



Technical Documentation

Shadow Protocol

Contents

1	Introduction	3
2	Core Architecture	3
2.1	Main Components	3
3	Key Features	3
3.1	Token Deployment	3
3.2	Token Creation	4
3.3	Position Management	6
3.4	Fee Collection	7
4	Security Features	7
4.1	Access Control	7
5	Uniswap V3 Integration	7
5.1	Liquidity Management	7
6	Conclusion	7

1 Introduction

Shadow Protocol is a smart contract system that enables automated deployment of ERC20 tokens with initial liquidity on Uniswap V3. It features sophisticated price management mechanisms and fee collection systems. Once the token is deployed, the contract automatically creates a pool on Uniswap V3 with the token and the SHADOW token. Shadow Protocol is a modular contract that can be used to deploy an ERC20 token on any EVM compatible chain. As we want to develop a project which could be multi-chain, the project will be using LayerZero solution to communicate between chains.

2 Core Architecture

2.1 Main Components

- `contracts/`
 - `Shadow.sol` - Main deployment contract
 - `Token.sol` - ERC20 token implementation
 - `LogCalculator.sol` - Price calculation utilities
 - `FeeDistributor.sol` - Fee management
- `scripts/`
 - `deployToken.js` - Token deployment script
 - `deploy.js` - Shadow deployment script
 - `collectFees.js` - Fee collection utilities

3 Key Features

3.1 Token Deployment

The `deployToken` function allows creation of a new token with customizable parameters:

- Name
- Symbol
- Supply
- Liquidity
- Max Wallet percentage

```

1 function deployToken(
2     string calldata name,
3     string calldata symbol,
4     uint256 supply,
5     uint256 liquidity,
6     uint24 fee, //Corresponding to Uniswap V3 values
7     bytes32 salt, //Calculated by the contract
8     address deployer,
9     uint256 maxWalletPercentage
10 ) external payable {
11     // Validation checks
12     require(msg.value > 0, "Must send ETH for initial buy");
13     require(!_isShadowTokenEnabled, "Shadow token payment not enabled");
14     require(shadowToken != address(0), "Shadow token not set");
15     // ... additional checks ...
16 }

```

3.2 Token Creation

The token creation process follows several key steps:

1. Predict the token address

```

1 function predictTokenAddress(){
2     // Generate a random salt based on severals
3     // parameters and using pre-determined salt-value
4     // to avoid snipers in the token creation block
5 }

```

2. Generate the salt

```

1 function generateSalt(){
2     // Generate a random salt using predictTokenAddress
3     // function until the token address is smaller than
4     // the SHADOW token address (Uniswap V3 requirement)
5 }

```

3. Initial Checks

- Verify SHADOW token payment for initial buy (Enabled, Token address, Fees, Fee recipient)
- Validate deployment parameters (name, symbol, supply, etc.)

4. Token Deployment

```

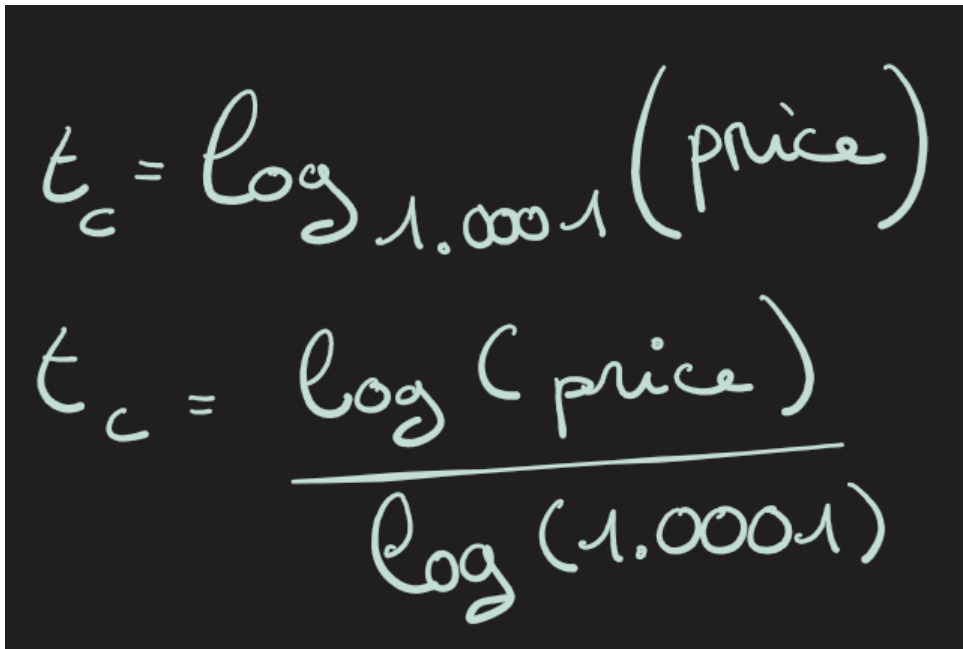
1 Token token = new Token{salt: create2Salt}(
2     finalName,
3     symbol,
4     supply,
5     deployer,
6     address(uniswapV3Factory),
7     address(positionManager),
8     maxWalletPercentage
9 );

