# Introduction

Smart contract programming languages, similar to those used in conventional software development, facilitate developers' transition to smart contract development. Due to reasons such as newly discovered vulnerabilities and continuous iteration of requirements, the issue of smart contract maintenance has increasingly gained attention. However, recent incidents have shown that existing smart contract maintenance technologies cannot adequately handle data state changes, resulting in operational complexity and risks to user financial security. Therefore, it is crucial to study methods to ensure the correct maintenance of stateful smart contracts. A significant aspect is the capability to deploy new versions of smart contracts and ensure consistency in the smart contract state before and after the maintenance. This study mainly focuses on evaluating the effort needed to maintain stateful smart contracts (i.e., deploying new smart contracts and state migration between old and new versions) in the context of the Ethereum blockchain and the commonly used Solidity programming language.

In this study, there are two main tasks and several sub-tasks. Please execute each task in the given order across all 10 contract samples, each representing different levels of complexity. For each sample, measure the time and gas cost required for each task, and answer the questions on the provided answer sheet. Make sure to read through the task descriptions in this document before starting.

We provide a pre-configured VM (username: study_user, password: 12345) with all necessary tools and an answer sheet installed. You will find all relevant files in the "Smart Contract Maintenance " directory, which is easily accessible from the Desktop link. You can access the local Remix IDE through the web browser at http://localhost:8080, either by using the browser bookmarks or the desktop icon. In this way, you can test the functionalities of the smart contract.

# Starting Questionnaire:

1、Please answer the following questions.
**Q1:** Have you ever conducted research in the field of Blockchain?
**Q2:** Did you write Solidity code in the last two weeks?
**Q3:** Have you previously worked on a production-grade Solidity-based Ethereum contract?
**Q4:** How many years of experience do you have in Solidity development?

2、Please answer the following questions by rating on a scale from 1 to 7, where 1 means you are not familiar at all (e.g., you have heard of the topic but have no understanding), and 7 means that you are most familiar (e.g., you have expert knowledge and extensive experience in the field).
**Q5:** How familiar are you with the Ethereum Blockchain in particular?
**Q6:** How familiar are you with the Solidity programming language?
**Q7:** How familiar are you with Solidity contract maintenance techniques?

# Task 1: Convert Contracts to Follow Delegatecall-based Pattern

In this task, you will convert a smart contract to follow *the delegatecall-based pattern.* Unlike software, smart contracts' code cannot be changed once they are deployed. However, for various reasons (e.g., bug fixes and new feature implementation), there has been a great demand for making smart contracts upgradable, and the most popular method is *the delegatecall-based pattern.*

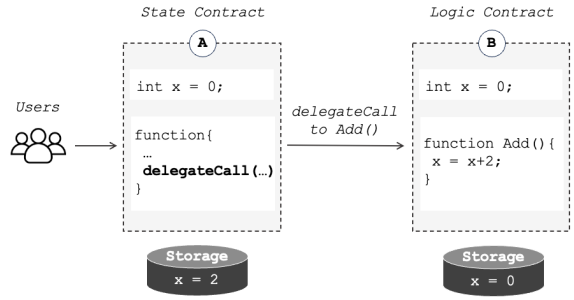


Figure 1: The delegatecall-based pattern.

The delegatecall-based pattern splits one contract into two: a state contract with address *A* and a logic contract with address *B*. The state contract holds the storage state as well as a changeable pointer to the logic contract. As shown in Figure 1, the state contract redirects the users' call to the logic contract by using `delegatecall` and executes the actual `Add()` in the state contract's context, resulting in an update of variable *x* in the state contract.

Your task is to convert an existing contract into this delegatecall-based pattern. If you are not familiar with it, you first need to understand the complexities of using this pattern and how it works.

Finally, you will use an automated tool, *SmartUpdater*, to convert the contract with hyperproxy-based contract maintenance pattern. In this pattern, the contract separates state and logic into contracts which are further divided into multiple state sub-contract and logic sub-contracts to minimize overall deployment and maintenance gas costs. Additionally, a hyperproxy contract is introduced to manage the updates of the underlying logic and state contracts, ensuring that any updates to the state or logic are transparent to end users.

## Step 1 – Manual Conversion

Begin by navigating to the task-1/1-manual directory and opening the contract within the Remix IDE. Your primary task involves implementing the delegatecall-based pattern for these contract samples, with its source code located in `.sol` files. After implementing the delegatecall-based pattern, proceed to test the new contract by deploying it directly from the Remix IDE.

Please note that no additional guidance will be provided to simulate varying levels of prior knowledge. You are encouraged to utilize any available resources to complete this task. To ensure the task is completed within the allotted study time, please set a timer and limit your work to a maximum of 3 hours.

## Step 2 – Conversion with OpenZeppelin

Begin by navigating to the task-1/2-openzeppelin directory and opening the contract within the Remix IDE. Your primary task is using the maintenance method provided by OpenZeppelin to

implement the delegatecall-based pattern for these contract samples.

You can follow the guide in OpebZeppelin documents (https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable). After implementiation, proceed to test the new contract by deploying it directly from the Remix IDE.

## Step 3 – Automatic Conversion with SmartUpdater

Navigate to the task-1/3-SmartUpdater directory. You can open the terminal and run the SmartUpdater deployment script:

```
$ python smartupdater_deploy.py --help

usage: smartupdater_deploy.py [-h] <contract_source>

SmartUpdater Command Line Interface for Contract Conversion


positional arguments:
  contract_source      Path to the Solidity contract source file

optional arguments:
  -h, --help           Show this help message and exit
```

You need to provide the path of the contract, for example:

```
Example usage:

  $ python smartupdater_deploy.py ./source.sol
```

SmartUpdater will then automatically convert the given contract to follow the hyperproxy-based contract maintenance pattern.

