Peer to Peer Systems and Blockchains
Academic Year 2017/2018

Final Term
COBrA: Fair COntent Trade on the BlockchAin
Deadline 11-06-2018

1 COBrA: description of the System

The goal of this assignment is to implement a decentralised content publishing service, which enables the authors of new contents to submit their creation (song, video, photo,...) and to be rewarded accordingly to customers' fruition. The users of COBrA are:

- *Customers*: entities that wish to access a new content.
- *Authors*: creators of a new contents

The same user can have different roles at the same time.

Customers may buy two kinds of accounts: *Standard* and *Premium*. *Standard accounts* require a payment for every single content access request, while *Premium accounts* require the payment of a fixed sum coverig a period of time, so enabling an unrestricted access to any content ~~for *x* hours~~, for x time (expressed either in hours or in block height difference) starting from the subscription time. Once a customer has obtained the access to a content, it can consume the content, *at most one time*. Do note that granting access to contents and content consuming are two different operations, that can happen at different times.

A customer can gift another customer, by buying access rights for someone else instead of itself, both for *Standard* and for *Premium Accounts*. Premium accounts cannot gift for free single contents to other users.

2 COBrA: Implementation

The system is implemented through a single *Catalog Smart Contract*, one or more *Content Management Contracts*, and any number of customers. Both the number of content management contracts and customers are dynamic and not capped, no assumption must be done on their value. Each *Content Management Contract* controls exactly one unique content.

The *Catalog Smart Contract* acts like an intermediary between customers and authors, by linking customers to newly published content managed by *Content management contracts*. An author submits a new content by deploying a new smart contract in charge of managing access to it. Such a contract implements a predefined set of functionalities by extending a predefined *Base Content Management Contract*. Afterwards, the author submits a new content publishing request to the *Catalog Contract* to link the new contract to the Catalog. This linking is generated

by calling a method of the *Catalog contract* with the needed information about the *content management contract*.

Customers access the *Catalog contract* to request access to contents. The access to a content is simulated by the *Content Management Contract*. A suggestion is to simulate an access right granting by a dummy function *grantAccess*, i.e. a function that simply sets a content access right to true for that user and the content fruition through another dummy function *consumeContent* which checks the access rights. It is required that the proposed solution takes into account the gas efficiency, in particular regarding the access right checking for Premium accounts.

The Catalog contract collects all the payments from the users and redistribute payouts among authors, granting a fixed payout for every v content views (the payout is triggered each time v new views are reached). The Content Management Contract must be written to prevent views tampering. For the sake of simplicity, we do not consider Premium Account content fruitions in the view count.

The *Catalog contract* collects all the payments from the users and redistribute payouts among authors, ~~granting a fixed payout for every content access)~~ granting a fixed payout for every v content views (the payout is triggered each time v new views are reached). The *Content Management Contract* must be written to prevent views tampering. For the sake of simplicity, we do not consider *Premium Account* content fruitions in the view count.

The system must be fully decentralized, i.e. the creator of the *Catalog Contract*, after having deployed it, can only close the contract, after having payed the remaining value to authors proportionally to current content views, and cannot receive any reward for itself.

The basic set of functions to be implemented must contain at least the following functions which do not change the state of the contract:

- *GetStatistics()*: returns the number of views for each content.
- *GetContentList()*: returns the list of contents without the number of views.
- *GetNewContentsList(x)*: returns the list of x newest contents.
- *GetLatestByGenre(x)*: returns the most recent content with genre x.
- *GetMostPopularByGenre(x)*: returns the content with genre x, which has received the maximum number of views
- *GetLatestByAuthor(x)*: returns the most recent content of the author x.
- *GetMostPopularByAuthor(x)*: returns the content with most views of the author x.
- *IsPremium(x)*: returns true if x holds a still valid premium account, false otherwise.

and the following functions which modify the state of the contracts:

- *GetContent(x)*: pays for access to content x.

- *GetContentPremium(x)*: requests access to content x without paying, premium accounts only.
- *GiftContent(x,u)*: pays for granting access to content x to the user u.
- *GiftPremium(u)*: pays for granting a Premium Account to the user u.
- *BuyPremium()*: starts a new premium subscription.

The implementation of further functions is left to students discretion.

Note that the project does not require to take into account the customers' privacy. This means that it is not requested to mask in any way customers subscription status and content views history.

3 Requirements

The student must:

- write the smart contracts in *Solidity*, i.e. the *Catalog Contract*, the *Base Content Management Contract* and each single *Content Management Contract* and compile/deploy them on *Remix* [1].
- log important events generated by contracts notifying the outcome of a set of operations. The student may choose which operations to consider, but it is mandatory to fire an event for each granted content access.
- provide a list of operations (contracts deployment, transactions sent, etc...), temporally ordered, enabling to run a meaningful simulation of the system.
- provide an estimation of the *gas* consumed by at least one of the *non trivial functionalities*, (i.e. not completed by a single contract funcion call) implemented by the system, for instance the cost of obtaining the access right to a content.
- define the parameters of the system (for instance, the amount to pay for each content, the lasting time of a premium account, etc.). Their choice must be discussed, when critical.

The assignment must be done individually and its deadline is 11 June 2018. The assignment can be submitted even if the mid term has not been submitted/ is not sufficient. If the evaluation of both the mid and of this final term will be positive, the student will be relieved from the oral exam. Submit the assignment through Moodle. Its evaluation will be notified through Moodle as well.

The assignment is not mandatory, if it is not presented, the student will be required to pass the oral exam on the second part of the course.

References

[1] *Remix - Solidity IDE* https://remix.ethereum.org/.