness):* At every stage of the protocol, the deck contains exactly 52 distinct curve points. This is verified after each shuffle and lock step.

Abort Conditions. The protocol transitions to the ABORTED state if:

- Any commitment verification fails during KEY_REVEAL.
- Any shuffle produces a deck with duplicate points (violation of I4).
- A phase deadline expires without completion (timeout).
- A player provides a card key that does not match its on-chain commitment.

Upon abort, all buy-in deposits are returned to players unless a specific cheating player is identified via cryptographic proof, in which case their deposit is forfeited (slashed) and distributed to honest participants.

4.3 Phase 1: Key Commitment

At the beginning of a hand, each player P_i generates their cryptographic material for the upcoming protocol execution.

Key Generation. Player P_i samples:

- A deck encryption key: $s_i \xleftarrow{\$} \mathbb{Z}_n^*$
- Fifty-two per-card encryption keys: $k_{i,1}, k_{i,2}, \ldots, k_{i,52} \xleftarrow{\$} \mathbb{Z}_n^*$
- An X25519 communication keypair: $(\text{sk}_i^{\text{comms}}, \text{pk}_i^{\text{comms}}) \leftarrow \text{X25519.KeyGen}()$

All random sampling uses a cryptographically secure pseudorandom number generator.

Commitment Computation. Player P_i computes a binding commitment to their public key material:

$$h_i = H\big(Q_i \parallel H(k_{i,1}) \parallel H(k_{i,2}) \parallel \cdots \parallel H(k_{i,52}) \parallel \text{pk}_i^{\text{comms}}\big)$$

where $Q_i = s_i \cdot G$ is the player's public deck key, $H(k_{i,j})$ are commitments to the per-card keys, and $\|$ denotes concatenation of the hexadecimal representations separated by a delimiter. Note that the secret scalar s_i and the per-card keys $k_{i,j}$ are never included directly in the commitment; only their public derivatives appear. This ensures that the commitment can be verified without revealing secret key material.

In the **coordinated variant** (optional optimization), the commitment instead takes the form $h_i = H(s_i \,\|\, k_{i,1} \,\|\, \cdots \,\|\, k_{i,52} \,\|\, \mathrm{pk}_i^{\mathrm{comms}})$, committing to the raw secret scalars directly, since these will be revealed to the server. **Warning:** this variant requires trusting the coordinating server with secret key material.

Broadcast. Player P_i sends h_i to the coordinator, which records it on-chain via `storeKeyCommitment(gameId, addr_i, h_i)`. The coordinator also records a separate commitment for the communication public key: $h_i^{\mathrm{comms}} = H(\mathrm{pk}_i^{\mathrm{comms}})$.

Transition Guard. The state advances from `KEY_COMMIT` to `KEY_REVEAL` when commitments have been received from all m players:

$$|\{i : h_i \text{ recorded}\}| = m$$

Purpose. The commitment phase prevents adaptive key selection. Because all commitments are recorded before any keys are revealed, no player can choose their key based on knowledge of other players' keys. This is essential for preventing a class of attacks in which a player selects a key that is algebraically related to another player's key in a way that leaks information about the deck permutation.

4.4 Phase 2: Key Reveal

After all commitments are recorded, each player reveals their public key material.

Reveal. Player P_i broadcasts:

- Their public deck key: $Q_i = s_i \cdot G$
- Per-card key commitments: $H(k_{i,j})$ for $j = 1, \ldots, 52$
- Their communication public key: $\mathrm{pk}_i^{\mathrm{comms}}$

In the **coordinated variant** (optional optimization), P_i may additionally reveal the full key set $(s_i, k_{i,1}, \ldots, k_{i,52})$ to the server. This enables the server to perform shuffle and lock operations on the player's behalf, reducing client-side computation, but requires trusting the server with secret key material. See the On-Chain Record paragraph below for further discussion of this tradeoff.

Verification (Trustless Default). The coordinator and every participant verify:

1. **Commitment match:** $H(Q_i \,\|\, H(k_{i,1}) \,\|\, \cdots \,\|\, H(k_{i,52}) \,\|\, \mathrm{pk}_i^{\mathrm{comms}}) = h_i$
2. **Public key well-formedness:** Q_i is a valid point on Secp256k1 (i.e., $Q_i \in \mathbb{G} \setminus \{\mathcal{O}\}$)
3. **Communication key commitment match:** $H(\mathrm{pk}_i^{\mathrm{comms}}) = h_i^{\mathrm{comms}}$

Note that only public information (Q_i, per-card key hashes, communication public key) is verified. The secret scalar s_i and the per-card keys $k_{i,j}$ are never revealed to the coordinator or other players at this

stage. This is the **trustless (decentralized) default** and ensures that the server never possesses the cryptographic material needed to decrypt any card.

If any verification fails, the protocol aborts and all buy-ins are returned.

Verification (Coordinated Variant). In the optional coordinated variant, the server additionally receives the full key set $(s_i, k_{i,1}, \ldots, k_{i,52})$ and verifies the stronger commitment $H(s_i \parallel k_{i,1} \parallel \cdots \parallel k_{i,52} \parallel \mathrm{pk}_i^{\mathrm{comms}}) = h_i$ along with key validity checks ($s_i \in \mathbb{Z}_n^*$ and $k_{i,j} \in \mathbb{Z}_n^*$ for all j). **Warning:** this variant requires trusting the coordinating server with secret key material. A compromised server in the coordinated variant can decrypt cards and violate player privacy. The coordinated variant is appropriate only when latency reduction is prioritized over trust minimization (e.g., casual low-stakes games with a trusted operator).

Transition Guard. The state advances from KEY_REVEAL to SHUFFLE when all m key reveals have been verified:

$$\forall i \in \{1, \ldots, m\} : \mathrm{Verify}(Q_i, \{H(k_{i,j})\}_{j=1}^{52}, \mathrm{pk}_i^{\mathrm{comms}}, h_i) = \texttt{true}$$

4.5 Phase 3: Encrypt and Shuffle

The shuffle phase produces a uniformly random permutation of the encrypted deck, such that no single player (and no proper subset of players) knows the resulting ordering.

Protocol. The shuffle proceeds sequentially through all m players. Let $\mathscr{D}_0 = (C_1, \ldots, C_{52})$ denote the initial (public) deck.

For each player P_i in order $i = 1, 2, \ldots, m$:

1. P_i receives the current deck $\mathscr{D}_{i-1} = (D_1^{(i-1)}, \ldots, D_{52}^{(i-1)})$.
2. P_i encrypts every card with their deck key: $E_j^{(i)} = s_i \cdot D_j^{(i-1)}$ for $j = 1, \ldots, 52$.
3. P_i selects a random permutation $\pi_i \xleftarrow{\$} S_{52}$ (implemented as a Fisher-Yates shuffle seeded by a cryptographically random value).
4. P_i applies the permutation: $D_j^{(i)} = E_{\pi_i(j)}^{(i)}$ for $j = 1, \ldots, 52$.
5. P_i outputs $\mathscr{D}_i = (D_1^{(i)}, \ldots, D_{52}^{(i)})$.

In TrueShuffle's implementation, the Fisher-Yates shuffle uses a deterministic pseudorandom permutation derived from a seed σ_i:

$$\pi_i = \mathrm{FisherYates}(\sigma_i)$$

where successive random indices are generated by iterated hashing: $r_j = H_s(r_{j-1})$ with $r_0 = \sigma_i$.

After all players shuffle. The final shuffled deck is:

$$\mathscr{D}_m[j] = \left(\prod_{i=1}^{m} s_i \right) \cdot C_{\pi(j)}$$

where $\pi = \pi_m \circ \pi_{m-1} \circ \cdots \circ \pi_1$ is the composed permutation. Each card point is encrypted under the

product of all players' deck keys, and the deck ordering is determined by the composition of all players' secret permutations.

Permutation Commitment. After shuffling, each player computes and broadcasts a commitment to their permutation and seed:

$$c_i^{\pi} = H(\pi_i \parallel \sigma_i \parallel s_i)$$

This commitment is recorded on-chain via `commitShuffle(gameId, addr_i, c_i^{\pi})`. It enables post-hoc verification: at showdown or during dispute resolution, a player can reveal (π_i, σ_i, s_i) and any observer can verify c_i^{π} and recompute the shuffle to confirm correctness.

Deck Hash Chain. Before the first shuffle, the coordinator records $h_{\text{init}} = H(\mathscr{D}_0)$ on-chain. After each player P_i's shuffle, the coordinator extends the chain:

$$\text{extendDeckHashChain}(\text{gameId}, \text{addr}_i, H(\mathscr{D}_{i-1}), H(\mathscr{D}_i), \texttt{SHUFFLE})$$

The contract verifies that the input hash matches the previous entry's output hash, ensuring chain continuity and preventing substitution of intermediate deck states.

Blind Shuffle (Trustless Variant). In the fully decentralized mode, each player performs the shuffle locally and submits only the resulting encrypted deck and the permutation commitment to the coordinator. The coordinator never sees the deck key s_i or the permutation π_i. The `submitShuffleBlind` function accepts the encrypted deck points directly and verifies only that the deck contains 52 distinct points.

Key Property. After the shuffle phase, no single player P_i knows the composed permutation π. Player P_i knows only their own permutation π_i and the deck as it existed after their shuffle step. Since each subsequent player re-encrypts and permutes, the relationship between card positions and card identities is hidden by the unknown permutations of all other players.

4.6 Phase 4: Lock (Deck-Level to Per-Card Keys)

The lock phase replaces each player's deck-level encryption with per-card encryption, enabling selective decryption of individual cards during the deal phase.

Protocol. The lock proceeds sequentially through all m players in the same order as the shuffle. Let $\mathscr{D}_m$ denote the fully shuffled deck.

For each player P_i in order $i = 1, 2, \ldots, m$:

1. P_i receives the current deck $\mathscr{L}_{i-1}$ (where $\mathscr{L}_0 = \mathscr{D}_m$).
2. For each card position $j = 1, \ldots, 52$, P_i performs:

$$\mathscr{L}_i[j] = k_{i,j} \cdot s_i^{-1} \cdot \mathscr{L}_{i-1}[j]$$

That is, P_i removes their deck-level encryption (s_i^{-1}) and applies their per-card key ($k_{i,j}$).

After all players lock. The final locked deck satisfies:

$$\mathscr{L}_m[j] = \left(\prod_{i=1}^{m} k_{i,j} \right) \cdot C_{\pi(j)}$$

Each card at position j is now encrypted under the product of all players' per-card keys for that specific position, applied to the original card point at the permuted position $\pi(j)$.

On-Chain Checkpoints. During the lock phase, two critical on-chain records are created:

1. **Deck hash chain extension:** After each player's lock step, $H(\mathscr{L}_i)$ is appended to the on-chain chain with `stepType = LOCK`.
2. **Card key commitments:** Each player's per-card keys are committed on-chain as:

$$\text{commit}_{i,j} = H(k_{i,j}) \quad \text{for all } j \in \{1, \ldots, 52\}$$

   These commitments are stored via `storeCardKeyCommitments(gameId, addr_i, positions, commitments)` and serve as binding references for the deal and showdown phases.

On-Chain Lock Verification. The correctness of each player's lock step can be verified on-chain by the `verifyLockStep` function, which checks:

$$\forall j \in \{1, \ldots, 52\} : \quad k_{i,j} \cdot s_i^{-1} \cdot \mathscr{L}_{i-1}[j] = \mathscr{L}_i[j]$$

Equivalently, using the formulation that avoids transmitting s_i^{-1}:

$$\forall j \in \{1, \ldots, 52\} : \quad s_i \cdot \mathscr{L}_i[j] = k_{i,j} \cdot \mathscr{L}_{i-1}[j]$$

which the contract verifies via two `ecMul` operations per card and a point equality check.

Immutability. After the lock phase completes, the locked deck $\mathscr{L}_m$ is finalized and its hash is recorded on-chain. No card may be dealt from a deck whose hash does not match this recorded value. This ensures that the deck state is immutable from the moment locking completes until all cards have been dealt and verified.

4.7 Phase 5: Deal (E2E Key Exchange)

The deal phase reveals specific cards to their intended recipients while maintaining confidentiality against all other parties, including the coordinating server.

Card Position Assignment. The coordinator assigns deck positions to game roles:

- **Hole cards:** For each player P_t, positions $\text{hole}_t = \{j_{t,1}, j_{t,2}\}$ (two cards for Texas Hold'em).
- **Community cards:** Positions $\text{comm} = \{j_{\text{flop}_1}, j_{\text{flop}_2}, j_{\text{flop}_3}, j_{\text{turn}}, j_{\text{river}}\}$.
- **Burn cards:** Positions $\text{burn} = \{j_{b_1}, j_{b_2}, j_{b_3}\}$.

The assignment is committed on-chain before shuffling begins (via `commitDealPositions`) and revealed after shuffling (via `revealDealPositions`), preventing the coordinator from adaptively choosing positions after observing the shuffle.

Hole Card Reveal. To reveal card at position j to target player P_t:

1. Each other player P_i (where $i \neq t$) retrieves their per-card key $k_{i,j}$ for position j.
2. P_i encrypts $k_{i,j}$ using X25519 E2E encryption targeted at P_t:

$$\mathrm{ct}_{i \to t} = \mathrm{XSalsa20Poly1305}\big(\mathrm{X25519}(\mathrm{sk}_i^{\mathrm{comms}}, \mathrm{pk}_t^{\mathrm{comms}}), \mathrm{nonce}, k_{i,j}\big)$$

3. The ciphertext $\mathrm{ct}_{i \to t}$ is relayed through the coordinator to P_t. The coordinator cannot decrypt it because it does not possess $\mathrm{sk}_t^{\mathrm{comms}}$ or $\mathrm{sk}_i^{\mathrm{comms}}$.
4. P_t decrypts each received ciphertext to obtain $k_{i,j}$ for all $i \neq t$.
5. P_t decrypts the card:

$$C_{\pi(j)} = k_{t,j}^{-1} \cdot \left(\prod_{i \neq t} k_{i,j}^{-1} \right) \cdot \mathscr{L}_m[j]$$

   which simplifies, by the definition of the locked deck, to:

$$C_{\pi(j)} = \left(\prod_{i=1}^{m} k_{i,j} \right)^{-1} \cdot \left(\prod_{i=1}^{m} k_{i,j} \right) \cdot C_{\pi(j)} = C_{\pi(j)}$$

6. P_t looks up $C_{\pi(j)}$ in the card point table to identify the card.

Community Card Reveal. Community cards are revealed to all players simultaneously. Each player P_i broadcasts their per-card key $k_{i,j}$ for the community card position j to all other players, encrypted individually via their pairwise X25519 channels. Once all keys are collected, every player can independently decrypt the community card.

Privacy Guarantee. Only the target player P_t receives the per-card keys from all other players for their hole card positions. No other player receives these keys, and the coordinator (which relays the ciphertexts) cannot decrypt them. Therefore, only P_t can determine the identity of their hole cards.

4.8 Phase 6: Betting

The betting phase implements standard No-Limit Texas Hold'em across four betting rounds, with cryptographic community card reveals between rounds.

Betting Rounds. The hand proceeds through up to four betting rounds:

1. **Preflop:** After hole cards are dealt, the player to the left of the big blind acts first. Players may fold, call the big blind, or raise.
2. **Flop:** Three community cards are revealed (see below). The first active player to the left of the dealer acts first.
3. **Turn:** One additional community card is revealed. Betting proceeds as in the flop round.
4. **River:** The final community card is revealed. A final betting round occurs.

Community Card Reveals. Between betting rounds, community cards are revealed using the mechanism described in Phase 5. All players broadcast their per-card keys $k_{i,j}$ for the relevant community card positions via their pairwise E2E encrypted channels. Once all keys are collected, every player independently decrypts the community cards.

Action Recording. Each player action (fold, check, call, bet, raise, all-in) is transmitted to the coordinator via the WebSocket connection. The coordinator validates the action against the current game state (correct player, legal action, valid amount) and broadcasts the updated state to all participants.

On-Chain Escape Hatch. If the coordinator is unresponsive or suspected of censorship, any player may submit their action directly on-chain via `submitAction(gameId, action, amount)` on the `PokerBetting` contract. The on-chain action record provides a censorship-resistant fallback that prevents the coordinator from selectively ignoring a player's actions. The coordinator must incorporate on-chain actions into its state or face a provable liveness violation.

Timing. Each player has a configurable time bank (default 30 seconds) per action. If the time bank expires, the player's hand is automatically folded by the coordinator. The time bank is enforced by both the coordinator and the on-chain phase deadline mechanism.

4.9 Phase 7: Showdown and Verification

At showdown, remaining players must prove the identity of their hole cards. This proceeds in two steps.

Key Revelation. Each player P_t who must show their hand reveals their per-card keys $k_{t,j}$ for their hole card positions $j \in \text{hole}_t$. Since the other players' keys for these positions were already shared during the deal phase (to enable P_t to decrypt), the complete set of all players' per-card keys for each showdown position is now known.

On-Chain Verification. The `verifyShowdownCard` function on the `MentalPokerGame` contract performs end-to-end cryptographic verification:

1. **Card key commitment verification:** For each player P_i and each showdown position j, verify $H(k_{i,j}) = \text{commit}_{i,j}$ (the commitment stored during the lock phase).
2. **Composite decryption verification:** Compute the product of all per-card keys: $K_j = \prod_{i=1}^{m} k_{i,j} \bmod n$. Verify that $K_j \cdot C_{\text{claimed}} = \mathscr{L}_m[j]$ using the `ecMul` precompile, where C_{claimed} is the claimed decrypted card point.
3. **Card validity verification:** Verify that C_{claimed} is a member of the set of 52 registered valid card points.

If all three checks pass, the card at position j is cryptographically confirmed to be the claimed card. Hand evaluation then proceeds on-chain (or off-chain with on-chain verification) to determine the winner and distribute the pot atomically via the escrow contract.

4.10 Protocol Properties

We now state and prove the three core security properties of the TrueShuffle protocol.

Theorem 3.1 (Fairness). *In a game with m players, no coalition of fewer than m players can determine the identity of any unrevealed card, assuming the hardness of the ECDLP on Secp256k1.*

Proof sketch. Consider an unrevealed card at position j in the locked deck. Its value is:

$$\mathscr{L}_m[j] = \left(\prod_{i=1}^{m} k_{i,j} \right) \cdot C_{\pi(j)}$$

To determine the card identity $C_{\pi(j)}$, the adversary must compute:

$$C_{\pi(j)} = \left(\prod_{i=1}^{m} k_{i,j}\right)^{-1} \cdot \mathscr{L}_m[j]$$

A coalition of $m-1$ players knows $k_{i,j}$ for $m-1$ values of i, but is missing the key $k_{t,j}$ of the honest player P_t. The adversary therefore needs to compute:

$$k_{t,j}^{-1} \cdot \left(\prod_{i \neq t} k_{i,j}^{-1} \cdot \mathscr{L}_m[j]\right)$$

The inner expression $Q = \prod_{i \neq t} k_{i,j}^{-1} \cdot \mathscr{L}_m[j] = k_{t,j} \cdot C_{\pi(j)}$ is computable by the coalition. To recover $C_{\pi(j)}$ from $Q = k_{t,j} \cdot C_{\pi(j)}$, the adversary must either determine $k_{t,j}$ (which requires solving ECDLP, since $k_{t,j}$ was sampled uniformly and only its commitment hash was published) or determine which of the 52 candidate cards satisfies $Q = k_{t,j} \cdot C_c$ for some card C_c (which also requires solving ECDLP to find the scalar relating Q to each C_c). Under the ECDLP hardness assumption, both approaches are computationally infeasible. □

Theorem 3.2 (Completeness). *If all m players follow the protocol honestly, every card can be correctly revealed to its intended recipient.*

Proof sketch. Consider a card at position j dealt to player P_t. During the deal phase, P_t receives $k_{i,j}$ from every other player P_i via E2E encrypted channels. Player P_t then computes:

$$k_{t,j}^{-1} \cdot \prod_{i \neq t} k_{i,j}^{-1} \cdot \mathscr{L}_m[j] = \left(\prod_{i=1}^{m} k_{i,j}\right)^{-1} \cdot \left(\prod_{i=1}^{m} k_{i,j}\right) \cdot C_{\pi(j)} = C_{\pi(j)}$$

The card point $C_{\pi(j)}$ is then looked up in the public card point table to identify the card. Since all arithmetic is performed in the group $\mathbb{G}$ of prime order n, and all keys are in $\mathbb{Z}_n^*$ (hence have well-defined inverses), the computation is always well-defined and correct. □

Theorem 3.3 (Verifiability). *Given the complete protocol transcript (on-chain commitments, deck hash chain, revealed keys, and locked deck), any observer can verify that the game was played correctly.*

Proof sketch. The on-chain record provides the following verification chain:

1. **Key commitment integrity:** The revealed keys match the committed hashes (verified during KEY_REVEAL and enforced by the contract).
2. **Deck hash chain integrity:** The chain $h_{\text{init}} \to h_1 \to \cdots \to h_{2m}$ is continuous (each step's input hash matches the previous step's output hash) and rooted at the known initial deck hash.
3. **Shuffle correctness:** For each player P_i who reveals (π_i, σ_i, s_i), the verifier can recompute the shuffle: encrypt $\mathscr{D}_{i-1}$ with s_i, apply permutation π_i, and verify the result matches $\mathscr{D}_i$ (whose hash is recorded on-chain). The permutation commitment c_i^{π} binds the player to the revealed values.
4. **Lock correctness:** For each player P_i, the `verifyLockStep` contract function confirms $s_i \cdot \mathscr{L}_i[j] = k_{i,j} \cdot \mathscr{L}_{i-1}[j]$ for all j.
5. **Card key commitment integrity:** Each revealed per-card key $k_{i,j}$ satisfies $H(k_{i,j}) = \text{commit}_{i,j}$.
6. **Showdown card verification:** The `verifyShowdownCard` function confirms that the decrypted card point, when re-encrypted with the product of all per-card keys, yields the locked deck entry;

and that the decrypted point is a valid card.

