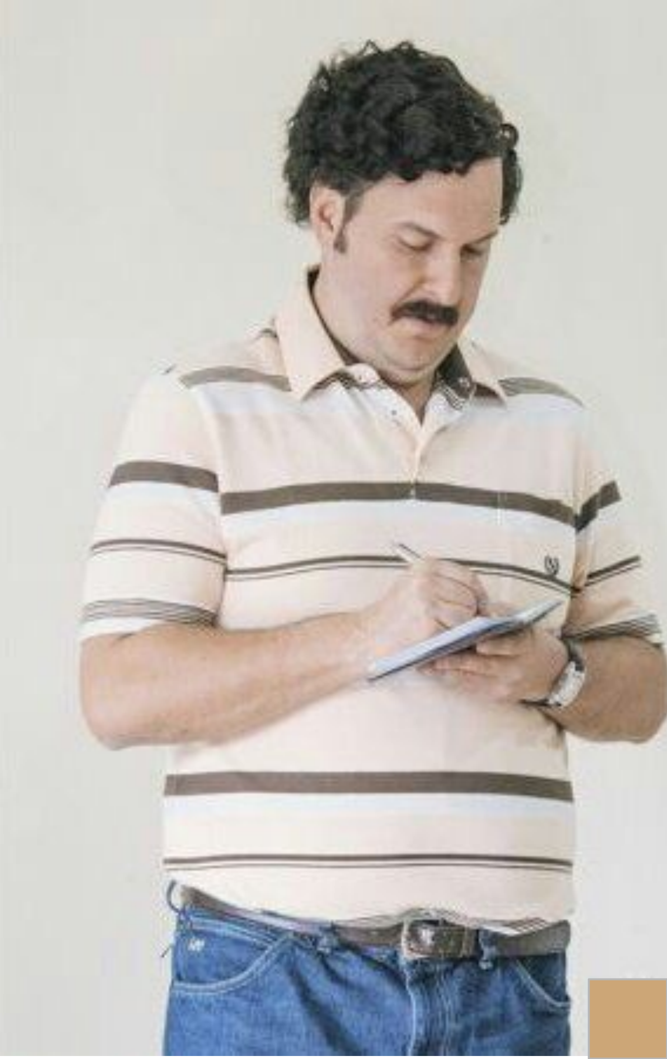
The Card DAO

Team members:

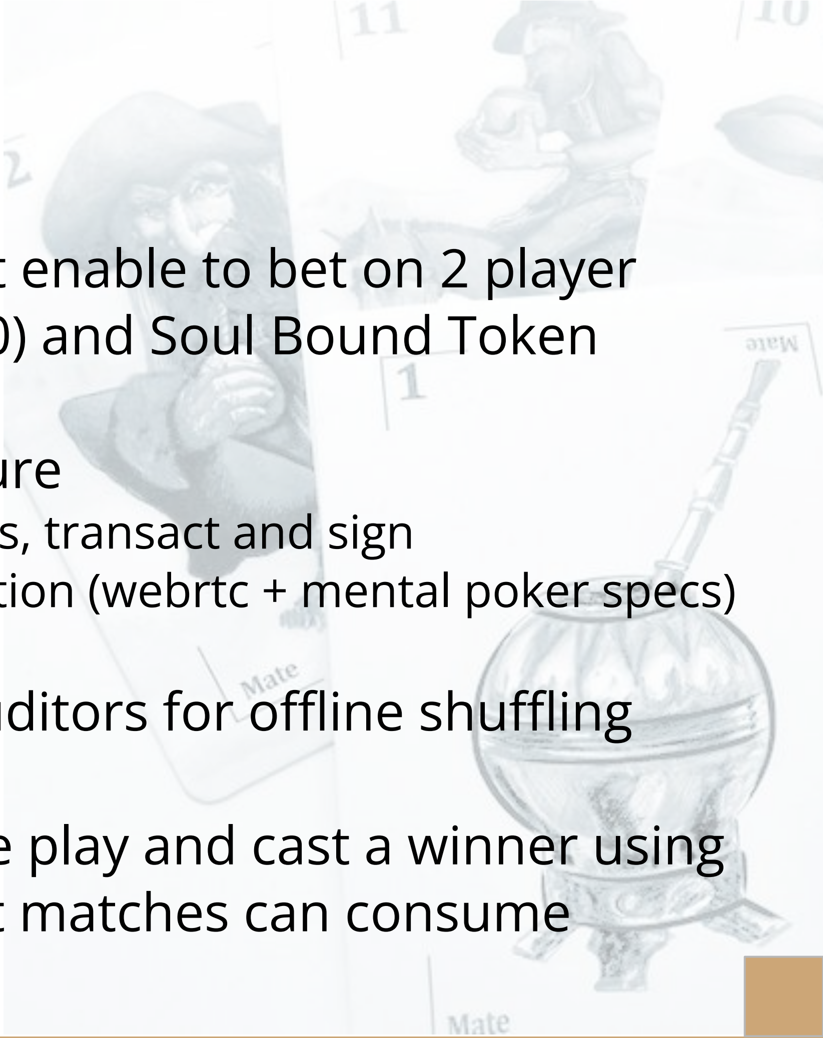
- Julián Murphy
- Emiliano Moscato
- Julián Paredes

Agenda

- Implementation scope and concessions
- What gives value to our token (Trucoin)
- Journey to decentralized shuffling
- The missing piece: CardDAO Impartial Verifier
- DEMO TIME 🤓
- Deep dive into Smart Contracts Architecture
- Explore the future of The DAO

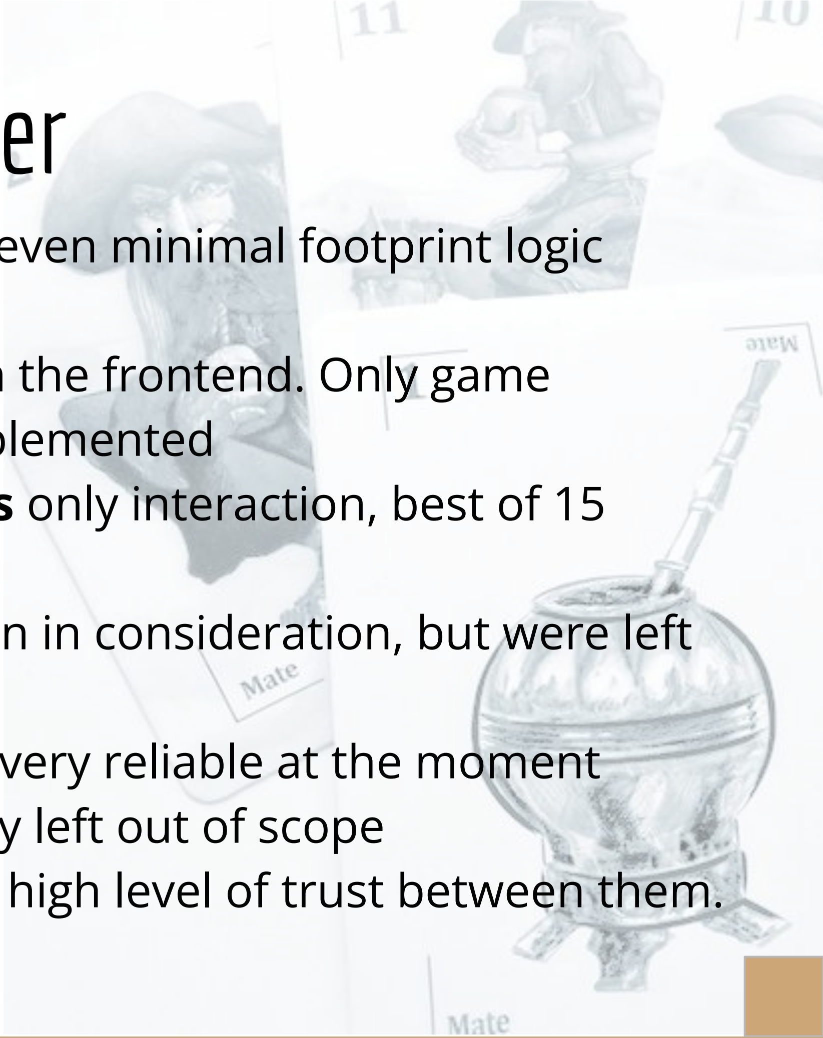


Implementation Scope



- Set of EVM compatible contracts that enable to bet on 2 player truco matches using Trucoins (ERC-20) and Soul Bound Token award for match winner
- React Frontend: serverless architecture
  - Wallet interaction: bet, deploy matches, transact and sign
  - Decentralized deck shuffling coordination (webrtc + mental poker specs)
  - Basic game UI
- Impartial Verifiers (IV) external key auditors for offline shuffling protocol
- Game Engine that ensures valid game play and cast a winner using an standard interface (ERC-3333) that matches can consume

Some Concessions / Disclaimer



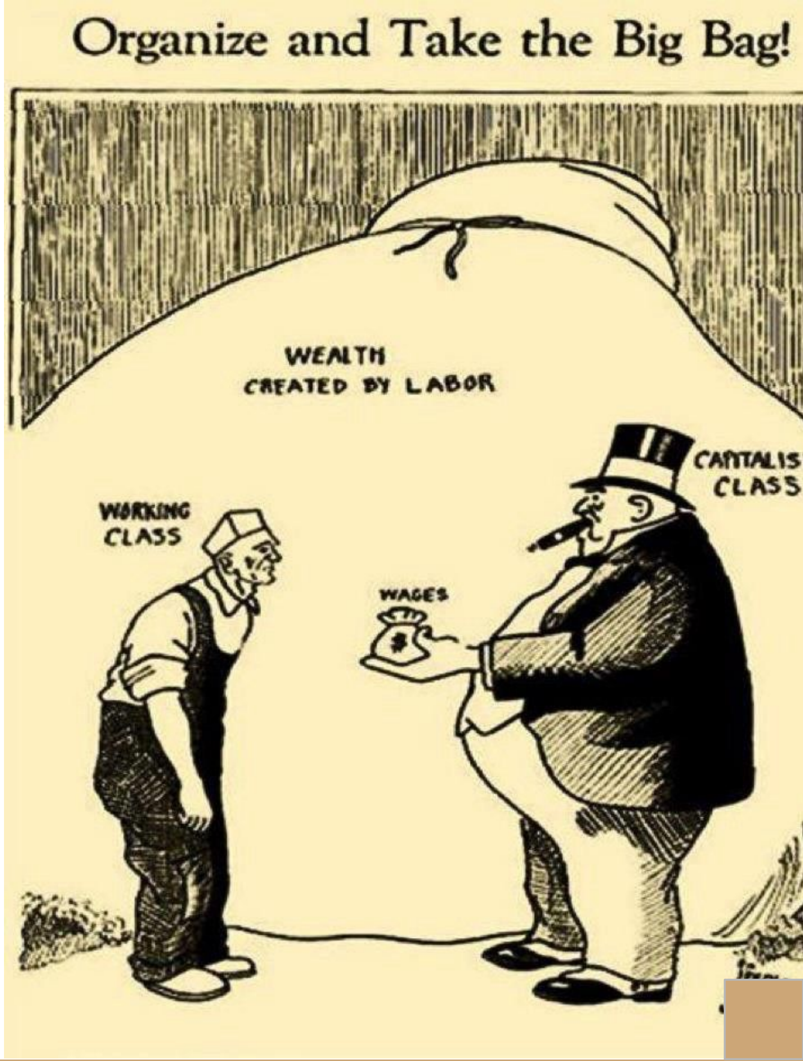
- No big efforts spent on GAS efficiency since even minimal footprint logic makes L1 deployment costs unaffordable
- Match browsing and listing was left out from the frontend. Only game creation/deployment and gameplay was implemented
- Shuffling/Engine logic designed for **2 players** only interaction, best of 15 points wins
- Multisig for critical contracts ownership taken in consideration, but were left out this implementation
- P2PT has its quirks with peer discovery. Not very reliable at the moment
- Means for liquidity of Trucoins were arbitrary left out of scope
- Match IVs are players themselves, assuming high level of trust between them.

