Decentralized Escrow Via Ethereum Smart Contracts

1 Abstract

The online marketplace and escrow system stand to benefit tremendously from blockchain technology and decentralization. Such systems save transactors money, minimize privacy risks, and give users control over their transactions, rather than relying on often unreliable, restrictive, and expensive third-party platforms. For these reasons, we implement a decentralized, Ethereum-based escrow marketplace that provides a setting for untrusted parties to initiate transactions without authority-bearing third-party arbiters in order to maintain anonymity, transaction integrity, and transparency. With blockchain technology, we are able to leverage smart contracts and use Ether to ensure that both parties remain compliant to their agreement, while also minimizing many of the aforementioned risks and costs of traditional third-party platforms. By implementing smart contracts through the Remix IDE and deploying them to an Ethereum network, and with the use of cryptocurrency wallets such as MetaMask, we are able to run transactions anonymously and securely store them on the Ethereum blockchain. This protocol creates a strong monetary incentive for both parties to go ahead with their transaction, while also keeping their transaction private and secure.

2 Introduction

The emergence of blockchain technology has increased possibilities of decentralization in many industries, one of which is the online marketplace and escrow system. Currently, online marketplace exchanges like Ebay require the private information of users, a cut of the transaction's profits ranging from 0.5% to 15% of the total amount of the sale,[6] and are not very resistant to online faults and attacks. All of these pose risks to buyers and sellers while also costing sellers and buyers more money. However, a decentralized, Ethereum based escrow system would only require one's public wallet address and negligible gas fees instead of a

substantial portion of the seller's profits. Trusted third-party firms such as Ebay were critical prior to blockchain technology and smart contracts since they address the issue of trust between unknown buyers and the sellers in a digital marketplace. These third parties would establish trust between the buyer and seller by acting as a mediator that ensures that the buyer will pay and the seller will deliver. However, with smart contracts, the open source code may replace such firms as mediators, while also eliminating the need for trust between transactors, but without the aforementioned drawbacks of centralized third parties.

Additionally, decentralized platforms are more tolerant to faults and more resistant to attacks, making them a better alternative than traditional online marketplace and escrow platforms.[1] For example, attacks such as DDOS are more difficult to execute against decentralized networks because nodes are spread all across the world, making DDOS attacks more expensive to implement due to this distributed system of blockchain.[4] Thus, since blockchains are distributed on multiple nodes, there is no central entity managing the chain, but rather a decentralized peer-to-peer network that prevents single points of failure, which third parties are often vulnerable to. A decentralized peer-to-peer network allows for each node to verify blocks or reject blocks that are tampered with and come to consensus with the rest of the network before the block is added throughout the network. This idea makes blockchain much more secure and reliable than third parties, since dishonesty or attacks would be less feasible as they would require tampering with all the blocks on the chain and getting the network's consensus, in turn eliminating single points of failure. Furthermore, data leaks are a growing concern, since third party firms often store their users' personal information, this makes them a common target for hackers and security breaches, greatly jeopardizing the privacy of their users. However, since cryptocurrency wallets have no links to personal information, blockchain

significantly reduces data privacy concerns since only the transactors' wallet address and transaction details are stored on the blockchain.

A decentralized escrow smart contract would greatly reduce the need for human interaction and minimize the risk of litigation, in turn reducing costs as well. Via smart contract code, users may have more trust and sense of reliability, as smart contracts are open source and transparent, unlike third party firms that subject their users to rely on often obscure and undisclosed methods. Thus, by deploying a smart contract through the Ethereum network (Goerli test Ethereum network) using blockchain technology, buyers and sellers may interact with the contract through actions such as making deposits to initiate a transaction, canceling transactions, confirming transactions, and sending receipts. Ultimately, we propose that smart contracts are a more viable option than third party arbiters for a trustless marketplace exchange, since our smart contract will securely hold funds until the item is received by the buyer and all conditions of the smart contract are met. Using an escrow mechanism that requires buyers and sellers to make double deposits, creates a trustless system since the extra deposit locked in the contract acts as a collateral, giving the transactors a strong incentive to complete the transaction.

3 Overview

According to the Ethereum website, Ethereum is the "foundation for building apps and organizations in a decentralized, permissionless, censorship-resistant way." The EVM (Ethereum Virtual Machine) is a computer that the network works with and anyone on the network may request computations from the computer. These computations are verified by nodes, or others on the network and committed to the updated "state" of the Ethereum network. These EVM state updates come with a cost of Ether, the native cryptocurrency of Ethereum, some of which is

rewarded to nodes who verify these updates to keep the network secure. All of this data is stored within blocks that are then stored on the blockchain where a user can track all of their previous transactions, this makes the blockchain "traceable" and prevents issues such as "double-spending" where the same currency is used twice .