Together, these checks form a complete audit trail from the initial public deck through every cryptographic transformation to the final revealed cards. Any deviation from honest play — an incorrect shuffle, a substituted key, a misidentified card — is detectable from the on-chain record alone, without requiring trust in any participant or the coordinator. □

5 Security Analysis

The preceding chapters established the TrueShuffle protocol's cryptographic foundations and operational mechanics. We now subject the protocol to rigorous security analysis, beginning with a formal adversary model, proceeding through a computational security argument grounded in ECDLP hardness, cataloguing concrete attack vectors with their corresponding mitigations, and culminating in a formal fairness proof under the Elliptic Curve Discrete Logarithm assumption.

5.1 Threat Model

A meaningful security analysis requires a precise specification of the adversary's capabilities. We define three classes of adversary, each strictly more powerful than the last, and demonstrate that TrueShuffle maintains its core fairness properties against all three.

Passive adversary (eavesdropper). The passive adversary can observe all messages transmitted between players and the coordinator. This adversary captures every ciphertext, every commitment hash, and every protocol control message. The adversary does not inject, modify, or suppress messages. This model captures network-level surveillance, compromised routers, or a curious coordinator that faithfully relays messages while recording their contents.

Active adversary (message injection). The active adversary possesses all capabilities of the passive adversary and can additionally inject fabricated messages, modify messages in transit, reorder message delivery, or selectively delay messages. This model captures a fully compromised coordinator or a man-in-the-middle attacker with write access to the communication channel.

Byzantine adversary (arbitrary behavior). The Byzantine adversary controls one or more protocol participants who may deviate arbitrarily from the protocol specification. Corrupt players may submit malformed encryptions, apply non-random permutations, withhold decryption keys, or collude with one another and with the coordinator. This is the strongest adversary model we consider and corresponds to a worst-case scenario in which up to $m - 1$ of the m players at a table are controlled by a single adversary.

Throughout this analysis, we assume that the Elliptic Curve Discrete Logarithm Problem (ECDLP) is computationally hard on the Secp256k1 curve. Specifically, given a generator G and a point $Q = sG$ on Secp256k1, no probabilistic polynomial-time algorithm can recover s with non-negligible probability. The Secp256k1 curve operates over a 256-bit prime field, providing approximately 128 bits of symmetric-equivalent security [3]. Secp256k1 possesses a well-known GLV endomorphism that accelerates scalar multiplication but does not weaken the ECDLP. The GLV method can speed up Pollard's rho attack by a factor of approximately $\sqrt{3}$, yielding an effective security level of approximately 127 bits — still well within the accepted security margin. This assumption underpins the security of Bitcoin, Ethereum, and the broader ecosystem of ECDSA-based digital signatures.

The coordinator is modeled as an untrusted entity. In the best case, the coordinator is honest-but-curious: it follows the protocol faithfully but attempts to extract private information from observed messages. In the worst case, the coordinator is actively malicious: it may forge, suppress, or modify messages. The protocol is designed to maintain fairness in both cases, degrading only in liveness (not safety) under an actively malicious coordinator.

The critical resilience property of TrueShuffle is that the protocol requires only a single honest player to guarantee fairness. Formally, if at least one of the m players at a table follows the protocol faithfully, then no adversary — regardless of capability class — can determine the identity of an unrevealed card with probability exceeding $1/k$, where k is the number of remaining undealt cards. This property is a direct consequence of the commutative encryption structure and is proven formally in Section 4.4.

5.2 Computational Security Model

We analyze the information content of the encrypted deck at each stage of the protocol to establish that card secrecy is maintained under standard computational hardness assumptions.

After the SHUFFLE phase, each card position j in the deck D_m has been encrypted by all m players' keys in sequence. Concretely, let $C_1, C_2, \ldots, C_{52}$ denote the initial card encodings (publicly known curve points). After all m players have applied their respective permutations $\pi_1, \pi_2, \ldots, \pi_m$ and encryptions with scalar keys $s_1, s_2, \ldots, s_m$, card position j contains:

$$D_m[j] = s_m \cdot s_{m-1} \cdots s_1 \cdot C_{\pi_m(\pi_{m-1}(\cdots \pi_1(j) \cdots))}$$

By the commutativity of scalar multiplication on elliptic curves, this is equivalent to:

$$D_m[j] = \left(\prod_{i=1}^{m} s_i \right) \cdot C_{\sigma(j)}$$

where $\sigma = \pi_m \circ \pi_{m-1} \circ \cdots \circ \pi_1$ is the composed permutation. To decrypt a card without the cooperation of all m players, an adversary must recover $\sum s_i \pmod{n}$ (or equivalently, $\prod s_i$ in the multiplicative sense of scalar composition), which requires solving the ECDLP for the aggregate key — a problem assumed to be infeasible.

Consider the scenario in which $m - 1$ players collude, pooling their private keys $s_1, s_2, \ldots, s_{m-1}$. The colluding parties can compute the partial decryption:

$$s_1^{-1} \cdot s_2^{-1} \cdots s_{m-1}^{-1} \cdot D_m[j] = s_{\text{honest}} \cdot C_{\sigma(j)}$$

To proceed further, the coalition must solve $s_{\text{honest}} \cdot C_{\sigma(j)} = Q$ for either s_{honest} or $\sigma(j)$, which constitutes an instance of the ECDLP. Therefore, even a coalition of $m - 1$ players cannot determine the identity of an unrevealed card.

Information leakage analysis. While the ciphertext values are secure under the ECDLP assumption, the protocol may leak information through side channels. We identify and address the following:

- *Timing of encrypt/shuffle operations.* The time required for a player to encrypt and shuffle the

52-card deck is approximately constant (dominated by 52 EC scalar multiplications), but variations in network latency or computational load could theoretically reveal information about the permutation applied. We mitigate this by requiring all players to add random padding delays, normalizing the observable timing profile.

- *Message sizes.* Each shuffle message contains exactly 52 compressed elliptic curve points, each 33 bytes, yielding a fixed message size of approximately 1,716 bytes plus protocol overhead. The fixed-size property prevents message-length side channels.
- *Access patterns.* During the DEAL phase, the specific cards requested for decryption reveal which positions are being dealt (e.g., positions 0–1 are hole cards for player 1). This is inherent to the protocol and does not leak card identities, only the publicly known dealing order.

Entropy analysis. Each encrypted card is represented as a point on the Secp256k1 curve, which resides in a group of order $n \approx 2^{256}$. The compressed representation of each point requires 256 bits plus a sign bit. For a 52-card deck, the total entropy of the encrypted deck state is:

$$H(\text{encrypted deck}) = 52 \times 256 = 13{,}312 \text{ bits}$$

This vastly exceeds the $\log_2(52!) \approx 225.58$ bits of entropy required to represent an arbitrary permutation of 52 cards, providing a substantial margin against any information-theoretic attack on the permutation itself.

5.3 Attack Vectors and Mitigations

We enumerate the principal attack vectors against the TrueShuffle protocol and describe the corresponding defenses.

Card marking. In a physical card game, an attacker may mark cards to identify them in subsequent rounds. The digital analogue is an attacker who embeds identifiable structure in the ciphertext during the SHUFFLE phase, enabling later identification of specific cards without decryption. TrueShuffle mitigates this attack through re-encryption: each player applies a fresh random scalar s_i to every card during their shuffle turn. Even if player i marks cards by introducing a detectable pattern, player $i+1$'s encryption with an independent random scalar s_{i+1} destroys the pattern. Formally, for any function f that player i uses to mark card j, the output after player $i+1$'s encryption is $s_{i+1} \cdot f(D_{i-1}[j])$, which is computationally indistinguishable from a random curve point under ECDLP hardness.

Shuffle manipulation. An adversary who controls one or more players may apply a non-random or identity permutation during the SHUFFLE phase, attempting to control the final card ordering. This attack is ineffective provided at least one honest player applies a uniformly random permutation. The composition of any permutation with a uniformly random permutation is itself uniformly random: if π_{honest} is drawn uniformly from S_{52}, then for any fixed permutation $\pi_{\text{adversary}}$, the composition $\pi_{\text{honest}} \circ \pi_{\text{adversary}}$ is uniformly distributed over S_{52}. This is a standard result in group theory and constitutes the fundamental reason why mental poker protocols require only one honest participant for fairness [6].

Key reuse across hands. If a player reuses the same encryption key s_i across multiple hands, an attacker who observes the encrypted decks from two hands may attempt a differential analysis. Specifi-

cally, if the same key is applied to a known card encoding in two different hands, the attacker can verify the card's identity by checking whether the ratio of ciphertexts matches the ratio of known plaintexts. TrueShuffle eliminates this vector by generating fresh ephemeral keys for every hand. Each hand begins with a KEY_COMMIT phase in which all players commit to newly generated keys, ensuring that no key material is reused [7].

Replay attack. An attacker (or compromised coordinator) may attempt to resubmit protocol messages from a previous hand, causing players to process stale data. TrueShuffle defends against replay through a combination of unique hand identifiers (cryptographically random UUIDs) and monotonically increasing sequence numbers within each hand's message stream. Every protocol message includes the hand ID and sequence number; messages with mismatched hand IDs or out-of-sequence numbers are rejected by the receiving player's protocol engine.

Man-in-the-middle attack. A compromised coordinator may modify messages in transit — for example, substituting one player's shuffle output with a different ciphertext deck. TrueShuffle detects such tampering through commitment hashes. At each protocol phase, the sending player computes a SHA3-256 (Keccak-256) hash of the message payload and publishes the hash to the on-chain state. The receiving player independently hashes the received message and compares it against the on-chain commitment. Any discrepancy triggers a dispute, and the dispute resolution functions within the MentalPokerGame contract adjudicate responsibility based on the committed hashes.

Timing side-channel. If the time required to decrypt a specific card depends on the card's identity (e.g., due to variable-time modular arithmetic), an attacker could infer card identities by observing decryption latency. The TrueShuffle TypeScript implementation employs constant-time elliptic curve operations from the `@noble/curves` library, which provides resistance to timing side-channels through careful implementation of field arithmetic that avoids data-dependent branching and memory access patterns [3].

5.4 Formal Fairness Proof

We now state and prove the central security theorem of the TrueShuffle protocol.

Theorem 4.1 (Card Indistinguishability). *Let $m \geq 2$ players execute the TrueShuffle protocol over the Secp256k1 curve. Assume the ECDLP is hard on Secp256k1. Then no probabilistic polynomial-time adversary $\mathscr{A}$ controlling up to $m-1$ players and the coordinator can determine the identity of an unrevealed card at position j with probability greater than $1/k$, where k is the number of cards whose identities have not yet been revealed.*

Proof sketch. We proceed by reduction. Suppose adversary $\mathscr{A}$ can distinguish the identity of an unrevealed card at position j with probability $1/k + \epsilon$ for some non-negligible ϵ. We construct an algorithm $\mathscr{B}$ that uses $\mathscr{A}$ to solve the ECDLP.

$\mathscr{B}$ receives an ECDLP challenge (G, Q) where $Q = s^*G$ for unknown s^*. $\mathscr{B}$ simulates the TrueShuffle protocol, playing the role of the honest player (player h) and embedding the challenge into the honest player's encryption key by setting $s_h = s^*$.

The adversary $\mathscr{A}$ controls all other $m - 1$ players and the coordinator. $\mathscr{A}$ chooses their keys $s_1, \ldots, s_{h-1}, s_{h+1}, \ldots, s_m$ and their permutations. The honest player's permutation π_h is drawn uniformly at random by $\mathscr{B}$.

After the SHUFFLE phase, an unrevealed card at position j has the form:

$$D_m[j] = \left(\prod_{i=1}^{m} s_i \right) \cdot C_{\sigma(j)}$$

The adversary knows all s_i except $s_h = s^*$ and all π_i except π_h. Define $S_{\text{known}} = \prod_{i \neq h} s_i$. Then:

$$D_m[j] = s^* \cdot S_{\text{known}} \cdot C_{\sigma(j)}$$

The adversary can compute $S_{\text{known}}^{-1} \cdot D_m[j] = s^* \cdot C_{\sigma(j)}$. To identify the card, the adversary must determine $\sigma(j)$, which requires knowing $\pi_h(j)$ (since the adversary knows all other components of σ). But π_h is a uniformly random permutation unknown to $\mathscr{A}$, so $\sigma(j)$ is uniformly distributed over the k remaining (unrevealed) card positions.

If $\mathscr{A}$ can nonetheless identify the card with probability $1/k + \epsilon$, then $\mathscr{A}$ must be extracting information from the ciphertext $s^* \cdot C_{\sigma(j)}$. Since the C_i are publicly known and distinct, distinguishing between $s^* \cdot C_a$ and $s^* \cdot C_b$ for distinct cards a, b with non-negligible advantage implies the ability to decide the Decisional Diffie-Hellman (DDH) problem on Secp256k1, which contradicts the ECDLP hardness assumption.

Therefore, no such adversary $\mathscr{A}$ exists, and the unrevealed card's identity is computationally hidden up to the uniform distribution over remaining cards. □

The key insight underlying this proof is compositional: card j's plaintext identity is $C_{\sigma(j)}$ where $\sigma = \pi_m \circ \pi_{m-1} \circ \cdots \circ \pi_1$. Even if the adversary knows all permutations except π_h, the honest player's uniformly random permutation makes the composition uniformly random over the unrevealed card positions. The ECDLP hardness ensures that the adversary cannot bypass the permutation uncertainty by directly decrypting the ciphertext.

5.5 Coordinator Security Properties

The TrueShuffle coordinator occupies a privileged network position — all messages between players transit through it — yet the protocol is designed to ensure that this position confers no meaningful advantage to an adversary.

Confidentiality. Card-related data exchanged between players is encrypted end-to-end using X25519 Diffie-Hellman key exchange [5]. The encryption topology differs by protocol phase. During the SHUFFLE and LOCK phases, data is passed sequentially between consecutive players in the shuffle order ($P_1 \to P_2 \to \cdots \to P_m$), so E2E encryption for these messages is established between each consecutive pair (P_i, P_{i+1}). During the DEAL phase, card reveal keys must be sent from each player to the card's intended recipient; for this phase, each pair of players (i, j) at the table establishes a shared secret k_{ij}, providing all-pairs E2E encryption. In both cases, the coordinator observes only ciphertext and cannot recover the plaintext without solving the Computational Diffie-Hellman problem on Curve25519.

Even a fully compromised coordinator cannot:

1. *Read any player's cards.* Card data is E2E encrypted; the coordinator never possesses the decryption keys.
2. *Predict game outcomes.* The coordinator cannot determine the deck permutation, and thus cannot predict which cards will be dealt to which players.
3. *Modify encrypted data undetectably.* All protocol messages are accompanied by commitment hashes published on-chain. Any modification by the coordinator produces a hash mismatch, which any player can detect and escalate to on-chain dispute resolution.

A compromised coordinator retains the ability to mount liveness attacks:

- *Message delay.* The coordinator may delay forwarding messages, stalling the game. TrueShuffle mitigates this through configurable timeouts: if a player does not receive an expected protocol message within 30 seconds, the player may submit actions directly to the MegaETH smart contracts, bypassing the coordinator entirely.
- *Selective message suppression.* The coordinator may drop messages from specific players, effectively silencing them. Players detect suppression via monotonic sequence numbers: a gap in the expected sequence triggers an alert. The affected player can escalate by posting the missing message directly on-chain, along with proof that the coordinator received but did not relay it.
- *Selective service denial.* The coordinator may refuse to seat certain players or deny table access. Since the coordinator's registration and seating functions are replicated in on-chain contracts, players can invoke the smart contract directly to assert their right to participate.

In all cases, the coordinator's misbehavior affects liveness (the game may slow down or stall) but never safety (card secrecy and fairness are preserved). This separation of liveness and safety is a deliberate architectural decision: liveness failures are inconvenient but recoverable, while safety failures would be catastrophic and irrecoverable.

5.6 Smart Contract Security

The on-chain settlement layer constitutes the final line of defense in TrueShuffle's security architecture. The smart contracts deployed on MegaETH enforce protocol invariants that no off-chain actor — including a compromised coordinator and a full coalition of corrupt players — can violate.

Immutability. All TrueShuffle smart contracts are deployed without proxy patterns, admin keys, or upgrade mechanisms. Once deployed, the contract bytecode is immutable and can be audited by any party. This eliminates the risk of a privileged administrator modifying contract logic to steal funds or manipulate game outcomes — a risk that has materialized in numerous DeFi exploits [10].

Reentrancy protection. All functions that transfer ETH or tokens employ the checks-effects-interactions pattern and are additionally guarded by reentrancy locks. The PokerEscrow contract, which custodies player funds, is particularly hardened: the `resolveGame()` and `withdraw()` functions update internal accounting state before executing any external calls, preventing reentrant withdrawal attacks.

Integer overflow protection. All contracts are compiled with Solidity 0.8+, which provides built-in overflow and underflow checking for all arithmetic operations. This eliminates the class of integer overflow vulnerabilities that plagued earlier Solidity contracts [9].

Conservation invariant. The PokerEscrow contract enforces a strict conservation law: at every point during a hand, the total escrowed balance must equal the sum of all player stacks plus the current pot value:

$$\sum_{i=1}^{m} \text{escrow}_i + \text{pot} = \sum_{i=1}^{m} \text{buyIn}_i$$

This invariant is checked on every state-modifying transaction. Any violation causes the transaction to revert, preventing fund creation or destruction through accounting errors. The invariant extends to side pots: the sum of all side pot values plus remaining player stacks must equal the total buy-in for the hand.

Commitment verification. The MentalPokerGame contract stores Keccak-256 hashes of all protocol messages (key commitments, shuffle data, lock data). During dispute resolution, the MentalPokerGame contract's dispute functions verify that revealed data matches these commitments. Since Keccak-256 is collision-resistant, no adversary can produce a different message that hashes to the same commitment, ensuring that the on-chain record faithfully represents the protocol execution.

Gas limits and denial-of-service. All contract functions are designed to operate within predictable gas bounds. The hand evaluation function, in particular, operates in $O(1)$ time (constant with respect to the number of community cards, since a poker hand always consists of exactly 5–7 cards). This prevents gas-based denial-of-service attacks in which an adversary submits transactions designed to consume excessive gas.

6 Anti-Collusion and Sybil Resistance

Cryptographic fairness guarantees that no party can manipulate the deck or predict unrevealed cards. However, fairness of the deal does not, by itself, preclude collusion — the coordinated sharing of private information among ostensibly independent players to gain a collective advantage. Collusion is widely regarded as the most significant integrity threat in online poker [8], and any protocol aspiring to trustless operation must address it explicitly. This chapter examines the collusion problem in the context of TrueShuffle, delineates the boundaries of cryptographic protection, and presents the statistical and economic mechanisms that complement the protocol's cryptographic foundations.

6.1 The Collusion Problem in Online Poker

Collusion in poker occurs when two or more players at the same table coordinate their actions to exploit opponents who play independently. The practice is as old as the game itself, but the online environment dramatically lowers the barrier to collusion: colluding players can communicate in real time via a side channel (voice call, messaging application, screen sharing) with no risk of physical detection.