Don't wait, go check the code!

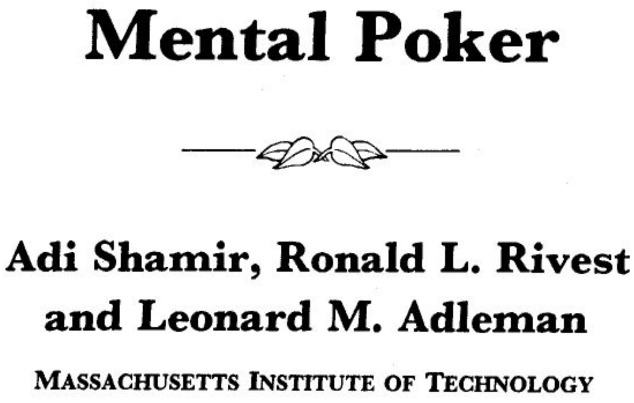
https://github.com/CardDAO

What gives value to Trucoin?

- It is CardDAO’s mission to constantly improve protocol contracts
- Ongoing matches are bug protected by “on-the-spot” bug fixes as soon as they are spotted or reported by the community
- Real value is produced within the blockchain and given in exchange of fees (and not from speculation!)
- Fees are collected **only in trucoins**
- Trucoin liquidity by DAO’s liquidity pools in Most prominent web3 exchanges on the network



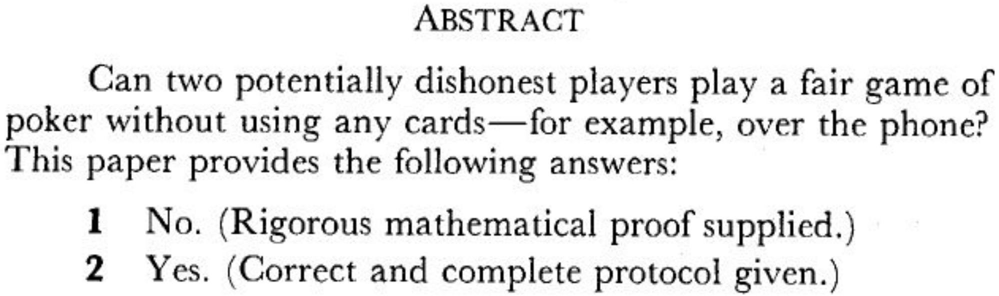
¿How we solve decentralized shuffling challenge?



Mental Poker

Adi Shamir, Ronald L. Rivest and Leonard M. Adleman

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

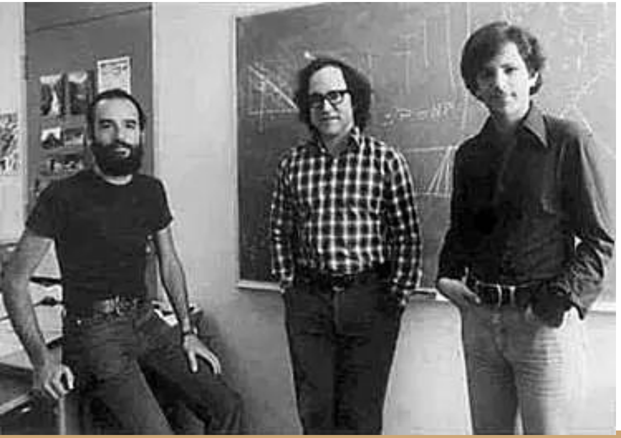


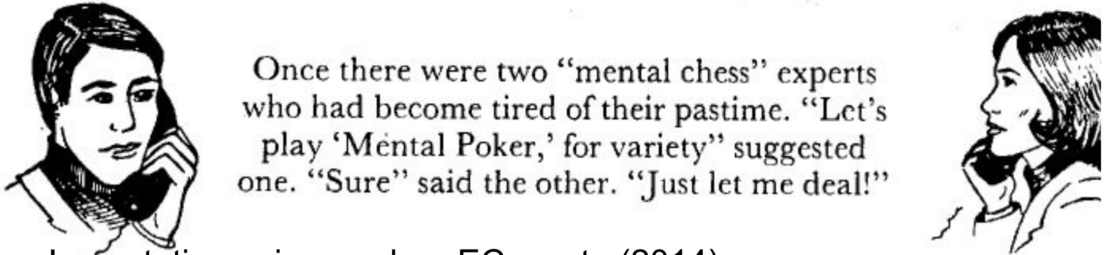
ABSTRACT

Can two potentially dishonest players play a fair game of poker without using any cards—for example, over the phone? This paper provides the following answers:

1 No. (Rigorous mathematical proof supplied.)

2 Yes. (Correct and complete protocol given.)





Once there were two "mental chess" experts who had become tired of their pastime. "Let's play 'Mental Poker,' for variety" suggested one. "Sure" said the other. "Just let me deal!"

Our implementation using modern EC crypto (2014):
https://github.com/kripod/mental-poker/blob/master/specs/thesis.pdf
Original paper (1979): https://apps.dtic.mil/dtic/tr/fulltext/u2/a066331.pdf
Mathematical Gardner (1981): https://people.csail.mit.edu/rivest/pubs/SRA81.pdf
Bibliography: http://liinwww.ira.uka.de/bibliography/Misc/MentalPoker.html

Our off-chain decentralized implementation: Codewords, Shuffling, Locking and Secp256k1 curve

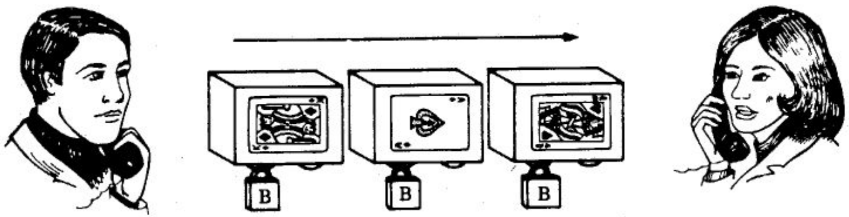
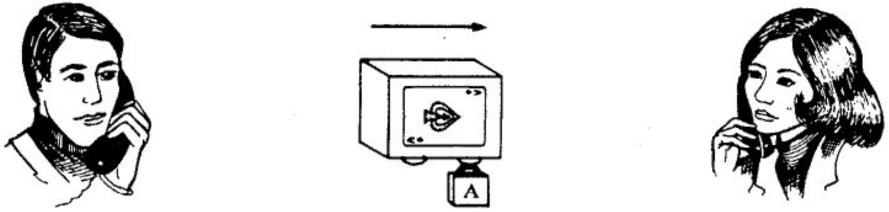
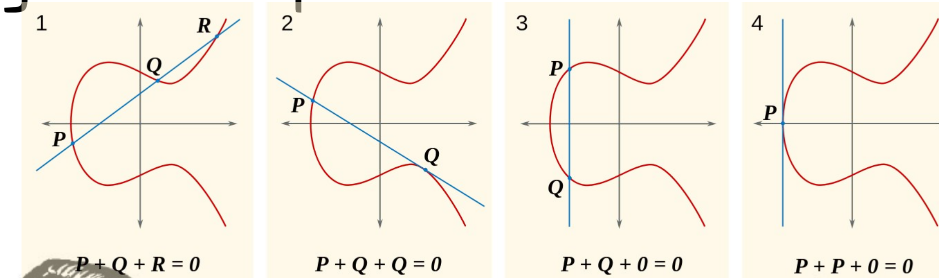


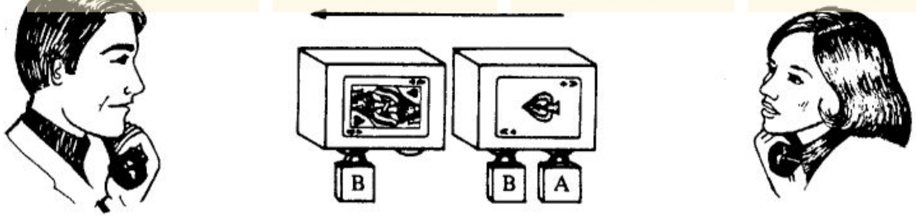
FIGURE 1

Bob encrypts the cards and sends them to Alice in scrambled order.



