### CrowdSourceChain:Smart-contract implementation using Script-Hash address



*By*

BONAGIRI THIRU SATYA SURYA MAHAVEER

*Under the Supervision*

*Of*

**Prof. JAYDEEP HOWLADER**

*DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING*

*National Institute of Technology Durgapur West Bengal-713209*

*India*

A Final Year Project submitted in partial fulfilment for the Degree of Bachelors of Technology in Information Technology and Conputer Science Department of Computer Science and Engineering

National Institute of Technology, Durgapur.

**ACKNOWLEDGEMENT**

We wish to express my heartiest gratitude to my project supervisor **Prof. Jaydeep Howlader**. Lecturer of the Department of Computer Science, NIT Durgapur for his valuable advice and encouraging support. He not only supervised me to materialize the work but also showed me the way to think in different dimensions and work steadily from the very beginning of the project. His continuous support helped me keep my confidence level high through the project.

**Declaration**

I hereby declare that this submission is my own work and that to the best of my knowledge and belief, it contains no material previously published or written by another person nor material has been accepted for the award of any other degree or diplomas of the university or the other institutes of higher learning, except which due acknowledgement has been made on this text.

CERTIFICATE

It is certified that the work contained in the project report entitled **“CrowdSourceChain**:Smart-contract implementation using Script-Hash address**”** has been carried out by

**BONAGIRI THIRU SATYA SURYA MAHAVEER (15/CS/25).**

under the guidance of **Prof. JAYDEEP HOWLADER**, the data reported herein is original and has not been submitted to any other University or Institute for the award of any degree or diploma.

This is to certify the above declaration is true.

**Prof. Jaydeep Howlader**

Asst. Professor, Dept. of CSE National Institute of Technology Durgapur-713209, India

Place:

Date:

# Chapter 1

* 1. Introduction

Over the past few years, crowdsourcing has gained considerable interest and adoption since it is coined in 2006 by Jeff Howe. It is a distributed

problem-solving model through an open call for solutions. Nowadays, many large companies choose crowdsourcing as a problem-solving method, ranging from web and mobile development to t-shirt designs. There are numerous famous crowdsourcing applications such as Upwork, Amazon Mechanical Turk and UBER.

However, despite the prosperity of the crowdsourcing systems, they are subject to the weaknesses of traditional trust-based model, which brings about some inevitable challenges. The majority of existing crowdsourcing systems rely on central servers, which are subject to the weaknesses of traditional trust-based model, such as single point of failure. They are also vulnerable to distributed denial of service (DDoS) and Sybil attacks due to malicious users involvement. In addition, high service fees from the crowdsourcing platform may hinder the development of crowdsourcing. How to address these potential issues has both research and substantial value.

*Can we design a decentralized crowdsourcing system with reliability, fairness, security and low services fee?*

To answer this question, we propose a blockchain-based decentralized framework for crowdsourcing. The framework has many advantages such as increasing user security and service availability, enhancing the flexibility of crowdsourcing with Turing-complete programming language and lowering cost. Therefore, our framework has the potential to disrupt the traditional model in crowdsourcing.

# Demerits of traditional Croudsourcing

* + - **Vulnerable To Attacks :** Traditional crowdsourcing systems are vulnerable to DDoS attacks, remote hijacking and mischief attacks, which makes the services unavailable.

Elance and oDesk, operated by Upwork presently, downed services for many workers due to be hit by DDoS attacks in May 2014.

* + - **Single Point Failure :** The majority of crowdsourcing systems run business on a centralized server, which suffers from single point of failure inherently.

In April 2015, a service outage emerged due to hardware failure in Uber China, which caused passengers can’t stop the order at the end of services.

* + - **Privacy :** User’s sensitive information (e.g. name, email address and phone number) and task solutions are saved in the database of crowdsourcing systems, which has the risk of privacy disclosure and data loss.

For example, one of the most prevalent crowdsourcing systems Freelancer was reported to breach the Privacy Act for uncovering a user’s true identity which contains IP addresses, active account and dummy accounts by Office of the Australian Information Commissioner (OAIC) in December 2015.

* + - **Service Charge :** Crowdsourcing companies are interested in maximizing their own benefits and require requesters paying for services, which in turn increase user’s costs. Currently, most of the crowdsourcing systems could demand a sliding services fee for 5% to 20%.

