

**ISTANBUL TECHNICAL UNIVERSITY
FACULTY OF COMPUTER AND INFORMATICS
ENGINEERING**

Blockchain-based e-Tendering System

Graduation Project Final Report

**Samed Kahyaoğlu
150120047**

**Department: Computer Engineering
Division: Computer Engineering**

Advisor: Asst. Prof. Dr. Mehmet Tahir Sandıkkaya

August 2021

Statement of Authenticity

I hereby declare that in this study

1. all the content influenced from external references are cited clearly and in detail,
2. and all the remaining sections, especially the theoretical studies and implemented software/hardware that constitute the fundamental essence of this study is originated by my/our individual authenticity.

İstanbul, August 2021

Samed Kahyaoğlu

Acknowledgments

I would like to thank my teacher Mehmet Tahir Sandıkkaya, who helped me a lot in this study, and my family who supported me throughout my education.

Blockchain-based e-Tendering System

(SUMMARY)

States generally purchase public services and products through tenders. Tenders prioritize efficiency in public expenditures and providing equal economic opportunity. They were designed to spend the taxpayers' money properly and efficiently, but in many countries they have become a tool for corruption. Blockchain can reduce corruption problems in auctions with openness and inclusiveness. This project aims to implement the tender procedure as a smart contract.

Blockchain Based e-Tender System aims to perform the basic functions of a tender on a smart contract. The the of this contract changes over time. The definition of tender is derived from the Turkish Public Procurement Law's definition of open tender procedure. According to this definition, the tender commission determines an estimated value for a tender, and a minimum and maximum amount is determined based on the estimated value. If the tender is not concluded between the minimum and the maximum, the tender commission reviews the situation. In the ideal scenario, the best (lowest) bid is within these limits. The project eliminates human intervention in bidding and determining the winner in the ideal scenario. In the open tender procedure, the minimum and maximum values are kept confidential until the end of the tender, and the bids of the participants are also kept confidential. But once the tender is complete, this data becomes publicly accessible for transparency and trust.

The project uses Ethereum and the programming language Solidity to develop smart contracts. Ethereum is an open blockchain, but people can also run their own private Ethereum blockchains. In this project, the public Ethereum network or a similar Ethereum Virtual Machine compatible public blockchain is considered an ecosystem. While major security concerns are delegated to a public network that is theoretically secure, trust and transparency is protected. People can use their own tools or trusted web3 applications that the community has developed to interact with smart contracts.

A fundamental problem with using an Ethereum network is that all data on the network is accessible to everyone. In order to ensure inter-bid confidentiality, the life cycle of the tender contract is determined in three separate phases: bidding, evaluation and post-tender. When creating the bid, the bid organizer uses a hash of data containing the minimum, maximum and estimated values and the stage lengths of the auction. In the bid collection stage, users transmit the bid and the hash value of a random character string to the smart contract, instead of their actual bid. During the evaluation phase, users verify their bids corresponding to this hash value and the most appropriate bid is calculated. In the post-tendering phase, the bid organizer reveals the hidden limitations of the tender, and if the owner of the most suitable bid has submitted a bid that meets the limits, the winner is determined. The data of the smart contract and the contract never change after the tender is finished by the organizer.

Within the scope of the project, a simple Python program was written as a precaution against

the problems that users may experience in creating hash values. Thanks to this program, it is possible to create mixed value as a tender organizer or bidder. The program will also help programmers as a draft that can be developed to be more secure. Since the hash values created with this Python code are created by the same method as the standard Keccak hash function calculated by Solidity, there is no problem in verifying them afterwards. With this helper it is possible for anyone to start using the tendering smart contracts immediately.

The related smart contracts were developed using Ethereum Remix, uploaded and transferred to Ethereum networks such as Ethereum Ropsten, BSC Testnet via Remix and wallets created with Metamask. While testing, various usage scenarios covering all functions and faults were used. The common writing style of the Ethereum developer community is used and all source code is available on GitHub¹.

¹<https://github.com/urtuba/open-tendering-in-ethereum>

Blokzincir Tabanlı e-İhale Sistemi

(ÖZET)

Devletler kamu hizmet ve ürün alımlarını genellikle ihaleler üzerinden yaparlar. İhaleler, kamu harcamalarında verimliliği ve ekonomik fırsat eşitliği sağlamayı önceler. Toplanan vergilerin doğru ve verimli şekilde harcanması için dizayn edilmişlerdir ama birçok ülkede yolsuzluk için bir araç haline gelmişlerdir. Blockchain ihalelerdeki bu problemlerin bazılarını açıklık ve kapsayıcılık ile elimine edebilir. Bu proje, ihale prosedürünü bir akıllı kontrat olarak gerçekleştirmeyi amaçlar.

Blokzincir Tabanlı e-İhale Sistemi bir ihalenin temel işlevlerini akıllı sözleşme üzerinde gerçekleştirmeyi amaçlar. İşbu sözleşmenin davranışı, zaman içerisinde değişim gösterir. İhale tanımı olarak Türk Kamu İhale Kanunu'nun açık ihale usulü tanımı baz alınmıştır. Bu tanıma göre ihale komisyonu bir tahmini ücret belirler, ve bu ücrete göre bir minimum ve maksimum rakam belirlenir. İhalenin minimum ile maksimum arasında sonuçlanmaması halinde ihale komisyonu toplanır ve durumu gözden geçirerek karar verir. İdeal senaryoda en iyi (en düşük) teklif bu sınırlar içerisinde. Proje, ideal senaryo içerisinde teklif verme, kazanan belirleme konularında insan müdahalesini ortadan kaldırır. Açık ihale usulünde minimum ve maksimum değerler ihale bitene kadar gizli tutulur, katılımcıların teklifleri de aynı şekilde gizli tutulur. Ama ihale tamamlandığında şeffaflık ve güven oluşması için bu veriler kamuya açık hale gelir.