In traditional centralized poker platforms, collusion detection is the exclusive responsibility of the operator. Platforms such as PokerStars employ dedicated security teams that analyze hand histories for statistical anomalies indicative of coordinated play. These teams have access to the complete game record — every card dealt, every action taken, every timing interval — and apply proprietary detection

algorithms. The fundamental limitation of this approach is that it requires trusting the operator to be both competent and honest, a trust that has been violated repeatedly in the industry's history [1], [2].

In a trustless system where players control their own encryption and no central authority observes the game state, the collusion problem acquires a new dimension. The protocol cannot prevent players from sharing their decrypted hole cards over an out-of-band channel, because this information resides on the player's own device after decryption. The question, then, is not whether collusion can be prevented at the cryptographic layer — it cannot — but what combination of cryptographic, statistical, and economic mechanisms can minimize its prevalence and impact.

We distinguish two categories of collusion by severity:

Soft collusion involves the sharing of private game information without explicit coordination of actions. Two players who reveal their hole cards to each other gain an information advantage: each knows which cards are no longer available, improving hand evaluation and pot odds calculations. Soft collusion is difficult to detect because the colluding players may act independently on the shared information, producing action patterns that differ only subtly from honest play.

Hard collusion involves explicit coordination of betting actions to extract value from opponents. Techniques include chip dumping (one player intentionally loses chips to a confederate), ghost raising (a confederate raises to drive out other players, then folds to the primary colluder), and whipsawing (confederates on either side of a target alternate raising to inflate the pot). Hard collusion produces more pronounced statistical signatures and is correspondingly more detectable.

6.2 Cryptographic Anti-Collusion Properties

The TrueShuffle mental poker protocol provides structural protection against certain forms of advantage that would be available to colluders in a centralized system, while acknowledging fundamental limitations inherent to any protocol in which players ultimately decrypt and observe their own cards.

Deck integrity. The protocol's strongest anti-collusion property is that colluding players cannot influence which cards are dealt. The deck permutation is the composition of all players' individual shuffles, and as proven in Theorem 4.1 (Section 4.4), a single honest player's uniformly random permutation ensures that the composed permutation is itself uniformly random. Colluding players may share information about dealt cards, but they cannot arrange for favorable cards to be dealt to their positions. This is a strictly stronger guarantee than any centralized platform provides, where a compromised server could deal selectively.

Information compartmentalization. During the DEAL phase, each card's decryption key is revealed only to the card's intended recipient, encrypted under an E2E channel established via X25519 key exchange [5]. No player — honest or corrupt — receives decryption keys for cards dealt to other players. This means that a player's hole cards are cryptographically inaccessible to all other participants until voluntarily revealed at showdown.

Irreducible vulnerability. Once a player decrypts their hole cards on their local device, the plaintext exists in the player's memory. No cryptographic protocol can prevent the player from communicating this plaintext to a confederate via an out-of-band channel. This is a fundamental limitation shared by all mental poker protocols and, indeed, by physical poker: a player can always describe their hand to a confederate by phone. The protocol guarantees that the *only* way to obtain card information is

through legitimate protocol execution or voluntary disclosure by the cardholder.

The practical consequence is that soft collusion — sharing hole card information — is always possible and cannot be prevented by cryptographic means alone. The protocol's contribution is ensuring that colluding players gain *only* information advantages, never the ability to manipulate the deck, the deal order, or the evaluation of hands. The residual risk of information sharing must be addressed through complementary detection and deterrence mechanisms.

6.3 Statistical Collusion Detection

Given that cryptographic measures cannot prevent information sharing, TrueShuffle employs statistical analysis of gameplay data to identify likely collusion. These methods operate on publicly observable data — betting actions, showdown results, and timing — and do not require access to players' private card information.

Win rate anomaly detection. Over a sufficient sample of hands, each player's win rate should converge to a value determined by their skill level and the stakes played. When two colluding players consistently avoid confrontation with each other (folding rather than calling each other's bets), their mutual win rate against the rest of the table will be anomalously high. We track the pairwise confrontation rate r_{ij} for each player pair (i, j):

$$r_{ij} = \frac{|\text{hands where } i \text{ and } j \text{ both reached showdown}|}{|\text{hands where } i \text{ and } j \text{ both had opportunity to play}|}$$

A statistically significant depression in r_{ij} relative to the baseline confrontation rate at the table triggers further investigation.

Action correlation analysis. We compute the mutual information $I(A_i; A_j)$ between the action sequences of each player pair, conditioned on the public game state:

$$I(A_i; A_j | S) = \sum_{a_i, a_j, s} P(a_i, a_j, s) \log \frac{P(a_i, a_j | s)}{P(a_i | s) \cdot P(a_j | s)}$$

where A_i and A_j are the action random variables for players i and j, and S is the public state (community cards, pot size, position). Independent players should exhibit low mutual information conditioned on the public state; elevated mutual information suggests that the players' actions are informed by a common private signal — consistent with shared hole card information.

Comprehensive pair metrics. For each player pair observed over a window of N hands, the system tracks:

- *Confrontation frequency*: how often the pair enters pots together relative to baseline.
- *Showdown avoidance ratio*: how often one player folds to the other's aggression before showdown, relative to their fold rate against other opponents.
- *Chip transfer patterns*: the net flow of chips from one player to the other, tested against the expected distribution under independent play.
- *Timing correlation*: whether the two players' action timing shows synchronization artifacts (e.g., consistently similar decision times, suggesting communication during the hand).

Formal hypothesis testing. We apply the Kolmogorov-Smirnov (KS) test to compare the observed distribution of a player pair's metrics against the expected distribution under the null hypothesis of independent play. The KS statistic D_n measures the maximum deviation between the empirical cumulative distribution function and the reference distribution:

$$D_n = \sup_x |F_n(x) - F_0(x)|$$

where F_n is the empirical CDF of the observed metric and F_0 is the expected CDF under independence. A sufficiently large D_n (corresponding to a p-value below a chosen significance threshold) constitutes statistical evidence of collusion.

Bayesian collusion scoring. The individual statistical tests are synthesized into a composite Bayesian collusion score. Let $\mathcal{O}$ denote the vector of observed metrics for a player pair. The posterior probability of collusion is:

$$P(\text{colluding} \mid \mathcal{O}) = \frac{P(\mathcal{O} \mid \text{colluding}) \cdot P(\text{colluding})}{P(\mathcal{O})}$$

The prior $P(\text{colluding})$ is estimated from historical data across the platform. The likelihood $P(\mathcal{O} \mid \text{colluding})$ is calibrated using known collusion cases (from dispute resolution outcomes and confirmed violations). Player pairs whose posterior collusion probability exceeds a configurable threshold are flagged for automated action or manual review.

6.4 Sybil Resistance

A Sybil attack in the TrueShuffle context involves a single entity operating multiple bot accounts at the same table, effectively achieving hard collusion with perfect coordination. The following mechanisms raise the cost and reduce the effectiveness of Sybil attacks.

Economic barrier to entry. Each bot participating in TrueShuffle must maintain a bankroll deposit in the PokerEscrow smart contract. To seat a bot at a table, the player must transfer a buy-in from their bankroll, locking real economic value for the duration of the hand. Operating k Sybil bots at a single table requires k independent bankrolls, each meeting the table's minimum buy-in requirement. At higher-stakes tables, the capital requirement for a Sybil attack becomes prohibitively expensive relative to the expected collusion profit.

Per-address table limits. The smart contract enforces a maximum of N bots per wallet address at any single table (with $N = 1$ as the default for competitive tables). While an attacker can create multiple wallet addresses, each address requires an independent bankroll deposit, multiplying the capital cost.

Bankroll-weighted seating priority. When table demand exceeds capacity, seating priority is determined by bankroll age and history. Bots associated with wallets that have maintained a bankroll for longer periods and have played more hands receive priority. This creates a temporal cost for Sybil accounts: freshly created wallets are deprioritized relative to established participants, reducing the effectiveness of rapid Sybil deployment.

Reputation and staking (future work). The roadmap includes an on-chain reputation system in which bots accumulate a reputation score based on the number of hands played, dispute outcomes,

and community evaluations. Bots with low or no reputation would be required to post a larger stake as a bond against detected collusion. This mechanism creates an asymmetry: legitimate participants build reputation over time and face decreasing friction, while Sybil accounts must repeatedly post bonds that are forfeited upon detection.

6.5 Economic Disincentives

Beyond detection and prevention, TrueShuffle employs economic mechanisms that reduce the expected profitability of collusion, rendering it irrational for a profit-maximizing adversary in many scenarios.

Rake structure. The platform charges a rake (percentage fee) on each pot. Colluding players who engage in ghost raising or pot inflation pay additional rake on the inflated pot, reducing the net profit from the manipulation. For a rake rate of r and a pot inflation of ΔP due to collusion, the colluding group pays an additional $r \cdot \Delta P$ in rake. If the expected profit from collusion is π_c, collusion is unprofitable when $\pi_c < r \cdot \Delta P$. At typical rake rates of 2.5–5%, this constraint eliminates the profitability of many common collusion strategies, particularly at lower stakes where the absolute profit margin is already thin.

Chip dumping detection. Chip dumping — the intentional transfer of chips from one player to another through deliberate losing play — produces distinctive statistical signatures: highly asymmetric chip flow between two players, combined with the dumping player's anomalously high fold rate to the recipient's bets. The statistical detection system (Section 5.3) flags such patterns, and confirmed violations result in deposit forfeiture.

Smart contract enforcement. When collusion is confirmed through the dispute resolution mechanism or by governance action, the PokerEscrow contract can freeze and forfeit the colluding players' bankroll deposits. This penalty is enforced at the smart contract level and cannot be circumvented by the penalized players. The forfeited deposits are distributed to affected opponents as compensation, creating a direct economic deterrent: the expected cost of detection and forfeiture must be weighed against the expected collusion profit.

The combination of cryptographic deck integrity, statistical detection, Sybil resistance, and economic disincentives creates a layered defense against collusion. No single mechanism is sufficient in isolation, but their composition raises the cost and risk of collusion to the point where rational, profit-maximizing adversaries are deterred in most practical scenarios. The irreducible residual risk — that two skilled colluders sharing hole cards will marginally outperform independent play without triggering detection thresholds — is a property shared by all poker platforms, online and physical alike, and is bounded by the fundamental information-theoretic limits of the game itself.

7 System Architecture

The security and fairness guarantees established in the preceding chapters are only as valuable as the system that realizes them. This chapter describes the concrete software architecture of TrueShuffle: the components, their responsibilities, the communication topology that connects them, the message protocol that governs their interactions, and the fault tolerance mechanisms that maintain system integrity under adverse conditions.

7.1 Design Principles

The TrueShuffle architecture is governed by four foundational principles that inform every design decision.

Coordinator as untrusted relay. The coordinator server occupies a privileged network position but is granted no trust. It cannot read card data (E2E encrypted), cannot modify messages undetectably (commitment hashes on-chain), and cannot prevent players from interacting with the smart contracts directly. The coordinator's role is strictly that of a message router and convenience layer: it improves latency and user experience but is not required for protocol correctness.

Defense in depth. No single mechanism is relied upon for security. Cryptographic guarantees (ECDLP hardness, commitment schemes) form the first layer. Economic incentives (bankroll deposits, rake-based disincentives) form the second. Smart contract enforcement (escrow invariants, dispute resolution) forms the third. Statistical monitoring (collusion detection, anomaly analysis) forms the fourth. An adversary must simultaneously defeat all layers to compromise the system.

Client-side cryptographic execution. All cryptographic operations — key generation, encryption, shuffling, decryption — are performed in TypeScript using the `@noble/curves` library (Secp256k1) and `@noble/ciphers` (XSalsa20-Poly1305), executing within the bot's local process. Private keys never leave the player's device. The Bot SDK provides a TypeScript framework for building bots that connect to the coordinator via WebSocket. A Tauri v2 desktop application is planned as a future component to provide a native GUI for bot management and local monitoring.

Minimal on-chain footprint. Only data essential for verifiability and settlement is written to the MegaETH blockchain [4]: commitment hashes, protocol checkpoints, betting actions, and fund transfers. The full protocol transcript (encrypted decks, shuffle data) remains off-chain, transmitted peer-to-peer through the coordinator. This separation keeps gas costs low while preserving the ability to reconstruct and verify any disputed hand from its on-chain commitments.

7.2 Component Overview

The TrueShuffle system comprises four major components — the Bot SDK, the Coordinator Server, the MegaETH Smart Contracts, and the Web Frontend — each designed for a specific operational domain. A fifth component, a Tauri v2 desktop application, is planned for future development.

7.2.1 Bot SDK (TypeScript)

The Bot SDK is the primary execution environment for all cryptographic and game-logic operations. Written entirely in TypeScript, it runs as a Node.js process on each bot operator's machine.

The **Protocol Engine** implements the complete mental poker protocol in TypeScript. It manages the state machine transitions through KEY_COMMIT, KEY_REVEAL, SHUFFLE, LOCK, DEAL, BETTING, SHOWDOWN, and COMPLETE phases. Each phase transition is validated against the protocol specification before execution, and the engine rejects any message that would cause an invalid state transition. The engine maintains an append-only log of all protocol messages for dispute evidence.

The **Crypto Module** provides the low-level cryptographic primitives. It uses `@noble/curves` for Secp256k1 elliptic curve operations (scalar multiplication, point addition, key generation), `@noble/ciphers` for XSalsa20-Poly1305 authenticated encryption (establishing E2E encrypted channels

between player pairs), and `@noble/hashes` for Keccak-256 hashing (commitment generation and verification). All elliptic curve operations leverage `@noble/curves`' constant-time implementations to resist timing side-channel attacks.

The **Bot Brain** is the decision engine that evaluates the current game state and selects actions. It operates against a parameterized strategy profile — the Bot DNA — which encodes aggression frequencies, positional adjustments, hand range definitions, and bluffing thresholds. The Bot Brain receives the public game state (community cards, pot size, opponent actions, position) and the player's private information (hole cards) and produces a betting action (fold, check, call, raise with sizing). The decision logic is entirely local; no game-state information is transmitted to the coordinator or any other party beyond the selected action.

The **WebSocket Client** manages the persistent connection to the coordinator server. It handles connection lifecycle (establishment, keepalive, reconnection), message serialization and deserialization, sequence number tracking, and E2E encryption of protocol payloads. The client implements exponential backoff for reconnection and maintains a local message queue for outbound messages during transient disconnections.

The **On-Chain Client** uses `ethers` (or `viem`) to interact directly with the MegaETH smart contracts. It submits commitment hashes, publishes protocol checkpoints, executes betting actions on-chain when the coordinator is unavailable, initiates disputes, and manages bankroll deposits and withdrawals. The on-chain client maintains a local nonce counter and implements gas estimation to ensure reliable transaction submission.

7.2.2 Desktop Application (Planned — Tauri v2)

A Tauri v2 desktop application is planned as a future component to provide a native GUI for bot management, local configuration, real-time monitoring, and wallet integration. The desktop app will wrap the Bot SDK's TypeScript runtime and provide a graphical interface for operators who prefer not to interact with the SDK directly via the command line.

7.2.3 Coordinator Server (Node.js + Fastify)

The coordinator server provides the real-time communication infrastructure and convenience services for the TrueShuffle platform. It is implemented in Node.js with the Fastify framework for its high-throughput HTTP handling and low overhead.

The **Table Manager** creates and destroys tables, manages player seating and buy-in validation, enforces table configuration parameters (minimum/maximum players, blind structure, buy-in range), and maintains the table lifecycle from creation through player seating to game initiation and eventual teardown.

The **Protocol Relay** is the core message-forwarding engine. It receives protocol messages from players, validates their structural integrity (correct format, valid sequence number, matching hand ID), and forwards them to the appropriate recipients. The relay does not inspect or modify the encrypted payload of protocol messages — it operates exclusively on the unencrypted message envelope. All relayed messages are logged with timestamps for auditability.

The **WebSocket Hub** manages bidirectional real-time connections with all connected players. It han-

dles connection multiplexing (a single coordinator instance may serve hundreds of concurrent tables), heartbeat monitoring, and graceful connection teardown. The hub implements per-connection backpressure to prevent slow consumers from degrading system performance.

The **REST API** provides endpoints for bot registration (associating a wallet address with the platform), table listing (querying available tables and their configurations), state queries (retrieving the current public state of a table or hand), and historical data (completed hand results for statistical analysis). All REST endpoints are rate-limited and authenticated via ECDSA signatures tied to the bot's Secp256k1 key pair.

The **Event Broadcaster** disseminates public game events — betting actions, community card reveals, showdown results — to spectators who are not active players at the table. Spectator connections receive a filtered view of the game state that excludes all private information (hole cards, encryption keys, shuffle data). The broadcaster uses a publish-subscribe model to minimize per-spectator overhead.

7.2.4 MegaETH Smart Contracts (Solidity)

The on-chain layer consists of six core smart contracts (with additional supporting contracts including PokerTable, OnChainLeaderboard, and Secp256k1), each with a single well-defined responsibility.

Contract	Responsibility	Key Functions
MentalPokerGame	Game lifecycle, protocol phases, dispute resolution	`createGame()`, `commitKeys()`, `submitShuffle()`, `submitLock()`, `initiateDispute()`, `resolveDispute()`
MentalPokerCoordinator	Table management, player registration	`registerTable()`, `joinTable()`, `leaveTable()`
PokerBetting	Betting actions, pot management	`submitAction()`, `settleBets()`
PokerEscrow	Fund custody, settlement	`deposit()`, `withdraw()`, `createGameEscrow()`, `resolveGame()`
PokerHandEvaluator	Hand ranking, winner determination	`evaluateHand()`, `compareHands()` — pure, no state
PokerGameResolver	Game outcome resolution	`resolveFromBetting()`, `resolveFromShowdown()`

The contracts are designed for minimal surface area: each exposes only the functions strictly necessary for its responsibility, reducing the attack surface for smart contract exploits. Inter-contract calls are restricted to well-defined interfaces, and all payment-related functions employ the checks-effects-interactions pattern with reentrancy guards.

7.2.5 Web Frontend (React 19 + TanStack Router)

The web frontend provides browser-based access to the platform's public-facing features. It is built with React 19 and uses TanStack Router for type-safe client-side routing.

The **Spectator UI** renders live game state for non-participating viewers, displaying community cards, betting actions, pot sizes, and showdown results in real time via WebSocket subscription to the Event Broadcaster. The **Bot Studio** provides a graphical interface for configuring Bot DNA parameters, backtesting strategies against historical hand data, and deploying bots to the desktop runtime. The **Leaderboard** aggregates and displays bot performance metrics — win rate, profit/loss, hands played, Elo rating — computed from on-chain settlement data. The **Explorer** provides a block-explorer-style interface for inspecting individual hands, viewing the on-chain protocol transcript, and verifying that commitment hashes match revealed data.

7.3 Network Topology

TrueShuffle employs a star topology in which all bot clients connect to the coordinator server via persistent WebSocket connections. The coordinator occupies the hub position, and all inter-player communication transits through it.

Within this star topology, end-to-end encryption creates logical point-to-point channels between each pair of players at a table. During the KEY_COMMIT phase, each player pair (i, j) performs an X25519 key exchange to establish a shared secret k_{ij}. Subsequent protocol messages containing sensitive data are encrypted under k_{ij} before transmission. The coordinator transports these ciphertexts but cannot decrypt them, effectively reducing its role to that of an encrypted message router.

This architecture offers operational simplicity and low latency: messages require only two hops (sender to coordinator, coordinator to recipient), and the coordinator can apply quality-of-service policies (prioritization, rate limiting) at the application layer. The tradeoff is a single point of failure for liveness — if the coordinator becomes unavailable, real-time communication ceases. This tradeoff is acceptable because the on-chain fallback path (Section 6.5) ensures that game safety is never compromised by coordinator failure.

The long-term architectural roadmap includes migration to a libp2p-based mesh network [11] for direct player-to-player communication, eliminating the coordinator from the data path entirely. In this topology, players would discover peers via the on-chain Coordinator contract and establish direct encrypted connections using libp2p's Noise protocol. The coordinator would be reduced to an optional convenience service for NAT traversal and peer discovery, with no role in message transport.

7.4 Message Protocol

All communication between players and the coordinator follows a JSON-RPC-inspired message format transported over WebSocket frames. Each message consists of an unencrypted envelope and an optionally encrypted payload.