# System model of traditional

Crowdsourcing

The human intelligence-based crowdsourcing consists of three groups of roles: requesters, workers and a centralized crowdsourcing system (Fig. 1.1).

Requesters submit tasks which are challenging for computers but easy for human to complete through the crowdsourcing system. A set of workers who are interested in this task compete and submit solutions to the crowdsourcing system, while requesters will then select a proper solution (usually the first or the best one that solves the task) and grant the corresponding workers the reward.

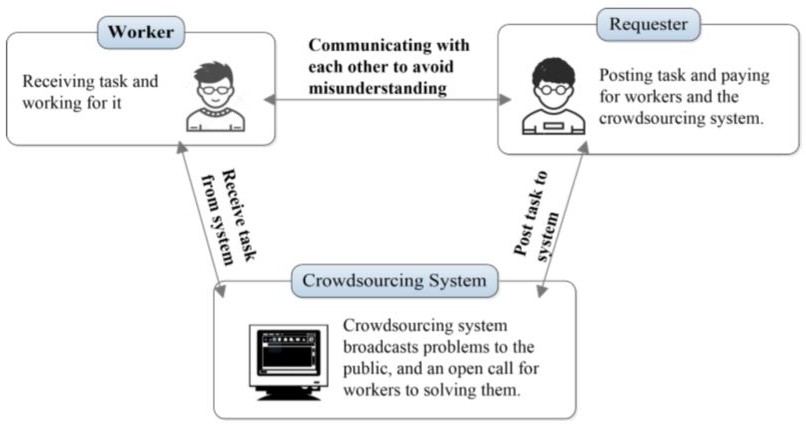


Fig. 1.1 : System model of traditional crowdsourcing

# Features of Blockchain-based

Decentralized Framework for Crowdsourcing

* + - The transactions stored in the blocks are contained in millions of computers participating in the chain. Hence it is decentralized. There is no single point failure.
    - The blocks are immutable. Transactions are organised as a Merkle tree.The Merkle Root is used to construct a block hash. If you change a transaction you need to change the subsequent block hash.
    - Users’ privacy can be guaranteed and only low transaction fees are required.

# Chapter 2

* 1. Blockchain

A blockchain is a decentralized, distributed and public digital ledger that is used to record transactions across many computers so that any involved record cannot be altered retroactively, without the alteration of all subsequent blocks.This allows the participants to verify and audit transactions independently and relatively inexpensively. A blockchain database is managed autonomously using a [peer-to-peer](https://en.wikipedia.org/wiki/Peer-to-peer) network and a distributed timestamping server. They are [authenticated](https://en.wikipedia.org/wiki/Authentication) by [mass collaboration](https://en.wikipedia.org/wiki/Mass_collaboration) powered by [collective](https://en.wikipedia.org/wiki/Collective)

[self-interests](https://en.wikipedia.org/wiki/Self-interest).