Akıllı kontratlar Ethereum Sanal Makinesi'nde çalıştırılmak üzere Solidity dili ile yazılmıştır. Proje ihalenin genel matematiksel ve mantık işlemlerini kapsamayı amaçlar. Ancak, ihalelerde bulunan kimi prosedürler diğer kamu kurumları ya da hizmetleriyle bağlantılı olabilir. Bu yüzden sistemin daha entegre ve verimli hale getirilebilmesi, diğer kamu hizmetlerinin de blokzincir ekosistemine taşınması ile sağlanabilir. Proje özelinde ekosistem olarak Ethereum veya Ethereum Sanal Makinesi uyumlu bir açık blokzincir ağı baz alınır. Bu sayede blok zincirdeki verinin güvenliği sağlanırken, şeffaflık ve kamu güveni artırılır. Tüm işlemler geriye dönük olarak merkezsiz blokzincirine kaydedilir. Kullanıcılar kendi programları ya da topluluk tarafından geliştirilen güvenli uygulamalar yoluyla akıllı sözleşmelerle etkileşime geçebilir.

Ethereum ağı kullanmanın temel bir sorunu, ağdaki tüm verilerin herkes için erişilebilir olmasıdır. Tekliflerarası gizliliğin sağlanması için ihale sözleşmesinin yaşam döngüsü üç ayrı aşamada belirlenmiştir: teklif toplama, hesaplama ve ihale sonrası. İhale organizatörü ihaleyi yaratırken en az, en fazla ve tahmin edilen değerleri içeren bir verinin karmasını ve ihalenin aşama uzunluklarını kullanır. Teklif toplama aşamasında kullanıcılar gerçek teklifleri yerine, teklif ve rasgele bir karakter dizisinin karma değerini akıllı sözleşmeye iletirler. Hesaplama aşamasında kullanıcılar bu karma değerine karşılık gelen tekliflerini doğrularlar ve en uygun teklif hesaplanır. İhale sonrası aşamada ise ihale organizatörü ihalenin gizli sınırlamalarını doğrular, eğer en uygun teklifin sahibi sınırlara uygun bir teklif vermişse kazanan olarak belirlenir. İhale organizatör tarafından bitirildikten sonra akıllı sözleşme ve sözleşmenin verisi asla değişmez.

Kullanıcıların hatalı işlemleri doğru sanmaları ya da yanlış zamanda yaptıkları geçersiz işlemler sebebiyle maddi kayıp yaşamamaları amacıyla fonksiyonlarda sadece doğru gerçekleştirmeleri için uygun durumlar göz önünde bulundurulmuştur. Süre geçtikten sonra teklif vermek gibi bir yanlış etkileşim halinde akıllı sözleşme hata belirten bir mesaj dönmek yerine, o fonksiyonu geri döndürür. Yapılmış tüm işlemler geçersiz sayılır ve sözleşme durumu korunur. Kullanıcılar da geri döndürülecek fonksiyonları önceden bildiren IDE'ler sayesinde gerçekleştirmeyecek bir işlem için ücret ödemekten kurtulabilirler. Ancak ne sebeple işlemlerinin gerçekleşmediğini anlamak için ihale süreci hakkında bilgilendirilmeleri gerekir.

Proje kapsamında kullanıcıların karma değeri oluşturma konusunda yaşayabilecekleri sorunlara önlem olarak basit bir Python programı yazılmıştır. Bu program komut satırından çağırıldığında, ihale organizatörü ya da teklif veren olarak karma değeri oluşturmak mümkün olur. Program daha güvenli olması yönünde geliştirilebilecek bir taslak olarak da programcılara yardım edecektir. Bu Python kodu ile oluşturulan karma değerler, Solidity tarafından standart olarak hesaplanan Keccak öz alma fonksiyonuyla aynı yöntemle oluşturulduğu için sonradan doğrulanması problem yaratmaz. Bu yardımcı dosya ile birlikte akıllı sözleşmeleri herhangi bir insanın hemen kullanmaya başlaması mümkündür.

İlgili akıllı sözleşmeler Ethereum Remix kullanarak geliştirilmiş, Metamask ile oluşturulan cüzdanlar aracılığıyla Remix üzerinden Ethereum Ropsten, BSC Testnet gibi Ethereum ağlarına yüklenmiş ve edilmiştir. Test edilirken toplamda tüm fonksiyonları ve hataları kapsayan kullanım senaryoları kullanılmıştır. Ethereum geliştirici topluluğunun yaygın yazım stili kullanılmıştır ve tüm kaynak kodlar GitHub üzerinde açık haldedir².

