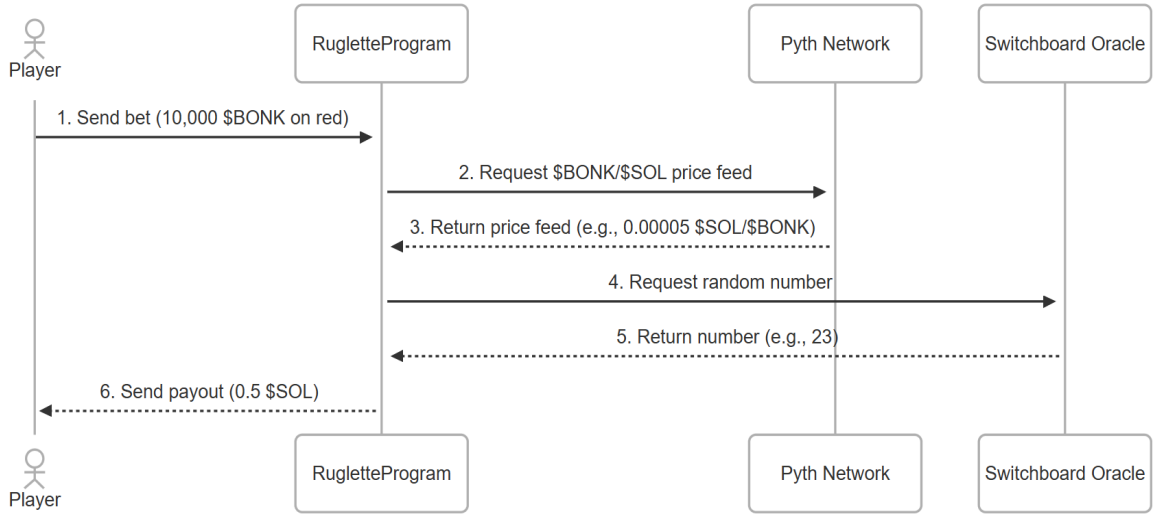
RUGLETTE - On-chain Meme Coin Roulette

Protocol POC Requirements:

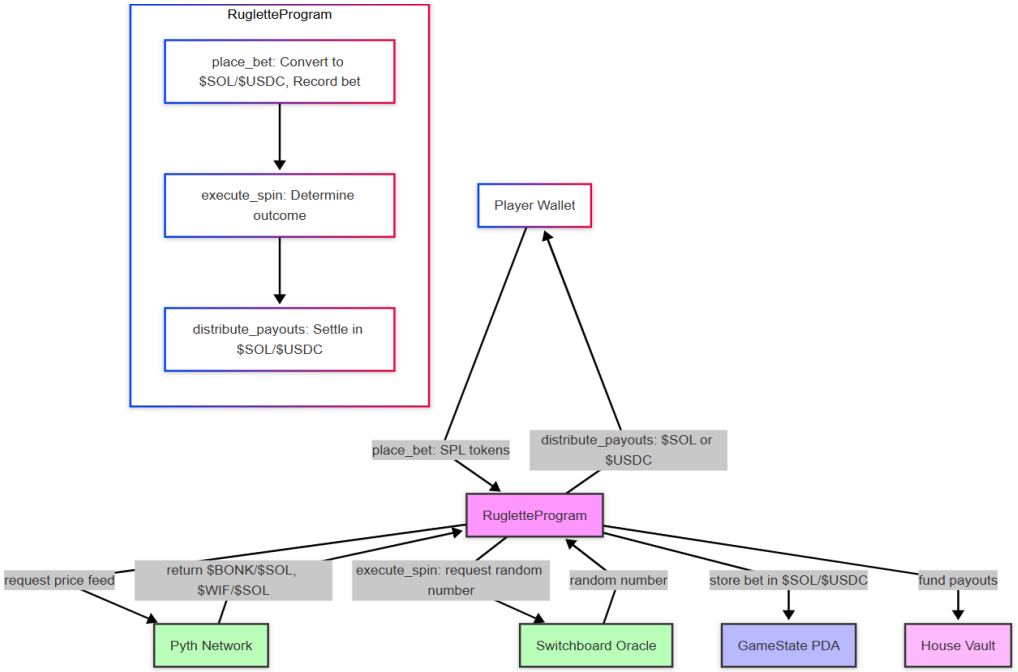
- The protocol shall allow a user to deposit/bet meme coins (e.g., $BONK, $WIF) from a token address whitelist maintained in the program's configuration, allowing only vetted tokens.
- The protocol shall allow a user to deposit/bet $SOL or $USDC directly.
- The protocol shall convert meme coin bets to $SOL/$USDC equivalents using Pyth Network price feeds.
- The protocol shall request a random number (0-36) from Switchboard Oracle to determine the roulette outcome (37 slots, European style).
- The protocol shall distribute payouts in $SOL or $USDC, based on the player's chosen payout currency at bet placement.
- The protocol shall claim a house edge (2.7%, European roulette) on all bets, collecting losses in the house vault.
- The protocol shall initialize the house vault with an admin-funded balance of $SOL and $USDC to ensure payout liquidity, with periodic rebalancing via an AMM (e.g., Raydium).
- The protocol shall integrate with a compliance oracle (e.g., Chainalysis) to verify player wallet addresses against sanction lists before accepting bets.
- The protocol shall allow a user to claim their $SOL/$USDC payout after the spin, if they win.
- The protocol shall include unit tests for bet conversion, spin execution, and payout distribution, with end-to-end testing on Solana devnet.
- The protocol shall undergo a security audit by a reputable firm (e.g., OtterSec) before mainnet deployment.

Overview:



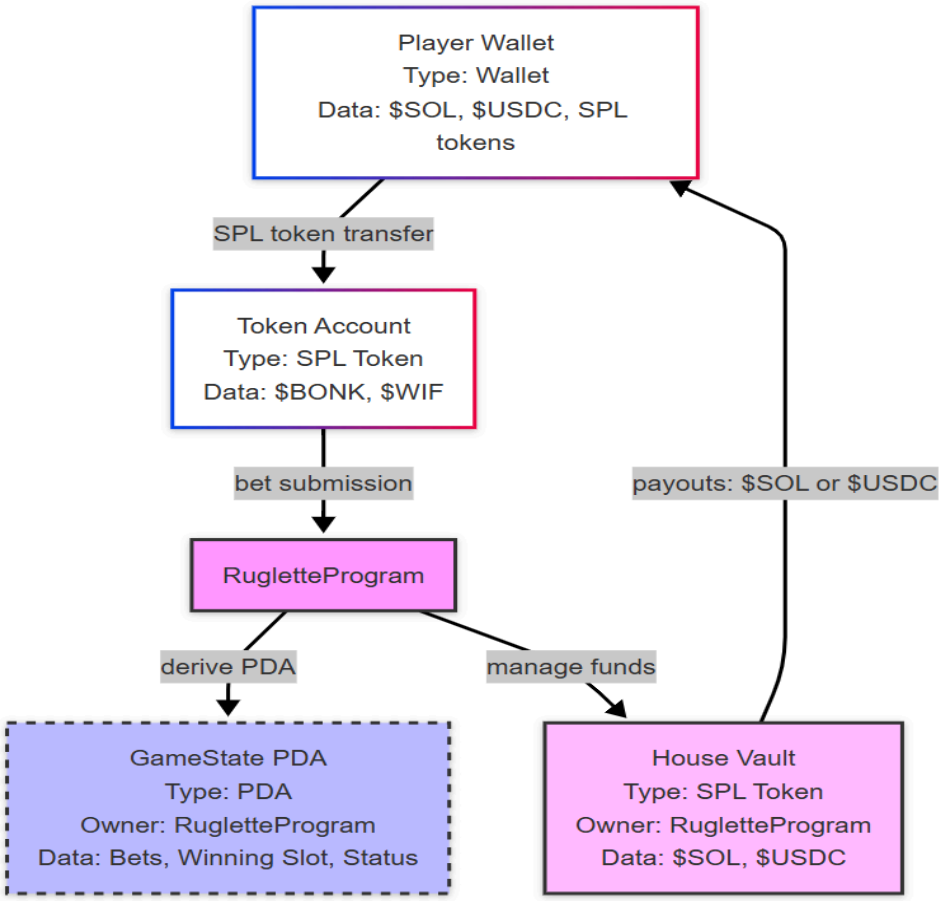
1. **Player Deposits Meme Coins or $SOL/$USDC**
    a. on-chain protocol account (GameState PDA) is created with rent-exempt lamports, funded by the player, to store bet details.
    b. Tokens are deposited in the player's SPL token account and recorded in the GameState PDA.
2. **Bet is Converted to $SOL/$USDC**
    a. The program will convert meme coin bets to $SOL/$USDC using Pyth Network. If the price feed fails, the bet is rejected with an error.
    b. Bet details (amount in $SOL/$USDC, type, numbers) are stored in the GameState PDA.
3. **Player Executes Spin**
    a. The program requests a random number from Switchboard Oracle. If the request fails, the spin is aborted with an error.
    b. The program determines the winning slot (0-36) based on the random number.
4. **Program Distributes Payouts**
    a. The program calculates payouts using standard European roulette odds (e.g., red/black 1:1, single number 35:1), enforcing a 2.7% house edge.
    b. The player receives a $SOL/$USDC payout if they win; otherwise, the house vault collects the bet.
5. **Player Receives Payout**
    a. After spin completion, the program automatically transfers the $SOL/$USDC payout to the player's wallet if they win, or resets the GameState PDA if they lose.