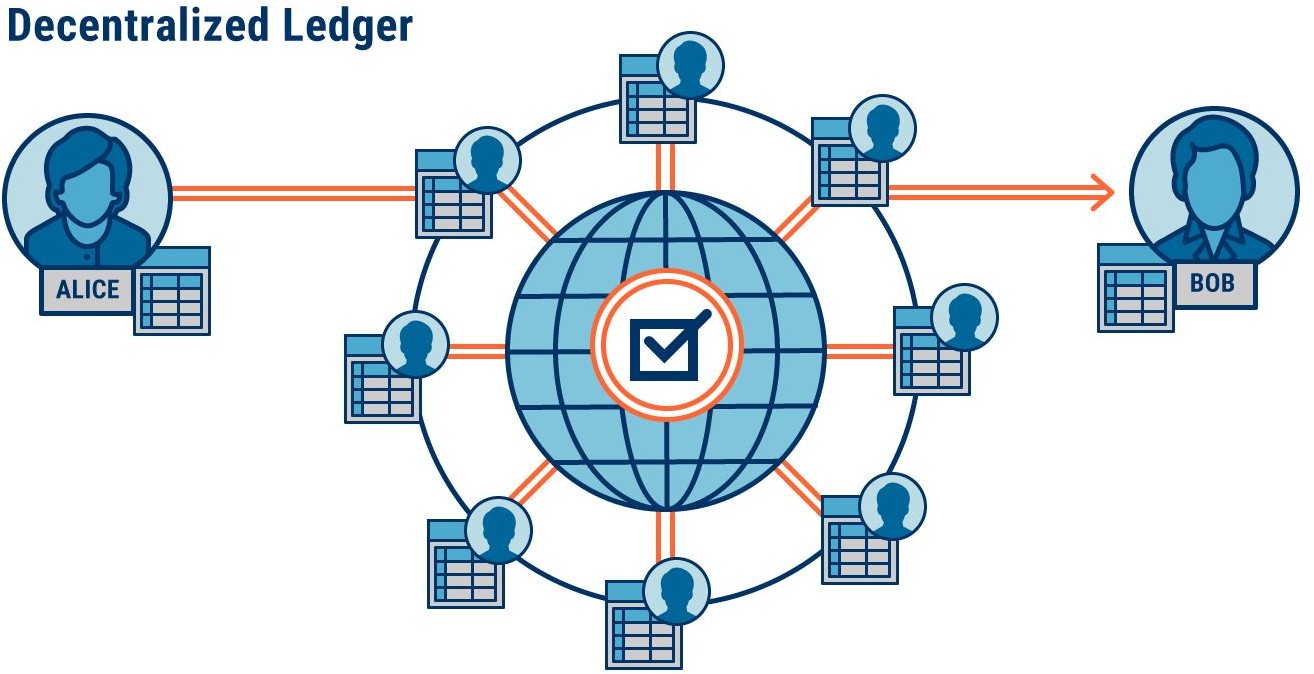


Fig. 2.1 : Decentralized Ledger

# Merkle Tree

Merkle trees, named after their creator Ralph Merkle, are binary hash trees used for efficient verification of data integrity. An example of a Merkle tree can be seen in Fig. 2.2. Leaves are computed directly as hashes over data blocks, whereas nodes further up the tree are computed by concatenating and hashing their respective children.

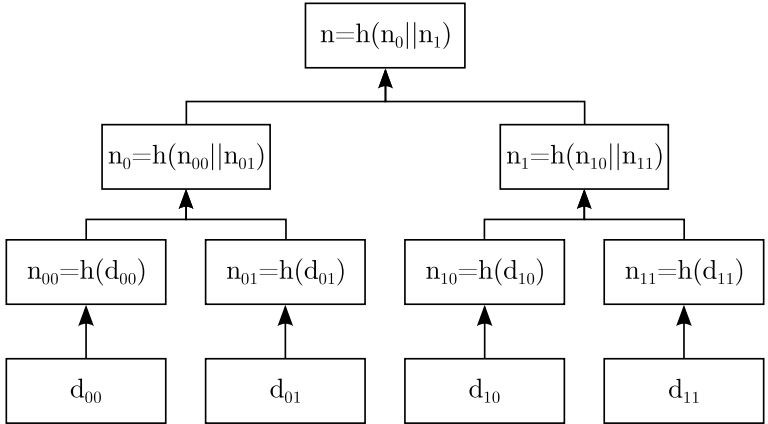


Fig. 2.2 : Merkle Tree

The main advantage of Merkle trees is that when one data block changes it is not necessary to compute a hash over all the data, as opposed to naive hashing. Assume data block d00 is modified, then n00 has to be re-computed as well as all nodes along the branch until the root node. Therefore, the number of required hash computations scales logarithmically in the number of data blocks. Since both data blocks and hashes are relatively small in size, this process is fairly efficient.

# Blocks

Each block is composed of a header and a payload. The header stores the current block header version (nVersion), a reference to the previous block (HashPrevBlock), the root node of the Merkle tree (HashMerkleRoot), a timestamp (nTime), a target value (nBits) and a nonce (nNonce). Finally, the payload stores the number of transactions (#vtx ) and the vector of transactions (vtx ) included in the block.