The message envelope contains: a message type identifier, the sender's public key (compressed Secp256k1 point), the hand ID (UUID), a monotonically increasing sequence number, and a timestamp. The envelope is signed by the sender using ECDSA on Secp256k1, enabling recipients to verify message authenticity and detect tampering.

The protocol defines five core message types, corresponding to the phases of the mental poker protocol:

- `key_commit`: player publishes a hash commitment to their per-hand encryption key.
- `key_reveal`: player reveals their encryption public key (after all commitments are collected).
- `shuffle_step`: player transmits the re-encrypted and permuted deck to the next player in the shuffle order.
- `lock_step`: player transmits the locked (individually encrypted) deck.
- `card_keys`: player provides the requested per-card decryption keys, encrypted under the E2E channel with the recipient.

Game actions and state updates use separate schemas outside the core protocol message types. The `BotActionSchema` defines betting actions (fold, check, call, raise with amount), and the `SpectatorEventSchema` defines the event stream broadcast to spectators and non-participating observers.

Sequence numbers serve dual purposes: replay prevention and message ordering. Each player maintains independent send and receive sequence counters per hand. A message with a sequence number that does not equal the expected next value is rejected. This prevents both replay attacks (resubmitting old messages) and reordering attacks (delivering messages out of sequence to induce incorrect state transitions).

Message sizes are predictable and bounded. The largest message type is `shuffle_step`, which contains 52 compressed elliptic curve points at 33 bytes each, totaling approximately 1,716 bytes of curve point data plus envelope overhead (approximately 200 bytes), yielding a total message size of approximately 2 KB. All other message types are substantially smaller. The bounded message size enables predictable bandwidth requirements and prevents denial-of-service through oversized messages.

7.5 Fault Tolerance

The TrueShuffle architecture is designed to maintain safety (no incorrect outcomes, no fund loss) under all failure modes, while preserving liveness (continued game progress) under most failure modes.

Player disconnect during setup. If a player disconnects during the KEY_COMMIT, KEY_REVEAL, SHUFFLE, or LOCK phases — before any betting has occurred — the hand is voided. All buy-ins are returned to the respective players' bankrolls via the PokerEscrow contract. No player is penalized for a setup-phase disconnection, as no information advantage has been gained.

Player disconnect during play. If a player disconnects after betting has begun, a timeout mechanism activates. The disconnected player has 30 seconds (configurable per table) to reconnect and submit their action. If the timeout expires, the player is automatically folded for the current hand. Their remaining stack (minus any chips already committed to the pot) is preserved in escrow and returned upon reconnection. Repeated timeouts trigger escalating penalties, up to and including ejection from the table.

Coordinator failure. If the coordinator becomes unavailable, the off-chain real-time communication channel is severed, but the game's safety properties are unaffected. Players can submit actions directly to the MegaETH smart contracts: the PokerBetting contract accepts betting actions via on-chain transactions, and the MentalPokerGame contract accepts protocol messages (key reveals, card decryption keys) via on-chain submission. This fallback path is higher latency (limited by MegaETH's block time

of approximately 10 milliseconds plus network propagation) but ensures that no hand is permanently stalled by coordinator failure.

On-chain dispute resolution. After the coordinator submits a hand result to the PokerEscrow contract for settlement, a dispute window of 3 minutes opens. During this window, any player may challenge the reported outcome by calling `initiateDispute()` on the MentalPokerGame contract, accompanied by the protocol transcript evidence and a bond deposit. The contract verifies the evidence against on-chain commitments and adjudicates the dispute via `resolveDispute()`. If the challenge is valid, the faulty party is penalized and the challenger's bond is returned. If the challenge is invalid, the challenger forfeits their bond. This mechanism ensures that even a coordinator that submits fraudulent hand results can be held accountable.

Byzantine fault tolerance. The system tolerates Byzantine behavior from any single actor — whether a player, the coordinator, or a smart contract interactor — without compromising fairness. As established in Theorem 4.1 (Section 4.4), card secrecy requires only one honest player. Liveness may degrade under Byzantine faults (e.g., a malicious player may delay the protocol by withholding messages until timeout), but safety is preserved unconditionally. The combination of cryptographic commitments, on-chain verification, and economic penalties ensures that Byzantine behavior is detectable, attributable, and punishable, creating a strong deterrent even when prevention is impossible.

8 On-Chain Settlement Layer

The cryptographic protocol guarantees that no party can observe or manipulate unrevealed cards. The system architecture ensures that these guarantees are realized in a practical, fault-tolerant system. However, neither mechanism addresses the most consequential question in any wagering platform: who gets paid, and how can players be certain that payouts are correct? The on-chain settlement layer answers this question by executing all financial operations — deposits, buy-ins, pot distributions, and dispute resolutions — as atomic smart contract transactions on the MegaETH Layer 2 network [4]. This chapter describes the smart contract architecture, the escrow mechanics, the on-chain hand evaluation algorithm, the dispute resolution protocol, and the rationale for selecting MegaETH as the settlement layer.

8.1 Smart Contract Architecture

The complete contract architecture is detailed in Section 6.2. This section summarizes the settlement-relevant contracts and their roles.

The **MentalPokerGame** contract serves as the on-chain record of the cryptographic protocol's execution. It stores commitment hashes for each player's per-hand encryption key (submitted during KEY_COMMIT), the revealed public keys (submitted during KEY_REVEAL), and checkpoint hashes of the encrypted deck at each protocol phase (SHUFFLE, LOCK). These on-chain records serve as the ground truth for dispute resolution: if any party claims that the protocol was not followed correctly, the MentalPokerGame contract's dispute functions (`initiateDispute()`, `resolveDispute()`, `abortAndSlash()`) compare the disputed data against these commitments.

The **PokerBetting** contract implements the betting state machine for Texas Hold'em. It tracks the current betting round (preflop, flop, turn, river), the action position, the minimum raise, and each

player's committed chips. All betting actions are submitted through a single `submitAction(gameId, action, amount)` function, where the `action` parameter is drawn from an Action enum (fold, check, call, raise, all-in). The contract enforces the rules of No-Limit Hold'em: a raise must be at least the size of the previous raise, a player cannot bet more than their remaining stack, and the action must proceed in seat order. Each betting action is recorded as an on-chain event, creating an immutable audit trail.

The **PokerEscrow** contract is the custodial core of the settlement layer. It holds all player funds during gameplay and executes all transfers. The contract enforces the conservation invariant (Section 7.2) at every state transition, ensuring that funds can neither be created nor destroyed through accounting errors or exploits. The escrow contract is the only contract authorized to transfer ETH or tokens, and all payment-related functions are protected by reentrancy guards.

The **PokerHandEvaluator** contract is a pure (stateless, side-effect-free) contract that accepts a set of cards and returns a hand score. It is invoked at showdown to determine the winner and, in the event of a dispute, to verify that the coordinator's reported hand ranking is correct. Because it maintains no state and performs no external calls, it presents a minimal attack surface and can be formally verified independently.

The **PokerGameResolver** contract implements game outcome resolution, providing `resolveFromBetting()` (when all but one player folds) and `resolveFromShowdown()` (when hands are compared at showdown). It coordinates between PokerHandEvaluator for hand scoring and PokerEscrow for fund distribution.

The **MentalPokerCoordinator** contract provides on-chain table management and peer discovery. Bots register their public keys and connection endpoints, and the contract maintains a registry of active participants. This enables players to discover peers and establish direct connections without relying on a centralized directory service. The coordinator contract also records the mapping between table IDs and their seated players, enabling any participant to verify table composition.

8.2 Escrow Mechanics

The PokerEscrow contract implements a two-tier fund management system: a persistent bankroll layer and a per-hand escrow layer.

Deposit and bankroll. A player deposits ETH (or approved ERC-20 tokens) into the PokerEscrow contract by calling `deposit()`. The deposited funds are credited to the player's bankroll balance, which persists across hands and sessions. The bankroll represents the player's total available funds on the platform. Players may withdraw any portion of their bankroll that is not currently locked in an active hand by calling `withdraw()`.

Buy-in and hand escrow. When a hand begins, the `createGameEscrow()` function transfers the specified amount from each player's bankroll to the hand's escrow pool. The buy-in amount is locked for the duration of the hand and cannot be withdrawn until the hand completes. Each player's escrowed amount represents their starting stack for the hand.

The escrow contract enforces a strict conservation invariant at every state transition:

$$\sum_{i=1}^{m} \text{escrow}_i + \text{pot} = \sum_{i=1}^{m} \text{buyIn}_i$$

where escrow_i is player i's remaining stack, pot is the current pot (including all side pots), and buyIn_i is player i's initial buy-in for the hand. This invariant is checked on every betting action and every settlement operation. Any transaction that would violate the invariant is reverted, providing a mathematical guarantee against fund leakage.

Settlement. Upon hand completion, the coordinator submits the hand result to the PokerEscrow contract by calling `resolveGame()`. This function receives the hand scores (from PokerHandEvaluator or from the coordinator's report) and distributes the pot to the winning player(s). In the case of a split pot (tied hand scores), the pot is divided equally among the tied players, with any indivisible remainder awarded to the player closest to the dealer position.

Side pot calculation. When a player goes all-in for less than the current bet, a side pot is created. The PokerEscrow contract handles side pot computation as follows: the main pot contains each player's contribution up to the all-in amount, and a side pot contains the excess contributions from players who remain active. If multiple players go all-in for different amounts, the contract creates a cascade of side pots, each eligible to be won only by the players who contributed to it. The conservation invariant extends to side pots:

$$\sum_{k} \text{sidePot}_k + \sum_{i \in \text{active}} \text{remaining}_i = \sum_{i=1}^{m} \text{buyIn}_i$$

where the sum ranges over all side pots k and all active (non-folded) players i.

8.3 Hand Evaluation On-Chain

The PokerHandEvaluator contract implements a deterministic, gas-efficient algorithm for scoring poker hands. The scoring system maps every possible 5-card poker hand to a unique integer, enabling comparison by simple integer ordering.

Base-13 encoding. Card ranks are encoded as integers from 0 (deuce) to 12 (ace). The hand score is computed as a composite integer using a base-13 positional system:

$$\text{score} = \text{handRank} \times 13^4 + k_1 \times 13^3 + k_2 \times 13^2 + k_3 \times 13 + k_4$$

where handRank is an integer from 0 to 9 representing the hand category, and k_1, k_2, k_3, k_4 are the kicker values in descending order of significance. The hand categories and their corresponding rank values are:

Rank Value	Hand Category	Example
9	Royal Flush	A-K-Q-J-10 suited
8	Straight Flush	9-8-7-6-5 suited
7	Four of a Kind	Q-Q-Q-Q-7
6	Full House	J-J-J-4-4
5	Flush	K-J-8-4-3 suited
4	Straight	10-9-8-7-6
3	Three of a Kind	8-8-8-K-2

Rank Value	Hand Category	Example
2	Two Pair	A-A-5-5-Q
1	One Pair	10-10-K-7-3
0	High Card	A-Q-9-6-3

The base-13 encoding ensures that the hand rank is always the most significant component of the score, guaranteeing that any hand of a higher category beats any hand of a lower category. Within the same category, kickers are compared in order of significance. For example, a pair of aces with a king kicker scores higher than a pair of aces with a queen kicker, regardless of the remaining kickers.

Seven-card evaluation. In Texas Hold'em, each player's hand is the best 5-card combination from 7 available cards (2 hole cards + 5 community cards). The evaluator examines all $\binom{7}{5} = 21$ possible 5-card combinations and returns the maximum score. This brute-force approach is feasible on-chain because 21 evaluations of a constant-time scoring function remain well within MegaETH's gas limits.

Edge cases. The evaluator handles all standard edge cases: the ace functions as both high (in A-K-Q-J-10 straights) and low (in 5-4-3-2-A straights); flush detection correctly identifies 5-card flushes within 7-card hands that may contain cards of multiple suits; and split pots are determined by exact score equality across all kicker positions.

Determinism and verifiability. Because the evaluator is a pure function with no state dependencies, its output is entirely determined by its inputs. Any party can independently verify a hand evaluation by invoking the contract with the same cards. This property is essential for dispute resolution: if a challenger claims that the coordinator reported an incorrect hand ranking, the MentalPokerGame contract's dispute resolution logic can re-evaluate the hand on-chain and compare the result.

8.4 Dispute Resolution Protocol

The dispute resolution protocol provides a cryptographic and economic mechanism for challenging incorrect hand outcomes. It operates in two paths: the normal (uncontested) path and the dispute (challenged) path.

Normal path. After a hand completes (all betting rounds are finished and remaining players reveal their cards at showdown), the coordinator computes the hand result and submits it to the PokerEscrow contract via `resolveGame()`. The submission includes the final hand scores and the pot distribution. A 3-minute dispute window opens upon submission. If no challenge is filed within this window, the settlement is finalized and funds are distributed to the designated winners.

Dispute path. Any player who participated in the hand may challenge the coordinator's submitted result by calling `initiateDispute()` on the MentalPokerGame contract within the 3-minute window. The challenger must provide:

1. A bond deposit (denominated in ETH) that will be forfeited if the challenge is found to be frivolous.
2. The protocol transcript: the sequence of encrypted deck checkpoints, key commitments, key reveals, card decryption keys, and betting actions that the challenger recorded during protocol execution.

Upon receiving a challenge, the MentalPokerGame contract's dispute resolution logic performs a four-stage verification:

Stage 1: Commitment verification. The contract hashes each submitted key commitment and compares it against the commitment hash stored in the MentalPokerGame contract during the KEY_COMMIT phase. If any commitment does not match, the submitting player is identified as the faulty party (they submitted a key that does not match their commitment, indicating an attempt to substitute keys after seeing other players' actions).

Stage 2: Shuffle and lock verification. The contract hashes the submitted shuffle and lock data and compares the hashes against the checkpoints stored in MentalPokerGame during the SHUFFLE and LOCK phases. A mismatch indicates that either the submitting player or the coordinator tampered with the encrypted deck data. The commitment chain allows attribution: each checkpoint is associated with the player who submitted it, enabling precise identification of the tampering party.

Stage 3: Card reveal consistency. The contract verifies that the card decryption keys revealed during the DEAL phase are consistent with the locked deck. Specifically, for each revealed card at position j, the contract applies the revealed decryption keys to the locked deck entry $D_{\text{lock}}[j]$ and verifies that the result is a valid card encoding C_i for some $i \in \{1, \ldots, 52\}$. An inconsistent reveal indicates that a player provided a false decryption key, attempting to alter the card's apparent identity.

Stage 4: Hand evaluation verification. The contract invokes the PokerHandEvaluator with the verified card identities and compares the resulting scores against the coordinator's submitted hand result. A discrepancy indicates that the coordinator reported an incorrect ranking, either through error or malice.

Adjudication. If any verification stage identifies a faulty party, the dispute is resolved in favor of the challenger. The faulty party's escrowed funds are slashed (a configurable penalty, up to the full escrow amount), and the slashed amount is distributed as follows: the challenger receives their bond back plus a bounty (a fraction of the slashed amount, incentivizing legitimate challenges), and the remainder is distributed to the other affected players. If all verification stages pass — indicating that the coordinator's submitted result was correct — the challenge is rejected, and the challenger forfeits their bond to the coordinator as compensation for the frivolous dispute.

The bond requirement serves a dual purpose: it deters frivolous challenges (which waste gas and delay settlement) while ensuring that legitimate grievances can be raised. The bond amount is calibrated to be significant enough to deter abuse but small enough relative to typical pot sizes that a player with a genuine dispute is not economically prevented from filing.

8.5 Why MegaETH

The choice of settlement layer is a critical architectural decision. The settlement chain must provide sufficient throughput, latency, and cost efficiency to support real-time poker without introducing perceptible delays or prohibitive overhead. We evaluated several EVM-compatible chains against these requirements.

Chain	Block Time	Finality	Approx. Cost per Tx	Viable for Poker?
Ethereum L1	12 s	~6 min	$52+	No
Polygon PoS	2 s	~30 min	$0.45	Marginal
Arbitrum One	250 ms	~7 min	$0.12	Possible
Optimism	2 s	~7 min	$0.10	Possible
MegaETH	10 ms	<1 s	$0.008	Yes

A typical 6-player Texas Hold'em hand generates between 15 and 25 on-chain transactions: key commitments (6), key reveals (6), shuffle/lock checkpoints (2–4), betting actions (variable, typically 4–8 per hand across all rounds), and settlement (1). On Ethereum L1, this volume of transactions would cost approximately $780–$1,300 per hand at current gas prices, rendering real-time on-chain poker economically infeasible. Even on Arbitrum or Optimism, costs of $1.50–$3.00 per hand would consume the platform's rake margin at lower stakes.

MegaETH's transaction cost of approximately $0.008 per transaction yields a total on-chain cost of $0.12–$0.20 per hand. At a standard rake rate of 2.5–5% on a $20 pot (typical of micro-stakes), the rake revenue is $0.50–$1.00, comfortably absorbing the on-chain settlement cost. This cost structure makes on-chain settlement economically viable across all stake levels, from micro-stakes training tables to high-stakes competitive play.

Latency. MegaETH's 10-millisecond block time and sub-second finality mean that on-chain transactions confirm before a human player (or bot with typical network latency) would perceive any delay. In contrast, Ethereum L1's 12-second blocks and 6-minute finality would introduce pauses that fundamentally disrupt the pace of play. Even Arbitrum's 250-millisecond soft confirmations require waiting for L1 finality (~7 minutes) for true settlement assurance, creating a window during which dispute resolution cannot be finalized.

EVM compatibility. MegaETH is fully EVM-compatible, enabling the TrueShuffle smart contracts to be written in standard Solidity, tested with standard tools (Hardhat, Foundry), and audited by any EVM-experienced security firm. No custom virtual machine, novel programming language, or unfamiliar toolchain is required. This compatibility also facilitates future multi-chain deployment: the contracts can be redeployed on any EVM chain if market conditions or technology evolution warrant migration.

Finality guarantees. MegaETH provides sub-second finality, meaning that once a transaction is included in a block, it is irreversible. This property is essential for the dispute resolution protocol: the 3-minute dispute window can be measured precisely because finalized transactions have a definitive timestamp. On chains with probabilistic finality, the dispute window would need to account for potential chain reorganizations, complicating the protocol and increasing the minimum window duration.

The combination of low cost, low latency, strong finality, and EVM compatibility makes MegaETH the optimal settlement layer for TrueShuffle's requirements. The architecture is designed with chain-agnosticism as a secondary goal: the smart contracts are standard Solidity with no MegaETH-specific dependencies, enabling redeployment on any EVM-compatible chain should the need arise.

9 Game Theory & Mechanism Design

The TrueShuffle protocol does not exist in a strategic vacuum. Poker is, by its very nature, the canonical game of imperfect information — the problem domain that motivated the development of game theory itself. This chapter analyzes TrueShuffle through the lens of game theory and mechanism design, demonstrating that the protocol satisfies key desiderata: incentive compatibility, individual rationality, budget balance, and collusion resistance in the dealing phase. We further analyze the emergent dynamics of the autonomous bot ecosystem, which constitutes a meta-game whose strategic landscape is shaped by evolutionary pressures.

9.1 Poker as an Imperfect Information Game

The formal study of strategic interaction began with poker. John von Neumann's seminal 1928 paper on the theory of parlor games [12] was motivated explicitly by the challenge of reasoning under uncertainty — a challenge that poker embodies more purely than perhaps any other game. In poker, each player possesses private information (hole cards) that is hidden from opponents, the game tree branches at chance nodes (community cards) whose outcomes are unknown at decision time, and the optimal action depends not merely on one's own holdings but on a probabilistic model of opponents' hidden states.

Formally, poker is a finite extensive-form game of imperfect information. A game state comprises the public board cards B, each player's private hole cards h_i, the current pot P, the betting history $\mathscr{H}$, and each player's remaining stack s_i. Player i's information set at any decision point consists of all game states consistent with i's observations: $\mathscr{I}_i = \{(B, h_i, P, \mathscr{H}, s_i)\}$. Critically, player i does not observe h_j for $j \neq i$, and therefore must reason about opponents' likely holdings through Bayesian updating on the observed betting history.

In two-player (heads-up) poker, the minimax theorem guarantees the existence of a Nash equilibrium — a strategy profile in which neither player can improve their expected payoff by unilaterally deviating [13]. This equilibrium strategy, known in the poker community as Game Theory Optimal (GTO) play, is characterized by mixed strategies that make the opponent indifferent between their available actions. A player executing GTO strategy perfectly cannot lose in expectation against any opponent, regardless of the opponent's strategy. The equilibrium is unique in the sense that all Nash equilibria yield the same expected payoffs for both players.