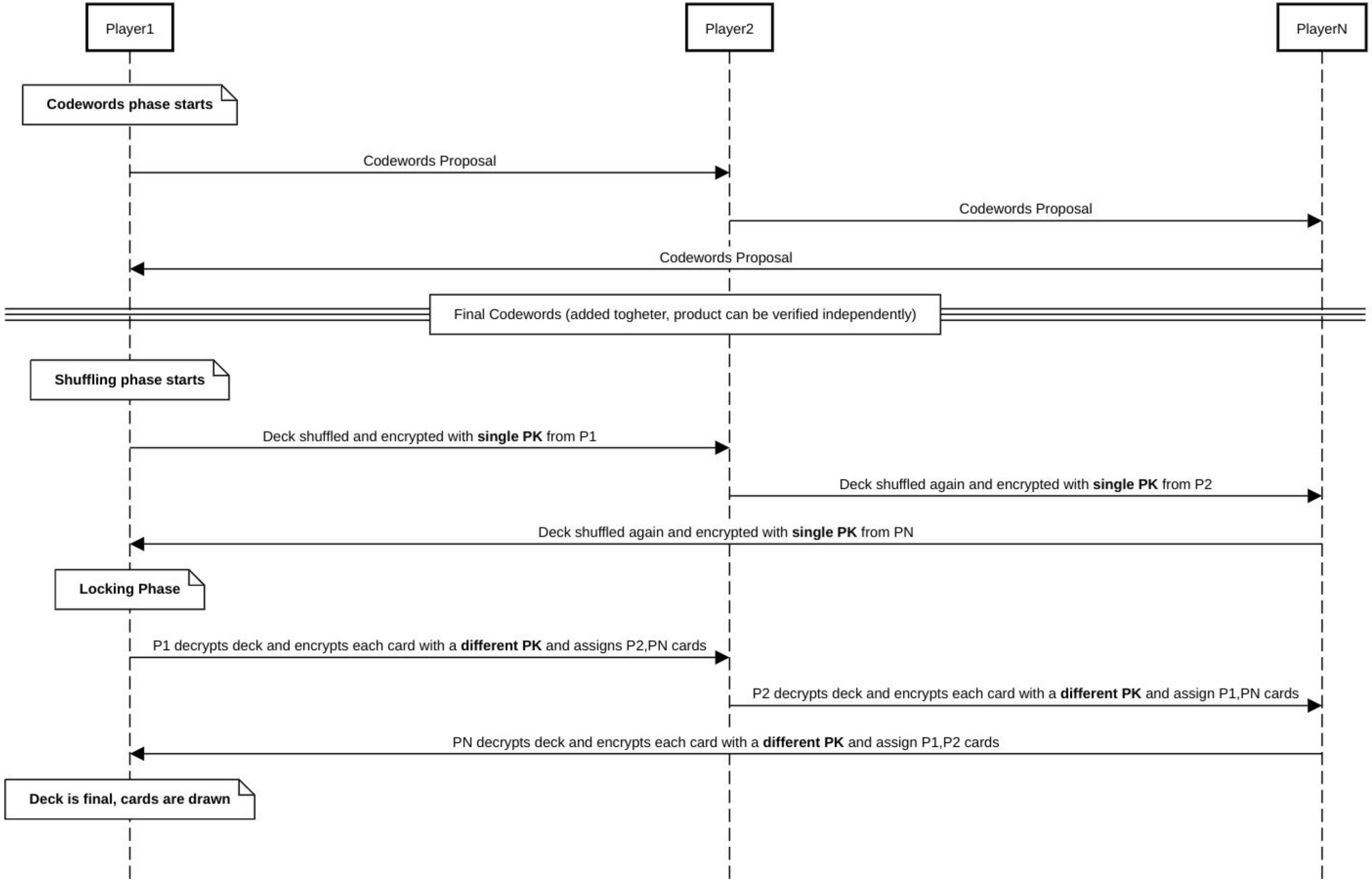
Bob decrypts both cards, and returns Alice's encrypted card to her.





Alice chooses one for Bob, and encrypts another for herself, and sends them both to Bob.

Decentralized Shuffling: N players general case



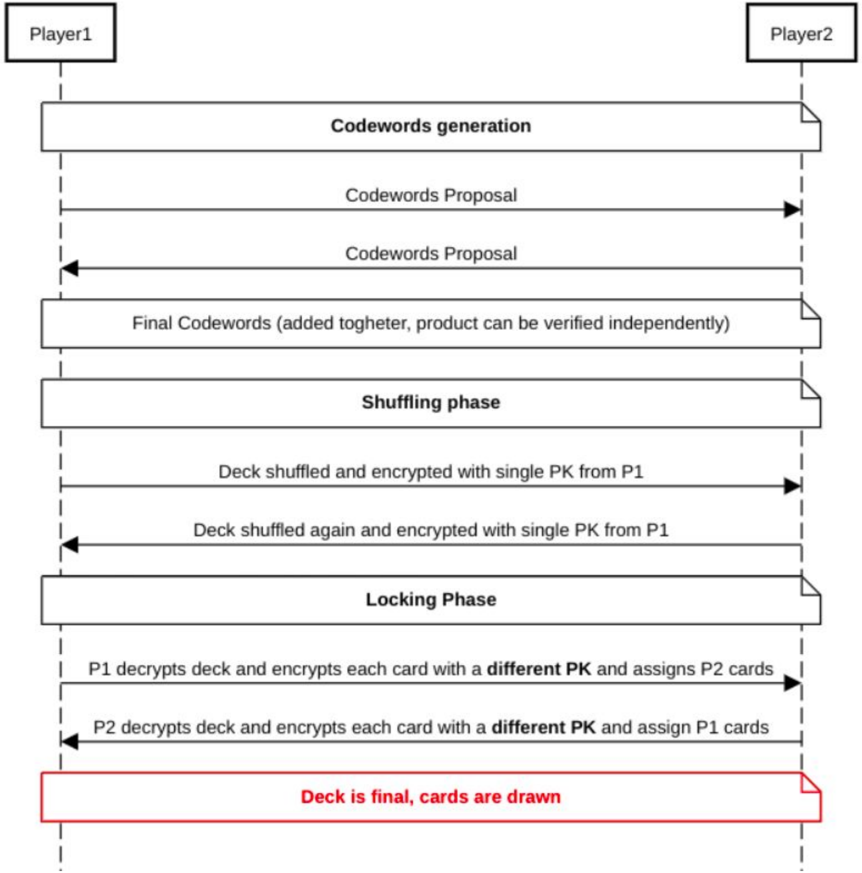
Player1
Player2
PlayerN
Codewords phase starts
Codewords Proposal
Codewords Proposal
Codewords Proposal
Final Codewords (added togheter, product can be verified independently)
Shuffling phase starts
Deck shuffled and encrypted with **single PK** from P1
Deck shuffled again and encrypted with **single PK** from P2
Deck shuffled again and encrypted with **single PK** from PN
Locking Phase
P1 decrypts deck and encrypts each card with a **different PK** and assigns P2,PN cards
P2 decrypts deck and encrypts each card with a **different PK** and assign P1,PN cards
PN decrypts deck and encrypts each card with a **different PK** and assign P1,P2 cards
Deck is final, cards are drawn

Our implementation: 2 players interaction over WebRTC

BitTorrent™

WebRTC

- Uses WebSocket Trackers as signalling servers
- Use a magnet uri as identifier built from match address which is converted to a valid Info Hash and sent to WebTorrent trackers who will give us a list of web peers (players participating, IVs or other peers witnessing the shuffle process)



Player1

Player2

Codewords generation

Codewords Proposal

Codewords Proposal

Final Codewords (added togheter, product can be verified independently)

Shuffling phase

Deck shuffled and encrypted with single PK from P1

Deck shuffled again and encrypted with single PK from P1

Locking Phase

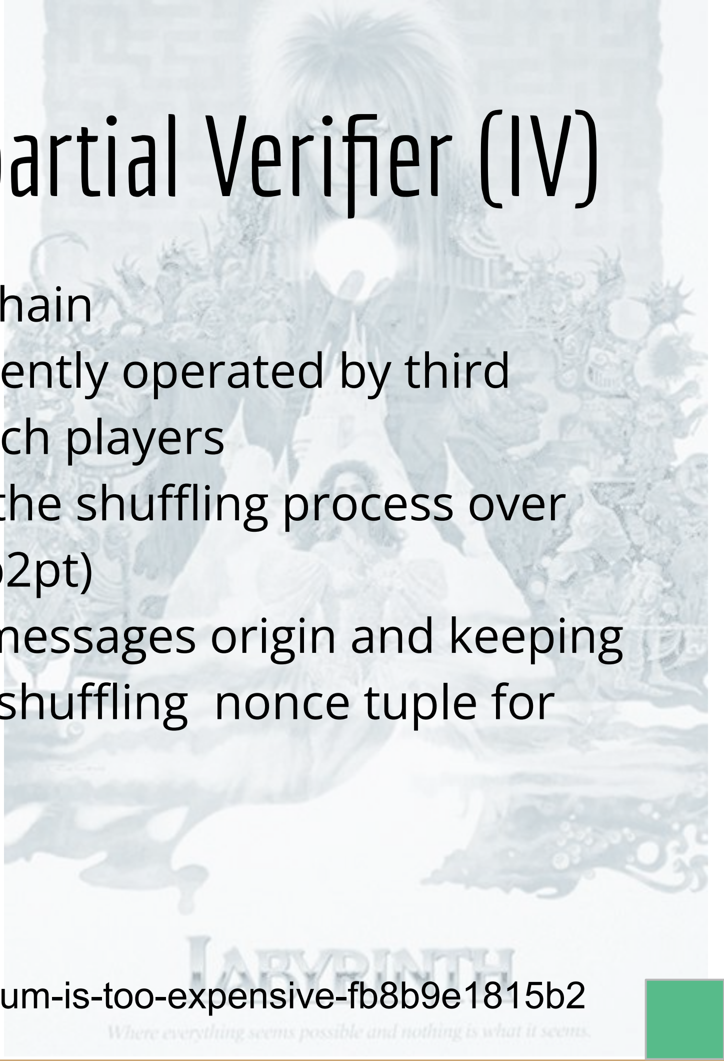
P1 decrypts deck and encrypts each card with a **different PK** and assigns P2 cards

P2 decrypts deck and encrypts each card with a **different PK** and assign P1 cards

Deck is final, cards are drawn

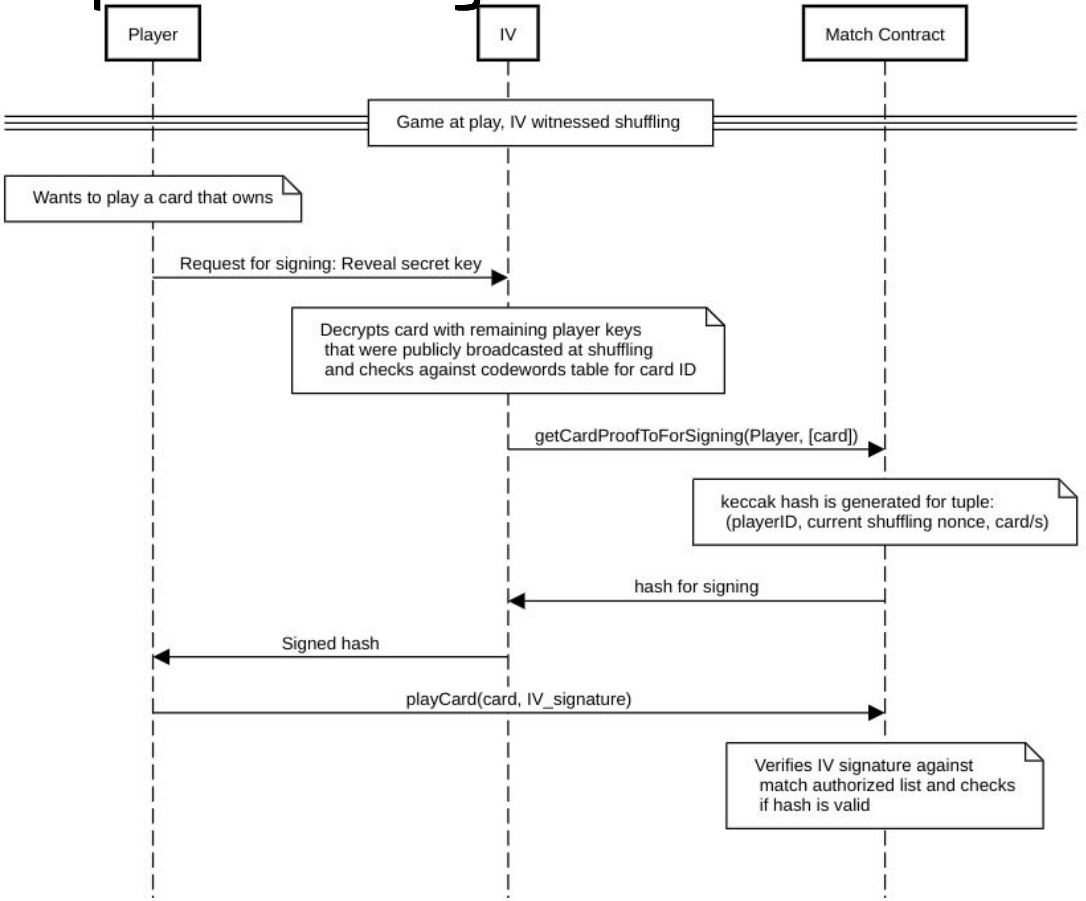
The missing piece: CardDAO Impartial Verifier (IV)