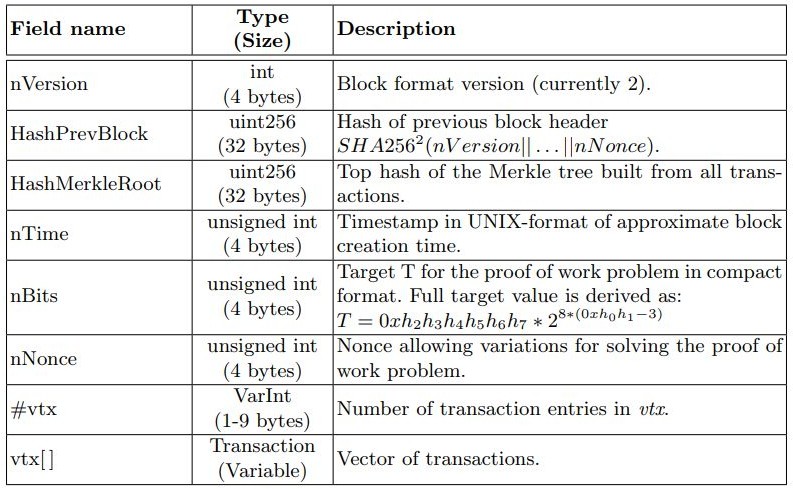


Fig. 2.3 : Block Structure

**HashPrevBlock** field stores the reference to the previous block, computed as hash over the block header as depicted in Fig. 2.4.

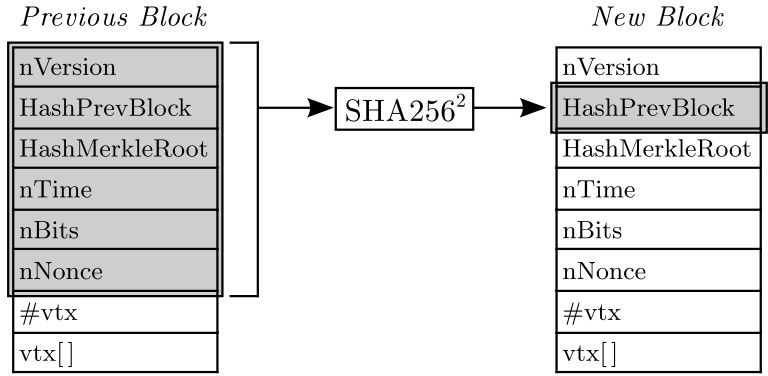


Fig. 2.4 : Block Reference Computation

A double-SHA256 hash is calculated over the concatenation of all elements in the previous block header:

**SHA2562 (nVersion||HashP revBlock||HashMerkleRoot||nTime||nBits||nNonce)**

The reference functions as a chaining link in the blockchain. By including a reference to the previous block, a chronological order on blocks, and thus transactions as well, is imposed.

# Transactions

Each block in the blockchain includes a set of transactions. Every transaction consists of a transaction version (nVersion), a vector of inputs (vin) and a vector of outputs (vout), both preceded by their count, and a transaction inclusion date (nLockTime).

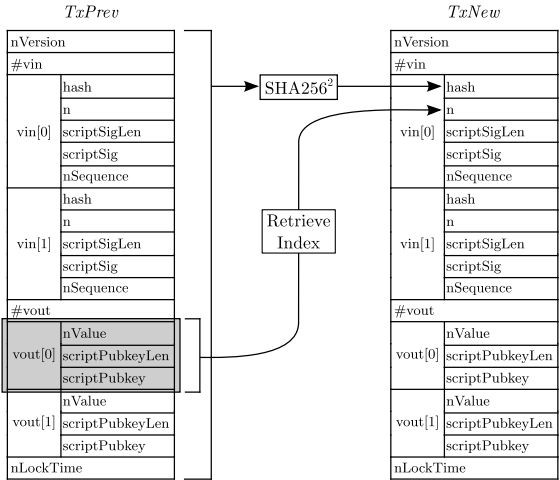


Fig. 2.4 : Transaction Output Reference Computation

## vin

The vin field stores a vector of one or more transaction inputs. Each transaction input is composed of a reference to a previous output (hash,n), the length of the signature script field in bytes (scriptSigLen), the signature script field (scriptSig) itself and a transaction sequence number (nSequence).

- (hash,n)

A previous output is uniquely identified by the tuple (hash,n). The hash field, also referred to as the transaction ID (TxID), is computed as a double-SHA256 hash of the raw transaction:

## T xID = SHA256^2 (Transaction)

Hence, whilst a transaction is uniquely identified by its hash, the specific output within that transaction is identified by the output index n. An example is given below in Fig. 2.5.