Smart contracts are one of the main benefits of the Ethereum blockchain. They are Ethereum accounts just like Ethereum wallet addresses' that can send and receive transactions, but instead of being controlled by a user, "they are deployed to the network and run as programmed."[3] Simply put, smart contracts use a snippet of code to enforce the conditions that are previously defined on them. In the case of our project, smart contracts would run the transaction between the two parties who use our service. For example, if user A would like to sell an item to user B, they would note the transfer in the smart contract along with the transfer of Ether (the currency which our system will use). Upon completion of the condition in the smart contract, user A will receive the Ether and user B will receive the item that they desired from user A. This is a basic summary of the smart contract, but when implemented, there will be other aspects as well such as an Ether amount working as collateral, to ensure both parties stay honest and do not cheat each other of the item or Ether. If the transaction is aborted by the seller, the smart contract is not completed and the seller gets his ether back from the contract. Upon completion of a smart contract, the transaction will be stored on the Ethereum blockchain and can always be verified by the involved parties.

4 Literature Review

Several decentralized crypto-currency-based trading systems similar to our model have been deployed. These platforms have been developed for the aforementioned advantages of

removing the mediating third party from commodity exchanges, which eliminates any potential high transaction fees, restrictions, or censorship while also ensuring the anonymity and data security of the transactors.[5]

For example, in N. I. Schwartzbach's (2020) paper, it is indicated that smart contracts can work as the arbiter and only invoke if disputes arise.[8] Normally, the buyer will transfer the value of goods in cryptocurrency as escrow and notify the contract when receiving the goods. The money as escrow will then be transferred to the seller, which terminates the contract. However, if disputes occur during the transaction, one party will place money of arbitrary value as a wager. The other party will also place the same amount of money as a wager to counter this dispute. This paper develops an arbiter based on mathematical deduction. According to the paper, the arbiter will judge in favor of the honest party based on evidence. The winner of the dispute will gain back the cost of the goods plus the wager, while the loser gains nothing. However, errors in judgment will always happen because the judgment is based on probability, thus a smart contract working as an arbiter in this case isn't the best approach as the stakes for errors may be high. This differs from our smart contract protocol for a decentralized exchange because, in our protocol, both buyers and sellers will need to put twice the amount of the transaction as escrow, which incentivizes the completion of the transaction and reduces the chance of arbitration.

Furthermore, Zimbeck's (2014) paper established a trustless escrow system that relies on the concept of two-party double deposits and the world's first decentralized smart contract protocol, BitHalo.[8] BitHalo was developed to address the issues of early smart contract protocols which lacked a feasible mechanism for trustless transactions. Additionally**,** transactions on early smart contract protocols like the one in Schwartzbach's paper were often vulnerable to transaction malleability, where raw transaction data may be changed or exchanged before

broadcasting, ultimately invalidating the transaction. Such vulnerabilities allowed for extortion and attacks such as double-spend attacks.[8] As a result, BitHalo introduced a two-party double deposit protocol that not only addressed these vulnerabilities mentioned above but also introduced a mechanism for trustless online marketplaces that rely on code rather than a mediating entity.

BitHalo's smart contract protocol for a decentralized exchange is similar to ours, since twice the amount of the transaction is held as escrow to incentivize honest behavior and the completion of the terms of the smart contract. According to BitHalo's design principles, if any of the parties break the commitment, or not finishing the transaction in a timely manner, the escrow will time out and destroy itself, thus both parties lose money. This leads BitHalo to be the "mother of unbreakable contracts".

However, one fundamental difference is that our smart contract protocol will run on the Ethereum blockchain whereas BitHalo is software based and uses BitCoin as currency, however the contract is not on the BitCoin blockchain (i.e., record of contracts not stored on blockchain).[2] It is important to consider that BitHalo's off-blockchain decentralized smart contract protocol has the advantage of not bloating the blockchain and allowing for greater complexity contracts, whereas Ethereum smart contracts have a max size of 25 KB due to gas limitations.[3, 2] On the other hand, the benefits to our protocol via Ethereum smart contracts is that they are able to execute on blockchain and stored in a distributed manner along with cryptographic mechanisms for a greater degree of security in terms of being tamper-proof and immutable. Moreover, our protocol also has the benefit of running on the Ethereum blockchain, which is proof-of-stake, resulting in lower overhead and a significantly lower carbon footprint than other blockchains that often run on proof-of-work.