²<https://github.com/urtuba/open-tendering-in-ethereum>

List Of Abbreviations

DBMS	Database management system
DSP	Double spending problem
DLT	Distributed ledger technology
TPPL	Turkish Public Procurement Law
OPT	Open procedure tender
EVM	Ethereum Virtual Machine
EOA	Externally owned accounts
OOP	Object oriented programming
SHA	Secure hash algorithms
NIST	National Institute of Standards and Technology
BSC	Binance Smart Chain

List of Tables

VIII

6.1	Time constraints for functions	16
6.2	Address constraints for functions	16
6.3	An example scenario and test results	17

List of Figures

1.1	Decision tree to determine use of blockchain [1]	2
2.1	Data payload of transaction and data of mined transaction	4
4.1	Model of Bidding Procedure	9
4.2	Revealing bid transaction	9
5.1	Inheritance of contracts	12
5.2	Deployment interface for compiled Tender.sol	13
7.1	Remix warns that transaction is going to fail	20

List of Algorithms

1	create hash for a bid	14
2	makeBid (_bidHash)	14
3	updateBid (_bidHash)	14
4	validateBid (_bidValue, _randomStr)	15

Contents

1	Introduction and Project Summary	1
1.1	Background	1
1.2	Motivation	1
1.3	Legal Basis	3
2	Technical Background	4
2.1	Ethereum	4
2.2	Solidity and Smart Contracts	5
3	Comparative Literature Survey	7
4	Bypassing Ethereum's Limitations	8
4.1	Achieving Confidentiality	8
4.2	Comparison Problem	9
5	Design And Implementation	11
5.1	Environment	11
5.2	Design And Implementation	11
5.2.1	Contract Deployment	13
5.2.2	Bidding Phase	14
5.2.3	Evaluation Phase	15
5.2.4	Post-Tendering Phase	15
6	Comparative Evaluation and Discussion	16
6.1	Testing Functions	16
6.2	Coverage	18
6.3	Confidentiality And Security	18
7	Conclusion and Future Work	20
7.1	The Project	20
7.2	Future Work	20

1 Introduction and Project Summary

Public procurement is a public trust problem about how taxes are spent. Countries use tenders to achieve trust and efficiency in governmental expenditures. However, corruption is a universal problem and exists in many countries [2]. Blockchain technology is claimed to be a solver of trust problems that require trusted intermediaries. In this project, the open procedure in Turkish Public Procurement Law is implemented using blockchain. Ethereum and its programming language Solidity is used to code smart contracts. The implementation covers the standard procedure. Exceptional cases are not solved. Thus, decisions taken by humans are not eliminated completely, yet reduced.

1.1 Background

Database is the generic name for any data collection. In information technologies, database management systems are implemented to store data in a structured way. In financial industry and public affairs, people's records are stored in a central database using a DBMS. DBMS has database administrators and other authorized users to manipulate data. Authorized people to insert, update or delete critical data is a potential point of failure. People can make mistakes or abuse their authority. DBMS's always need trusted intermediaries to keep data correct and up-to-date. This problem is solved with Bitcoin using distributed ledger technology named blockchain. As human intervention is no longer necessary, blockchain technology is a new paradigm in how data is managed. In blockchain, trusting math replaces trusting people [3].

Digital assets are easily copied, unlike physical assets. Therefore a digital money is created, it can be spent multiple times. The problem of spending the same money more than once is called double spending problem. In the monetary system, DSP is always managed by financial institutions. People trust banks, banks trust authorized personnel. People cannot claim ownership of any digital asset without referencing a trusted third party. When bitcoin and underlying blockchain technology were first conceptualized in 2008, the double-spending problem had a solution without any trusted intermediary [4]. Bitcoin is an electronic cash system that is free of the possible weaknesses of a trusted third party [4]. Blockchain is a distributed database system that has pseudonymous privacy, security, inclusivity, and immutability in its nature; a database that has approved correctness by consensus of participants. It is used to ensure the correctness of transaction records in electronic cash systems called cryptocurrency.

1.2 Motivation

Considering blockchain as a distributed database technology instead of only an e-cash transfer protocol led to further application fields. Blockchain has changed perspective on problems in many industries, including government and public services. Smart contracts on blockchains

such as Ethereum provide different solutions to conventional trust problems. Ethereum combines DLT with a computer named Ethereum Virtual Machine that relies on distributed computing [5]. Ethereum is a platform to develop distributed applications that are controlled by contracts. These contracts are immutable and they can replace some solutions to real-world problems which have to be solved by trusted intermediaries until nowadays. In a study comparing blockchain and classical databases, a decision tree was created on which problems the use of blockchain is appropriate [1].