Program Structure Visualization:



- **Program**: A single Solana program (RugletteProgram) handles betting, spin execution, and payouts for simplicity and modularity.
- **Program Initialization**: The RugletteProgram is deployed on Solana devnet with an admin authority to initialize the house vault and whitelist meme coins.
- **Responsibilities**:
    - Accept bets in $SOL or meme coins (via SPL tokens).
    - Request random numbers from an oracle (e.g., Switchboard).
    - Calculate and distribute payouts based on European roulette rules (37 slots, 2.7% house edge).
- **Instructions**:
    - **place_bet**: Records a player's bet (amount, type, numbers) and calls Pyth Network for meme coin conversion to $SOL/$USDC. Reverts with an error if the bet is invalid (e.g., unsupported token, failed price feed).
    - **execute_spin**: Requests a random number (0-36) from Switchboard Oracle, validates the range, and determines the outcome. Reverts with an error if the oracle request fails.
    - **distribute_payouts**: Calculates and sends $SOL/$USDC payouts to the player if they win, or transfers the bet to the house vault if they lose.
- **Interactions**: The program interacts with player wallets, a GameState PDA, and an oracle for randomness. No cross-program invocations (CPIs) are needed for this MVP, keeping it self-contained. Future iterations may introduce CPIs for features like jackpots or tournaments.

Account Structure Mapping:



Accounts:

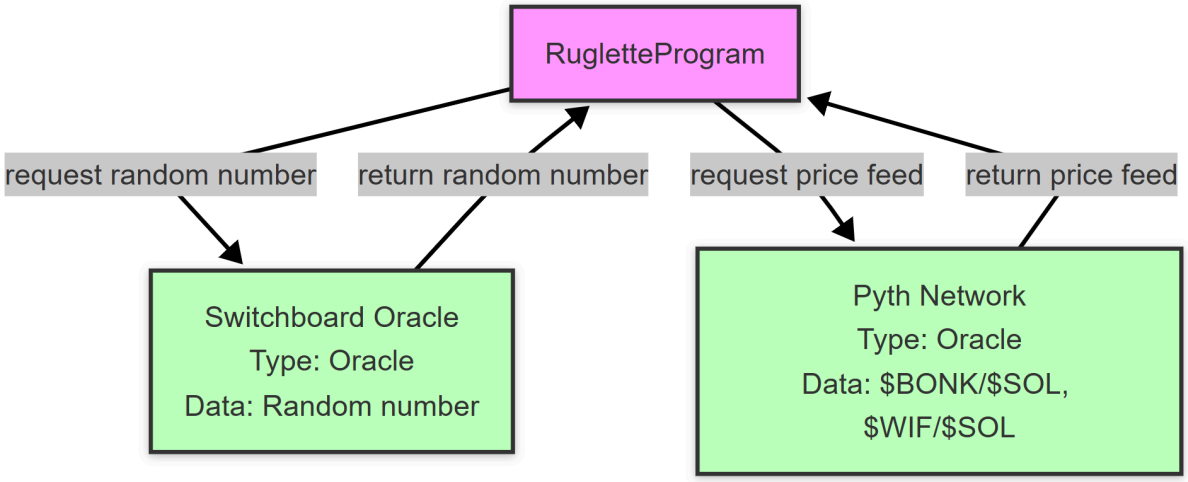
- **Player Wallet**: Owned by the player, holds $SOL or meme coins (SPL tokens). Used to send bets and receive payouts.
- **GameState PDA**: Owned by RugletteProgram, stores current game state (bets, winning slot, status). Initialized with rent-exempt lamports funded by the player.
- **House Vault**: A token account owned by RugletteProgram, collects losses and funds payouts. Holds only $SOL/$USDC for payouts and bet collection
- **Token Accounts**: SPL token accounts for meme coins (e.g., $BONK, $WIF) and $SOL/$USDC, created by the player's wallet if needed. The program ensures token transfers are completed before spin execution.

PDAs:

- The GameState PDA is derived from the program ID and a game ID seed (e.g., ["game", game_id]).
- Stores bet details (player pubkey, amount, bet type) and spin outcome.

Relationships: The program manages the PDA and interacts with token accounts for transactions. The GameState PDA is reset after each spin, returning any remaining rent to the player.

