A Lightweight Onchain CAPTCHA for the Detection of Non-Human Accounts on Ethereum (*A Position Paper*)

Roshan Singh

`https://kalikho.github.io`
`roshancsofficial@gmail.com`

Abstract. The open and permissionless nature of blockchain allows anyone and everyone to create unlimited accounts and become a part of the ecosystem. Blockchains by default do not provide any identification and classification mechanism that can distinguish between a human vs. a non-human aka bot account. This attribute makes the Ethereum network a breeding space for activity by software robots (bots). Recently, bots have gained widespread attention for all the bad reasons such as disrupting proper airdrops, sandwich attacks, and MEV attacks. Therefore, effective identification of non-human accounts is the need of the hour for minimizing the financial losses caused by the bots. This work is one such step in that direction. The paper proposes a Lightweight on-chain approach for detecting Non-Human Accounts on the Ethereum blockchain.

Keywords: Bot Detection · Unicode Based CAPTCHA · Airdrop.

1 Introduction

Addresses are the first-class citizens on the Ethereum blockchain. Addresses on the Ethereum blockchain mainly consist of two types. The first one is the contract accounts generated as a result of a contract deployment. The other one is the EOA or Externally Owned Account which the external users generate. EOA can be either owned by a human or a non-human. In this work, we refer to non-human accounts as a bot account. Bots are automated software that monitors the blockchain and executes and arranges transactions to maximize the incentives earned. With no restriction on the number of accounts to be created, many bots-controlled accounts have popped up on the Ethereum blockchain. These bot accounts outsmart the benign human-controlled accounts in executing transactions causing losses to the human users. This has a direct implication for the user community of the blockchain. Handling bot-controlled accounts is a challenge.

2 Background

This section provides a brief background of the technologies used in the work.

- **CAPTCHA**: Completely Automated Public Turing test to tell Computers and Humans Apart is a test to determine if the user interacting with the system is a human or not. Initially, CAPTCHAs were text-based, asking the user to identify a distorted and incorrect text sequence. The user was regarded as human if they could accurately identify the word; otherwise, they were not. Google created reCAPTCHA in 2009; these captchas were made up of jumbled words and sentences that optical character recognition algorithms were unable to decipher. Google created "No Captcha reCaptcha," a new captcha system that just needs a click or touch from the user to verify they are human, in 2014. This system determines whether a user is genuinely human or not by utilizing cutting-edge technology like machine learning and user behavior analysis.
- **Smart Contracts**: A smart contract is a self-executable block of code that is deployed on the blockchain. The code specifies the rules that govern the interaction between the participating parties of the blockchain. Smart contracts ensure bias-free implementation of the rules. This work uses Solidity as the language of choice for implementing the smart contract.

3 Proposed Approach

This section provides the proposed approach in detail.

- **Challenger**: A human or an entity such as a crypto project releasing an airdrop.
- **Prover**: A human or a non-human entity with an EOA. The Prover interacts with the smart contract to prove itself as a human.
- **Unicode Based CAPTCHA**: Unicode is the world standard for text and Emojis[4]. Solidity version 0.7.0 and higher introduced support for Unicode characters in the smart contract. The character can be accessed with the help of a Getter function where the Unicode character can be returned prefixed with the *unicode* keyword. The Unicode character is written as a byte(UTF-8). Unicode 15.1 specifies a total of 3,782 emoji[4]. This work uses Unicode characters to build CAPTCHAs on the smart contract. With the available number of CATCHAs in hand a large set of CAPTCHAs can be generated. These CAPTCHAs are not only lightweight but are also not easily interpretable and solvable by machines and bots that lack human intelligence. Using the Unicode characters several logic-based CAPTCHAs can be formed. Logic-based CAPTCHAs make machines difficult to interpret the problem and guess the correct answer, even if they decode the meaning of the unicode characters. At the same time, such CAPTCHAs are easy to understand by humans. One such CAPTCHA is defined below:- Consider the image in Fig.1.

  Question 1:- Interpreting the sequence of images from Left to Right, find out the image with a single eye.

Here the correct answer is option B. The rest of the options have two eyes.

The Fig. 1 is composed of 4 Unicode characters

1. The first image is the full moon face (Unicode: U+1F31D).
2. The second image is a first-quarter moon with a face(Unicode: U+1F31B).
3. The third image is a snowman(Unicode: U+2603).
4. The fourth image is a prince(Unicode: U+1F934).