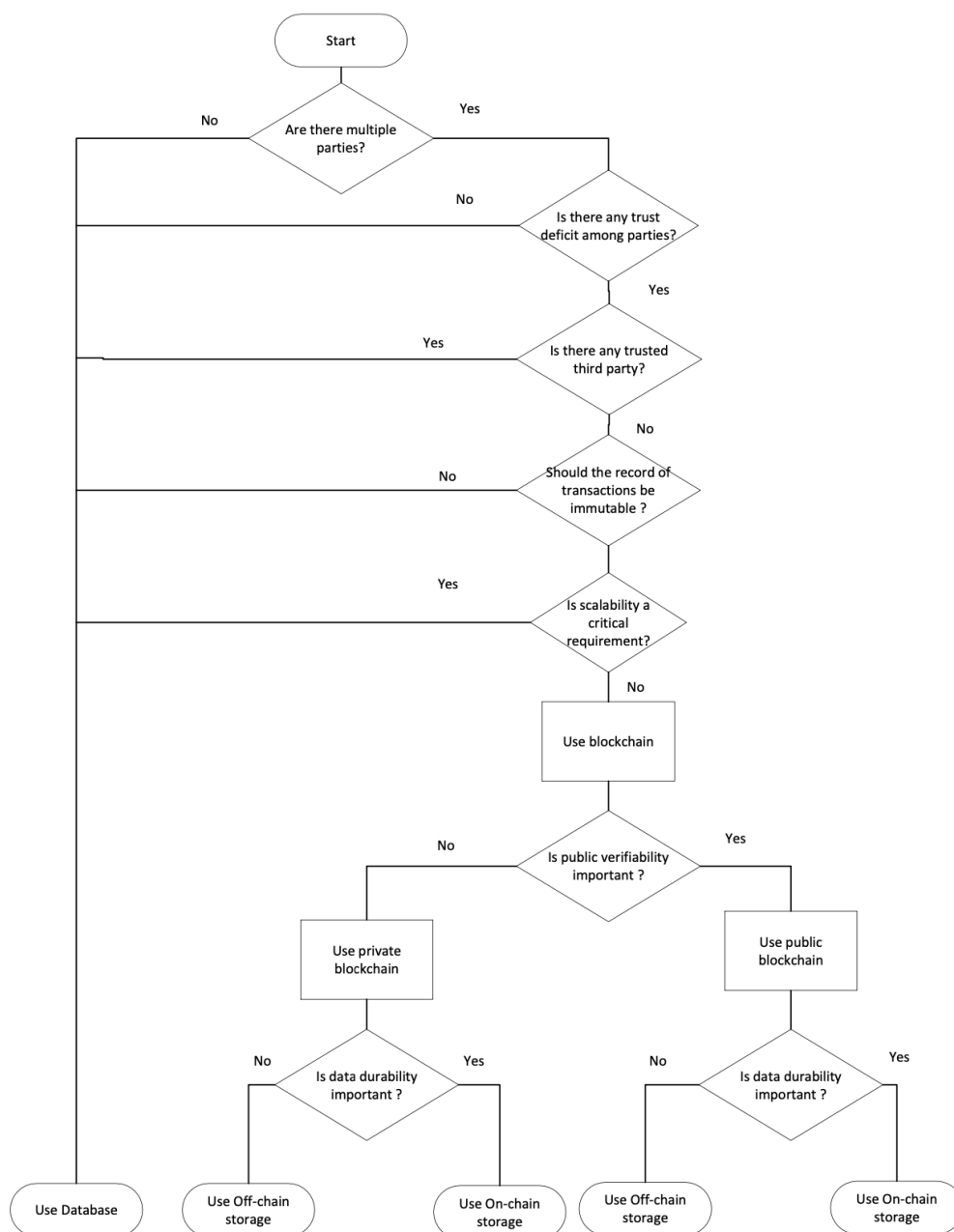


Figure 1.1: Decision tree to determine use of blockchain [1]

The tender process has multiple stakeholders: applicants, government and taxpayers. The trust between these parties is important for the efficiency of public expenditures, equal economic opportunity and public trust. To ensure these values, government agencies and officials are involved as trusted third parties. Each tender has its own application scale, which can be considered as small data that must be verified by the public. This data must be open to retrospective investigation. When all these features are evaluated together with the decision tree in Figure 1.1, the public tendering process is fit to be solved with blockchain.

1.3 Legal Basis

Different types of tenders are described in Turkish Public Procurement Law. This article and solution have been prepared specifically for the open procedure (açık ihale usulu), a tender with open procedure is named open procedure tender in the paper. An OPT allows any participant who satisfies requirements to join. The commission determines the estimated price and decides the upper and lower limits for bids. The estimated price and limits are not shared with the public until decision day. Bidders submit their bids within the given period and deposit a specified amount of money as a guarantee. The best price is accepted unless there are legal concerns or exceptions. Some exceptions may affect decisions such as the best price beyond the lower limit or the best price above the upper limit [6]. The ideal scenario (excluding exceptions) for OPTs is in this paper's area of interest. However, there are different cases or parts of OPT which are not deterministic enough to implement in EVM yet.

2 Technical Background

2.1 Ethereum

Ethereum is a cryptocurrency-backed open-source blockchain that allows the development of blockchain applications. It is used to develop games, decentralized exchanges, tokens and various decentralized applications. Ethereum has a native currency named ether, it is used to incentive good behaviour and pay for EVM processing power. Transactions simply transfer Ether, but they can have an additional data field. This data field has byte-codes to run by EVM [5]. Each instruction in the EVM instruction set requires a certain amount of gas units to be executed. The price of the unit process is called gas price in Ethereum. Gas prices are determined in the free market. Each transaction has a 'gas price' field specifying the sender's offer and 'start gas' for the maximum amount of gas units to use. Miners execute the best offers for themselves [7]. In Ethereum, blockchain interactions between accounts are done with transactions, thus each transaction has a sender account and a receiver account. Mined transactions are written in blocks. Each block updates machine state; account balances and stored changes.

Transaction	Mined Transaction
nonce	hash
gasprice	block hash
startgas	block number
to	gasprice
value	gas
data	from
v, r, s (ECDSA)	to
	value
	input
	index

Figure 2.1: Data payload of transaction and data of mined transaction

Ethereum has two types of accounts: externally owned accounts and contract accounts. EOAs have an ether balance, they are controlled by private keys by the owners and have no code associated with them. Contract accounts have codes, they have no owner after deployment, and they can interact with other contract accounts and EOAs if triggered by a message [8]. An EOA

can call functions in contract accounts and they can return value responses [5]. As an analogy, each EOA is a user, each contract account is a computer application, and EVM is a distributed computer. Miner population guarantees security. Private keys and addresses provide confidentiality while allowing transparency.

A fundamental advantage of Ethereum over previous blockchain networks is that smart contracts running on it are aware of the blockchain network. The contract code can read block data and addresses. This allows Ethereum smart contracts to be coded to change behaviour over time. Block time can be used to predict and scope future events. By using block time, a smart contract can change behaviour of own functions. Since tendering has time constraints, block awareness is a must-have for blockchain-based tendering. The features mentioned above are Ethereum's advantages to develop such systems in Ethereum.