In multiplayer poker ($n > 2$), the situation is fundamentally more complex. While Nash's existence theorem guarantees that an equilibrium exists in any finite game, several complications arise. First, Nash equilibria in multiplayer games are not interchangeable: different equilibria may yield different payoffs for the same player, and there is no canonical method for selecting among them. Second, even computing an approximate Nash equilibrium in a general multiplayer game is PPAD-complete, suggesting that no polynomial-time algorithm exists under standard complexity-theoretic assumptions. Third, a Nash equilibrium provides no guarantee of individual rationality in multiplayer settings — a player following an equilibrium strategy may still lose to a coalition of opponents coordinating off the equilibrium path.

Given these computational barriers, practical poker AI has turned to regret minimization as an alternative solution concept. Counterfactual Regret Minimization (CFR), introduced by Zinkevich et al. [14], iteratively refines a strategy profile by minimizing each player's cumulative regret — the difference be-

tween the player's actual payoff and the payoff they would have achieved by always selecting their best action in hindsight. Over T iterations, CFR produces a strategy whose exploitability (the maximum an opponent can gain by deviating) converges at a rate of $O(1/\sqrt{T})$. CFR and its successors have proven remarkably effective: Libratus (2017) and Pluribus [15] achieved superhuman performance in heads-up no-limit and six-player no-limit Hold'em respectively, demonstrating that approximate equilibrium strategies can be computed for poker variants with game trees exceeding 10^{164} states.

9.2 TrueShuffle's Bot Ecosystem as a Game

The TrueShuffle platform supports an ecosystem of autonomous bots, each parameterized by a DNA configuration that encodes strategic preferences: aggression, tightness, bluff frequency, and a prioritized rule system. This ecosystem constitutes a meta-game — a game whose strategies are themselves poker strategies — that operates on a longer time scale than any individual hand or session.

We model this meta-game formally. Let $\mathcal{S}$ denote the space of all possible bot configurations (DNA vectors and rule sets). Each bot i selects a strategy $s_i \in \mathcal{S}$. The payoff function for bot i is its expected profit rate against the field:

$$u_i(s_i, s_{-i}) = \mathbb{E}\left[\text{profit}_i \mid \text{DNA}_i = s_i, \text{opponents} = s_{-i}\right]$$

where s_{-i} denotes the strategy profile of all other bots in the ecosystem. This payoff depends on the distribution of opponents that bot i encounters, which is determined by the table assignment mechanism and the population composition.

The meta-game is a repeated game with strategy modification between rounds. After each session, bots may adjust their DNA parameters based on performance data, effectively selecting a new strategy for the next round. The dynamics of this process depend on the adaptation mechanism, the information available to each bot, and the rate at which the population evolves.

A natural question arises: does this ecosystem converge to a Nash equilibrium of the meta-game? That is, does there exist a strategy profile $s^* = (s_1^*, s_2^*, \ldots, s_n^*)$ such that no bot can improve its expected profit by unilaterally changing its DNA?

9.3 Convergence vs. Arms Race Dynamics

The question of convergence admits two qualitatively different outcomes, each with distinct implications for the platform.

Outcome 1: Convergence. The bot population converges to an approximate Nash equilibrium in which all bots play near-GTO strategies. In this regime, no bot can systematically exploit another, and profits are determined primarily by rake efficiency — the ability to minimize losses to the rake while extracting marginal edges from small strategic imperfections. The ecosystem stabilizes, and the primary differentiator among bots becomes computational efficiency rather than strategic innovation.

Outcome 2: Arms Race. The bot population enters a perpetual cycle of exploitation and counter-exploitation. When a strategy s_A becomes prevalent, an exploitative strategy s_B emerges that targets s_A's weaknesses. As s_B gains adoption, a counter-exploit s_C appears, and the cycle continues indefinitely. The meta-game never reaches equilibrium; instead, the strategic landscape exhibits chaotic or

oscillatory dynamics.

The theoretical analysis of these outcomes draws on the theory of fictitious play and its extensions. The bot adaptation mechanism — adjusting DNA parameters based on observed opponent behavior and outcome data — implements a form of approximate fictitious play, in which each bot updates its strategy based on an empirical model of the opponent distribution. A classical result in game theory establishes that fictitious play converges to Nash equilibrium in two-player zero-sum games. However, this convergence guarantee does not extend to multiplayer games. Shapley [16] demonstrated that fictitious play can cycle indefinitely in certain 3×3 games, and subsequent work has shown that convergence failures are generic rather than pathological in games with three or more players.

The practical expectation for TrueShuffle lies between these extremes. We anticipate convergence within skill bands — groups of bots with similar capability levels — but persistent arms race dynamics across skill bands. Within a band, bots face similar opponents and converge to locally optimal strategies. Across bands, however, strategies that are effective against weaker opponents may be exploitable by stronger ones, creating a stratified landscape with distinct strategic regimes at each level.

This mixed outcome is, in fact, beneficial for the platform. Pure convergence would render the bot ecosystem static and uninteresting. Pure chaos would make strategic investment futile. The intermediate regime — stable within levels, dynamic across levels — rewards continuous improvement while maintaining a meaningful skill ladder.

9.4 Incentive Compatibility

A critical property of any protocol for competitive play is incentive compatibility: rational participants should have no motivation to deviate from the prescribed protocol. We establish this property for TrueShuffle's shuffle and deal phases.

Theorem 8.1 (Incentive Compatibility of Honest Participation). *Under the TrueShuffle protocol, honest participation in the shuffle, lock, and deal phases is a weakly dominant strategy for all players.*

Proof. Consider a player P_i deciding whether to follow the protocol honestly or to deviate. We analyze the two components of the protocol in which deviation is conceivable: the shuffle phase and the key reveal phase.

Shuffle phase. In the shuffle phase, each player receives an encrypted deck, applies a permutation π_i and re-encrypts each card with their secret scalar k_i. A dishonest player might attempt to apply a non-random permutation (e.g., the identity permutation) or to encrypt non-uniformly. However, the composed permutation is $\pi = \pi_m \circ \pi_{m-1} \circ \cdots \circ \pi_1$. If at least one player P_j applies a uniformly random permutation π_j, the composed permutation π is uniformly random regardless of all other players' choices. Since each player's permutation is hidden by encryption, a deviating player cannot target their non-random permutation to cancel out another player's randomness. The deviating player therefore gains no informational advantage.

Key reveal phase. During card dealing, a player is asked to reveal specific per-card decryption keys. A dishonest player might attempt to reveal incorrect keys. However, all keys are committed on-chain during the `KEY_COMMIT` phase via $K_i^{\text{pub}} = k_i \cdot G$, where G is the Secp256k1 generator. A revealed key k_i is verified against this commitment by checking $k_i \cdot G \stackrel{?}{=} K_i^{\text{pub}}$. Revealing an incorrect key fails this verification, triggering the `abortAndSlash` mechanism, which forfeits the deviating player's escrowed

deposit. Thus, the expected payoff of deviating (loss of deposit) strictly dominates the payoff of honest play only in the impossible case where the player benefits more from the deviation than from their entire escrowed stake.

Since deviating provides no informational advantage in the shuffle phase and results in deposit forfeiture in the key reveal phase, honest participation weakly dominates deviation. No rational player has an incentive to deviate from the protocol. □

9.5 Mechanism Design Properties

Beyond incentive compatibility, the TrueShuffle protocol satisfies several classical desiderata from mechanism design theory. We analyze each in turn.

Individual Rationality. A mechanism is individually rational if every participant's expected utility from participating is at least as great as their utility from non-participation (their outside option). In TrueShuffle, a player's expected utility from sitting at a table is:

$$\mathbb{E}[u_i] = \mathbb{E}[\text{winnings}_i] - \text{rake}_i - \text{gas}_i$$

For a player with a strategic edge (positive expected winnings), participation is individually rational provided the edge exceeds the combined rake and gas costs. For a player with no edge, the expected utility is negative, and non-participation is rational. This is precisely the correct outcome: a fair mechanism should not guarantee profit to unskilled participants, but should guarantee that no player is systematically disadvantaged by the mechanism itself (as opposed to by their own strategic inferiority). The fairness of the dealing protocol ensures that any disadvantage arises from strategic play, not from protocol manipulation.

Budget Balance. The mechanism is weakly budget-balanced: the platform collects rake from each pot but does not create or destroy chips. The conservation law holds:

$$\sum_{i=1}^{n} \Delta s_i + \text{rake} = 0$$

where Δs_i is the change in player i's stack over a hand. The total chips in play (plus accumulated rake) remain constant. This property ensures that the platform operates as a venue, not as a counterparty — a critical distinction from casino games where the house has a mathematical edge built into the rules.

Truthfulness in Showdown. At showdown, players must reveal their decryption keys to expose their hole cards. This revelation is truthful (players cannot misrepresent their hands) because:

1. **Not revealing** results in a timeout, which is treated as a fold. The non-revealing player forfeits their pot equity — a strictly dominated action when the player holds a winning hand.
2. **Revealing incorrect keys** is detectable. The verification $k_i \cdot G \stackrel{?}{=} K_i^{\text{pub}}$ fails, triggering dispute resolution and deposit forfeiture. The expected cost of detection (loss of entire deposit) far exceeds any potential gain from misrepresentation.

Thus, truthful revelation is the unique rational action at showdown.

Collusion Resistance. The TrueShuffle protocol provides strong collusion resistance in the dealing phase: no coalition of $k < m$ players can determine the identity of any card not dealt to a coalition member. This is a direct consequence of the cryptographic security analysis presented in Chapter 4. In the strategic play phase, collusion resistance is weaker: colluding players can share hole card information out-of-band and coordinate betting actions. This limitation is inherent to poker as a game and is not specific to TrueShuffle. However, the on-chain hand history provides a complete, immutable record of all actions, enabling statistical detection of collusion patterns — an audit capability that centralized platforms lack.

9.6 The Bot DNA Meta-Game

The autonomous bot framework gives rise to a richly structured meta-game whose dynamics merit detailed analysis. Each bot's DNA defines a point in a continuous strategy space:

$$\mathcal{S} = \{(a, t, b) \in [0, 100]^3\}$$

where a denotes aggression, t denotes tightness, and b denotes bluff frequency. The rule system adds discrete strategic dimensions, making the full strategy space a hybrid continuous-discrete object: $\mathcal{S}_{\text{full}} = [0, 100]^3 \times \mathcal{R}$, where $\mathcal{R}$ is the set of all valid rule configurations.

The learning loop implements a gradient-like ascent in strategy space. After each session, a bot's DNA is updated according to:

$$s_i^{t+1} = s_i^t + \alpha \nabla_{s_i} u_i(s_i^t, s_{-i}^t)$$

where α is a learning rate (bounded to prevent overcorrection) and $\nabla_{s_i} u_i$ is an empirical estimate of the payoff gradient with respect to bot i's strategy parameters. In practice, this gradient is estimated from performance data: if increasing aggression correlates with higher win rates in the recent session, the aggression parameter is nudged upward.

With thousands of bots adapting simultaneously, this process approximates a population dynamics model. Let $f(s, t)$ denote the density of bots playing strategy s at time t. The evolution of this distribution is governed by a replicator-like equation:

$$\frac{\partial f(s,t)}{\partial t} = f(s,t) \cdot [u(s, f(\cdot, t)) - \bar{u}(f(\cdot, t))]$$

where $u(s, f(\cdot, t))$ is the expected payoff of strategy s against the current population distribution and $\bar{u}(f(\cdot, t))$ is the population average payoff. Strategies that outperform the average grow in frequency; those that underperform shrink.

This connection to evolutionary game theory is not merely analogical. The bot ecosystem exhibits the three hallmarks of an evolutionary system: variation (bots differ in their DNA configurations), selection (bots that perform well are copied by other users, while poorly performing configurations are abandoned), and heredity (successful DNA configurations are replicated, sometimes with modification, by users who observe the leaderboard). The resulting dynamics — stable equilibria punctuated by

invasions of novel strategies — mirror the evolutionary stable strategies (ESS) framework introduced by Maynard Smith and Price [17].

The practical implication for platform design is significant. The bot meta-game is self-sustaining: it generates strategic diversity, rewards innovation, and creates a continuously evolving competitive landscape. Unlike human-only poker, where strategic evolution occurs on time scales of months to years, the bot ecosystem can undergo rapid strategic shifts, producing a dynamic environment that sustains engagement and rewards the development of increasingly sophisticated strategies.

10 Autonomous Bot Framework

The TrueShuffle platform is designed from its inception to support autonomous agents as first-class participants. Rather than treating bots as adversaries to be detected and banned — the prevailing approach in traditional online poker — TrueShuffle embraces bot play as the primary mode of competitive interaction. This chapter specifies the autonomous bot framework in detail, covering the design philosophy, the Bot DNA specification language, the decision engine architecture, the hand classification system, and the mechanisms for learning, adaptation, and AI-assisted bot creation.

10.1 Design Philosophy

The bot framework is governed by four design principles that balance accessibility, expressiveness, determinism, and adaptability.

Accessibility. Bot creation must be accessible to users without programming expertise. The strategic intent of a bot — "play tight-aggressive, three-bet light from late position, never bluff into multiple opponents" — should be expressible in natural language or through a visual interface, with AI systems handling the translation to executable configuration. This principle ensures that the platform's competitive landscape is shaped by strategic insight rather than by the ability to write code.

Expressiveness. The configuration language must be powerful enough to encode sophisticated, multi-dimensional strategies. Simple trait-based parameterization (aggression, tightness) provides a useful baseline, but real poker strategy requires conditional reasoning: the correct action depends on position, hand strength, pot odds, stack depth, opponent tendencies, and the specific sequence of prior actions. The rule system provides this conditional expressiveness through a priority-ordered rule evaluation mechanism.

Determinism. Given identical game states and bot configurations, the decision engine must produce identical outputs. This determinism is essential for auditability: if a dispute arises about whether a bot acted correctly, any observer can reproduce the decision by replaying the game state through the decision engine. Non-deterministic elements (such as mixed strategies expressed as probability distributions) are resolved using a deterministic pseudorandom function seeded by the hand identifier and the player's secret key, ensuring reproducibility without sacrificing strategic randomization.

Data-Driven Adaptation. Bots should improve automatically based on performance data. The learning system analyzes hand histories, identifies systematic leaks, and proposes DNA modifications. This closed-loop adaptation process transforms each bot from a static strategy into an evolving agent that responds to the competitive environment.

10.2 Bot DNA Specification

A bot's strategic identity is encoded in a JSON configuration called its DNA. The DNA comprises two components: a personality vector that defines baseline behavioral tendencies, and a rule set that encodes conditional strategic overrides.

```
{
  "personality": {
    "aggression": 72,
    "tightness": 65,
    "bluffFrequency": 35
  },
  "rules": [
    {
      "priority": 100,
      "conditions": { "hand": "premium", "position": "any" },
      "action": "raise",
      "sizing": "pot"
    },
    {
      "priority": 90,
      "conditions": { "hand": "strong", "position": "late", "facingRaise": false },
      "action": "raise",
      "sizing": "2.5bb"
    },
    {
      "priority": 80,
      "conditions": { "hand": "weak", "potOdds": ">30%", "street": "flop" },
      "action": "call",
      "sizing": null
    }
  ]
}
```

Personality traits define a bot's default behavioral tendencies, serving as a fallback when no rule in the rule set matches the current game state. Each trait is an integer in the range $[0, 100]$:

- **Aggression** ($a \in [0, 100]$): the propensity to bet and raise rather than check and call. A bot with $a = 90$ will frequently apply pressure; one with $a = 20$ will play passively, preferring to check and call.
- **Tightness** ($t \in [0, 100]$): the selectivity of starting hand requirements. A bot with $t = 85$ enters few pots but does so with strong holdings; one with $t = 25$ plays a wide range of hands.
- **Bluff frequency** ($b \in [0, 100]$): the propensity to bet or raise without a strong hand. This trait governs the rate at which the bot attempts to win pots through aggression rather than hand strength.

Rules encode conditional strategy overrides that take precedence over personality-driven behavior.

Each rule consists of four fields:

- **Priority** (integer, higher values evaluated first): determines the evaluation order. When multiple rules match, the highest-priority rule is selected.
- **Conditions** (structured predicate): a conjunction of conditions that must all be satisfied for the rule to match. Available condition dimensions include hand classification (premium, strong, medium, weak), position (early, middle, late, blinds, or specific seats such as BTN, CO, HJ, LJ, SB, BB), pot odds (numeric threshold with comparison operator), stack-to-pot ratio (SPR), street (pre-flop, flop, turn, river), number of opponents remaining, facing a raise (boolean), and opponent tendency descriptors (tight, loose, aggressive, passive) derived from session statistics.
- **Action** (enum): the action to take if all conditions are met. Valid actions are `fold`, `check`, `call`, `raise`, and `allIn`.
- **Sizing** (string or null): for raise actions, specifies the bet size. Supported formats include fixed amounts (`"2.5bb"`, `"100"`), pot fractions (`"pot"`, `"0.67pot"`, `"0.33pot"`), and the keyword `"allIn"`. For non-raise actions, this field is null.

The rule evaluation semantics are straightforward: rules are sorted by descending priority, each rule's conditions are evaluated against the current game state, and the first rule whose conditions are fully satisfied determines the action. If no rule matches, behavior falls back to the personality-driven decision model.

10.3 Bot Brain — Decision Engine

The decision engine is the computational core that translates a bot's DNA and the current game state into a concrete action. It is implemented in TypeScript for portability, ecosystem compatibility, and deterministic execution guarantees. The engine follows a six-stage pipeline.

Stage 1: Receive. The engine receives a game state update from the coordinator via WebSocket. The message is a JSON payload containing the current street, board cards (if any), the bot's hole cards (decrypted via the mental poker protocol), the pot size, current bet to call, each player's stack, the action history for the current street, and summary statistics for each opponent (computed from prior hands in the session).

Stage 2: Classify. The engine classifies the bot's hand strength using precomputed lookup tables. Pre-flop classification is based on the 169 canonical starting hand groups (13 pocket pairs, 78 suited combinations, 78 offsuit combinations). Post-flop classification incorporates board texture, using a combination of hand rank evaluation and equity estimation against a range of likely opponent holdings.

Stage 3: Evaluate Position. The engine determines the bot's position relative to the dealer button. Position is a critical strategic variable: later positions act with more information (having observed earlier players' actions) and therefore can profitably play wider ranges. The canonical positions are BTN (button, latest position), CO (cutoff), HJ (hijack), LJ (lojack), SB (small blind), and BB (big blind).

Stage 4: Match Rules. The engine iterates through the bot's rule set in descending priority order. For each rule, every condition is evaluated against the current game state. A rule matches if and only if all its conditions are satisfied. The first matching rule determines the action and sizing. This evaluation is deterministic and efficient: with a typical rule set of 20–50 rules, the matching process completes in microseconds.

Stage 5: Fallback to Personality. If no rule matches the current game state, the engine falls back to the personality-driven decision model. The probability of each action is computed as a function of the personality traits and the current hand strength:

$$P(\text{raise}) = \frac{a}{100} \times f(h)$$

$$P(\text{fold}) = \frac{t}{100} \times (1 - f(h))$$

$$P(\text{call}) = 1 - P(\text{raise}) - P(\text{fold})$$

where a is the aggression trait, t is the tightness trait, and $f(h) \in [0, 1]$ is a monotonically increasing function of hand strength h. The bluff frequency trait modulates $f(h)$ by adding a bluff component: when the raw hand strength is low, a random draw (seeded deterministically from the hand identifier and player's secret key) with probability $b/100$ replaces $f(h)$ with an elevated value, causing the bot to act as though its hand were stronger than it is.

In practice, all probabilities are clamped to $[0, 1]$ and normalized to ensure $P(\text{raise}) + P(\text{fold}) + P(\text{call}) = 1$.

The action is then selected by a deterministic draw from the resulting probability distribution, using the hand identifier and player's secret key as a deterministic seed.

Stage 6: Size the Bet. If the selected action is a raise, the engine determines the bet size. Rule-specified sizing is used directly. For personality-driven raises, the sizing is computed as a function of the aggression trait, the pot size, and the hand strength:

$$\text{betSize} = \text{pot} \times \left(0.33 + 0.67 \times \frac{a}{100}\right) \times g(h)$$

where $g(h)$ is a scaling function that increases the bet size with hand strength (value betting) and occasionally produces large bets with weak hands (bluffing), governed by the bluff frequency trait.