- REST web service component that works off-chain
- Can be operated by CardDAO itself, independently operated by third parties or by mutual agreement between match players
- Plays a role of external auditor, being part of the shuffling process over the coordination protocol channel (webrtc + p2pt)
- Witness users shuffling exchange, validating messages origin and keeping a log of encrypted decks, codewords tables + shuffling nonce tuple for each match


LABYRINTH
Where everything seems possible and nothing is what it seems.

Privacy on Ethereum is too expensive:
https://medium.com/clearmatics/privacy-on-ethereum-is-too-expensive-fb8b9e1815b2

IV Flow Sequence Diagram



Player

IV

Match Contract

Game at play, IV witnessed shuffling

Wants to play a card that owns

Request for signing: Reveal secret key

Decrypts card with remaining player keys
that were publicly broadcasted at shuffling
and checks against codewords table for card ID

getCardProofToForSigning(Player, [card])

keccak hash is generated for tuple:
(playerID, current shuffling nonce, card/s)

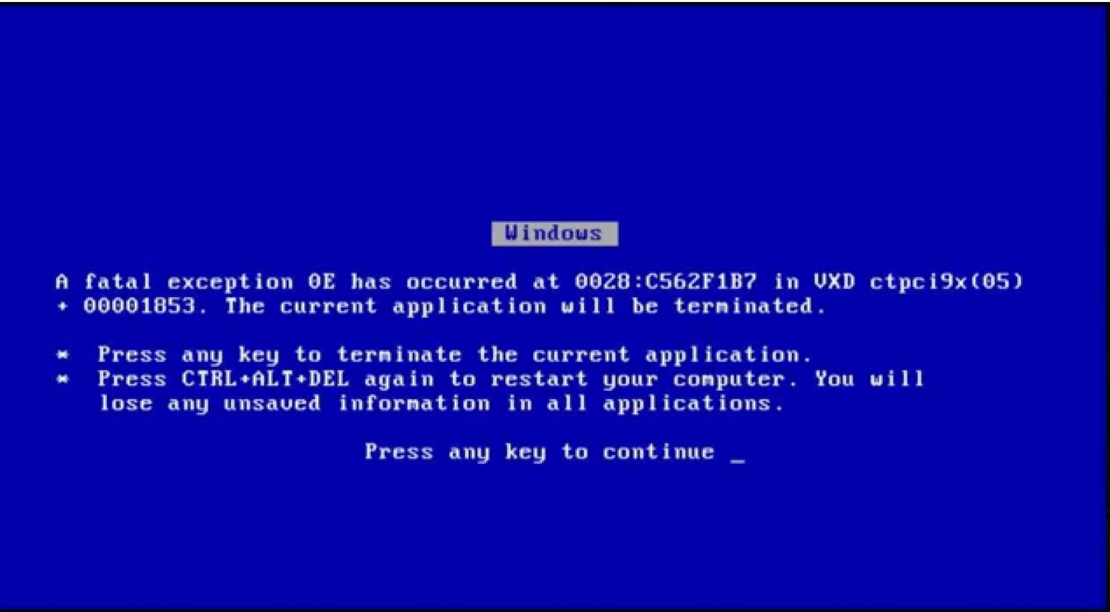
hash for signing

Signed hash

playCard(card, IV_signature)

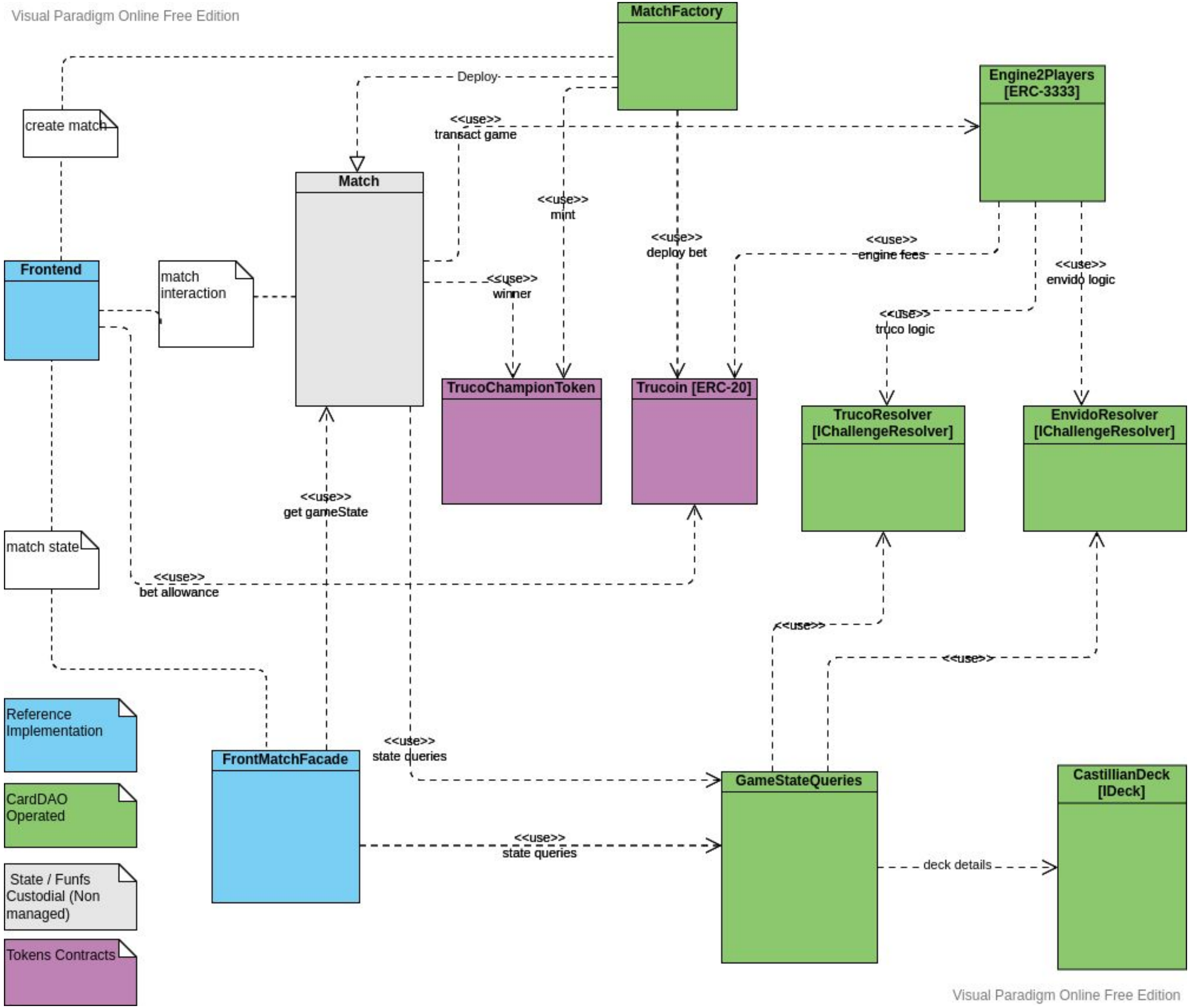
Verifies IV signature against
match authorized list and checks
if hash is valid

DEMO



```
                              Windows

 A fatal exception 0E has occurred at 0028:C562F1B7 in VXD ctpci9x(05)
 + 00001853. The current application will be terminated.

 *  Press any key to terminate the current application.
 *  Press CTRL+ALT+DEL again to restart your computer. You will
    lose any unsaved information in all applications.

                        Press any key to continue _
```

General Overview



Match & Deploy Architecture

Deployer

- Upgradeable and CardDAO maintained

Match:

- Not upgradeable (no privileged methods either)
- Acts as an escrow of bet funds till match ends
- Soul Bound Token minting for match winner

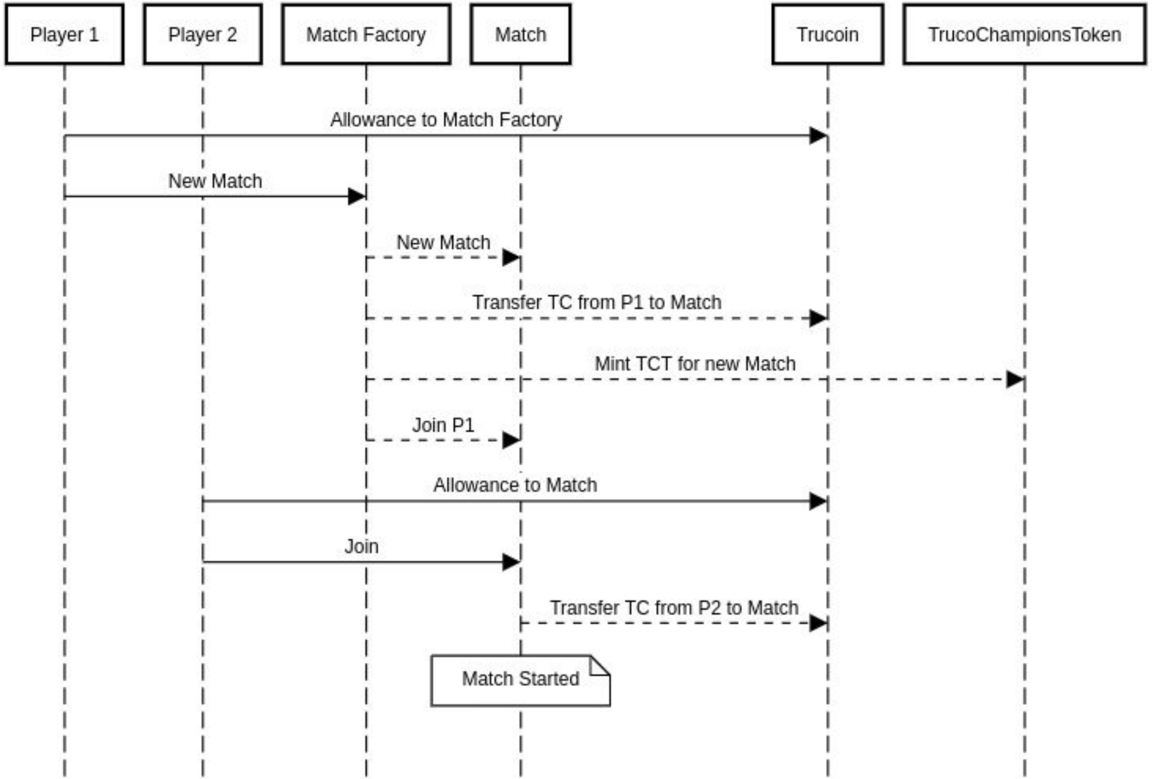
Truco Champions Token:

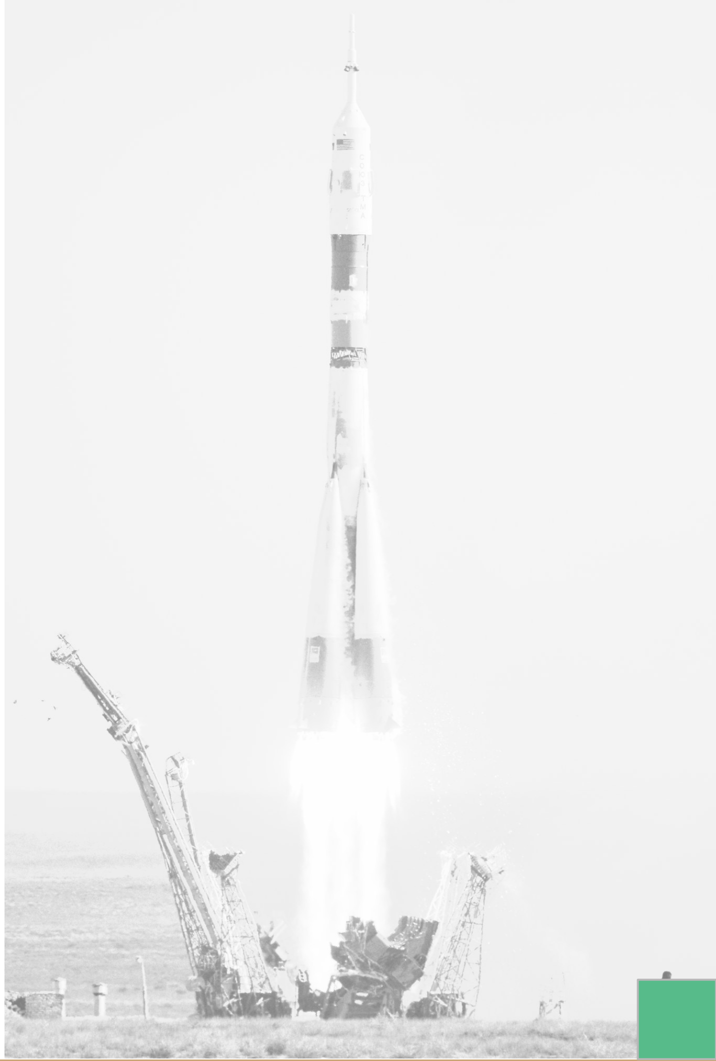
- Not upgradable
- Soul Bound Token

Trucoin:

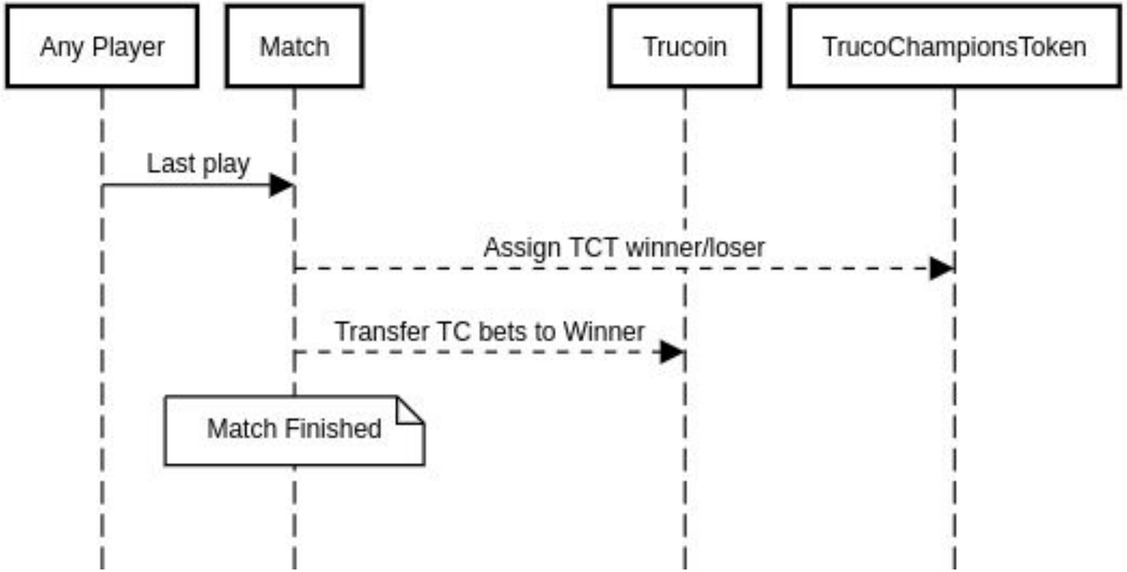
- Openzeppelin ERC-20 bet token

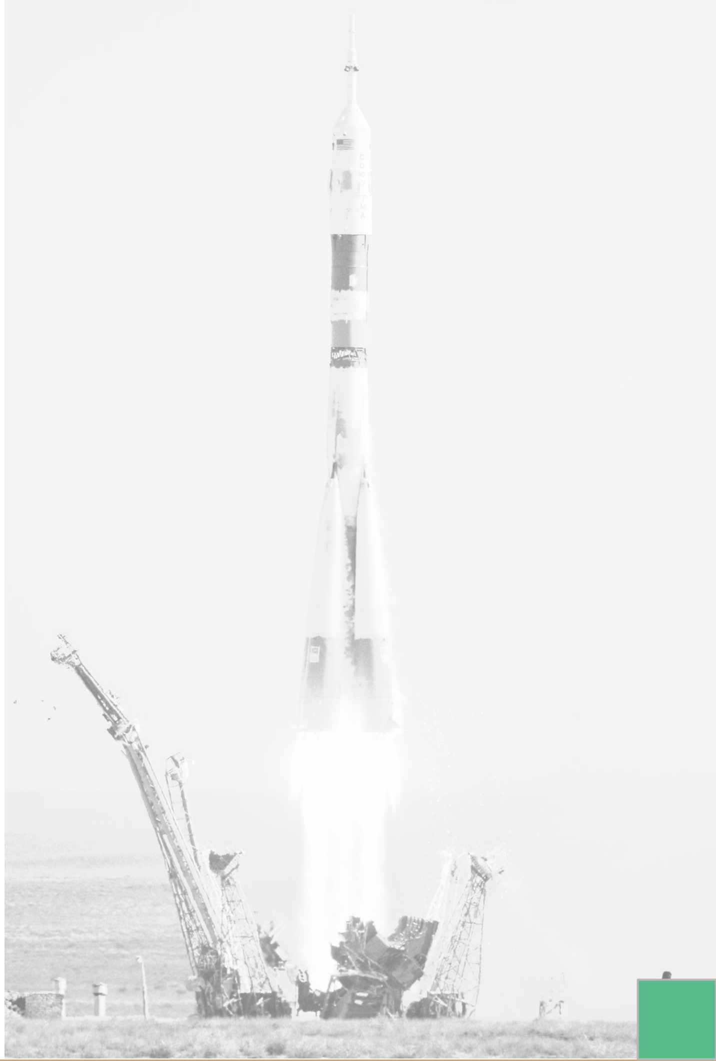
Match creation sequence



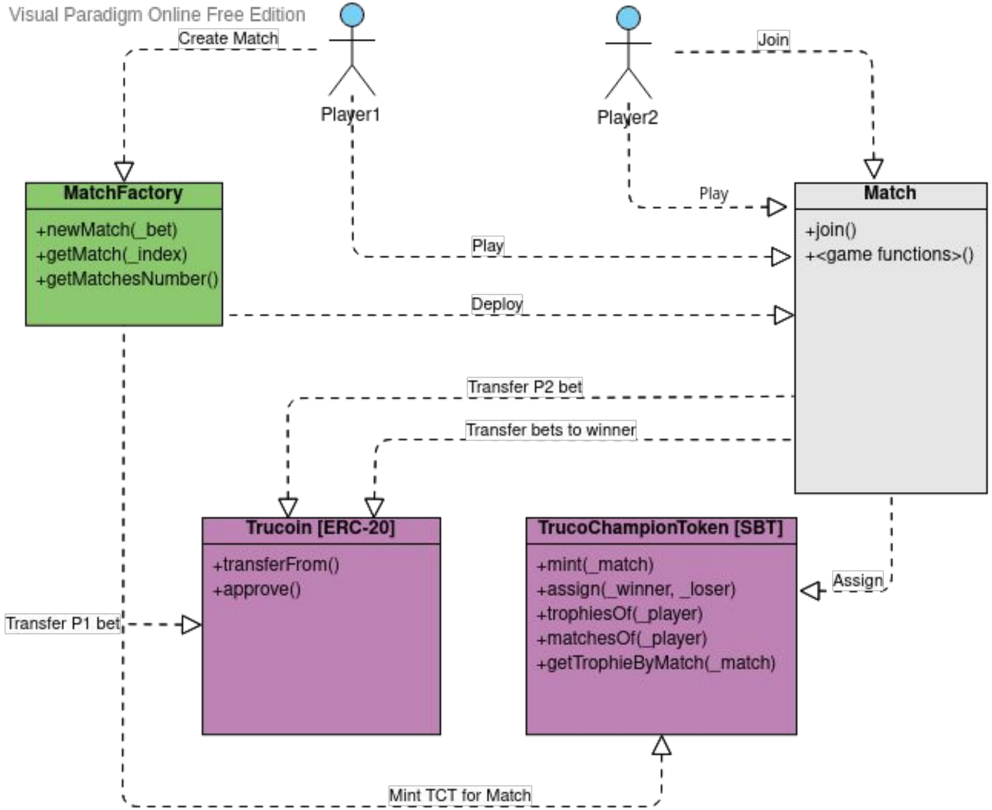


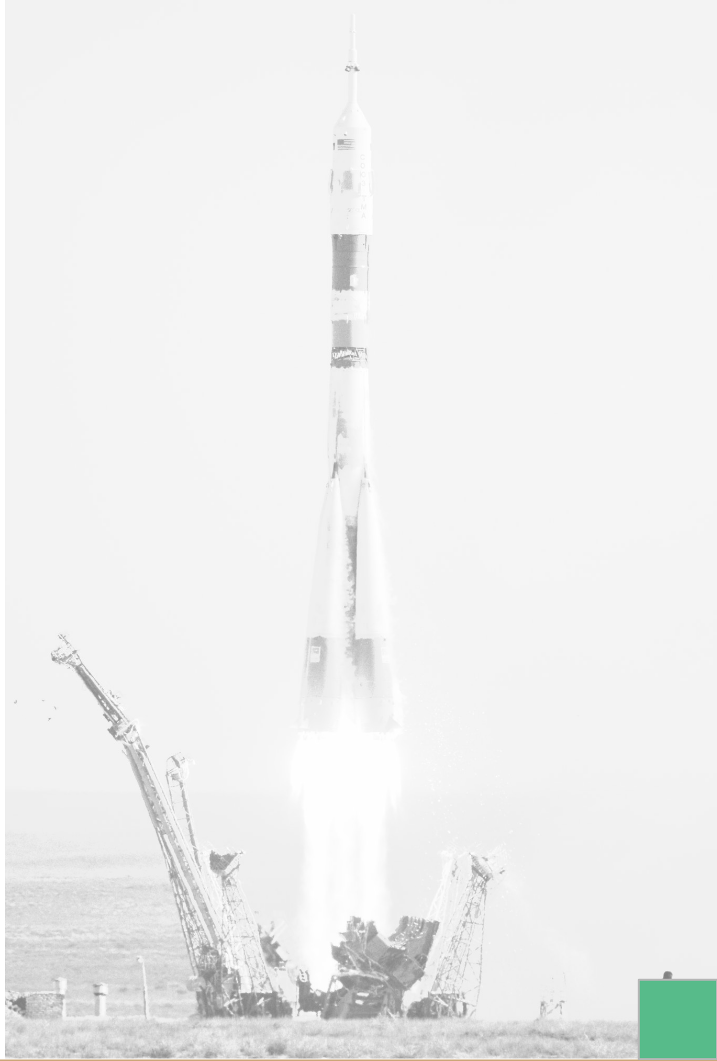
End of Match sequence





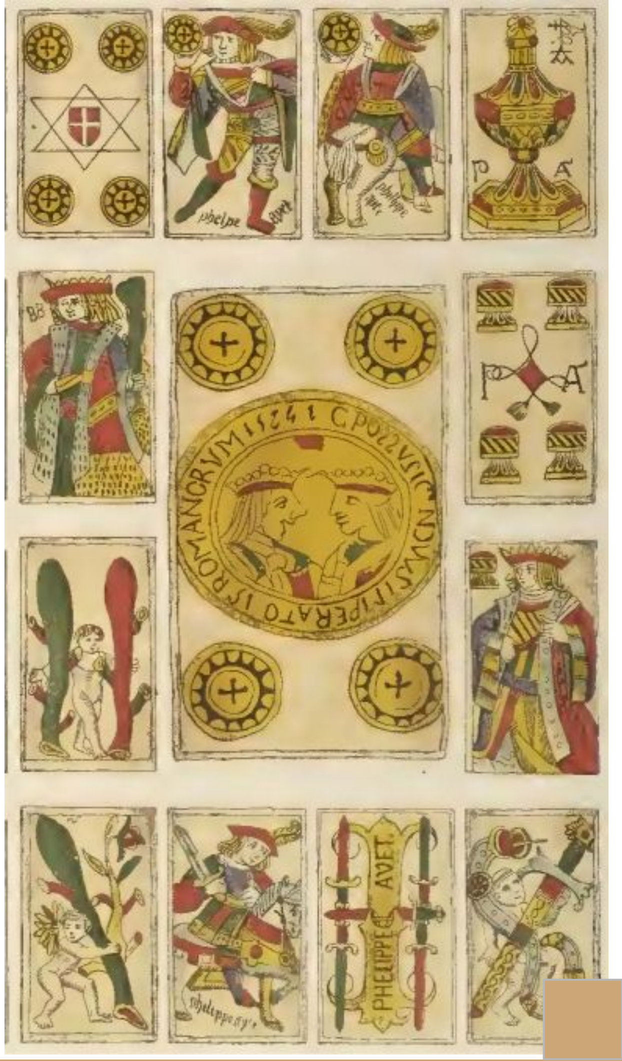
Life cycle class diagram



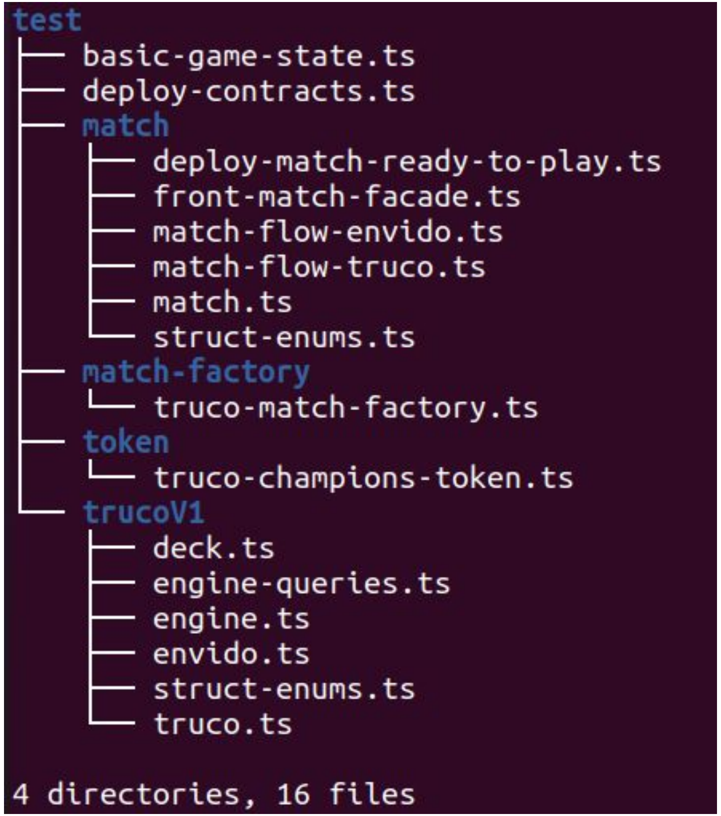


Engine Architecture

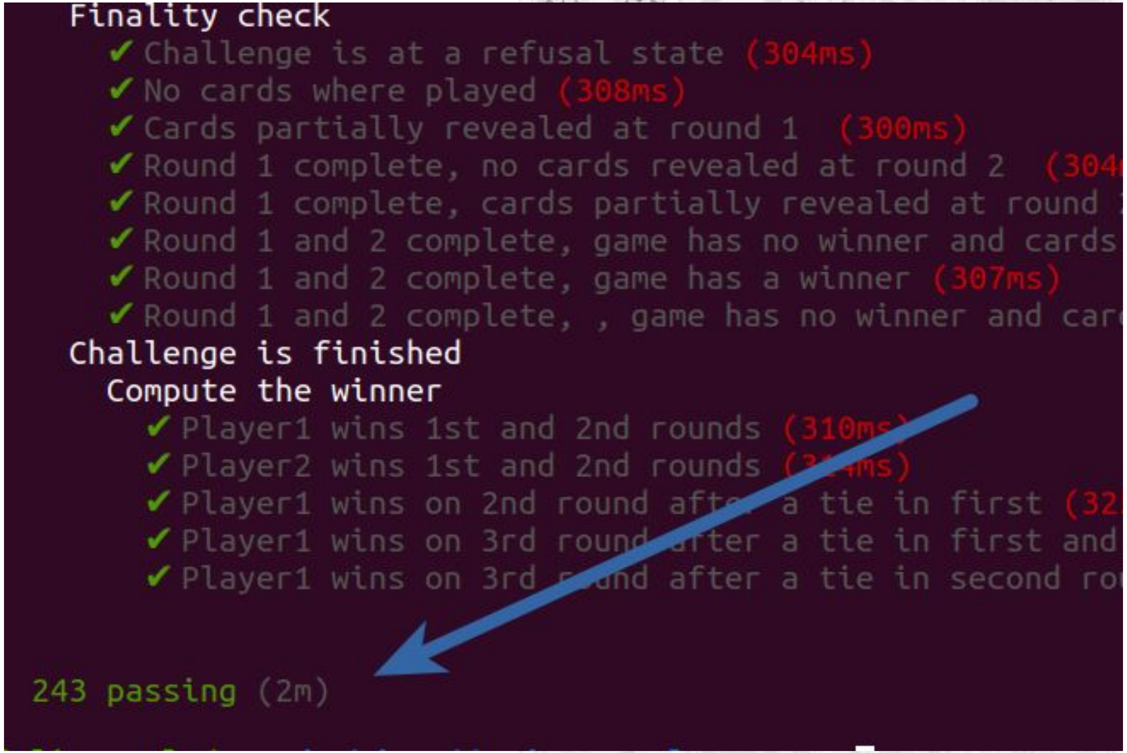
- Engine: **ERC-3333** proposed standard interface for on-chain truco game plays😜
- Resolvers for different game play logic (Truco and Envido)
- Deck Abstraction: Castilian Deck implemented
- General multi-consumer contract for non transactional game state related query logic
- No turn handling (handled at match level)



TDD Galore: A test case for almost every use case you can think of...