A public tender is a standardized process to run in each procurement. It can be implemented as a smart contract. Since the code on EVM is aware of the blockchain itself, it can use timestamps of blocks for time-dependent events, therefore we do not need to get the current time from an external source. The starting and ending times of tenders can be represented in the blockchain. Smart contracts' ability to communicate with others, help us to separate the general tendering process and instances of the process. Each tender may have a different address by using proxy contracts which delegates execution to the implementation contract. Thus, openness is more obvious since all citizens may inquire about any tender with its own Ethereum address.

2.2 Solidity and Smart Contracts

Solidity is a smart contract language, developed to be run on EVM. Solidity is an object oriented programming language, contracts are similar to the classes in other OOP languages. Polymorphism and libraries are supported as well as user defined data types. Solidity is statically-typed and Turing-complete [9]. It is the most-used smart contract language. Solidity has Ethereum-specific methods to interact with blockchain. Since Ethereum blockchain is open, any smart contract code or function call can be observable for anyone. Therefore, security practices of Solidity code may be different from general purpose programming languages.

Writing data into Ethereum blockchain is costly, therefore smart contracts are designed to be simple, less data-dependent programs. Large blocks of code are not preferred in smart contracts. A smart contract is identified by a contract address after it is created; it has its own storage, executable code and Ether balance. Contract code is a low-level byte code called EVM code. High-level Solidity code is compiled into EVM code. [10]. Contract has some functions that have to be triggered by transactions to the contract address. Executed contract code changes chain state immutably, therefore state changes are not recorded to blockchain before run and finish successfully. If a function call had a problem during execution, all previous operations are reverted despite recording into blockchain.

Solidity code is compiled using the relevant version of Solidity Compiler. Compiled code is deployed using 3rd party applications, Ethereum Remix or web3 libraries in programming languages. Ethereum network has a JSON-RPC endpoint to interact with, thus any user can interact with blockchain without running node. Smart contracts may refer to other smart contracts using their contract addresses. Therefore, previously created smart contract functions are capable of being used by newer ones for cost efficient deployments.

3 Comparative Literature Survey

Governmental expenditures are made through tenders by using citizens' taxes. Public tendering must be transparent to build trust in governance. Blockchain is also claimed to be transparent, and many blockchain projects are developed for transparent governance. Some tendering solutions using blockchain have been proposed and developed before. They are generally not about the implementation of existing rules but creating a transparent governance utopia. A procedure that designed for TPPL was not implemented before, however, there are similar solutions.

Tendering is one of the possible fields that blockchain may change in the future. However, there is no specific study, design, or implementation of about tendering in blockchain before 2018. One of the first studies on the topic [11] is appeared in 2018. Implementation is done with Solidity smart contracts as done in this project, but the system was a tendering framework rather than a realization of a specific law. This year can be assumed as the starting point of the discussion about tendering in Ethereum/blockchain.

In the following years, some implementations are proposed and implemented on tendering in the blockchain. Some of them prioritized open governance; others prioritized archiving or verification. A typical solution for tendering contracts is hashing bids, then transacting hashes into smart contracts. Then, validation of bids is done by trusted intermediaries using SHA-3 hashes that are recorded in the blockchain. However, some researchers implemented all steps of tendering, from bidding to declaring winner, in the blockchain [12]. This project aims to include all steps, with compliance to Turkish Procurement Law. Some similar examples exist, however, the subject is new, and implemented solutions are few and unlocalized. Thus, contributions to this topic could lead to more efficient and trustworthy tenders.

4 Bypassing Ethereum's Limitations

Besides Ethereum's advantages and compliances for tendering, it has some limitations. Confidentiality of bids and tender constraints is a problem since all data in the Ethereum blockchain is open to the public. This problem may be eliminated by hashing values, however the hash of data has its own problems. The proposed two-phase process eliminates the confidentiality problem successfully. An evaluation phase exists, besides the well-known bidding phase of tenders. Problems and solutions are covered below.

4.1 Achieving Confidentiality

Confidentiality is one of the major principles in an open public tender, however, Ethereum is transparent. Anybody can look for any data in the blockchain. Functions, variables, arguments, and anything except private keys are accessible [5]. Bidders reveal their addresses and bid prices if the offer is stored in the contract. A participant can investigate previous offers and determine the minimum (best) offer, therefore offering any price above the best offer is obviously illogical. Therefore, bids have to be hidden before sending them to the blockchain.

$$f(n) = h \quad (1)$$

$$f(n||s) = h \quad (2)$$

The solution is a hash function (f), that maps integers (n) to fixed-size byte string (h) that may hide the number (Equation 1). However, hashing without unexpected input does not provide security. Anybody can determine a reasonable price range for the tender, and create hashes for every integer in the range to get any offer's price. When a commitment scheme is applied to the bid, hashing the combined input is safe. A character string version of bid (n) is concatenated with a randomly generated string (s) to create a character string. This string is used in the hash function as input (Equation 2). Produced hashes are secure if the hashing algorithm is secure. In this case, the Keccak algorithm is used since it is natively supported by Solidity. Participants of the tender clarify their bids after the bidding phase by using bid values and randomly generated string. The procedure described above is simply visualized in Figure 3.1.

A similar commitment method is applied to tender constraint values that have to be kept as secret during the tendering process. The minimum, maximum and estimated values are concatenated with a randomly generated string to produce input string. The hash of the input is stored as a tender hash. Tender organizers reveal tender constraints after the bidding and evaluation phase. The winner of the evaluation phase is declared as winner if the bid satisfies conditions.