## Questionnaire

1、Please answer the following questions with either yes or no.
**T1Q1:** Have you previously used the delegatecall-based pattern in a Solidity contract?
**T1Q2:** Have you previously used the maintenance method provided by OpenZeppelin?
**T1Q3:** Have you previously used a different contract maintenance pattern?

2、Please answer the following questions by rating on a scale from 1 to 7.
**T1Q4:** How familiar are you with the delegatecall-based pattern? (1-not familiar, 7-most familiar)
**T1Q5:** How confident are you in the correctness of your manual conversion? (1-least confident, 7-most confident)
**T1Q6:** How confident are you in the correctness of your conversion using OpenZeppelin's method? (1-least confident, 7-most confident)
**T1Q7:** How confident are you in the correctness of the conversion using SmartUpdater ? (1-least confident, 7-most confident)

**T1Q8:** How difficult is the manual conversion? (1-easy, 7-most difficult)
**T1Q9:** How difficult is the conversion with OpenZeppelin's method? (1-easy, 7-most difficult)
**T1Q10:** How difficult is the automatic conversion with SmartUpdater? (1-easy, 7-most difficult)
**T1Q11:** How satisfied are you with the conversion using SmartUpdater? (1-very dissatisfied, 7-very satisfied)

3、Please report the time you required for the step.
**T1Q12:** How much time do you need to manually convert the given contract? (step 1)
**T1Q13:** How much time do you need to convert the given contract using OpenZeppelin's method? (step 2)
**T1Q14:** How much time do you need to convert the contract using SmartUpdater? (step 3)

## Task 2: Complete the Contract Maintenance

During the state update process, it can be summarized in two main steps: 1) creating the new state/logic contracts according to the update requirements, and 2) initializing states to maintain consistency between the old and new contracts. In this task, you will develop a new state contract using the smart contracts created in the previous task along with the provided state update requirements. Additionally, we will supply the state values from the old state contract, which you will need to transfer and recover in the new state contract you create. Finally, you will use the *SmartUpdater* to automate the maintenance process described above.

### Step 1 – Maintenance of Manually Converted Contract

First, navigate to task-2/1-manual_openz and open the require.pdf files. The require.pdf file outlines the specific update requirements of each contract sample for this task. Your task is to implement the new contract by adjusting the code you manual converted in the previous task to meet the update specifications detailed in the `require.pdf`. This includes any necessary modifications or additions to the contract's state structure. Subsequently, test the new contract by deploying it from within the Remix IDE.
Second, navigate to task-2/1-manual_openz and open the `data.xlsx` file that contains the state values from the old contract. Your task is to read these values and integrate them effectively, ensuring that the state is correctly initialized in the new state contract.

### Step 2 – Maintenance of OpenZeppelin-based Contract

The steps are similar to those in Step 1. However, in this step, you will work on the code you converted using OpenZeppelin in the previous task. Follow the same process: refer to the `require.pdf` file for update specifications and modify the contract accordingly. Additionally, use the `data.xlsx` file to integrate the state values from the old contract, ensuring proper initialization in the updated OpenZeppelin-based contract.

Important Note: It is difficult to obtain the data of mapping-type states in the contract from the Ethereum directly. Therefore, we will use a specific number as the count of elements included in

the mapping-type states and construct the corresponding data accordingly. Test these functions within the Remix IDE to verify that the state transfer is successful and that the new contract behaves as expected with the old data.

### Step 3 – Maintenance using SmartUpdater

Navigate to the task-2/2-SmartUpdater directory. First, you need to read the DSL design document and write the corresponding state modifying policy written in DSL according to the state update requirement of each contract sample in `require.pdf` file. With these DSL-written policies, you can open the terminal and run the SmartUpdater mintenance script:

```
$ python smartupdater_maintenance.py --help

usage:smartupdater_maintenance.py [-h] <contract_name> <policy_source>

SmartUpdater Command Line Interface for Contract Maintenance


positional arguments:
  contract_name         Name ofthe Solidity contract
  policy_source    Path to the policy source file


optional arguments:
  -h, --help          Show this help message and exit
```

You need to provide the paths of the contract file and the requirement file , for example:

```
Example usage:

  $ python smartupdater_maintenance.py name ./policy.txt
```

SmartUpdater will then automatically complete the maintenance process described above based on the requirement.

### Questionnaire

1、Please answer the following questions.

**T2Q1:** How confident are you in the correctness of the contract maintenance processed by yourself ? (1-least confident, 7-most confident; step 1 and step 2)

**T2Q2:** How confident are you in the correctness of the contract maintenance processed by using SmartUpdater ? (1-least confident, 7-most confident; step 3)

**T2Q3:** How difficult is the contract maintenance process by yourself? (1-easy, 7-most difficult; step 1 and step 2)

**T2Q4:** How difficult is it to write DSL-written policy by reading the design document? (1-easy, 7-most difficult)

**T2Q5:** How difficult is the maintenance process using SmartUpdater? (1-easy, 7-most difficult)

**T2Q6:** Do you know how to get the values of the state with dynamic storage strategy (e.g., mapping-type state) from the Ethereum blockchain? If you know, please explain in detail how to get them.

2、Please report the time you required for the step.

**T2Q7:** How much time do you need to perform maintenance of the manually converted contract? (step 1)

**T2Q8:** How much time do you need to perform maintenance of the OpenZeppelin-based contract? (step 2)

**T2Q9:** How much time do you need to perform maintenance using SmartUpdater? (step 3)

## Submitting the Results

To submit your results, please ensure that all edited files are saved and all questions in the `answer.xlsx` file are fully answered. By sending your results via email, you agree that your responses and any modifications to the source code will be analyzed and published anonymously as part of a scientific publication. Thank you for your valuable contribution to this study.