All decisions are logged with a complete reasoning chain: the game state, the matched rule (if any), the computed probabilities, the selected action, and the sizing calculation. This log enables post-session analysis and is stored immutably for audit purposes.

10.4 Hand Classification System

The hand classification system assigns each starting hand to one of four tiers, which serve as the primary input to both the rule matching system and the personality-driven fallback. This classification draws on established starting hand theory in Texas Hold'em.

Tier	Example Hands	Pre-flop Strategy
Premium	AA, KK, QQ, JJ, TT, AKs, AKo, AQs	Always raise or re-raise
Strong	99–77, AJs–ATs, KQs, AQo	Open-raise; call three-bets in position

Tier	Example Hands	Pre-flop Strategy
Medium	66–22, suited connectors (98s–54s), suited aces (A5s–A2s)	Position-dependent; open from late position
Weak	All remaining hands	Fold unless receiving sufficient pot odds

Post-flop hand classification is more nuanced, incorporating the interaction between the player's hole cards and the community board. The engine evaluates made hand strength (using a fast 7-card evaluator based on lookup tables), draw potential (flush draws, straight draws, combination draws), and board texture (wet vs. dry, paired vs. unpaired, monotone vs. rainbow). These factors are combined into a composite hand strength score $h \in [0, 1]$ that drives both rule matching and personality-driven decisions.

The classification system is designed to be extensible. Advanced users can define custom hand groupings and override the default tier assignments through the rule system, enabling strategies that deviate from conventional hand strength hierarchies — for example, a bot that treats small suited connectors as premium hands in specific positional and stack-depth contexts.

10.5 Learning & Adaptation

The learning subsystem closes the loop between play and strategy, enabling bots to improve automatically based on empirical performance data. After each session, the system executes a multi-stage analysis pipeline.

Performance Metrics. The system computes standard poker performance metrics from the session's hand histories: BB/100 (big blinds won per 100 hands, the canonical measure of win rate), VPIP (voluntarily put money in pot, a measure of hand selection looseness), PFR (pre-flop raise percentage), three-bet percentage, continuation bet percentage, fold-to-continuation-bet percentage, and showdown win rate. These metrics are computed both in aggregate and segmented by position, hand tier, and opponent type.

Opponent Modeling. Per-opponent statistics are maintained across sessions: VPIP, PFR, three-bet percentage, aggression factor (ratio of bets and raises to calls), and fold-to-pressure frequency. These statistics are used to classify opponents into archetypes (tight-passive, tight-aggressive, loose-passive, loose-aggressive) and to adjust the bot's strategy accordingly. Against a tight opponent who folds frequently to aggression, the bot increases its bluff frequency; against a loose-passive opponent who calls too often, the bot reduces bluffs and increases value bets.

Expected Value Tracking. Every decision point is tracked against its expected value:

$$\text{EV}(\text{action}) = P(\text{win}) \times \text{potSize} - P(\text{lose}) \times \text{betSize}$$

This formulation is simplified for heads-up scenarios; multi-way pots use a more detailed calculation that accounts for multiple opponents' ranges and the possibility of split pots. Here, $P(\text{win})$ and $P(\text{lose})$ are estimated from hand equity calculations and opponent range modeling. Decisions with negative EV are flagged as potential leaks.

Leak Detection. The leak detection approach is conceptually related to regret minimization [14]: the system identifies decisions where the bot's chosen action produced lower value than an alternative would have. The AI analysis engine identifies systematic mistakes — recurring decision patterns that produce negative expected value. Examples include: "calling three-bets out of position with medium-tier hands, losing 15 BB/100 in these spots," "continuation betting too frequently on wet boards against sticky opponents," or "failing to value bet the river with strong holdings, leaving an estimated 3 BB/100 on the table." Each identified leak is accompanied by a quantified impact estimate and a suggested corrective action.

DNA Evolution. Based on the analysis results, the system proposes modifications to the bot's DNA parameters. These modifications are bounded to prevent overcorrection: no trait may change by more than ±5 per session, and the system requires statistical significance (a minimum sample size and confidence interval) before proposing changes. For example, if a bot's bluff frequency is set to 40 but bluffs are losing money at a statistically significant rate, the system proposes reducing the bluff frequency to 35. Over multiple sessions, these incremental adjustments converge toward a locally optimal configuration for the bot's competitive environment.

Rule Generation. The learning system can generate new rules from observed winning patterns. If the hand history reveals that a specific action in a specific context produces consistently positive outcomes — for example, "three-betting suited connectors from the cutoff position won 8 out of 10 hands and produced a profit of 45 BB" — the system generates a candidate rule encoding this pattern and proposes it at an appropriate priority level. The user may accept, modify, or reject the proposed rule.

10.6 AI-Assisted Bot Creation

To fulfill the accessibility design principle, TrueShuffle provides an AI-assisted bot creation interface that enables users to specify strategic intent in natural language and receive a complete, executable Bot DNA configuration.

The creation flow proceeds as follows. The user provides a natural language description of their desired strategy: for example, "Build me a tight-aggressive bot that three-bets light from late position, plays cautiously out of position, and never slow-plays premium hands." A large language model (Gemini) parses this description, extracts the strategic intent, and generates a complete Bot DNA configuration — personality traits, and a rule set that encodes the specified conditional behaviors.

The generated configuration is presented to the user in the Bot Studio visual interface, where each trait and rule is displayed with a plain-language explanation of its effect. The user may iteratively refine the bot through continued conversation ("Make it more aggressive on the flop," "Add a rule to check-raise dry boards with strong draws") or through direct manipulation of the visual interface. Each modification is immediately reflected in the DNA configuration, and the user can preview the bot's decision logic against example game states.

This AI-assisted creation pipeline democratizes access to sophisticated strategy development, building on advances in superhuman poker AI [15] to inform the strategic foundations underlying generated configurations. A user with deep poker knowledge but no programming experience can create a bot that encodes nuanced, context-dependent strategy — something that previously required both strategic expertise and software engineering capability. Conversely, a user with limited poker knowledge can describe a general strategic archetype ("play like a rock," "be a maniac") and receive a reasonable

starting configuration that can be refined through play and learning.

The Bot Studio also provides analysis tools: strategy comparison (how does this bot's DNA differ from the current leaderboard leader?), expected performance estimation (based on historical data from similar configurations), and weakness analysis (what types of opponents are likely to exploit this strategy?). These tools enable informed iteration and reduce the time required to develop a competitive bot.

11 Scalability & Performance

A cryptographic poker protocol is of limited practical value if it cannot sustain the pace and throughput that real-time play demands. This chapter analyzes the computational, network, and on-chain costs of the TrueShuffle protocol, demonstrates that the overhead is negligible relative to poker's natural tempo, and outlines a scalability roadmap from the current centralized coordinator architecture to a fully decentralized peer-to-peer topology.

11.1 Computational Costs

The dominant computational cost of the TrueShuffle protocol is elliptic curve scalar multiplication on the Secp256k1 curve, which underlies all encryption, re-encryption, and decryption operations. Using the TypeScript `@noble/curves` library on modern consumer hardware (Apple M-series or Intel/AMD processors from 2022 onward), a single scalar multiplication completes in approximately 0.5 milliseconds. The `@noble/curves` library uses constant-time operations to prevent timing side-channel attacks, preserving the same security guarantees as native implementations.

The per-hand cryptographic cost is determined by the protocol phases:

- **Shuffle phase**: each player encrypts 52 card points with their secret scalar and applies a permutation. This requires 52 scalar multiplications, totaling $52 \times 0.5\text{ms} = 26\text{ms}$ per player.
- **Lock phase**: each player re-encrypts 52 card points with per-card keys, again requiring 52 scalar multiplications and $\approx$ 26ms per player.
- **Deal phase**: decrypting a single card requires one scalar multiplication per player who has encrypted it. For a 6-player table, decrypting one card costs $6 \times 0.5\text{ms} = 3\text{ms}$.

The total cryptographic overhead per hand scales linearly with the number of players m:

$$T_{\text{crypto}}(m) = 2m \times 26\text{ms} + d \times m \times 0.5\text{ms}$$

where d is the number of cards dealt during the hand (typically 9–23 depending on the variant and the number of players who reach showdown). For representative configurations:

Players	Shuffle + Lock	Deal (typical)	Total
2	104 ms	14 ms	~118 ms
6	312 ms	54 ms	~366 ms
9	468 ms	81 ms	~549 ms

Even the 9-player case completes in well under one second. Since poker's natural pace involves 10–30 seconds per player action, and the cryptographic operations occur during the dealing phase (before any

player decisions), the protocol overhead is imperceptible to participants. The shuffle and lock phases execute concurrently with the UI's dealing animation, fully masking the computational latency.

11.2 Network Overhead

The TrueShuffle protocol transmits encrypted card data as compressed elliptic curve points. Each compressed Secp256k1 point occupies 33 bytes (1 byte prefix + 32 bytes x-coordinate).

- **Shuffle message**: 52 compressed points = $52 \times 33 = 1{,}716$ bytes per player.
- **Lock message**: 52 compressed points = 1,716 bytes per player.
- **Deal message**: per-card key reveal = 32 bytes per card per player.
- **Commitment messages**: key commitments, shuffle hashes = ≈ 200 bytes per player.

For a 6-player table, the total protocol overhead per hand is approximately:

$$\text{Network}_{\text{total}} = 6 \times (1{,}716 + 1{,}716 + 200) + \text{deal keys} \approx 22\text{KB} + 2\text{KB} \approx 24\text{KB}$$

This is negligible by modern networking standards. A single high-definition video frame exceeds this by two orders of magnitude.

WebSocket latency between client and coordinator determines the perceived speed of protocol phases. Within a single cloud region, round-trip latency is typically under 50 ms. Cross-region communication incurs 100–200 ms, which remains acceptable for poker's pace of play. The protocol requires m sequential rounds of communication for the shuffle and lock phases (each player must process the deck in order), yielding a total communication latency of:

$$T_{\text{network}}(m) = 2m \times \text{RTT}$$

For $m = 6$ and RTT = 50ms, this is 600ms — again, imperceptible within the context of a dealing phase that typically spans 2–3 seconds including animations.

MegaETH transaction submission latency is approximately 100 ms: the RPC call to submit the transaction plus the approximately 10 ms block inclusion time on MegaETH's real-time block production schedule [4].

11.3 On-Chain Costs

TrueShuffle settles critical game events on MegaETH. The gas cost of each on-chain operation has been benchmarked on the MegaETH testnet:

Operation	Gas	Estimated Cost (USD)
`commitKeys`	~50,000	$0.0004
`submitShuffle` (hash only)	~30,000	$0.00024
`submitAction` (bet/call/raise)	~45,000	$0.00036
`resolveGame`	~120,000	$0.00096
`evaluateHand`	~80,000	$0.00064

A typical 6-player hand involves approximately 20 on-chain actions (key commits, shuffle hashes, 10–15 betting actions, pot settlement, and hand evaluation). The total gas cost per hand is approximately:

$$\text{Gas}_{\text{total}} \approx 6 \times 50{,}000 + 6 \times 30{,}000 + 15 \times 45{,}000 + 120{,}000 + 80{,}000 \approx 1{,}355{,}000 \text{ gas}$$

At MegaETH's current gas pricing, this translates to approximately $0.011 per hand. This cost is fully absorbed by the platform's standard 2.5% rake: for a hand with a 20 BB pot at $0.02 per BB, the rake is $0.01 — comparable to the gas cost. At higher stakes, the gas cost becomes a vanishingly small fraction of the rake collected.

11.4 Throughput Analysis

The system's throughput is determined by the coordinator's capacity to manage concurrent tables, route WebSocket messages, and submit on-chain transactions.

Single coordinator capacity. A single coordinator instance (running on a modern cloud VM with 8 vCPUs and 16 GB RAM) can manage approximately 500 concurrent tables. The bottleneck is WebSocket message processing and game state management, not cryptographic computation (which occurs client-side) or on-chain settlement (which is asynchronous). Each table produces approximately one hand every 2 minutes on average (accounting for player decision time, dealing, and between-hand pauses), yielding:

$$\text{Throughput}_{\text{coordinator}} = 500 \times \frac{1}{2} = 250 \text{ hands/minute} = 15{,}000 \text{ hands/hour}$$

Horizontal scaling. Coordinators are stateless with respect to each other: each coordinator manages a disjoint set of tables, and all persistent state resides on-chain. Adding a second coordinator doubles capacity to 1,000 tables and 30,000 hands per hour. The scaling is linear up to the point where on-chain transaction throughput becomes the bottleneck. MegaETH's target throughput of 100,000 transactions per second provides ample headroom: even at 100 coordinators (1.5 million hands per hour), the on-chain transaction rate is approximately 375 transactions per second — well within MegaETH's capacity.

11.5 Scalability Roadmap

The architecture is designed for incremental decentralization across four phases.

Phase 1 (Current): Single Coordinator, MegaETH Settlement. A single coordinator instance manages all tables, routes all messages, and submits transactions to MegaETH. This architecture prioritizes simplicity and rapid iteration. The coordinator is a trust-minimized relay: it cannot observe card values or manipulate outcomes, but it can censor messages (denial of service). Liveness depends on the coordinator's availability.

Phase 2: Multiple Coordinators, Load-Balanced Table Assignment. Tables are distributed across multiple coordinator instances via a load balancer. Each coordinator manages a subset of tables. A shared registry (on-chain or via a distributed coordination service) tracks which coordinator manages which table. Players connect to the appropriate coordinator based on their table assignment. This phase eliminates the single-coordinator bottleneck while maintaining the same trust model.

Phase 3: libp2p Direct Peer-to-Peer Communication. Players communicate directly via libp2p [11], with the coordinator reduced to a matchmaking and table assignment service. Once players are seated at a table, all protocol messages (shuffles, locks, key reveals) flow peer-to-peer, eliminating the coordinator as a potential censorship point. The coordinator is consulted only for table creation, player seating, and dispute initiation.

Phase 4: Fully Decentralized. The coordinator is eliminated entirely. Players discover each other via an on-chain table registry, negotiate table parameters through smart contract interactions, and communicate directly via libp2p. Dispute resolution is handled entirely on-chain. This architecture achieves full decentralization: no single entity can censor, delay, or manipulate any aspect of the protocol. The trade-off is increased complexity in matchmaking and connection establishment, which may increase the time to seat a full table.

Each phase preserves the core security guarantees of the mental poker protocol. The cryptographic properties — fair dealing, card secrecy, shuffle integrity — are independent of the communication topology. The scalability roadmap expands capacity and reduces trust assumptions without compromising the protocol's foundational guarantees.

12 Economic Model

The sustainability of the TrueShuffle protocol depends on an economic model that aligns the incentives of all participants: players (and their bots), the platform, and the broader ecosystem. This chapter specifies the revenue structure, bankroll mechanics, incentive alignment properties, player reward mechanisms, and cost structure, demonstrating that the protocol achieves a self-sustaining economic equilibrium in which platform profitability and player welfare are complementary rather than adversarial.

12.1 Revenue Structure

TrueShuffle's primary revenue source is rake — a percentage of each pot collected by the platform as a fee for providing the infrastructure, matchmaking, and settlement services.

The rake parameters are as follows:

- **Rate**: 2.5% of each pot.
- **Cap**: 3 big blinds (BB) per hand, regardless of pot size.
- **Collection mechanism**: the `PokerEscrow` smart contract automatically deducts the rake from the pot before distributing winnings to the victor. Rake is transferred atomically to the DAO treasury address as part of the `resolveGame` transaction.

The cap ensures that high-stakes pots do not incur disproportionate fees. For a pot of 120 BB, the uncapped rake would be 3 BB (2.5% × 120 = 3), which equals the cap. For pots exceeding 120 BB, the effective rake percentage decreases: a 200 BB pot incurs only 1.5% rake. This structure is competitive with traditional online poker platforms, which typically charge 3–5% rake with higher caps.

Revenue per coordinator can be estimated from throughput and average pot size. At 15,000 hands per hour with an average pot of 20 BB at a representative stake of $0.02 per BB:

$$\text{Revenue}_{\text{hourly}} = 15{,}000 \times \min(0.025 \times 20, 3) \times 0.02 = 15{,}000 \times 0.50 \times 0.02 = \$150/\text{hour}$$

At higher stakes or with larger average pots, revenue scales proportionally up to the cap.

12.2 Bankroll Economics

All player funds are held on-chain in the `BankrollManager` smart contract. Players deposit funds to their on-chain bankroll, from which buy-ins are deducted when joining a table. Winnings are credited back to the bankroll upon leaving a table or at the conclusion of each hand.

The platform defines table tiers with corresponding blind structures and buy-in ranges:

Tier	Blinds (BB/SB)	Buy-in Range	Target Audience
Micro	1/2	40–200 chips	New bots, experimentation
Low	2/5	100–500 chips	Learning and calibration
Mid	5/10	200–1,000 chips	Competitive play
High	25/50	1,000–5,000 chips	Elite bots

New users receive a starting bankroll of 10,000 chips at no cost, enabling immediate experimentation without financial commitment. This free bankroll serves as an onboarding mechanism: users can create, test, and iterate on bot configurations at the micro and low tiers before committing real funds. To prevent abuse, new accounts require wallet verification, and free chips are non-transferable until a minimum play threshold is met.

The buy-in range at each tier enforces a minimum and maximum stack depth relative to the blinds. The minimum buy-in (typically 20 BB) prevents players from engaging in excessively short-stacked play that distorts strategic dynamics. The maximum buy-in (typically 100 BB at lower tiers, 100–200 BB at higher tiers) limits the variance impact of any single hand and creates a more standardized strategic environment.

12.3 Incentive Alignment

The economic model is designed to align platform incentives with player interests — a structural property that distinguishes TrueShuffle from both traditional online poker and casino-style gambling.

The platform profits from volume, not from player losses. The rake is a fixed percentage of the pot, collected regardless of which player wins. The platform's revenue is proportional to the total amount wagered, not to the total amount lost by players. This creates a direct incentive for the platform to maximize the number of hands played, which in turn requires maintaining a large, active player base.

Fair dealing increases trust, which increases volume. If the platform were to manipulate card dealing — favoring certain accounts, engineering improbable outcomes, or otherwise compromising the integrity of the game — the inevitable discovery of such manipulation would destroy player trust and, with it, the volume on which revenue depends. The TrueShuffle protocol makes such manipulation cryptographically impossible, but the economic argument is independent of the cryptographic one: even a platform that *could* cheat would be economically irrational to do so, because the long-term revenue loss from destroyed trust would far exceed any short-term gain from manipulation.

Contrast with adversarial models. In casino-style gambling, the house has a built-in mathematical edge: the rules of the game guarantee that the expected value of play is negative for the player and positive for the house. This creates an inherently adversarial relationship. In poker, the platform has no edge in any individual hand — the rake is a transaction fee, not a house advantage. The platform's economic interest is in sustaining a fair, active, and growing ecosystem, which is precisely aligned with the players' interest in fair, competitive play.

12.4 Rakeback & Player Rewards

To incentivize sustained participation and reward high-volume players, TrueShuffle implements a rakeback and rewards program.

Volume-based rakeback. Players who generate significant rake through active play receive a percentage of their contributed rake returned to their bankroll. The rakeback rate increases with volume, creating a tiered incentive structure:

Monthly Rake Generated	Rakeback Rate
< 100 BB	0%
100–500 BB	10%
500–2,000 BB	15%
2,000–10,000 BB	20%
> 10,000 BB	25%

This structure reduces the effective rake for the platform's most active participants. A high-volume bot generating 10,000 BB in monthly rake pays an effective rake of approximately 1.875% (2.5% × 0.75) — competitive with the lowest rates offered by any online poker platform.

Leaderboard prizes. The top-performing bots on monthly and seasonal leaderboards receive bonus rewards funded from the DAO treasury. Leaderboard rankings are computed from BB/100 win rate with a minimum hand threshold to prevent variance-driven rankings. These prizes incentivize strategic excellence and provide a competitive motivation beyond pure profit extraction.

Purpose. The rakeback and reward programs serve two economic functions. First, they reduce the effective cost of play for the platform's most valuable users (high-volume players who generate the most revenue and liquidity). Second, they create non-monetary incentives (leaderboard status, competitive recognition) that increase engagement and retention.

12.5 Cost Structure

The platform's cost structure is characterized by low marginal costs and moderate fixed costs, yielding favorable unit economics at scale.

On-chain gas costs. As analyzed in Chapter 10, the gas cost per hand on MegaETH is approximately \$0.007–\$0.011, depending on the number of players and actions. This is a variable cost that scales linearly with volume.