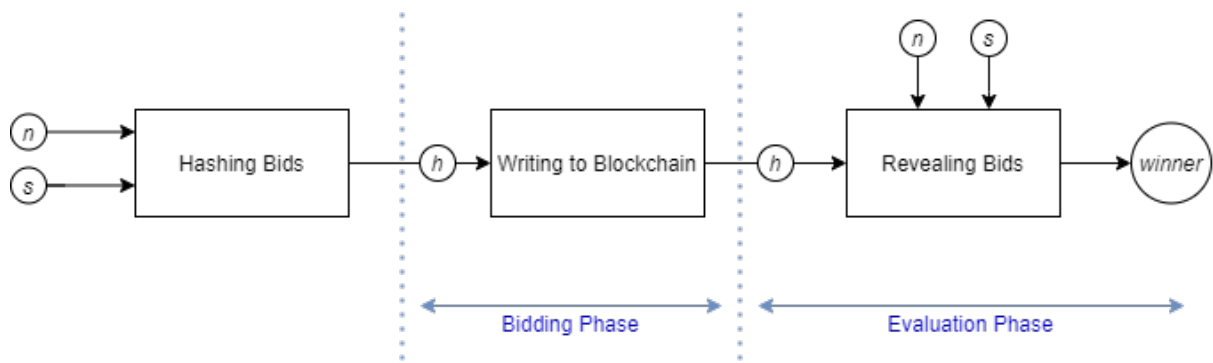


Figure 4.1: Model of Bidding Procedure

4.2 Comparison Problem

Comparison operations are required to list bids in order to declare the winner. However, Ethereum has limitations in code execution. Each execution should be triggered by a function call to contract. Automated programs are not possible in the Ethereum virtual machine. The solution is executing comparison operations when function calls are done. Another limitation is storing hash values; they are irreversible. Comparing two bids using their hash values is impossible. Thus, arithmetic/logic operations are executed while bids are being revealed. These operations are all done in the evaluation phase.

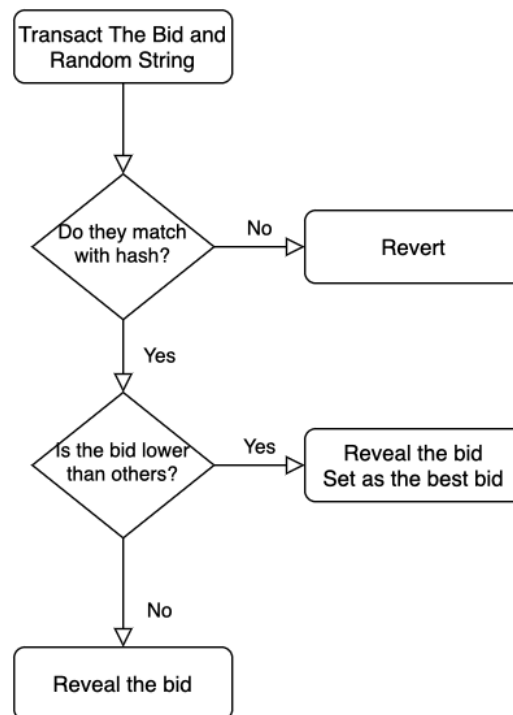


Figure 4.2: Revealing bid transaction

Each bid should be revealed in the evaluation phase to be valid. Bidders provide initial bid value and pseudo-random string. If the revealing transaction's Keccak-256 hash matches with one of the bid transactions, the original bid value is stored alongside the hash value. This transaction also triggers a piece of code which compares the previous best bid with the revealed bid. If the last transaction reveals a value that is lesser than the initial bid, it is recorded as the best bid instead of the previously stored value. When the evaluation phase ends, the revealed bids are considered valid, the winner and the winner's bid are already calculated.

5 Design And Implementation

5.1 Environment

To create and interact with smart contracts, we need externally owned accounts. Metamask, a Google Chrome extension, is used to create and manage EOAs. Metamask is capable of interacting with web3 applications like Ethereum Remix and Etherscan. Ethereum test networks are used for blockchain. Metamask can be used with any Ethereum network.

Solidity code is written in a text editor, compiled and deployed to the blockchain. The Ethereum Foundation has an online integrated development environment called Ethereum Remix. All development, debugging and deployment operations are supported by Remix. Remix also runs a JavaScript EVM for testing contracts locally. Remix can compile the code in any compiler version. Solidity versions between 0.4.0 and 0.6.0 (excluded) are used in the project. All smart contract development is done using Remix.

Some programming languages have libraries to develop web3 applications. Web3.py interface [13] in python is used to create hash values, in a way that can be reproduced in EVM. Web3.py is not used to interact with Ethereum blockchain.

5.2 Design And Implementation

The application is designed with the OOP approach. Variables and utilities with similar purposes are implemented together in different contracts. Inheritance is used to simplify code. Each contract has its own file. Contract files are all compiled together while compiling the inherited Tender contract. TenderLib contains the necessary function to convert integer to string. TenderTimer contract keeps track of time for the tender. TenderData holds hashed versions of bids and defines internal functionality to simplify tender implementation.

The following interactions are defined in the Tender contract: make bid (*makeBid*), update bid (*updateBid*), validate your bid after bidding ends (*validateBid*), end the tender if procedure is done (*endTender*) and a method to get winner (*getWinner*). All the data about the tendering process is publicly reachable. Compilers automatically create getter methods for public variables. Tender contract is the user interface for the tender, bringing inherited contracts' functionality together. Figure 5.1 shows the inheritance schema, plus sign refers to public members while minus sign stands for private or internal members.