5 Infrastructure

5.1 High-Level Infrastructure

- *Remix-Ethereum*: Remix-Ethereum is where we write our smart contract code. From Remix-Ethereum, we are able to deploy our smart contract. Any changes in code must be updated here and redeployed.
- *MetaMask Wallet*: The buyer/seller must connect to their MetaMask wallet (or a similar crypto currency wallet), in order to utilize the smart contract, as the currency (Ether) is contained in the MetaMask wallet. The MetaMask wallet is also used for its unique ID (public key) to recognize an account (unique identifier for security purposes such as verifying sufficient funds).
- *Goerli Test Networ*k: This is the network where our smart contracts run until our protocol is ready for production. It is a test Ethereum network that replicates the Ethereum Network (EVM - Ethereum Virtual Machine) and uses the test currency we are using, Goerli Test Ether. We use a test network and test Ethereum due to the expense of using actual Ethereum for contract deployment and testing.
- *Etherscan.io*: Etherscan is where most of the process occurs. It is where we run the smart contract, verify it (ensuring the bytecode for contract execution is the same as when it was deployed), publish it, read from it, write to it, and monitor it. We use Remix-Ethereum and MetaMask to connect to Etherscan where the rest of the process is conducted.

5.2 Design Principles

In order to ensure that both parties are incentivized to complete the transaction, the buyer and the seller will both each deposit twice the amount of ETH into the contract as escrow. The ETH payment will remain locked inside the contract until the buyer confirms receipt of the promised item, or the seller aborts the sale. If the buyer confirms receipt of the item, the seller will be sent its deposit back plus the cost of the item and the buyer will be sent their deposit back minus the cost of the item. However, if the seller aborts the sale, the buyer and seller will each be sent their full ETH deposits, see Figure 1. We will determine the success of our protocol by analyzing the time elapsed between the time transactions were initiated until the time they were resolved along with analyzing the distribution of successful transactions and cancellations. Thus, shorter times elapsed for transactions and higher rates of successful resolutions will be our metric for evaluating success. Furthermore, once our protocol goes into production, we intend to record transaction details in order to conduct analysis that will optimize our protocol. A potential approach to optimizing our protocol will be finding an optimal value for the escrow deposit amount. For instance, by setting the deposit amount as a factor that is more than the item value but no more than twice the amount of the item, depending on the transaction details and patterns learned from our analysis, in order to achieve shorter contract completion times and more contract completions.

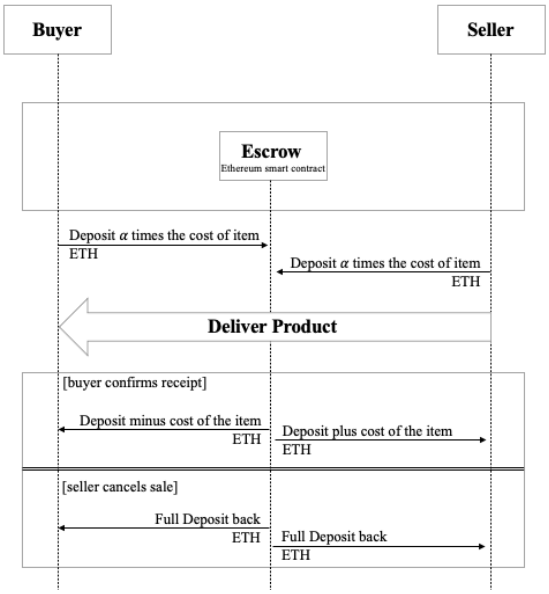

Figure 1. Diagram of Proposed Decentralized Escrow Smart Contract Protocol.