## - scriptSig

The signature script field contains a response script corresponding to the challenge script (see scriptPubkey field) of the referenced transaction output (hash,n). More precisely, whilst the challenge script specifies conditions under which the transaction output can be claimed, the response script is used to prove that the transaction is allowed to claim it.

# Script

Script is a stack-based, Turing-incomplete language designed specifically for the Bitcoin protocol. A script is essentially a set of instructions that are processed left to right. Script is used to encode two components - a challenge script and a response script:

* + - A challenge script (see scriptPubkey field) is part of a transaction output and specifies under which conditions it can be claimed.
    - A response script (see scriptSig field) is part of a transaction input and is used to prove that the referenced transaction output can be rightfully claimed.

For a given transaction, each transaction input is verified by first evaluating scriptSig, then copying the resulting stack and finally evaluating scriptPubkey of the referenced transaction output. If during the evaluation no failure is triggered and the final top stack element yields true, then the ownership has been successfully verified.

# 2.5 Signature

Signatures are a central cryptographic primitive in Bitcoin and play a significant role in transaction authorization. In a regular transaction, a signature is included in the signature script field (scriptSig) of every transaction input to prove that the referenced transaction output can be rightfully spent by the claimant.

## SIGHASH ALL

The default signature hash type SIGHASH ALL represents the simplest of the three base types. It signs the complete transaction, including all the transactions inputs and outputs, with the exception of the signature script fields. The coverage of the signature is illustrated below in Fig. 2.5 with grey fields.

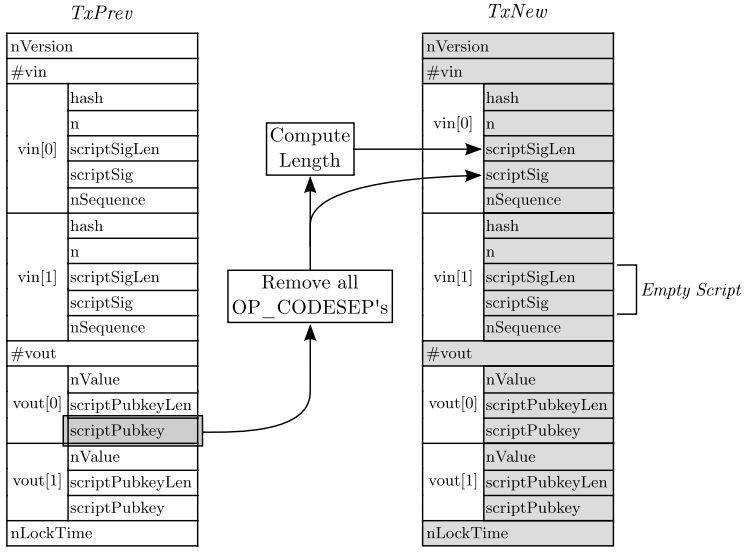


Fig. 2.5 : Signature Computation - SIGHASH ALL

## SIGHASH ANYONECANPAY

The SIGHASH ANYONECANPAY modifier is used in conjunction with a base type and affects the signature coverage of transaction inputs. It is used to only cover the currently signed input by the signature. For example, the transaction depicted in Fig. 2.6 illustrates that the second transaction input is excluded from the signature.

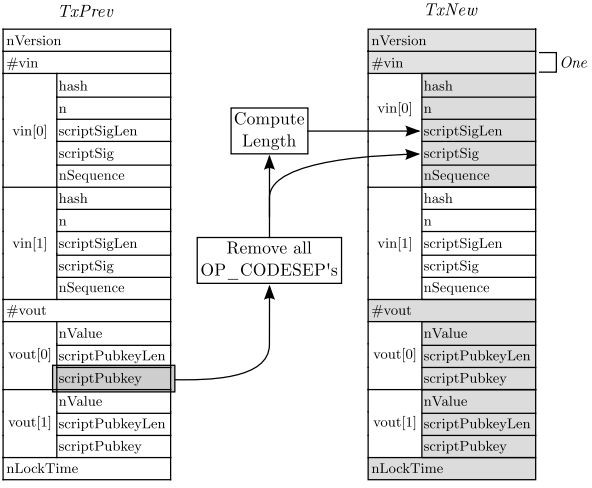


Fig. 2.6 : Signature Computation - SIGHASH ALL|SIGHASH ANYONECANPAY