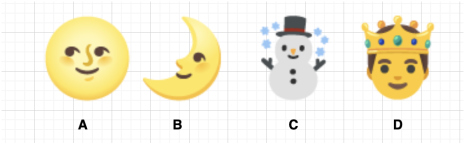


Fig. 1. Visual cues based CAPTCHA

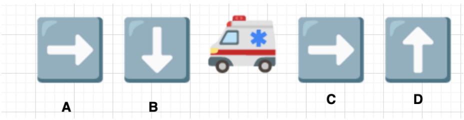


Fig. 2. Orientation based CAPTCHA

Question 2:- Consider the images in Fig. 2 and find the arrow whose nose is pointing toward the front of the ambulance.
Here the correct answer is option A.

- **Commit Reveal Scheme**: The proposed approach utilizes a commit reveal scheme where the response of the user against the CAPTCHA is recorded on the blockchain in a hashed form. This thwarts an attempt from malicious entities monitoring the blockchain transactions to look into the results submitted by a user against the CAPTCHA, as the blockchain transactions are released as plaintext. Provers are required to submit their hashed response against the CAPTCHA within a stipulated time frame. The prover generates the hash locally and off the chain with the random secret parameter.
  Once the deadline for submission of hashed responses by the provers against the challenge has expired. The Provers are required to disclose their submitted responses in plaintext. At the same time, the Challenger also submits the result in plaintext. The smart contract then matches the response submitted by the user with the response submitted by the Challenger if the response

matches; the Prover is considered as a human otherwise a bot. To increase the effectiveness of the approach we suggest conducting the CAPTCHA test over several interactions, such as the Prover needs to solve *n/2 + 1* number of CAPTCHAs correctly out of n, to prove human.

Use Case: We consider a use case of a modified Airdrop. In our modified Airdrop a user the Prover needs to prove itself a human by solving our Unicode CAPTCHA. Once the Prover can solve a sufficient number of CAPTCHA challenges, the user is allowed to transfer the defined amount of coins to itself via the smart contract. Otherwise, the Prover is considered to be a bot and is discarded from obtaining the coin from the Airdrop. The Challenger creates Unicode CAPTCHAs within the smart contract and deploys the smart contract on the Ethereum blockchain. The address of the contract is made publicly available. The rest of the steps are executed in two phases:-

Phase I, commit phase - The sequence of steps followed in Phase I is shown in Fig. 3, the terminologies used in the flowchart is defined below:-

- C_{BN} : Current block number.
- T_{BN} : Threshold block number
- A_c : Number of Attempted CAPTCHA by the Prover.
- T_c : Number of Total CAPTCHA available for challange.
- Opt_{No} : Option Number against a CAPTCHA. To be submitted by the Prover.
- *Res* : Hashed response submitted by the Prover against a CAPTCHA challenge to the smart contract.
- H : Hash function.
- *Answer*: The correct option against a challenge, disclosed by the Challenger after Phase I.

Phase II, reveal phase:- In the reveal phase, the Challenger discloses the correct options for the CAPTCHA-based challenge. The Prover also discloses the option number and the random number via the smart contract. The smart contract calculates the hash value of the response and the random number submitted by the Prover. It then matches the calculated hash value with the hash value submitted by the Prover. If the hash matches then the option submitted by the Prover is matched with the correct response submitted by the Challenger against the CAPTCHA challenge if the submitted response matches; then the Prover is considered to have passed the CAPTCHA test. The solidity code for a sample Unicode CAPTCHA can be found here [1]. The sequence of steps to be followed in Phase II is as shown in Fig. 5.