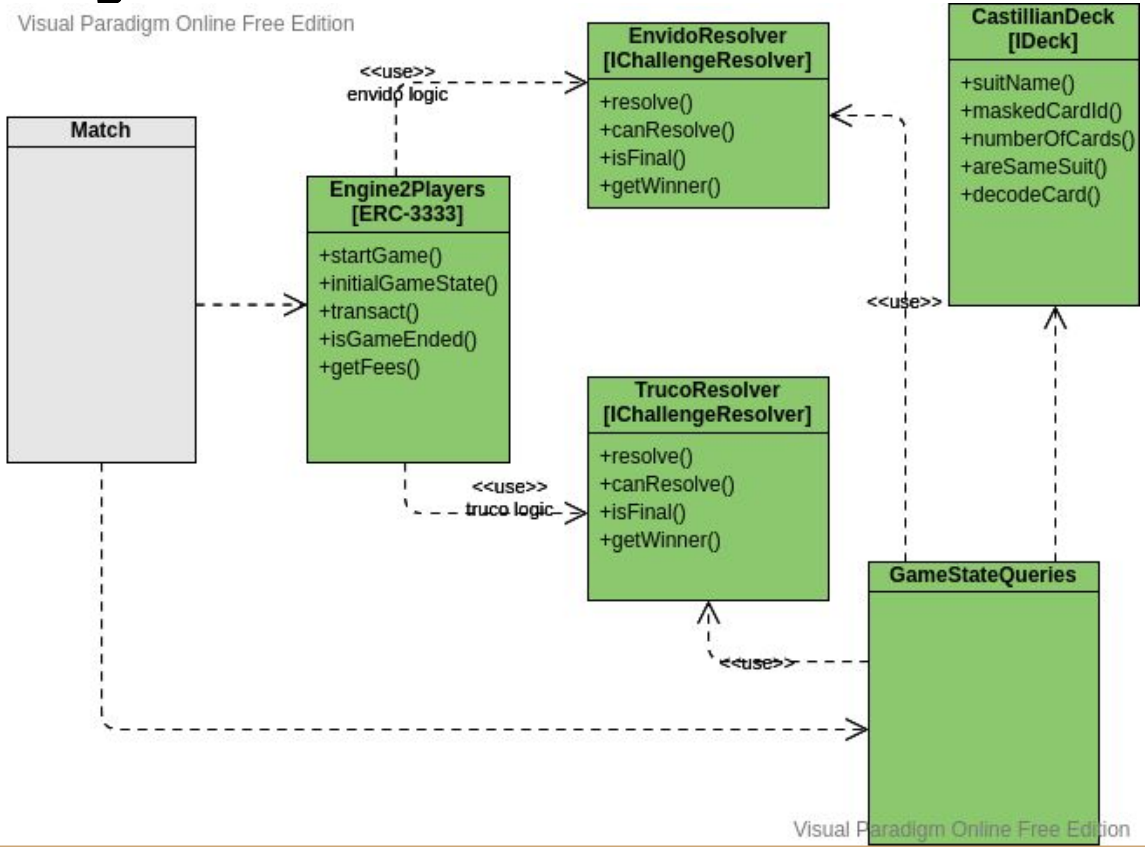


```
test
├── basic-game-state.ts
├── deploy-contracts.ts
├── match
│   ├── deploy-match-ready-to-play.ts
│   ├── front-match-facade.ts
│   ├── match-flow-envido.ts
│   ├── match-flow-truco.ts
│   ├── match.ts
│   └── struct-enums.ts
├── match-factory
│   └── truco-match-factory.ts
├── token
│   └── truco-champions-token.ts
└── trucoV1
    ├── deck.ts
    ├── engine-queries.ts
    ├── engine.ts
    ├── envido.ts
    ├── struct-enums.ts
    └── truco.ts

4 directories, 16 files
```



```
  Finality check
    ✓ Challenge is at a refusal state (304ms)
    ✓ No cards where played (308ms)
    ✓ Cards partially revealed at round 1  (300ms)
    ✓ Round 1 complete, no cards revealed at round 2  (304
    ✓ Round 1 complete, cards partially revealed at round
    ✓ Round 1 and 2 complete, game has no winner and cards
    ✓ Round 1 and 2 complete, game has a winner (307ms)
    ✓ Round 1 and 2 complete, , game has no winner and car
  Challenge is finished
    Compute the winner
      ✓ Player1 wins 1st and 2nd rounds (310ms)
      ✓ Player2 wins 1st and 2nd rounds (314ms)
      ✓ Player1 wins on 2nd round after a tie in first (32
      ✓ Player1 wins on 3rd round after a tie in first and
      ✓ Player1 wins on 3rd round after a tie in second ro


243 passing (2m)
```

Engine Architecture Overview



Match

Engine2Players
[ERC-3333]

+startGame()
+initialGameState()
+transact()
+isGameEnded()
+getFees()

EnvidoResolver
[IChallengeResolver]

+resolve()
+canResolve()
+isFinal()
+getWinner()

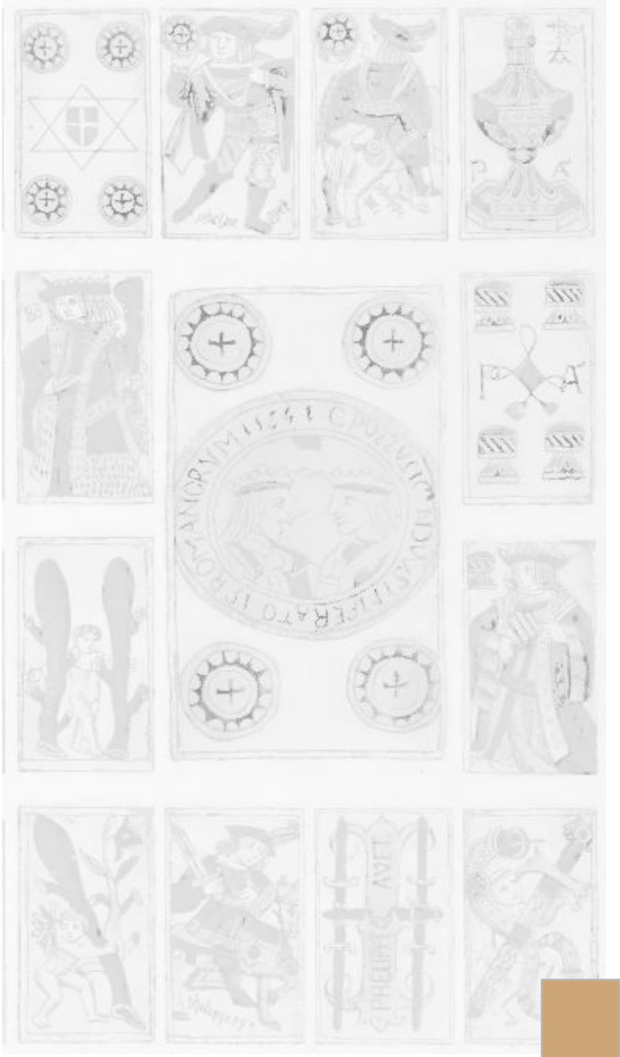
TrucoResolver
[IChallengeResolver]

+resolve()
+canResolve()
+isFinal()
+getWinner()

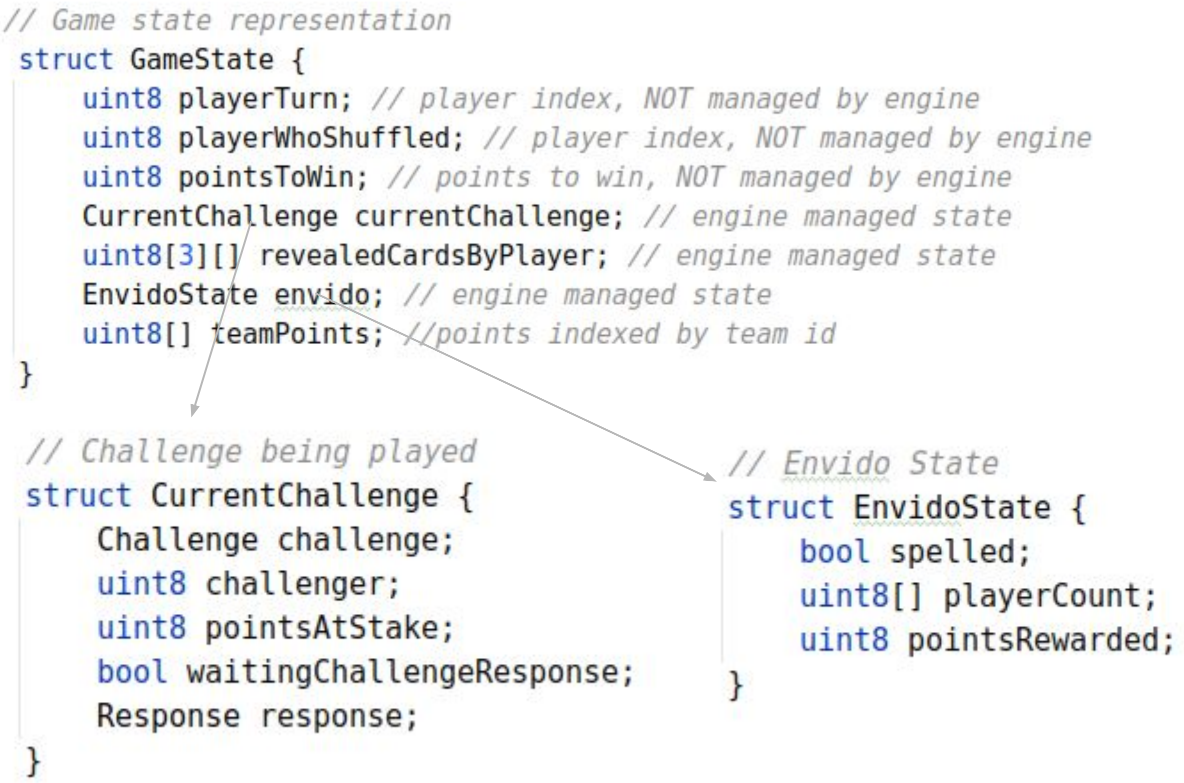
CastillianDeck
[IDeck]

+suitName()
+maskedCardId()
+numberOfCards()
+areSameSuit()
+decodeCard()

