Design Decisions and Testing Strategy:

OTCSwap Smart Contract

Design Decisions

Atomicity of Swaps

The primary requirement for this contract is to ensure that token swaps between two parties (Alice and Bob) are atomic. This means that either both tokens are exchanged successfully, or neither are, ensuring the integrity of the swap. The executeSwap function ensures atomicity by transferring tokens from both parties within a single transaction. If any part of the transfer fails, the transaction reverts, leaving the state unchanged.

Specifying Counterparties

To prevent unauthorized third parties from participating in the swap, the contract requires that the counterparty be explicitly specified during the swap creation. This is enforced in the executeSwap function, where only the designated counterparty can call the function to execute the swap.

Expiration of Swaps

Swaps are designed to expire after a specified timeframe to ensure that they do not remain pending indefinitely. This is managed by checking the current block timestamp against the swap's expiry timestamp in the executeSwap function. If the swap has expired, the execution is not allowed.

Swap Identifiers

Unique swap identifiers are generated using a hash of the swap parameters, including the initiator and counterparty addresses, token addresses, amounts, and expiry. This ensures that each swap is uniquely identifiable and prevents duplicate swaps with the same parameters.

Security Considerations

Reentrancy Protection

To protect against reentrancy attacks, the contract uses OpenZeppelin's ReentrancyGuard. This ensures that no nested (reentrant) calls can be made to the contract's functions, which could otherwise lead to unexpected behaviors or vulnerabilities.

Validating Inputs

The contract includes several require statements to validate inputs:
- Ensuring non-zero addresses for counterparties and tokens.
- Ensuring token amounts are greater than zero.
- Ensuring the expiry time is in the future.

These checks prevent invalid data from being processed and help maintain the integrity of the contract’s state.

Token Transfers

The contract relies on the ERC20 transferFrom function for token transfers. This requires that the involved parties have previously approved the contract to transfer their tokens on their behalf. This two-step process (approval followed by transfer) adds an extra layer of security by requiring explicit consent from the token holders.

Event Emissions

Events are emitted for key actions within the contract (swapCreated, swapExecuted, swapCancelled). These events provide an on-chain record of what actions have occurred, which can be useful for off-chain monitoring and debugging.

Testing Strategy for OTCSwap

To ensure the OTCSwap smart contract operates as intended, we need to thoroughly test its core functionalities, including swap creation, execution, and cancellation, as well as handle various edge cases.

First, for swap creation, we should test both successful and invalid scenarios. A successful swap creation involves deploying the contract, initiating a swap with valid parameters, and confirming the emission of the swapCreated event with correct details. Additionally, we should verify that the swap details are stored accurately. Testing invalid parameters involves attempting to create swaps with zero token addresses or amounts and ensuring the transaction reverts with appropriate error messages.

For swap execution, the focus is on ensuring correct token transfers and authorization. A successful execution test involves creating a swap, approving token transfers, executing the swap, and verifying that tokens are exchanged as expected, alongside checking the swapExecuted event. We also need to test unauthorized executions by attempting to execute the swap from a non-designated address, which should revert with an error message. Another critical test is ensuring that swaps cannot be executed after their expiry, which involves creating a swap, fast-forwarding time beyond its expiry, and attempting execution to confirm it reverts with an "expired" message.

Swap cancellation tests are necessary to confirm that only the initiator can cancel a swap before execution. This involves creating a swap, canceling it from the initiator's address, and verifying the swapCancelled event and removal of swap details. Testing unauthorized cancellations ensures that non-initiators cannot cancel swaps, which should revert with an appropriate error message.

Edge cases include testing for duplicate swap executions and non-existent swap executions. After executing a swap, any further attempt to execute the same swap should revert, ensuring it cannot be executed more than once. Similarly, attempts to execute swaps with invalid or non-existent IDs should revert with an error message. Lastly, creating swaps with past expiry times should be tested to ensure the contract rejects such attempts.

This comprehensive testing strategy covers key functionalities and edge cases, ensuring the OTCSwap smart contract's reliability and security in real-world scenarios.

Decentralized Marketplace Contract

Design Decisions

The design decisions for the Marketplace smart contract revolve around user registration, item listing, price updates, relisting, item purchases, and funds withdrawal. Users can register with unique usernames, ensuring username uniqueness and rejecting empty usernames. Registered users can list items with names, descriptions, and prices, with the requirement that prices must be greater than zero. Item owners have the ability to update item prices, but only for items that haven't been sold, and the new price must also be greater than zero. Additionally, owners can relist items that have been sold, updating their status and price accordingly. Items can be purchased by users, updating ownership and marking them as sold, provided the correct Ether value is sent, and the item hasn't been sold before. Users can also withdraw their available funds from sales.

Testing Strategy for DecentralizedMarketplace

The testing strategy encompasses various aspects of the contract's functionalities. It includes testing user registration with valid and invalid usernames, ensuring uniqueness, and rejecting empty usernames or registrations from already-registered users. Item listing testing involves verifying that items are correctly listed with accurate details and are initially marked as not sold, while also checking for rejection of zero-priced listings. Price update testing focuses on validating that only item owners can update prices, for items that haven't been sold, and rejects updates with zero values. Item relisting testing ensures that only owners can relist items that are not sold, updating their status and price, and rejecting zero-priced relists.

Item purchase testing covers scenarios where users purchase listed items, ensuring ownership updates, marking items as sold, rejecting incorrect Ether values for purchase, and preventing multiple purchases of the same item or purchases of already sold items. Funds withdrawal testing confirms that users can withdraw their available funds and that insufficient funds result in rejection.