Coordinator infrastructure. Each coordinator instance runs on a cloud VM (estimated cost: \$500/month for an 8-vCPU, 16 GB instance with adequate network bandwidth). A single coordinator

supports approximately 15,000 hands per hour, or roughly 10.8 million hands per month assuming 24/7 operation.

Break-even analysis. The break-even point can be computed from the fixed infrastructure cost and the net revenue per hand (rake minus gas):

$$\text{Break-even hands} = \frac{\text{Infrastructure cost}}{\text{Rake per hand} - \text{Gas per hand}}$$

At micro stakes (average pot 20 BB at \$0.02/BB), rake per hand is approximately \$0.01 and gas cost is approximately \$0.007, yielding net revenue of \$0.003 per hand:

$$\text{Break-even} = \frac{\$500}{\$0.003} \approx 167{,}000 \text{ hands/month} \approx 5{,}600 \text{ hands/day}$$

This is approximately 0.4% of a single coordinator's capacity, indicating that the infrastructure cost is recovered at very low utilization levels. At higher stakes, the break-even point decreases further: at mid stakes (average pot 20 BB at \$0.10/BB), rake per hand is approximately \$0.05, and break-even is reached at roughly 11,000 hands per month.

Marginal cost at scale. Beyond break-even, the marginal cost of an additional hand is dominated by gas (\$0.007–\$0.011), with negligible incremental infrastructure cost. This near-zero marginal cost structure enables aggressive pricing (low rake, high rakeback) while maintaining profitability, creating a sustainable economic model that scales favorably with adoption.

13 Governance & Upgradeability

The design of a decentralized protocol's governance framework presents a fundamental tension. On one hand, the capacity to upgrade and adapt is essential for responding to discovered vulnerabilities, evolving regulatory requirements, and community-driven feature requests. On the other hand, the entire value proposition of a trustless poker protocol rests on the assurance that the rules of the game will not change unilaterally after players have committed funds. TrueShuffle resolves this tension through a carefully stratified architecture that distinguishes between immutable core contracts — whose behavior is fixed permanently at deployment — and peripheral contracts that may be upgraded through a transparent, time-delayed governance process.

13.1 Immutability vs. Upgradeability Tradeoff

The smart contracts that constitute TrueShuffle's security foundation — PokerEscrow, PokerHandEvaluator, and MentalPokerGame — are deployed as immutable bytecode on MegaETH. They contain no proxy patterns, no admin keys, no `DELEGATECALL`-based upgrade mechanisms, and no owner-controlled state variables. Once deployed, their logic is fixed for the lifetime of the blockchain. This is not a limitation to be overcome; it is a deliberate and central design decision.

The rationale is straightforward. Players who buy in to a poker hand are entering a financial contract whose terms must be knowable and unalterable at the moment of commitment. If the hand evaluation logic could be modified between the deal and the showdown, the game would not be trustless — it would merely substitute one form of operator trust (the server) for another (the governance mechanism). The

immutability of core contracts provides a guarantee that no entity — not the protocol team, not a DAO, not a majority of token holders — can change the rules under which a player's funds are at risk.

This guarantee carries a cost: if a critical vulnerability is discovered in an immutable contract, the only remediation path is to deploy a new version of the contract and migrate users to it. There is no "hot fix" capability. We accept this tradeoff because the alternative — upgradeable core contracts — introduces an attack surface that is fundamentally incompatible with the trust model of a financial protocol. The history of decentralized finance is replete with exploits enabled by upgrade mechanisms: admin key compromises, malicious proxy implementations, and governance attacks that modify contract logic to drain funds [10]. By removing the upgrade mechanism entirely from core contracts, TrueShuffle eliminates this entire class of vulnerability.

Peripheral contracts — the Coordinator registry, table configuration contracts, and parameter stores — serve operational functions that do not directly custody funds or evaluate game outcomes. These contracts employ a timelock upgrade pattern: any proposed upgrade must be announced on-chain and subjected to a mandatory 7-day delay before execution. During this delay, the proposed new bytecode is publicly visible, enabling community review, security audit, and, if necessary, player exit. The timelock mechanism ensures that no upgrade can take effect covertly or instantaneously. Players who object to a proposed change have a full week to withdraw their funds and cease participation before the change takes effect.

The 7-day timelock is enforced by a `TimelockController` contract that queues upgrade transactions and reverts any attempt to execute them before the delay has elapsed. The controller itself is immutable, preventing any circumvention of the delay mechanism. This architecture provides upgradeability where it is operationally necessary while preserving the trust properties that players require.

13.2 Current Governance Model

TrueShuffle's governance in its initial deployment phase (Phase 1) is intentionally centralized in specific, limited respects. The protocol team controls the deployment of smart contracts, the operation of the coordinator server, and the configuration of initial table parameters. This centralization is transparent: deployment transactions are publicly verifiable on MegaETH, coordinator source code is open-source, and all parameter values are readable from on-chain state.

Critically, the protocol team's authority is bounded by the immutability of deployed contracts. The team cannot modify the hand evaluation algorithm, cannot alter escrow settlement logic, cannot redirect pot distributions, and cannot prevent dispute resolution from executing. The smart contracts have no admin keys — there is no `onlyOwner` modifier on any function that affects game outcomes or fund custody. The team's authority extends only to peripheral operations: registering new table configurations, updating coordinator endpoints, and adjusting operational parameters within ranges defined at deployment.

The coordinator server represents the sole centralized component of the live system. It relays WebSocket messages between players, manages table seating, and provides the real-time communication layer that the mental poker protocol requires. However, as established in the security analysis (Chapter 4), the coordinator cannot observe card data (which is end-to-end encrypted between players), cannot manipulate game outcomes (which are verified on-chain), and cannot misappropriate funds (which are custodied by immutable smart contracts). A compromised coordinator can affect liveness — stalling

games or denying service — but not safety.

Moreover, the coordinator's liveness monopoly is itself bounded: players can bypass the coordinator entirely by submitting protocol messages directly to the on-chain contracts. The dispute resolution mechanism is accessible without coordinator involvement, ensuring that even a completely unavailable coordinator cannot prevent players from recovering their escrowed funds.

Upgrade decisions during Phase 1 are made by the protocol team and communicated through public channels (governance forum, protocol blog, on-chain timelock announcements) prior to execution. The team commits to a policy of no surprise upgrades: every peripheral contract change is announced, explained, and subjected to the 7-day timelock without exception.

13.3 Progressive Decentralization Roadmap

The concentration of operational authority in the protocol team during Phase 1 is understood as a transitional arrangement, not an architectural steady state. TrueShuffle follows a progressive decentralization roadmap designed to systematically transfer control from the protocol team to the community and, ultimately, to no one.

Phase 2: Community Multisig. The protocol team's control over peripheral contract upgrades is transferred to a 3-of-5 community multisig. The five signers are drawn from independent entities: protocol team members, independent security researchers, and elected community representatives. The multisig retains the 7-day timelock constraint; no upgrade can be executed without both the requisite signatures and the passage of the delay period. This phase eliminates single-entity control over upgrades while maintaining the capacity for coordinated response to operational issues.

Phase 3: DAO Governance. The multisig is replaced by a decentralized autonomous organization (DAO) with token-weighted voting. The specific token mechanism (distribution, vesting, governance weight) is deferred to a separate tokenomics proposal, as the protocol's core value proposition is independent of any particular token design. The current design focuses on establishing the parameter boundaries and decision-making framework that any future governance token would operate within. Governance proposals undergo a structured lifecycle: submission, discussion period (7 days), voting period (7 days), timelock (7 days), and execution. The DAO's authority extends to protocol parameters — rake percentages, rake caps, minimum and maximum buy-in amounts, timeout durations, dispute bond amounts, and the addition of new game types — but explicitly excludes core contract modification, which remains impossible by construction. Quorum and supermajority requirements prevent governance attacks by small token concentrations. Vote delegation enables passive token holders to assign their voting weight to informed delegates.

Phase 4: Protocol Ossification. The final phase eliminates programmatic governance entirely. All contracts — core and peripheral alike — are rendered immutable. Parameter ranges are fixed at values determined by the DAO during Phase 3. The protocol becomes a public utility, operating without any controlling entity, upgradeable only through voluntary community migration to a successor protocol. Governance in Phase 4 is limited to social coordination: community consensus-building around potential successor protocols, ecosystem fund allocation through immutable distribution contracts, and informal coordination of coordinator operators.

The aspiration of Phase 4 is explicit: TrueShuffle should ultimately resemble Bitcoin in its governance properties — a protocol that runs without anyone in charge, whose rules are enforced by mathematics

and consensus rather than by any identifiable authority.

13.4 Parameter Governance

The separation between immutable security logic and governable operational parameters is not merely architectural convenience; it is a security boundary. TrueShuffle explicitly partitions its parameter space into two categories with fundamentally different governance properties.

Governable parameters include the rake percentage charged on pot distributions, the rake cap (maximum rake per hand), minimum and maximum buy-in amounts for each table tier, timeout durations for player actions and protocol phases, dispute bond amounts required to initiate on-chain dispute resolution, and the set of supported game variants and table configurations. These parameters are economic in nature: they affect the profitability, accessibility, and operational characteristics of the protocol but do not impact the fundamental security guarantees. Adjusting the rake from 3% to 5% changes the economics of play but does not compromise card secrecy or settlement integrity.

Non-governable parameters include hand evaluation logic (the ranking of poker hands and the resolution of ties), escrow invariants (the conservation law requiring that total escrowed funds equal total buy-ins at all times), cryptographic protocol rules (the sequencing of commit, shuffle, lock, deal, and reveal phases), commitment and verification algorithms (the hash functions and elliptic curve operations used for on-chain verification), and dispute resolution logic (the rules by which MentalPokerGame assigns fault and triggers escrow slashing). These parameters are security-critical: any modification could enable cheating, fund theft, or protocol subversion. They are embedded in immutable contract bytecode and are not subject to governance of any kind.

This separation ensures that the protocol can adapt its economic characteristics to market conditions, regulatory requirements, and community preferences, while its security properties remain provably fixed. A governance attack that captures the DAO and passes a malicious proposal can, at worst, set the rake to its maximum allowed value or adjust timeout durations to their boundary limits — inconvenient outcomes, but not catastrophic ones. The attacker cannot modify the rules of the game, cannot redirect funds, and cannot compromise the cryptographic protocol.

The parameter ranges themselves are defined at deployment and enforced by the immutable parameter store contract. For example, the rake percentage is constrained to the range $[0\%, 10\%]$, and any governance proposal that attempts to set it outside this range will be rejected by the contract. This bounded governance model provides a final safeguard against governance capture: even complete control of the DAO cannot push parameters beyond their predefined safety bounds.

14 Regulatory Considerations

The deployment of a decentralized poker protocol on public blockchain infrastructure raises novel regulatory questions that do not map cleanly onto existing legal frameworks. Traditional online poker regulation assumes a centralized operator: a licensed entity that controls the software, custodies player funds, enforces age and identity verification, and bears legal responsibility for compliance. TrueShuffle's architecture — in which card dealing is performed by a cryptographic protocol, fund custody is delegated to immutable smart contracts, and no single entity possesses the ability to observe or manipulate game outcomes — challenges this regulatory paradigm at its foundations. This chapter examines

the regulatory landscape, the distinction between protocol and platform, the classification of bot competition, responsible gaming considerations, and the compliance strategy that TrueShuffle adopts.

14.1 The Regulatory Landscape for Online Poker

The legality of online poker varies dramatically across jurisdictions, creating a fragmented regulatory environment with no global consensus.

In the United States, online poker regulation is determined at the state level following the Department of Justice's 2011 reinterpretation of the Wire Act [1]. As of this writing, online poker is legal and regulated in New Jersey, Pennsylvania, Michigan, Nevada, Delaware, West Virginia, and Connecticut, with additional states considering legislation. In regulated states, operators must obtain licenses from state gaming commissions, implement geolocation verification, maintain segregated player fund accounts, and submit to regular audits. In the remaining states, online poker for real money is either explicitly prohibited or exists in a legal gray area.

In the European Union, regulation varies by member state. The United Kingdom operates a comprehensive licensing regime under the UK Gambling Commission (UKGC), which requires operators to hold a remote gambling license, implement responsible gaming measures, and submit to technical standards testing. France, Italy, Spain, and Portugal operate ring-fenced markets with national licensing requirements. Germany implemented the Interstate Treaty on Gambling in 2021, establishing a federal licensing framework with strict deposit limits and mandatory cooling-off periods. Other EU member states range from permissive (Malta, Gibraltar) to restrictive (the Netherlands prior to 2021, Poland).

In Asia, the regulatory posture is overwhelmingly prohibitive. China, Japan, South Korea, and most Southeast Asian nations ban online gambling in all forms. Notable exceptions include the Philippines, which licenses offshore gaming operators through PAGCOR, and Macau, which permits land-based but not online gambling. India's regulation is determined at the state level, with some states classifying poker as a game of skill exempt from gambling prohibitions and others banning it outright.

The central regulatory question for TrueShuffle is whether a decentralized protocol — as distinct from a centralized platform — falls within the scope of existing gambling regulation. This question has no settled answer in any jurisdiction.

14.2 Decentralized Protocol vs. Centralized Platform

The distinction between TrueShuffle as a protocol and TrueShuffle as a platform is not merely semantic; it is architecturally fundamental and carries significant regulatory implications.

As a protocol, TrueShuffle is analogous to HTTPS, BitTorrent, or the Ethereum Virtual Machine itself: it is a set of rules and algorithms, implemented in open-source code, that enables a particular class of interaction. The smart contracts deployed on MegaETH are public infrastructure — they cannot be paused, censored, or shut down by any entity, including the protocol team. Any developer can build a client that interacts with these contracts, just as any developer can build a web browser that speaks HTTPS. The protocol does not custody funds (the smart contracts do, autonomously), does not observe card data (the commutative encryption ensures no party can), and does not control access (anyone with a blockchain wallet can interact with the contracts).

As a platform, however, TrueShuffle comprises identifiable services operated by an identifiable entity:

the coordinator server that relays WebSocket messages, the web application that provides the user interface, and the bot marketplace that facilitates strategy trading. These services have operators, domain names, hosting infrastructure, and, potentially, revenue streams. They are subject to the laws of the jurisdictions in which they operate and the jurisdictions whose residents they serve.

The legal strategy follows from this distinction. The protocol itself — the smart contracts, the cryptographic algorithms, the open-source libraries — is infrastructure. It is published under open-source licenses, and its deployment on a public blockchain is a speech act (the publication of code) rather than the operation of a gambling service. Specific deployments of the protocol — particular coordinator servers, particular web frontends, particular bot marketplaces — may constitute gambling services in certain jurisdictions and may require appropriate licensing.

This two-layer analysis is consistent with the regulatory treatment of other decentralized protocols. The Uniswap smart contracts, for example, are deployed on Ethereum and cannot be shut down, but the Uniswap Labs website (app.uniswap.org) complies with U.S. regulations by restricting access to certain tokens and implementing geo-blocking. TrueShuffle adopts an analogous posture: the protocol is permissionless and unstoppable; the services built atop it comply with applicable law.

14.3 Bot Competition vs. Gambling

TrueShuffle's bot arena mode introduces a classification question that is genuinely novel in gambling regulation: is a competition between autonomous software agents a form of gambling?

The argument that bot competition constitutes skill-based competition, rather than gambling, rests on several observations. First, the bot's strategy is deterministic — it is fully defined by its DNA configuration parameters prior to any hand being dealt. The bot does not make decisions based on intuition, emotion, or real-time judgment; it executes a fixed decision function parameterized by its DNA. Second, the skill resides in the design and optimization of the bot. A player who constructs a superior bot — one whose DNA encodes a more profitable strategy — will outperform inferior bots over a statistically significant sample, just as a superior chess engine outperforms inferior ones. Third, bot competitions are structurally analogous to activities that are not classified as gambling: algorithmic trading competitions, autonomous vehicle racing, chess engine tournaments (TCEC, Top Chess Engine Championship), and programming contests. In each case, the competitor's skill is embodied in the software they create, and the competition evaluates the quality of that software.

The counterargument acknowledges that the underlying poker game involves irreducible randomness in the card dealing process. A bot may execute a theoretically optimal strategy and still lose a given hand due to an unfavorable card distribution. Over a sufficiently large sample, skill dominates variance — this is the basis for poker's classification as a skill game in several jurisdictions — but the presence of per-hand randomness invites the application of gambling regulation.

The jurisdictional classification likely depends on how the relevant authority characterizes the human's role. If the human designs the bot and the bot plays autonomously, the activity resembles engineering competition more than gambling. If the human merely selects a bot from a marketplace and watches it play, the activity more closely resembles placing a bet on an outcome the human cannot influence. TrueShuffle's primary emphasis on bot creation and customization — the DNA framework, the AI-assisted strategy builder, the backtesting infrastructure — positions the platform toward the skill-based end of this spectrum.

14.4 Responsible Gaming Considerations

Regardless of regulatory classification, TrueShuffle is committed to responsible gaming principles that meet or exceed industry standards, adapting them to the unique characteristics of a decentralized platform.

Deposit limits. The PokerEscrow smart contract supports configurable deposit limits that cap the total amount a wallet address can have escrowed across all active tables. These limits are enforced at the contract level — they cannot be bypassed by the coordinator, the UI, or any off-chain component. Players can set personal limits below the contract maximum, and once set, a personal limit reduction takes effect immediately while a limit increase is subject to a 24-hour cooling-off period.

Session time tracking. The client application tracks continuous play duration and provides configurable notifications at intervals defined by the player (default: every 60 minutes). While session tracking cannot be enforced at the protocol level (a player can always connect with a different client), the reference client implementation includes prominent session duration displays and mandatory break reminders.

Voluntary self-exclusion. Players can invoke a self-exclusion function on the PokerEscrow contract that prevents their wallet address from joining any new table for a specified duration (minimum 24 hours, maximum 1 year). Self-exclusion is enforced on-chain and cannot be reversed before the expiration of the chosen period, even by the player themselves. This provides a commitment device that is stronger than any centralized platform's self-exclusion mechanism, which can typically be circumvented by contacting customer support.

Free play mode. TrueShuffle supports practice tables using non-transferable test tokens, enabling players to experience the full protocol without any financial exposure. The free play mode uses identical smart contracts and cryptographic protocols, ensuring that the gameplay experience is representative of real-money play.

Transparency. Perhaps the most significant responsible gaming feature of TrueShuffle is the unprecedented transparency of all game data. Every hand is recorded on-chain: the encrypted deck, the revealed cards, the betting actions, the pot distributions. Any player — or any observer — can independently verify the fairness of every hand ever played. This level of transparency is unmatched in the poker industry, where hand histories are proprietary, audit results are confidential, and players must trust the operator's assurances of fairness.

14.5 Compliance Strategy

TrueShuffle's compliance strategy operates at the application layer, preserving the protocol's permissionless nature while enabling regulatory compliance for specific deployments.

Geo-fencing. The coordinator server and web application implement IP-based and GPS-based geolocation verification, restricting access from jurisdictions where online poker is prohibited. This geo-fencing operates at the application layer and can, in principle, be circumvented by VPN usage. However, application-layer geo-fencing satisfies the regulatory requirements of most jurisdictions that mandate it, and the circumvention risk is no different from that faced by any web-based service. The protocol-layer smart contracts do not implement geo-fencing — they are jurisdiction-agnostic by design, accepting interactions from any valid blockchain address.

KYC/AML integration. For regulated markets that require Know Your Customer (KYC) and Anti-Money Laundering (AML) compliance, TrueShuffle supports integration with wallet-based identity verification services. Attestation protocols such as Ethereum Attestation Service (EAS) enable a licensed KYC provider to issue on-chain attestations to verified wallet addresses. The coordinator can then require a valid KYC attestation as a precondition for table access in regulated jurisdictions, without the protocol team itself collecting or storing personal data. This approach preserves user privacy (the protocol team never sees identity documents) while satisfying regulatory requirements (a licensed third party has verified the player's identity).

Tax reporting. All financial transactions — buy-ins, cash-outs, and net session results — are recorded immutably on-chain, providing a complete and auditable record for tax reporting purposes. The client application can generate standardized reports suitable for tax filing in major jurisdictions.