4 Related Works

There exist some works in the direction of identifying bots on the blockchain[2] [3]. However, these works are statistical and analytical, which needs monitoring

---

[1] https://github.com/kalikho/unicodeCAPTCHA

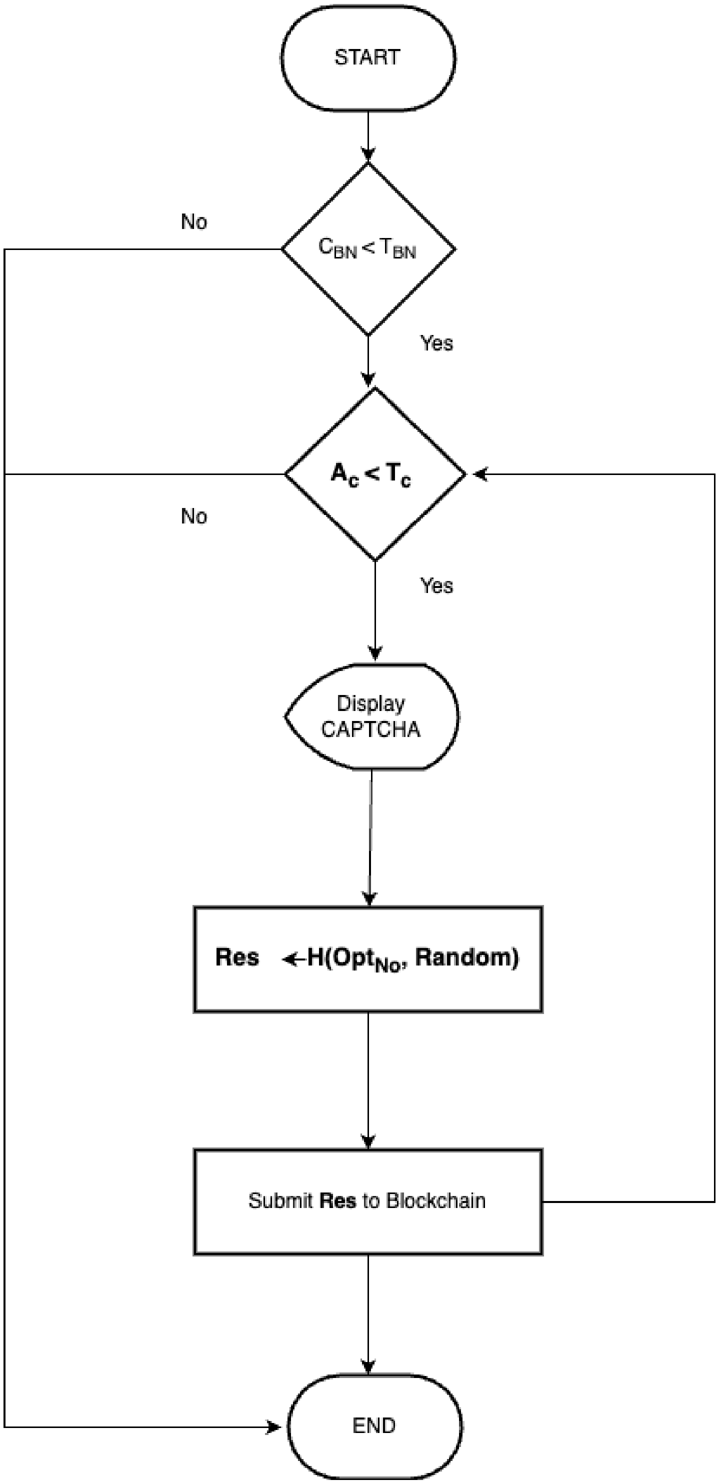


Fig. 3. Flowchart for commiting Prover response (Phase I)

and analysis of blockchain transactions over time. Such approaches can't provide an immediate classification of human vs. non-human accounts for accounts with no prior activity on the chain. However little effort has been made on interactive and reactive approaches for the identification of bots on the blockchain. Zkaptcha[1] attempts to address the challenge. However, it uses complex zero-knowledge proof and requires the images to be stored in external storage.