```

5. Price Calculation

```
1  function getShadowWETHPrice(){
2      // Get the price of the SHADOW token in WETH
3      // using the Uniswap V3 TWAP (Oracle)
4  }
5
6  function calculatePriceTokenShadow(){
7      // Return the price of the token in the pool
8      // based on the supply, the liquidity and the
9      // price of the SHADOW token in WETH
10 }
```

6. Calculate Tick



The image shows two handwritten equations on a black background. The first equation is $t_c = \log_{1.0001}(\text{price})$. The second equation is $t_c = \frac{\log(\text{price})}{\log(1.0001)}$.

Figure 1: Tick Calculation Process

The tick is calculated using the following steps:

```
1  function getTick(uint256 priceSHADOW) {
2      uint256 price = Log10(priceSHADOW);
3      int24 tick = price / Log(1.0001);
4      return tick;
5  }
```

This value is crucial for Uniswap V3 pool initialization and determines the starting price of the pool.

7. Pool Creation

```
1  uint160 sqrtPriceX96 = getSqrtRatioAtTick(tick);
2  address pool = uniswapV3Factory.getPool(params);
3  if (pool == address(0)) {
4      pool = uniswapV3Factory.createPool(params);
5      IUniswapV3Pool(pool).initialize(sqrtPriceX96);
6  }
```

8. Liquidity Provision

```
1    MintParams memory params = MintParams(params);
2    struct MintParams {
3        address token0;
4        address token1;
5        uint24 fee;
6        int24 tickLower;
7        int24 tickUpper;
8        uint256 amount0Desired;
9        uint256 amount1Desired;
10       uint256 amount0Min;
11       uint256 amount1Min;
12       address recipient;
13       uint256 deadline;
14   }
```

9. Initial Market Making

```
1        uint256 swapAmount = 1;
2        ISwapRouter.ExactInputSingleParams({
3            tokenIn: shadowToken,
4            tokenOut: address(token),
5            fee: fee,
6            recipient: address(this),
7            amountIn: swapAmount,
8            amountOutMinimum: 0,
9            sqrtPriceLimitX96: 0
10       });
```

The entire process is atomic - if any step fails, the entire transaction reverts, ensuring safety of user funds.

3.3 Position Management

The contract tracks Uniswap V3 positions for each token:

```
1 // Mapping to store position IDs for each token
2 mapping(address => uint256[]) public tokenPositions;
3
4 function _addTokenPosition(address token, uint256 tokenId) internal {
5     tokenPositions[token].push(tokenId);
6 }
```

3.4 Fee Collection

The collectAllUniswapFees function handles fee collection and distribution:

```
1 function collectAllUniswapFees() external {
2     uint256 totalAmount0;
3     uint256 totalAmount1;
4
5     for (uint256 i = 0; i < tokenPositions[msg.sender].length; i++) {
6         uint256 tokenId = tokenPositions[msg.sender][i];
7         if (positionManager.ownerOf(tokenId) == address(this)) {
8             // Collect and distribute fees
9         }
10    }
11 }
```

4 Security Features

4.1 Access Control

The contract implements various access controls:

```
1 modifier onlyDeployer() {
2     require(msg.sender == _deployer, "Only Deployer can call");
3     _;
4 }
5
6 modifier onlyOwner() {
7     require(msg.sender == owner(), "Only owner can call");
8     _;
9 }
```

5 Uniswap V3 Integration

5.1 Liquidity Management

- NFT position minting
- Price range calculation
- Initial liquidity distribution

6 Conclusion

Shadow represents a comprehensive solution for automated token deployment with liquidity. Its modular architecture and security mechanisms make it a robust platform for token creation and management.