GDPR and data protection. Wallet addresses recorded on-chain may constitute pseudonymous personal data under the European Union's General Data Protection Regulation (GDPR) and similar data protection frameworks. TrueShuffle's architecture mitigates data protection concerns through deliberate minimization of on-chain data: the protocol stores only cryptographic commitments (hashes of encrypted deck states, key commitments) and settlement amounts. No plaintext personal information — names, email addresses, IP addresses, or identity documents — is recorded on-chain. The tension between the right to erasure (GDPR Article 17) and blockchain immutability is substantially mitigated by the fact that on-chain data contains no plaintext personal information, only hashes and encrypted values that are computationally infeasible to link to a natural person without access to off-chain mapping data. Application-layer operators who collect personal data (for example, through KYC processes) remain subject to GDPR obligations and must implement appropriate data handling, retention, and erasure policies independently of the on-chain protocol.

The fundamental principle underlying TrueShuffle's compliance strategy is separation of concerns. The protocol is neutral infrastructure — it enforces cryptographic fairness and financial integrity without reference to any jurisdiction's laws. Compliance with specific regulatory requirements is implemented at the application layer by identifiable service operators, who bear the legal responsibility for the services they provide. This architecture enables TrueShuffle to serve both regulated and unregulated markets through different application-layer deployments, all interacting with the same immutable protocol-layer contracts.

15 Future Work & Roadmap

TrueShuffle, as presented in this paper, constitutes a complete and deployable system for trustless poker on an EVM-compatible Layer 2. However, the protocol's architecture admits substantial extensions across multiple dimensions: game format diversity, decentralization depth, bot capability sophistication, cryptographic efficiency, platform reach, and foundational research. This chapter outlines the planned development trajectory and identifies open research questions whose resolution would materially advance the state of decentralized gaming.

15.1 Protocol Extensions

The current TrueShuffle implementation supports No-Limit Texas Hold'em for 2–10 players at a single table. Several protocol extensions are under active development or planned for near-term implemen-

tation.

Multi-table tournaments (MTT). Tournament poker represents a significant fraction of online poker volume and introduces structural requirements beyond those of cash games. An MTT protocol must manage escalating blind levels on a synchronized schedule, dynamically balance players across tables as eliminations occur, compute prize pool distributions according to predetermined payout structures, and handle table breaks and merges without interrupting ongoing hands. The mental poker protocol itself requires no modification — each table within the tournament operates an independent instance of the shuffle-lock-deal protocol. The extension lies in a tournament coordination layer: an on-chain contract that tracks player chip counts across tables, enforces blind level transitions, triggers table rebalancing according to a deterministic algorithm, and distributes prizes from a pooled escrow upon tournament completion. The primary design challenge is ensuring that table rebalancing — which requires moving a player's chip state from one table contract to another — executes atomically and without creating opportunities for chip duplication or loss.

Mixed games. Supporting poker variants beyond Texas Hold'em — including Pot-Limit Omaha, Short Deck (6+ Hold'em), Omaha Hi-Lo, Stud, Razz, and mixed-game rotations — requires modifications to the hand evaluation logic and the dealing protocol (Stud games, for instance, require multiple rounds of individual card deals with interleaved betting). The core mental poker protocol remains unchanged: commutative encryption, shuffle, lock, and selective decryption operate identically regardless of the specific game variant. The extension requires variant-specific hand evaluators deployed as immutable contracts, variant-specific dealing sequences encoded in the table configuration, and a variant registry that maps game identifiers to their corresponding evaluator and dealing contracts.

Private tables. Invitation-only tables with custom parameters (buy-in amounts, blind structures, player limits) enable home-game-style play within the trustless protocol framework. Private tables are implemented through an access control list stored in the table configuration contract, where the table creator specifies a set of authorized wallet addresses. The mental poker protocol operates identically; the access restriction applies only at the seating phase.

Spectator betting. A prediction market layer enables spectators to place side bets on hand outcomes, tournament placements, or player performance metrics. Because all game events are recorded on-chain and all outcomes are publicly verifiable, the resolution of prediction markets is fully automated: the market contract reads the outcome from the game contract and settles positions without any oracle or manual intervention. This extension transforms every poker hand into a publicly verifiable event upon which permissionless markets can operate.

15.2 Decentralization Milestones

The coordinator server is the sole centralized component of the TrueShuffle architecture. While the security analysis (Chapter 4) demonstrates that the coordinator cannot compromise card secrecy or game fairness, its role as a communication relay creates a single point of failure for liveness. The long-term roadmap targets complete coordinator elimination through progressive decentralization of the communication layer.

Coordinator elimination via P2P networking. The coordinator's primary function — relaying WebSocket messages between players — can be replaced by a peer-to-peer mesh network built on libp2p [11]. In this architecture, players discover one another through an on-chain registry (the exist-

ing Coordinator contract, repurposed as a player directory) and establish direct WebRTC connections for protocol message exchange. Public game events (table creation, seating availability, hand results) propagate through a gossip protocol, enabling any peer to observe game state without connecting to a central server. The principal technical challenges are NAT traversal (establishing direct connections between players behind firewalls and routers), peer discovery latency (finding and connecting to table participants without a central directory service), and message ordering (ensuring that all players observe protocol messages in a consistent sequence without a central sequencer). STUN/TURN relay servers provide NAT traversal as a degraded fallback, but the long-term goal is fully direct connectivity using techniques such as hole punching and relay circuit protocols from the libp2p stack.

IPFS-hosted frontend. The web application is deployed to IPFS with an ENS domain name, eliminating dependence on centralized web hosting. Content addressing ensures that the frontend code served to users is immutable and verifiable: the ENS record points to a specific IPFS content hash, and any modification to the frontend produces a different hash, detectable by the user's browser.

Competitive coordinator market. As an intermediate step before full P2P, the protocol supports multiple independent coordinator operators. Any entity can deploy and operate a coordinator server, earning fees from the tables it manages. Players choose coordinators based on reputation, latency, and fee structure. This competitive market eliminates the single-operator dependency while preserving the simplicity of the client-server communication model for users who prefer it. The on-chain Coordinator contract registers authorized coordinator endpoints and tracks their performance metrics.

15.3 Advanced Bot Capabilities

The DNA-based bot framework presented in this paper provides a parameterized strategy space that is expressive, interpretable, and accessible to non-programmers. Future development extends the bot framework along several axes of capability.

Deep learning integration. Neural network-based decision engines — trained on large hand history datasets via supervised learning or through self-play reinforcement learning — can operate alongside or in place of the rule-based DNA system. The bot SDK exposes a decision interface that accepts game state observations and returns an action distribution; any model architecture that implements this interface can serve as a bot's decision engine. Integration with frameworks such as PyTorch or TensorFlow enables bots whose strategies are learned rather than manually specified, opening the platform to the machine learning research community [15].

GTO solver integration. Game-theory optimal (GTO) strategies, computed by counterfactual regret minimization (CFR) algorithms [14], can be embedded in bots as precomputed lookup tables or as real-time solvers that compute Nash equilibrium strategies for specific game subtrees. A bot equipped with a GTO solver computes the theoretically unexploitable strategy for each decision point, then optionally deviates from GTO based on observed opponent tendencies. This architecture — GTO baseline with exploitative adjustments — mirrors the approach of state-of-the-art poker AI systems such as Pluribus [15].

Meta-learning. Bots that learn how to learn represent a further capability tier. A meta-learning bot adjusts its own learning rate, exploration parameters, and adaptation strategy based on higher-order feedback: how quickly it is converging, whether its opponent appears to be adapting to its adaptations, and whether the current game population rewards exploitation or robustness. Meta-learning bots

employ algorithms from the meta-reinforcement learning literature to optimize their learning process itself, enabling rapid adaptation to novel opponents and game conditions.

Cross-table learning. In a multi-tabling environment, a bot that operates simultaneously at multiple tables can transfer insights across tables. Statistical observations about the game population (aggregate tendencies, common strategy archetypes, prevailing rake structures) inform strategy adjustments at each individual table. Cross-table learning requires a shared state layer that aggregates observations across table instances and propagates strategic updates.

Bot marketplace. A decentralized marketplace enables users to buy, sell, license, and auction bot DNA configurations. Smart contracts enforce licensing terms: a DNA configuration sold with a non-exclusive license can be used by multiple buyers, while an exclusive license restricts usage to a single address. Revenue-sharing arrangements — in which the DNA creator receives a percentage of the bot's winnings — are enforced automatically by escrow contracts. The marketplace creates an economy around poker strategy, rewarding skilled bot designers and providing access to sophisticated strategies for users who prefer to compete without building their own bots.

15.4 Cryptographic Improvements

The commutative elliptic curve encryption scheme employed by TrueShuffle is secure and efficient, but several cryptographic enhancements would improve privacy, reduce communication overhead, or prepare the protocol for future threats.

Zero-knowledge proofs. The current protocol verifies shuffle correctness through commitment hashing and on-chain replay: the smart contract stores hashes of each shuffle output and verifies consistency during dispute resolution. Replacing this mechanism with zero-knowledge proofs of shuffle correctness — specifically, ZK proofs that the shuffle output is a valid re-encryption and permutation of the shuffle input — would provide stronger privacy guarantees. A player could prove that they shuffled correctly without revealing their permutation, eliminating even the theoretical possibility of permutation recovery through side-channel analysis. ZK-SNARK and ZK-STARK constructions for shuffle proofs exist in the literature, though their prover time remains a practical constraint for real-time applications.

Threshold encryption. The current protocol requires $O(mn)$ point-to-point key exchanges during the deal phase (m players, n cards dealt), as each player must transmit their per-card decryption key to the card's intended recipient. A (t, n) threshold encryption scheme would enable a card to be decrypted when any t of n players contribute their key shares, reducing the communication complexity and enabling fault-tolerant decryption that proceeds even if some players disconnect before completing the key exchange.

Post-quantum readiness. The security of TrueShuffle's commutative encryption rests on the hardness of the Elliptic Curve Discrete Logarithm Problem, which is vulnerable to quantum algorithms (specifically, Shor's algorithm running on a sufficiently large quantum computer). While large-scale quantum computers capable of breaking 256-bit ECDLP are not expected in the near term, prudent protocol design includes a migration path to post-quantum cryptographic primitives. Lattice-based commutative encryption schemes, while less mature than their elliptic curve counterparts, represent a promising direction for post-quantum mental poker. The modular architecture of TrueShuffle's cryptographic layer — in which the encryption primitive is abstracted behind a well-defined interface —

facilitates such migration without requiring changes to the protocol logic or smart contract infrastructure.

15.5 Platform Growth

Mobile client. The Tauri v2 framework [18] supports compilation to iOS and Android targets, enabling a native mobile experience for bot management, table observation, and (for human-play modes) direct participation. The mobile client communicates with the same coordinator and smart contracts as the desktop client, ensuring a unified game ecosystem across platforms.

API ecosystem. A comprehensive public API enables third-party developers to build tools and services atop the TrueShuffle protocol: hand history analyzers, strategy backtesting engines, real-time visualization dashboards, portfolio trackers for bot operators, and tournament management interfaces. The API exposes both real-time WebSocket streams (for live game data) and historical REST endpoints (for completed hands, player statistics, and bot performance metrics).

Educational platform. An interactive tutorial system teaches poker strategy through the lens of bot construction. Users learn fundamental concepts — pot odds, position, hand ranges, bet sizing — by implementing them as DNA parameters and observing their bots compete. This approach transforms strategy education from passive study into active engineering, aligning with TrueShuffle's thesis that bot building is a skill discipline.

Cross-chain settlement. While MegaETH provides the performance characteristics required for real-time mental poker, supporting additional Layer 2 networks enables geographic optimization (players connecting to the L2 with lowest latency to their region), asset diversity (settlement in various ERC-20 tokens or stablecoins native to different chains), and ecosystem breadth (tapping into the user bases of multiple L2 communities). Cross-chain settlement requires bridge contracts that lock funds on the source chain and mint corresponding representations on the destination chain, with the mental poker protocol executing on a single canonical chain.

15.6 Research Directions

Several open research questions, if resolved, would materially advance the theoretical foundations and practical capabilities of decentralized mental poker.

Formal verification. The smart contracts comprising TrueShuffle's on-chain settlement layer are candidates for formal verification using tools such as Certora, Halmos, or the K Framework. Formal verification would provide mathematical proofs that the contracts satisfy their specified invariants (fund conservation, correct hand evaluation, faithful dispute resolution) under all possible execution paths, strengthening the trust guarantees beyond what testing and auditing alone can provide.

Game-theoretic population dynamics. As the bot population grows, the strategic landscape evolves: the profitability of any given strategy depends on the distribution of strategies in the population. Analyzing these dynamics through the lens of evolutionary game theory [12], [13] would yield insights into equilibrium strategy distributions, the rate of strategic innovation, and the conditions under which strategy monocultures emerge or diversify.

Information-theoretic bounds on side-channel leakage. While TrueShuffle employs constant-time cryptographic operations and fixed-size messages to minimize side-channel leakage, a rigorous

information-theoretic analysis would quantify the maximum information an adversary can extract from observable protocol metadata (timing, message frequency, connection patterns) and establish whether these bounds are achievable by the current implementation.

Efficient multi-party shuffle protocols. The current protocol requires $O(n)$ sequential rounds for n players to complete the shuffle phase, as each player must receive the deck, encrypt and permute it, and pass it to the next player. Research into parallel shuffle protocols — leveraging techniques from multi-party computation, such as mixing networks with logarithmic depth — could reduce the round complexity to $O(\log n)$, significantly improving the protocol's scalability to large player counts and tournament formats with many simultaneous tables.

16 Conclusion

Online poker suffers from a trust deficit that is not incidental but structural. The client-server architecture upon which the entire industry is built concentrates absolute informational and financial authority in the platform operator: the server generates the deck, observes every card, evaluates every hand, and custodies every dollar. This architecture does not merely permit abuse — it guarantees that abuse is undetectable by the victims and costless for the perpetrator, up to the moment of external discovery. The Ultimate Bet superuser scandal, the Full Tilt Poker insolvency, and the endemic bot rings that plague every major platform are not anomalies; they are the predictable consequences of a system that asks players to trust an entity whose incentives are misaligned with their own [1], [2].

This trust deficit cannot be resolved by policy, regulation, or reputation. Licensing requirements create barriers to entry but do not prevent licensed operators from cheating. Audits verify compliance at a point in time but cannot guarantee continuous honesty. Reputation is a lagging indicator that collapses catastrophically when fraud is eventually discovered. The only durable solution is to remove the need for trust entirely — to replace institutional assurances with mathematical guarantees.

TrueShuffle provides this replacement. By implementing the mental poker protocol of Shamir, Rivest, and Adleman [6] — refined through the elliptic curve construction of Barnett and Smart [7] — on a high-performance EVM-compatible Layer 2, TrueShuffle demonstrates the feasibility of production-scale operation for what decades of academic research proposed but never delivered: a poker system in which no party, including the server, can observe, predict, or manipulate the cards.

The system's contributions are concrete and verifiable:

1. **Provably fair dealing.** Commutative elliptic curve encryption over Secp256k1 ensures that the deck is a uniformly random permutation from the perspective of any coalition of fewer than m players. Each player encrypts, shuffles, and locks the deck with their own secret key; no card can be decrypted without the cooperation of all players whose keys protect it. The coordinator relays messages but never possesses decryption keys, rendering it cryptographically blind to the deck state. The fairness guarantee is not a policy commitment — it is a mathematical theorem, proven under the Elliptic Curve Discrete Logarithm assumption (Chapter 4).

2. **Trustless settlement.** All funds are custodied by immutable smart contracts deployed on MegaETH. The PokerEscrow contract enforces a strict conservation invariant: the sum of all player stacks and pot values equals the sum of all buy-ins at every point during a hand. Pot distributions execute atomically upon showdown verification. Dispute resolution operates on-chain

through the MentalPokerGame contract's dispute functions (`initiateDispute`, `resolveDispute`, `abortAndSlash`), which adjudicate protocol violations using committed hashes and cryptographic proofs. No admin key, no governance vote, and no human intervention can override the contract's logic or redirect its funds.

3. **Real-time performance.** MegaETH's approximately 10-millisecond block finality and sub-cent transaction costs [4] make per-action on-chain settlement economically and temporally practical for the first time. The complete cryptographic protocol — key commitment, shuffle, lock, deal — executes within seconds, and each betting action settles on-chain before the next player acts. The total on-chain cost of a complete hand is approximately $0.01, rendering the protocol viable for stakes ranging from micro-limits to high-roller tables.

4. **Accessible bot framework.** The DNA-based strategy parameterization system makes autonomous poker accessible to users without programming expertise. A bot's strategy is defined by a configuration of interpretable parameters — aggression, position awareness, hand range tightness, bluff frequency — that can be tuned manually, optimized through backtesting, or generated by AI-assisted tools. The bot SDK handles all cryptographic operations transparently, enabling bot creators to focus on strategy rather than protocol mechanics. This framework transforms poker from a game of real-time human decision-making into an engineering discipline of strategy design and optimization [15].

5. **Open and verifiable.** Every hand played on TrueShuffle is recorded immutably on-chain: the encrypted deck states, the commitment hashes, the revealed cards, the betting actions, the pot distributions. Any observer — player, regulator, researcher, or member of the public — can independently reconstruct and verify the fairness of any hand from the on-chain data alone. This represents a level of transparency unprecedented in the poker industry, where hand histories are proprietary, audit reports are confidential, and players must accept the operator's assurances on faith.

The significance of TrueShuffle extends beyond poker. The mental poker protocol is, at its core, a general-purpose solution to the problem of playing games with hidden information in the absence of a trusted dealer. Any card game — bridge, mahjong, rummy, blackjack — can be implemented using the same commutative encryption and selective decryption framework. More broadly, any multiparty interaction in which participants must commit to private information, reveal it selectively, and settle outcomes financially can benefit from the architectural pattern that TrueShuffle demonstrates: cryptographic protocol for information integrity, immutable smart contracts for financial settlement, and a coordinator that is untrusted by design.

The protocol is open-source. The smart contracts are immutable. The governance roadmap progresses toward complete decentralization, culminating in protocol ossification — a state in which no entity controls, administers, or can modify the system. The goal is a poker protocol that operates like a law of mathematics: indifferent to the identity or intentions of its participants, enforcing its rules with the same impartiality whether the stakes are one cent or one million dollars.

In a world where trust is scarce, cryptographic proof is abundant. TrueShuffle replaces the question "Do you trust the dealer?" with the statement "The math guarantees it."

[1] U. S. D. of Justice, "PokerStars and full tilt poker indictment." Press Release, 2011.

[2] H. Hintze, "Absolute poker and UltimateBet cheating scandal," *Poker News*, 2008.

[3] C. Research, "SEC 2: Recommended elliptic curve domain parameters." https://www.secg.org/sec2-v2.pdf, 2010.

[4] M. Foundation, "MegaETH: A real-time EVM-compatible layer 2." https://megaeth.com, 2025.

[5] D. J. Bernstein, "Curve25519: New diffie-hellman speed records," *Public Key Cryptography*, pp. 207–228, 2006.

[6] A. Shamir, R. L. Rivest, and L. M. Adleman, "Mental poker," *The Mathematical Gardner*, pp. 37–43, 1981.

[7] A. Barnett and N. P. Smart, "Mental poker revisited," in *Cryptography and coding*, in LNCS, vol. 2898. Springer, 2003, pp. 370–383.

[8] J. Castella-Roca, J. Domingo-Ferrer, A. Riera, and J. Borrell, "Secure multi-party computation protocols for card games," *Information Security*, pp. 370–385, 2005.

[9] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger." Ethereum Project Yellow Paper, 2014.

[10] V. Buterin, "A next-generation smart contract and decentralized application platform." Ethereum Whitepaper, 2014.

[11] P. Labs, "libp2p: A modular network stack." https://libp2p.io, 2023.

[12] J. Von Neumann, "Zur theorie der gesellschaftsspiele," *Mathematische Annalen*, vol. 100, no. 1, pp. 295–320, 1928.

[13] J. F. Nash, "Equilibrium points in n-person games," *Proceedings of the National Academy of Sciences*, vol. 36, no. 1, pp. 48–49, 1950.

[14] M. Zinkevich, M. Johanson, M. Bowling, and C. Piccione, "Regret minimization in games with incomplete information," *Advances in Neural Information Processing Systems*, vol. 20, 2007.

[15] N. Brown and T. Sandholm, "Superhuman AI for multiplayer poker," *Science*, vol. 365, no. 6456, pp. 885–890, 2019.

[16] L. S. Shapley, "Some topics in two-person games," *Advances in Game Theory*, pp. 1–28, 1964.

[17] J. Maynard Smith and G. R. Price, "The logic of animal conflict," *Nature*, vol. 246, pp. 15–18, 1973.

[18] T. Contributors, "Tauri v2: Build smaller, faster, and more secure desktop and mobile applications." https://v2.tauri.app, 2024.