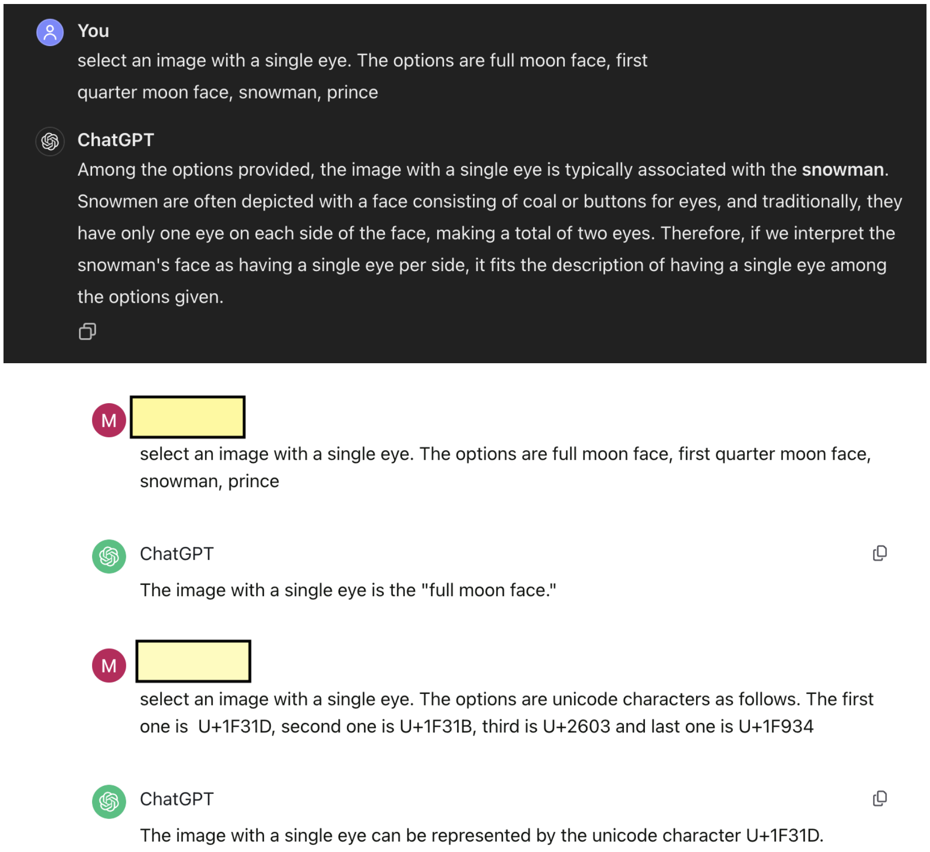
5 Discussion

This position paper aims to shed some light on the effectiveness and feasibility of Unicode-based CAPTCHA with fully on-chain operation and interaction. The CAPTCHA is made publically available through the smart contract. The proposed approach is extremely simple, and lightweight, and does not rely on any complex zero-knowledge setup and operation. The commit and reveal scheme provides secure and reliable interaction with our CAPTCHA system.

The Unicode characters and their definition are publicly available, allowing anyone to decode the meaning of a Unicode character[5][6]. With the advancement in language-based Artificial Intelligence such as the Large Language Model (LLM), it becomes crucial to discuss the resilience of Unicode-based CAPTCHA against such threats. We attempted to test our Unicode-based CAPTCHA against Chat-GPT[7] (version 3.5) a popular LLM. The Unicode CAPTCHA as in Question 1 was decoded and Chat-GPT was queried to find the image with a single eye. It was observed, that with the only definition available against the Unicode character, it is not sufficient to solve the carefully designed Unicode CAPTCHA.

References

1. ZKaptcha: https://ethglobal.com/showcase/zkaptcha-4q4qz
2. Detecting Bot Activity in the Ethereum Blockchain Network : Morit Zwang, Shahar Somin, Alex 'Sandy' Pentland, and Yaniv Altshuler
3. Detecting Financial Bots on the Ethereum Blockchain: Thomas Niedermayer et. al.
4. Unicode : https://www.unicode.org/
5. Emoji Charts : https://www.unicode.org/emoji/charts/full-emoji-list.html
6. https://apps.timwhitlock.info/emoji/tables/unicode
7. Chat GPT : https://chatgpt.com/



You

select an image with a single eye. The options are full moon face, first quarter moon face, snowman, prince

ChatGPT

Among the options provided, the image with a single eye is typically associated with the **snowman**. Snowmen are often depicted with a face consisting of coal or buttons for eyes, and traditionally, they have only one eye on each side of the face, making a total of two eyes. Therefore, if we interpret the snowman's face as having a single eye per side, it fits the description of having a single eye among the options given.

M [redacted]

select an image with a single eye. The options are full moon face, first quarter moon face, snowman, prince

ChatGPT

The image with a single eye is the "full moon face."

M [redacted]

select an image with a single eye. The options are unicode characters as follows. The first one is U+1F31D, second one is U+1F31B, third is U+2603 and last one is U+1F934

ChatGPT

The image with a single eye can be represented by the unicode character U+1F31D.