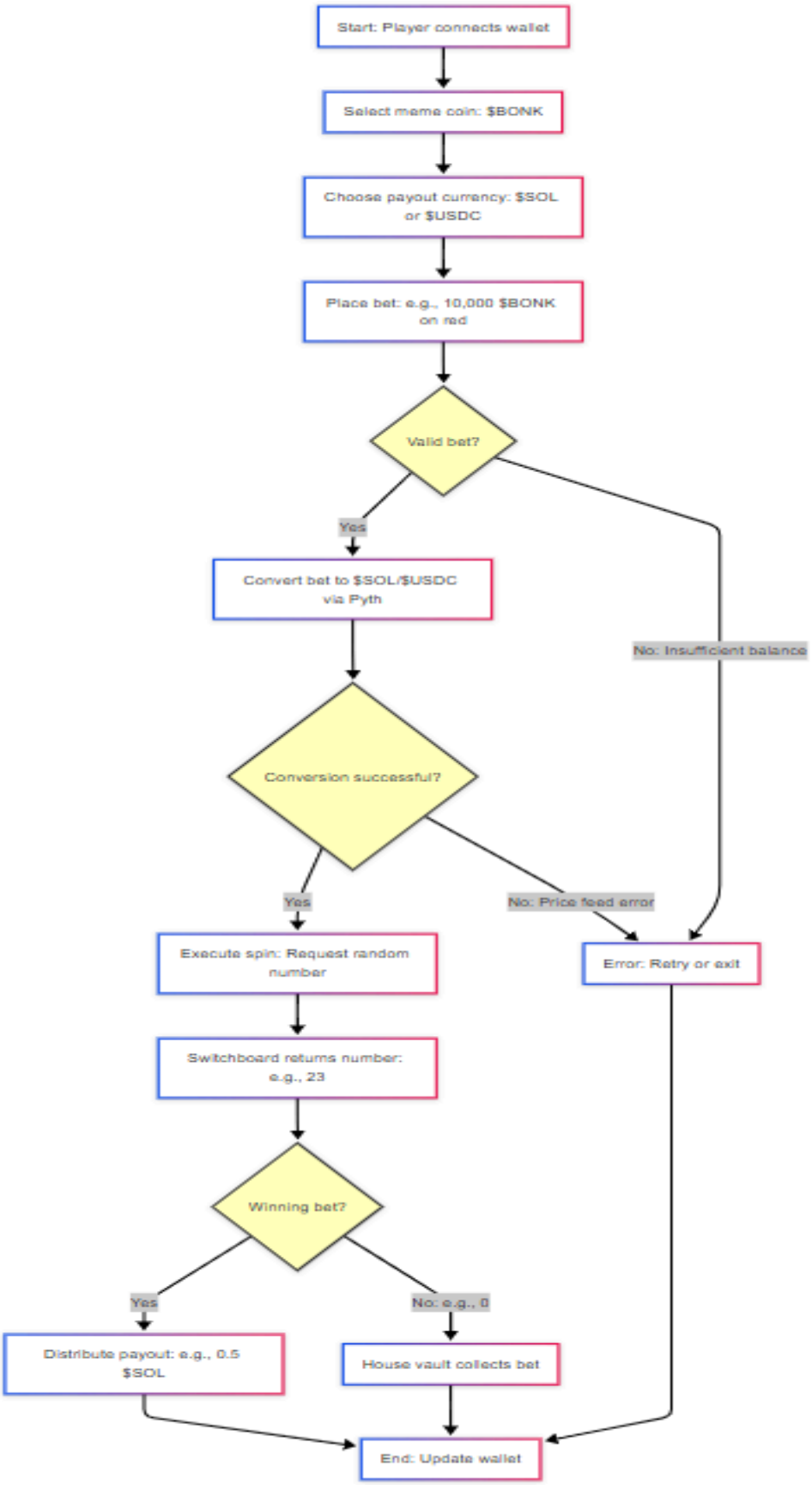
External Dependencies:



Explanation:

- **Dependencies**:
  - Switchboard Oracle: Provides provably fair random numbers (0-36) for the roulette spin.
  - Pyth Network: Supplies real-time price feeds for meme coins (e.g., $BONK/$SOL) to convert bets.
- **Interactions**:
  - RugletteProgram requests a random number from Switchboard during execute_spin, verifying the number is within the range 0-36. If the oracle request fails, the spin is aborted with an error.
  - The program queries Pyth Network for meme coin prices during place_bet to handle conversions. If the price feed is unavailable or stale, the bet is rejected with an error.

Complete User Interaction Flow:



Explanation:

- **Flow**:
    - Player connects wallet, selects a meme coin (e.g., $BONK), and chooses a payout currency ($SOL or $USDC).
    - Player places a bet (e.g., on red).
    - Program converts the bet to $SOL/$USDC equivalent via Pyth, based on the player's chosen payout currency.
    - Program requests a random number and determines the outcome.
    - Payouts are distributed in $SOL/$USDC if the player wins (e.g., 1:1 for red/black, 35:1 for single number); otherwise, the house vault collects the bet, enforcing a 2.7% house edge.
- **Decision Points:**
    - Valid bet? (Enough balance, supported token)
    - Winning bet? (Matches outcome, e.g., red vs. 0)
- **Error Paths**: Handle insufficient balance, invalid bet types, or oracle failures by rejecting the bet and notifying the player to retry or exit.