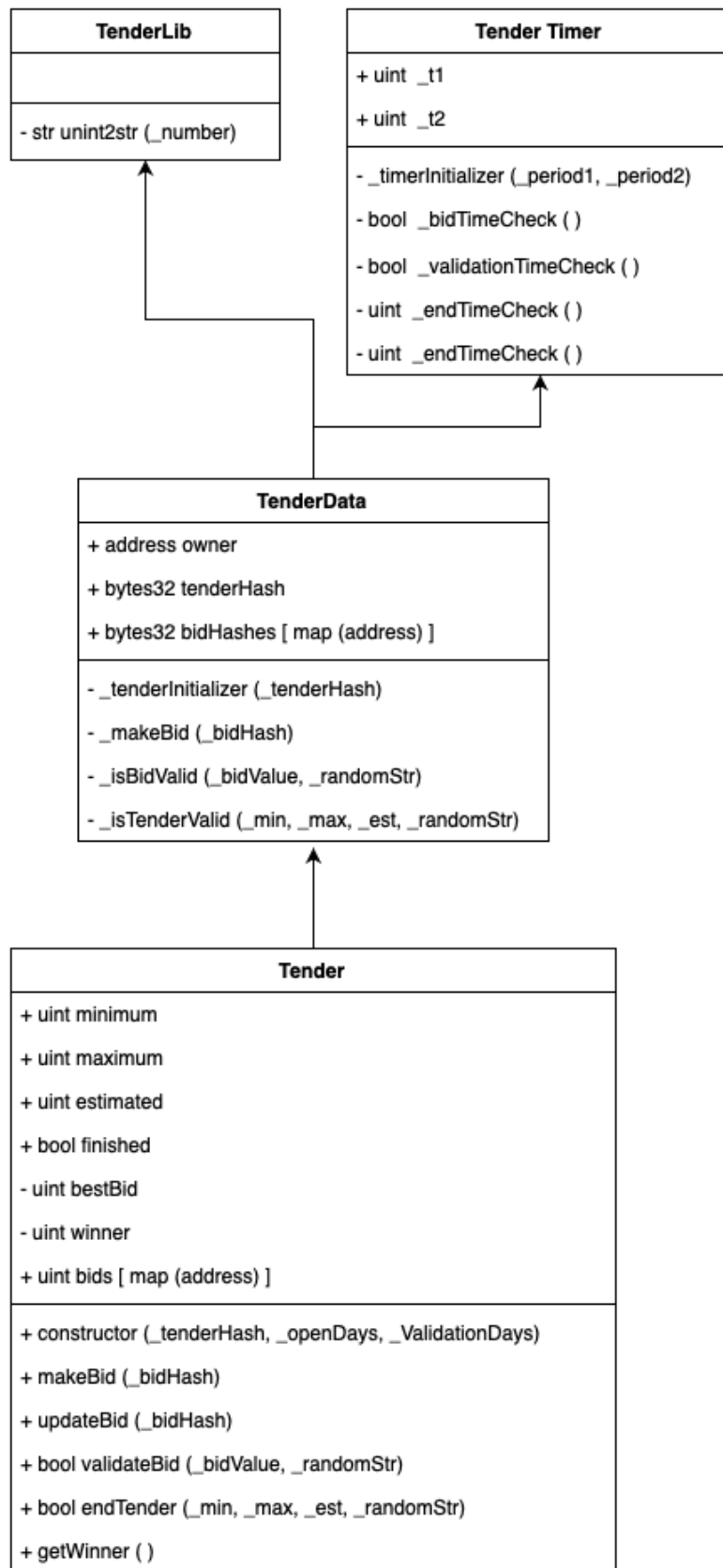
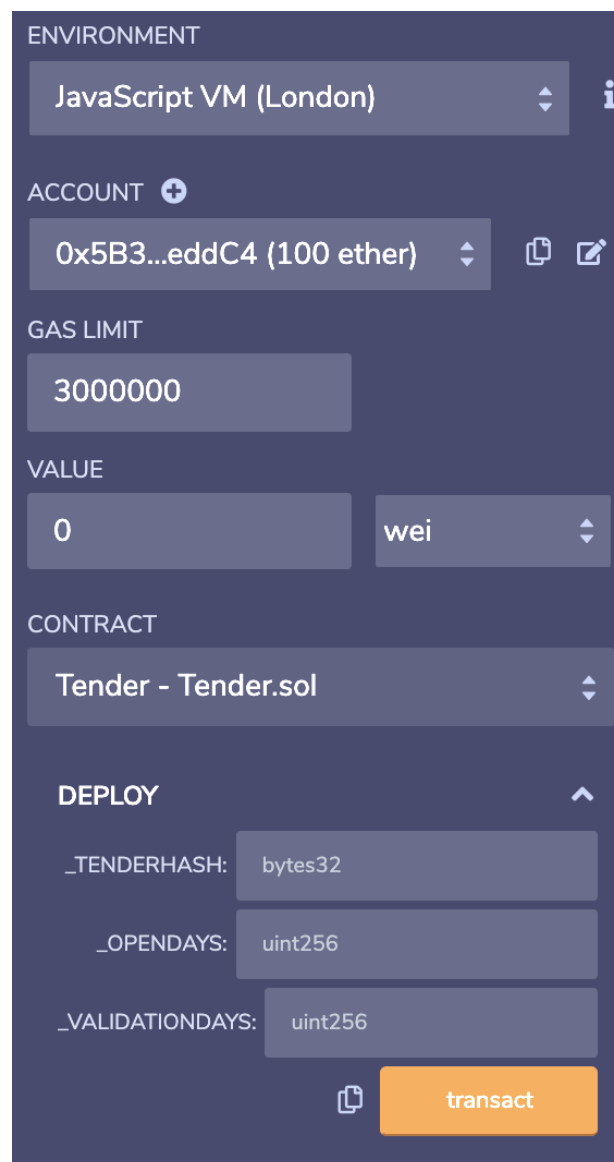


Figure 5.1: Inheritance of contracts

5.2.1 Contract Deployment

While deploying Tender contract using Remix, constructor parameters are given. While *openDays* specifies the length of the bidding phase, *validationDays* specifies the length of the evaluation phase. Constructor calls for *TenderData* and *TenderTimer* initializers. *TenderTimer* determines 2 time points while contract is created. *t1* is the time when bidding phase ends and the validation phase starts. *t2* is the time when validation ends. The tender timer module is used to check if transactions are not allowed in this phase. The account address that is the message sender for the contract creation call is recorded as the tender owner. The owner does not have permission to make a bid for its own tender. When the owner deploys the contract, the tendering process begins. The contract will have a life span consisting of three phases: bidding phase, evaluation phase and post-tendering phase.