Fig. 4. Response of Chat-GPT 3.5 against the Unicode CAPTCHA in Question 1

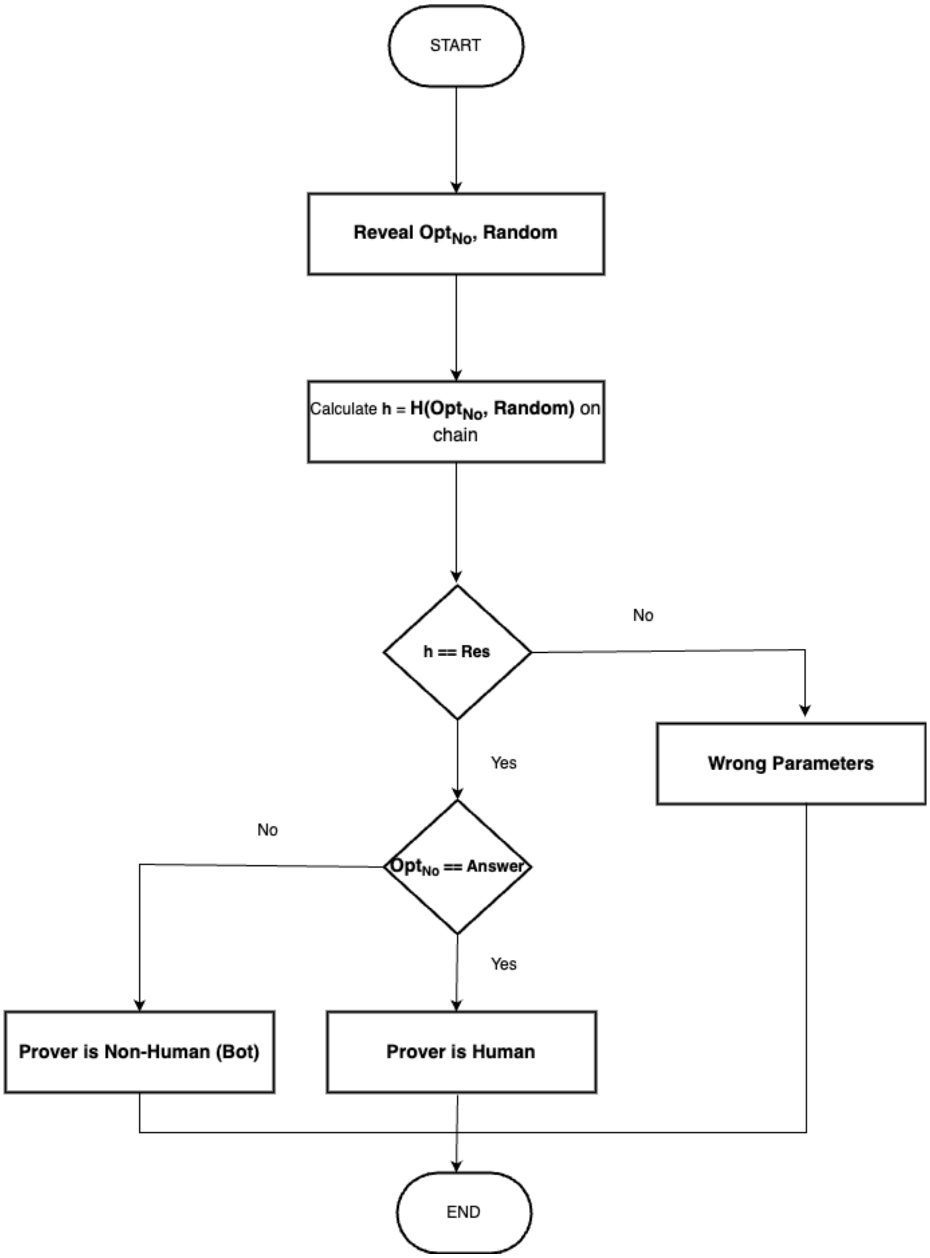

START
Reveal Opt_{No}, Random
Calculate h = H(Opt_{No}, Random) on chain
h == Res
No
Wrong Parameters
Yes
Opt_{No} == Answer
No
Yes
Prover is Non-Human (Bot)
Prover is Human
END

Fig. 5. Flowchart for Phase II