GameStateQueries



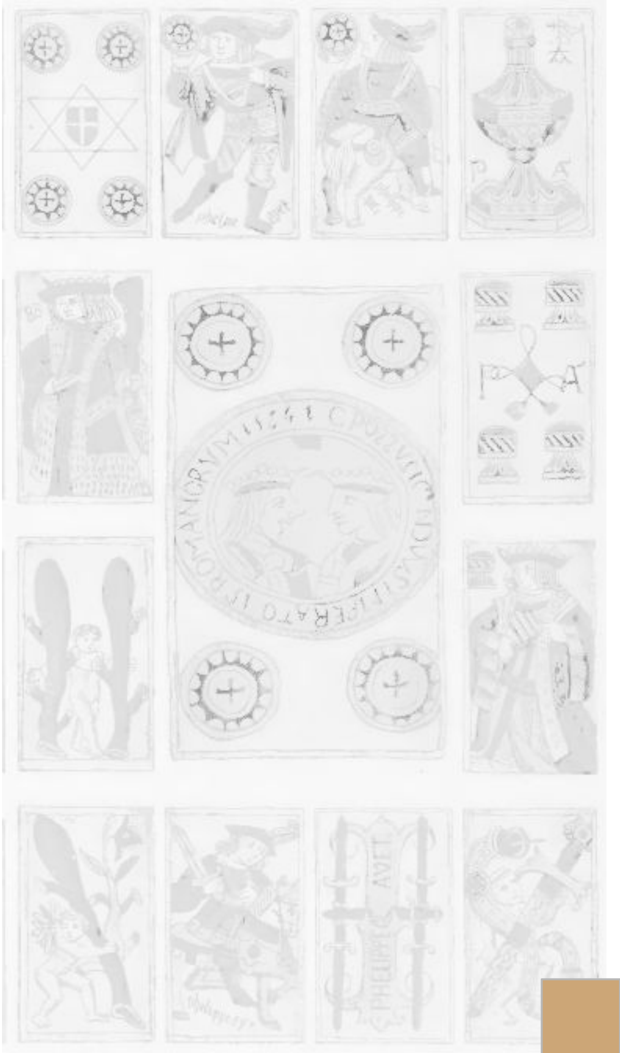
Engine Shared State



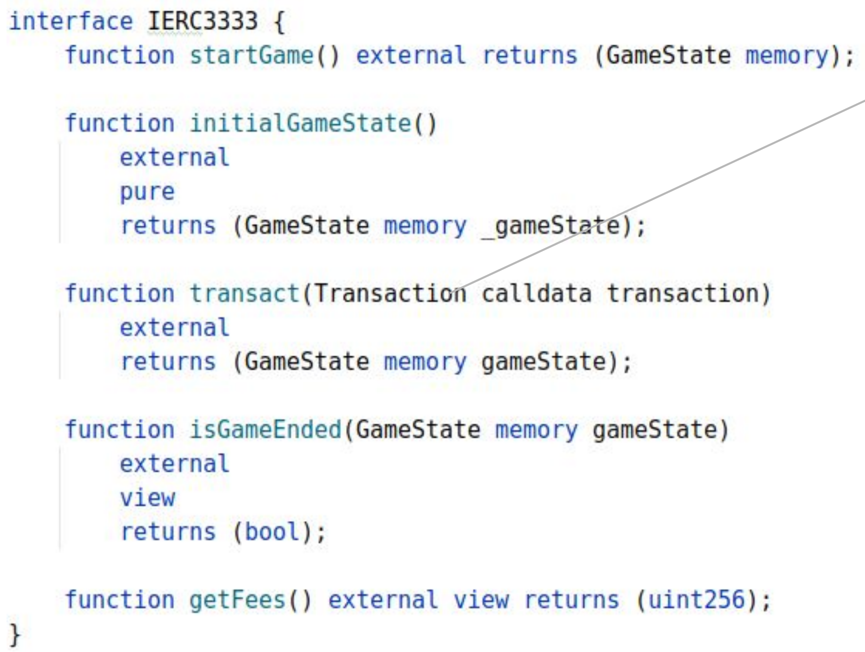
```
// Game state representation
struct GameState {
    uint8 playerTurn; // player index, NOT managed by engine
    uint8 playerWhoShuffled; // player index, NOT managed by engine
    uint8 pointsToWin; // points to win, NOT managed by engine
    CurrentChallenge currentChallenge; // engine managed state
    uint8[3][] revealedCardsByPlayer; // engine managed state
    EnvidoState envido; // engine managed state
    uint8[] teamPoints; //points indexed by team id
}
```

```
// Challenge being played
struct CurrentChallenge {
    Challenge challenge;
    uint8 challenger;
    uint8 pointsAtStake;
    bool waitingChallengeResponse;
    Response response;
}
```

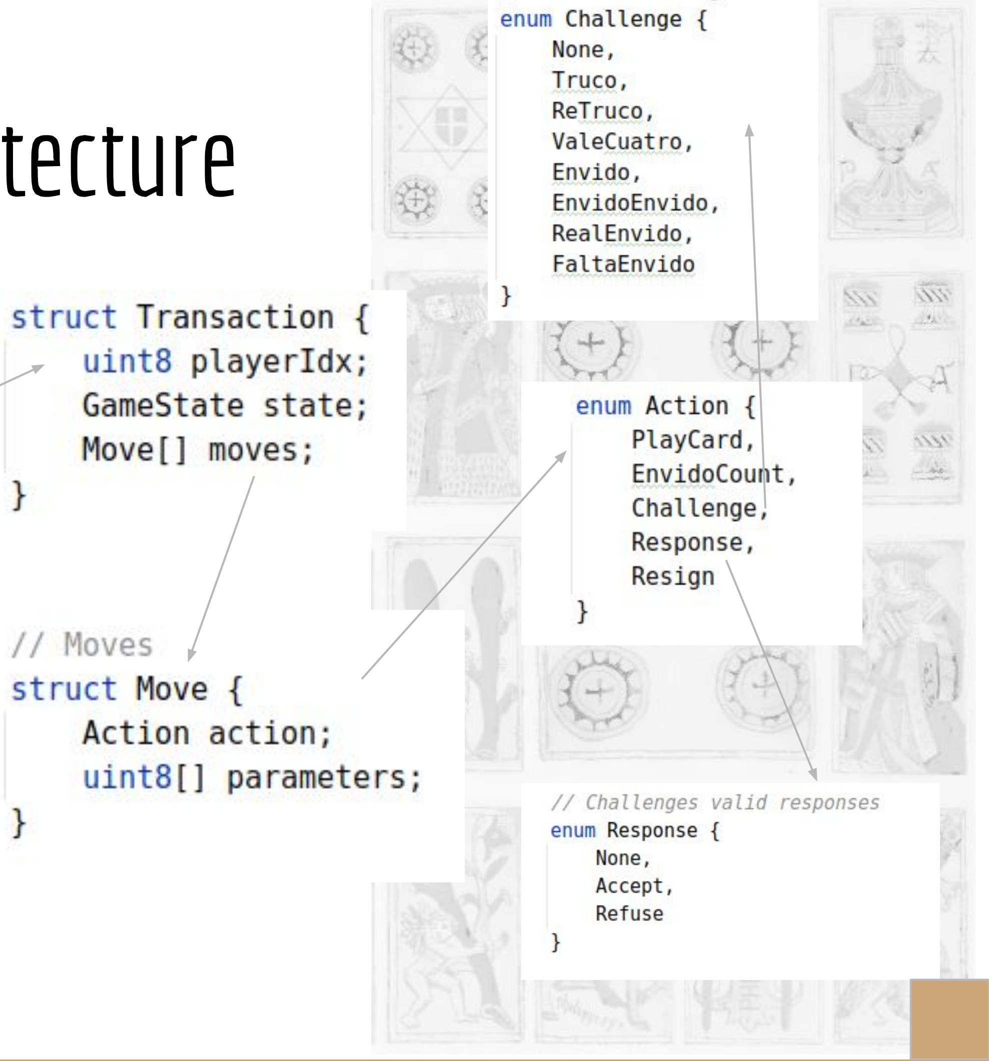
```
// Envido State
struct EnvidoState {
    bool spelled;
    uint8[] playerCount;
    uint8 pointsRewarded;
}
```



Engine Transaction Architecture



```
interface IERC3333 {
    function startGame() external returns (GameState memory);

    function initialGameState()
        external
        pure
        returns (GameState memory _gameState);

    function transact(Transaction calldata transaction)
        external
        returns (GameState memory gameState);

    function isGameEnded(GameState memory gameState)
        external
        view
        returns (bool);

    function getFees() external view returns (uint256);
}
```



```
struct Transaction {
    uint8 playerIdx;
    GameState state;
    Move[] moves;
}
```

```
// Moves
struct Move {
    Action action;
    uint8[] parameters;
}
```

```
enum Challenge {
    None,
    Truco,
    ReTruco,
    ValeCuatro,
    Envido,
    EnvidoEnvido,
    RealEnvido,
    FaltaEnvido
}
```

```
enum Action {
    PlayCard,
    EnvidoCount,
    Challenge,
    Response,
    Resign
}
```

```
// Challenges valid responses
enum Response {
    None,
    Accept,
    Refuse
}
```

The Future 🔮😛

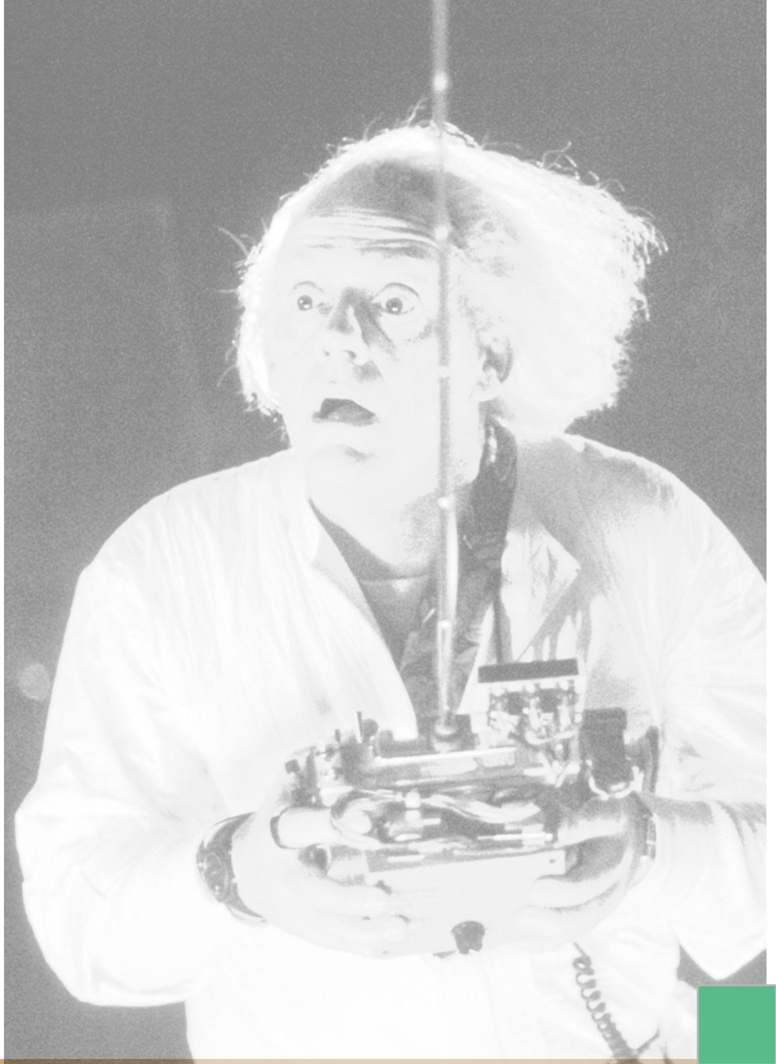
Community:

- Generate reputation for addresses
- Players could choose who to play with

DAO Services:

- DAO could host a website to do “game replays”
- Trophies “Wall of Fame” for reputation
- Community around reputation
- DAO will operate SBT website urls for SBT extra metadata

What else can **you** think of?



Thanks for listening! 🙏

Questions? 🤔 Like it? Want to finance? 🤑