6 Methods

6.1 Setting the Contract in Motion:

Our online escrow marketplace will be run through a smart contract via the Remix Ethereum IDE, and in order to understand the process of executing our smart contract, it is crucial to distinguish between some terminology. Each party in our escrow system will have their own MetaMask wallet, which contains their Ether and their unique wallet address, which is necessary for identifying the selling party and the buying party. For the purposes of our sample, we have two metamask wallet accounts which each represent a party in our sample transaction. Henceforth, we will refer to the purchasing party's MetaMask account as the "buyer", or the "buyer's account" and the selling party's account as the "seller", or the "seller's account". At the moment, our smart contract exists on the Remix IDE and on the Goerli test Ethereum network (due to the high cost of deploying and testing with actual Ethereum, although once our protocol is finalized we will deploy on the actual Ethereum Network with a more optimal frontend that

will be established in the future). The transaction must be commenced from the point of view of the seller. However, before the seller deploys the contract, they must set values for the item_price_in_ETH, item_name, item_details, item_description which will be passed into respective read only functions that may be utilized by the buyer before confirming purchase. Next the seller can deploy the purchase.sol file (the file of our smart contract) via the Remix Ethereum IDE website, with the necessary Ether cost attached to it (twice the product amount), to cover two times the transaction cost for escrow (not including the gas fee). At this point, the Ethereum state is in stage zero, in which our code defines the smart contract as officially created. The smart contract is then documented on the etherscan website for the buyer and seller to follow and must be verified and published by the seller.

6.2 Transaction Process:

Now that the seller has put the smart contract officially in motion, two times the product price is held in the contract and a buyer must make the next step. However, if a suitable buyer is not found, the seller has the ability to abort the process and this will refund their Ether. When the buyer is ready, they then confirm their purchase through the etherscan website (again, a more proper interface will be set up in the future) and deposit two times the transaction amount into the smart contract. With both parties double depositing, they will both bear the same risk if the contract is not completed, creating an equally strong incentive for both parties to complete the transaction. At this point, the buyer's MetaMask account is connected to the smart contract and recognized as the purchaser, by confirming the purchase. The Ethereum state will then move to stage one, where it is defined as "locked", as an agreement is being initiated between the two acknowledged parties. This locked state indicates that the Ethereum will be locked into the

contract until the buyer confirms receiving the promised item agreed upon. Thus, upon receipt of the purchased good, the buyer will confirm that they have received the purchase through Etherscan, at which point the buyer will be refunded half of their input into the smart contract (1x the price of the product) at which point the smart contract enters the penultimate state which is “Release”. Finally, the seller will choose to refund their stored Ether, to receive their whole input, as well as the revenue generated from the transaction (total of 3x the price of the product to account for the seller’s profit and their original deposit). The state of the smart contract finally moves to stage 3 where it is considered complete and the contract is defined as “inactivate”, concluding the arrangement and closing the contract.

6.3 Implementation and Contract Execution Model

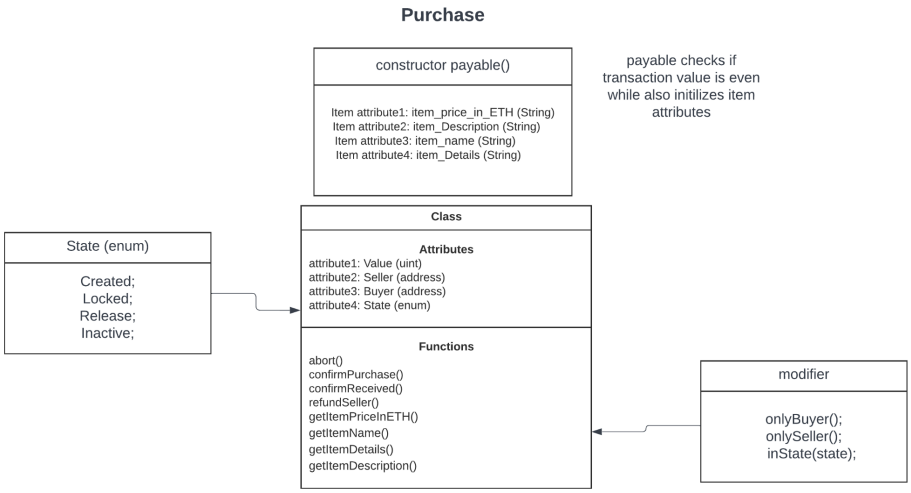

Figure 2. Class diagram of our implementation for a escrow smart contract (purchase.sol)

A single contract named Purchase contains one class including attributes such as value, seller address, buyer address, state, and various product information attributes (item_name,

item_price_in_ETH, item_details, item_description). The state is one of the most crucial attributes, which has four different values, and it determines the operation that the contract can conduct at a certain stage. Inside the class, we have a constructor payable that helps ensure the transaction value is even because it's double deposit escrow and the input value has to be twice the value of the goods. An odd double deposit value will result in a transaction cost that is a fraction which could pose problems as Solidity does not natively support fractions. It is important to note that the read only attributes also are initialized in the constructor prior to deployment. The class has four main functions (as well as four read-only functions for the product attributes), the successful operation of which will be based on modifier check and state check. Details regarding the 4 functions' backend flow are presented in the flow-diagram in figure 3.

