A Provably Fair Blockchain Raffle Utilizing Chainlink VRF

Fairness You Can Verify.

"Fairness equals Equality!"

Utilising revolutionary technology to create something for all of us!

100% fair raffles

Our blockchain raffle utilises Chainlink VRF to guarantee 100% fairness, ensuring that all participants have an equal chance of winning.

Low fees for High Rewards

Low ticket prices combined with the low gas fees of the Polygon Layer 2 Network ensure accessibility and inclusion for a global audience.
Let’s bring people closer! Let’s play together!

Permissionless 24 / 7

Our raffle can be accessed and played at any time, by anyone, 24/7, from anywhere in the world, making it convenient and accessible to everyone.

Fair

Verifiable

Decentralized

Zero Risk and Full Transparency

Thanks to Blockchain Technology the need for trusting software is so 20st century!
We can finally verify, relax and enjoy the game!

Verified Smart Contract on Polygon

Our Raffle Smart Contract Code is deployed on the Polygon Blockchain, verified and well documented so you can educate and verify yourself before joining the game!

User-Friendly Experience

Our platform provides an easy and transparent User Interface so you have a great time playing!

The Smart Contract Code

The essential functions and Logic!

function buyTicket() external payable

This function gets executed when you click the Buy tickets now! in the UI


```
function buyTicket() external payable {
        if (msg.value != i_entranceFee) {
            revert Raffle__InvalidEntranceFee();
        }
        if (s_raffleState != RaffleState.OPEN) {
            revert Raffle__RaffleNotOpen();
        }
        s_players.push(payable(msg.sender));
        emit EnteredRaffle(msg.sender);
    }
```

The transaction reverts if:

- The value of the transaction does not match the entrance fee
- The state of the raffle is not open, because it is currently drawing the next winner

We keep a small amount of each entrance fee to pay for:

- Chainlink Subscription & Automation
- Hosting the Website
- Maintenance (yes, also Food for the Developers)

The Smart Contract Code

The essential functions and Logic!

function checkUpkeep() public view returns (bool upKeepNeeded, ...)

This function let’s the Chainlink VRF check if it is time to draw the next winner!

```
function checkUpkeep(
        bytes memory /* checkData */
    ) public view returns (bool upKeepNeeded, bytes memory /* performData */) {
        bool itsTimeToDraw = (block.timestamp - i_interval) >=
            s_lastDrawTimestamp;
        bool isOpen = RaffleState.OPEN == s_raffleState;
        bool hasBalance = address(this).balance > 0;
        bool hasPlayers = s_players.length > 0;
        upKeepNeeded = (itsTimeToDraw && isOpen && hasBalance && hasPlayers);
        return (upKeepNeeded, "0x0");
    }
```

The function name must be “checkUpkeep” so Chainlink VRF can find it.

It is time to draw the next winner if

- enough time has passed since the last draw (itsTimeToDraw)
- the state of the raffle is currently open (not calculating)
- there is price money in the raffle contract
- there are any players signed up for the raffle!

The Smart Contract Code

The essential functions and Logic!

function performUpkeep(bytes calldata /* performData */) external

This function gets executed when it is time to draw a winner!
It updates the state of the raffle to CALCULATING, then asks for a random value via the Chainlink VRF subscription.

The function name must be “performUpkeep” in order to work with Chainlink VRF.

```
function performUpkeep(bytes calldata /* performData */) external {
        (bool upKeepNeeded, ) = checkUpkeep("");
        if (!upKeepNeeded) {
            revert Raffle__UpKeepNotNeeded(
                address(this).balance,
                s_players.length,
                uint256(s_raffleState)
            );
        }
        s_raffleState = RaffleState.CALCULATING;
        // Get the random number
        // Will revert if subscription is not set and funded.
        i_vrfCoordinator.requestRandomWords(
            i_gasLane, // gas lane (keyHash) see here:
https://docs.chain.link/vrf/v2/subscription/supported-networks
            i_subscriptionId, // subscription funded with link
            REQUEST_CONFIRMATIONS, // number of block confirmations
            i_callbackGasLimit, // gas limit for callback
            NUM_WORDS // number of random values requested
        );
    }
```

The transaction reverts if not enough time has passed to draw the next winner!

If enough time has passed “checkUpkeep” returns true for variable “upKeepNeeded”.

In programming, an exclamation mark infront of a boolean value checks for NOT, so if NOT upKeepNeeded, it is not time to draw the next winner.

The Smart Contract Code

The essential Functions and Logic!

function fulfillRandomWords(..., uint256[ ] memory randomWords) internal override

This function gets executed by the Chainlink Automation service and receives the random number needed to draw a winner and complete the Raffle!

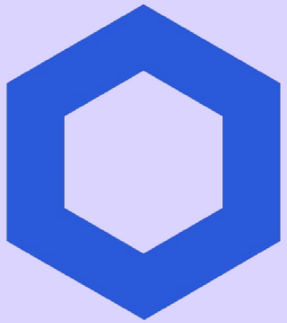
```
function fulfillRandomWords(
        uint256 /* requestId */,
        uint256[] memory randomWords
    ) internal override {
        /// @notice determines the index of the winner
        uint256 winnerIndex = randomWords[0] % s_players.length;
        /// @notice picks the winner from the list of players
        address payable winner = s_players[winnerIndex];
        /// @notice remembers the most recent winner
        s_recentWinner = winner;
        /// @notice sets the state to open again
        s_raffleState = RaffleState.OPEN;

        /// @notice the raffle is completed so we reset the players array
        s_players = new address payable[](0);
        /// @notice we reset the last draw timestamp so the raffle is restarted
        s_lastDrawTimestamp = block.timestamp;
        /// @notice sends the prize money to the winner of the raffle
        (bool success, ) = winner.call{value: getPricePool()}("");

        /// @notice emits the winner drawn event
        emit WinnerDrawn(winner);

        /// @notice in case something goes wrong with the transfer, we revert the transaction in
order to save the money
        if (!success) {
            revert Raffle__TransferFailed();
        }
        /**
         * @notice sends the remaining balance to the contract owner for refunding chainlink automation
 and VRF Subscription
         * and as incentive for the contract owner to maintain the project
         */

        (bool success2, ) = s_owner.call{value: address(this).balance}("");
        if (!success2) {
            revert Raffle__TransferFailed();
        }
    }
```

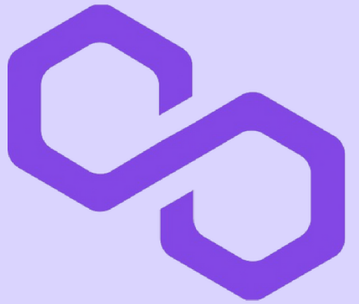
The transaction reverts if:

- the Price Money could not be sent to the winner! (Security First!)
- the Maintenance Money could not be sent to the developer

All money transfers are 100% automated!
No one, also not the developer of the Smart Contract, can withdraw any funds by themselves!

Learn more and verify further!





Look up and read through the verified Smart Contract code on Mumbai Polyscan and check that the code is indeed doing what we claim it is doing! => Verify here
If you have any more questions, feel free to contact me via e-mail at thomas@blissofkundalini.yoga or Join our Web3 Coding Group on Discord