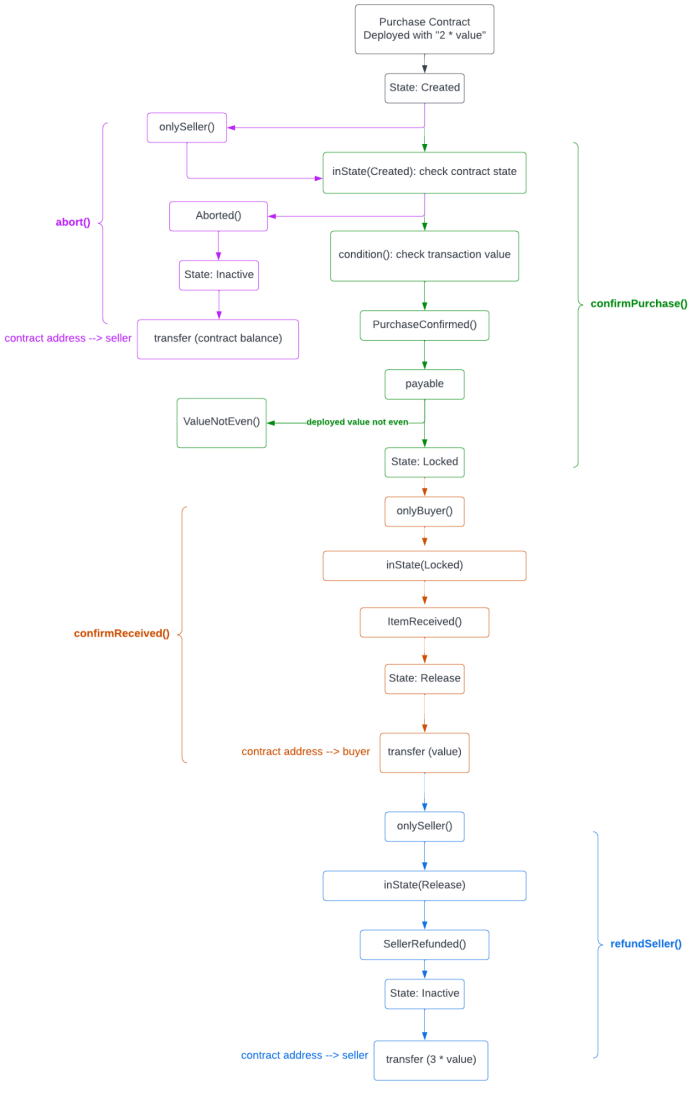

Figure 3. This diagram depicts the low-level mechanism behind each function in our smart contract. The high-level idea of each function is explained in section 6.1 and 6.2 above.

7 Video Demo of Code Execution

The video demo will walk through each step in the flow diagram above, demonstrating the process of contract deployment and execution.

Link to walkthrough video: https://www.youtube.com/watch?v=CihC_BC41w0

Link to Github: https://github.com/matin-g/DSC180a-Q1-final-code

8 Discussion

We successfully created an escrow-based marketplace that allows buyers and sellers to make transactions without the intervention of a third party and in a comparatively fair condition. However, there are some issues and limitations regarding the current contract, which we hope to address in the next phase of the project. Firstly, so far we haven't considered the situation that the contract ceases at a middle stage for a long period and stops moving forward. In other words, what if the transaction is not completed in a timely manner? Even though both seller and buyer are incentivized to finish the transaction and get their money back, it's possible that one party wants the transaction completed quickly more than the other group, thus would like to speed up the process. In that case, it is necessary to set an upper limit for the transaction period, otherwise, it may take longer than expected and cause anxiety to one of the parties, because the buyer/seller has no idea on how important the money is to the other group. Secondly, in our contract we did not include the solution to a dispute case, which is very likely to happen in real life. For instance, once the buyer confirms purchase and successfully receives the product, and they are not

satisfied with the quality of the product, and does not want to forward the contract to the next stage. In this scenario, the buyer and seller will most likely have a dispute and we need to find a way to fairly resolve the issue. Furthermore, if you look into our contract, you will realize that once a cycle of transactions completes, the contract will stay in the state = “inactive” forever, and cannot be reused later on. And if we hope to introduce more buyers to one seller, we need to consider how our state variable shall change after each buyer’s transaction. One simple solution would be initializing multiple state variables, and each buyer will correspond to his/her own state variable.

9 Future Work

In the next phase, we will extend our project so that it enables multiple buyers to purchase multiple products from one seller. Also, currently the transaction can only be done through the etherscan interface, which isn’t convenient for both sellers and buyers, and we aim to create a website that serves as the front-end of our smart contract. In the end, we hope the website can support multiple sellers and thus become a completely decentralized e-commerce platform that benefits both parties.

References

1. Ahsan, J. (2019, March 21). *Centralized vs. decentralized: The best (and worst) of both worlds*. LinkedIn. Retrieved October 30, 2022, from https://www.linkedin.com/pulse/centralized-vs-decentralized-best-worst-both-worlds-jiyad-ahsan

2. Bigi, G., Bracciali, A., Meacci, G., & Tuosto, E. (1970, January 1). *Validation of decentralised smart contracts through game theory and formal methods: Semantic scholar*. Retrieved October 30, 2022, from https://www.semanticscholar.org/paper/Validation-of-Decentralised-Smart-Contracts-Through-Bigi-Bracciali/2722e4c3bbabaaebd282b298224bf24d56a6d67f
3. *Introduction to smart contracts*. ethereum.org. (2022, September 1). Retrieved October 30, 2022, from https://ethereum.org/en/developers/docs/smart-contracts/
4. Moreland, K. (2022, October 25). *What are ddos attacks?* Ledger. Retrieved October 30, 2022, from https://www.ledger.com/academy/what-is-a-ddos-attack.
5. Prasad, R. V., Dantu, R., Paul, A., Mears, P., & Morozov, K. (2018). *A decentralized marketplace application on the ethereum blockchain*. IEEE Xplore. Retrieved October 30, 2022, from https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8537821
6. *Selling fees*. eBay. (n.d.). Retrieved December 1, 2022, from https://www.ebay.com/help/selling/fees-credits-invoices/frais-pour-les-vendeurs-particuliers?id=4822
7. Schwartzbach, N. I. (2020, August 24). *An incentive-compatible smart contract for Decentralized Commerce*. arXiv.org. Retrieved December 2, 2022, from https://arxiv.org/abs/2008.10326
8. Zimbeck, D. (2014). *Two party double deposit trustless escrow in cryptographic networks and ...* Retrieved October 31, 2022, from https://cryptochainuni.com/wp-content/uploads/BitHalo-whitepaper-two-party-double-deposit-trustless-escrow-in-cryptographic-networks-bitcoin.pdf

Contributions

Matin Ghaffari:

I helped by editing and adding to the introduction, abstract, literature review, and overview sections that were edited from the quarter 1 project checkpoint (using TA feedback). Furthermore, I implemented much of the code and performed much of the code debugging and testing. Also I created the code walkthrough/demo video. I also helped write about the high level infrastructure section and wrote the design principles section. I also contributed to the methods section and the implementation section.

Wenyuan Chen:

I helped by editing and adding to introduction, abstract, overview, literature review based on the feedback provided by TA from quarter 1 project checkpoint. I also helped with the creating class diagram and flow-diagram by going over the inner logics. I also helped with the implementation of the contract by providing ideas on how to add read functionality for product descriptions. I also deployed the contract and tested if it operated correctly.

Yu Huang:

I helped with modifying the literature review part from quarter 1 project checkpoint, looked into the paper about BitHalo and compared its difference to the platform that we aim to create. Also helped with the code debug, as we encountered issues with verifying and publishing our smart contract after the deployment on etherscan. Moreover, I came up with the class diagram and flow chart of the contract, which aims to provide a clear visualization in the report. Besides, I summarized the thoughts that our group has regarding the current state of the project and put them into the discussion section. Same for the future work section, where I briefly wrote about how we aim to build upon the current smart contract and improve in the next quarter.

Alan Amirian:

I helped with the introduction, abstract, infrastructure, and methods sections as well as testing and debugging the code. I collaborated with the group on much of the intro and abstract. In particular, I wrote the high level architecture (Infrastructure), Setting the Contract in Motion (Methods) and the Transaction Process (Methods). I also tested the solidity code on Remix Ethereum to ensure that it worked without flaws and to better document the process for the methods section.