The image shows the deployment interface in the Remix IDE. It is a dark-themed window with several sections:

- ENVIRONMENT:** A dropdown menu showing "JavaScript VM (London)" with an information icon to its right.
- ACCOUNT:** A dropdown menu showing "0x5B3...eddC4 (100 ether)" with a plus icon to its left and copy/paste icons to its right.
- GAS LIMIT:** A text input field containing "3000000".
- VALUE:** A text input field containing "0" and a unit dropdown menu showing "wei".
- CONTRACT:** A dropdown menu showing "Tender - Tender.sol".
- DEPLOY:** A section with three constructor arguments:
 - `_TENDERHASH:` with a value of "bytes32".
 - `_OPENDAYS:` with a value of "uint256".
 - `_VALIDATIONDAYS:` with a value of "uint256".
- At the bottom right, there is a "transact" button and a copy icon.

Figure 5.2: Deployment interface for compiled Tender.sol

5.2.2 Bidding Phase

Applicants of the tender do not transact their actual bids to achieve confidentiality. They have to create a hash code using the algorithm that is given below. The bid hash value is calculated using Web3.py Keccak function with parameter concatenated string of the bid and a random string padding. A similar process was also used in the creation of the tender hash. They are implemented to help applicants to create hash values.

Algorithm 1: create hash for a bid

```
str bid  $\leftarrow$  INPUT
str secureLenght  $\leftarrow$  32
str randomStr  $\leftarrow$  randomString(secure.length - lenght(bid))
str bidStr  $\leftarrow$  concat(bid, randomStr)
str hashStr  $\leftarrow$  Keccak(bidStr)
bytes32 hash  $\leftarrow$  toHex(hashStr)
```

While the current time is less than t_1 , the state of the contract is in the bidding phase. Applicants make bids without revealing their offers. Making and updating bids are separate because they are different procedures in reality, separating making and updating bids makes retrospective data readable. They both delegate the update operation to internal *_makeBid* function. After t_1 , all requests to make or update bids are reverted.

Algorithm 2: makeBid (*_bidHash*)

```
if bidHash[sender] is undefined then
  | if now <  $t_1$  then
  | | bytes32 bidHash[sender]  $\leftarrow$  _bidHash
  | end
end
```

Algorithm 3: updateBid (*_bidHash*)

```
if bidHash[sender] is defined then
  | if now <  $t_1$  then
  | | bytes32 bidHash[sender]  $\leftarrow$  _bidHash
  | end
end
```

5.2.3 Evaluation Phase

Since all bids are recorded with corresponding has values, bids are revealed to end tender. As mentioned in section 4.2, hash values cannot be compared to each other to sort bids. Bids should be stored as numbers for openness and to sort bids. Applicants call the bid validation function with their bid values as number and random string padding. Each successfully validated bid is recorded as a public address for integer mapping. If the applicant's bid is lower than the previous best bid, the best bid and address of the winner are updated with the applicant's data. All validation calls are reverted out of the interval $t1 < t < t2$ called evaluation phase.

Algorithm 4: validateBid (*_bidValue*, *_randomStr*)

```
if bidHash[msg.sender] is defined then
  if  $t1 < now$  and  $now < t2$  then
    str bidStr  $\leftarrow$  uint2str(_bidValue)
    bytes hash  $\leftarrow$  concat(bidStr, _randomStr)

    if _bidHash[msg.sender] = hash then
      int bits[msg.sender]  $\leftarrow$  _bidValue

      if _bidValue < bestBid then
        bestBid  $\leftarrow$  _bidValue
        winner  $\leftarrow$  msg.sender
      end

      RETURN (true)
    end
  end
end
```

5.2.4 Post-Tendering Phase

After the end of the evaluation phase, all function calls to change the contract state are disallowed. The owner of the tender contract reveals tender details after $t2$. If the revealed constraints are satisfied by the winner, *getWinner* function returns the address of the winner to all calls. If the winner's bid is out of bounds on minimum or maximum bids, *getWinner* reverts the transaction instead of returning to an address. Once the owner ends the tender, the contract storage and phase never changes.

6 Comparative Evaluation and Discussion

6.1 Testing Functions

The contract code is designed to respond to function calls when allowed. There are time and address constraints. For example, the owner cannot submit new bids and no bid is submitted after the bidding phase. All of these rules are implemented as requirements or if statements that prevent the code from reaching the return state. All function calls except the ones satisfying constraints are reverted. Thus, no operation can modify contract data and end successfully except valid function calls. There are three types of constraints: time, address (account) and dependent functions. *updateBid* function is used if *makeBid* is used before, *getWinner* is only returns result after *endTender*.

	Bidding Phase	Evaluation Phase	Post-Tendering Phase
makeBid	pass	revert	revert
updateBid	pass	revert	revert
validateBid	revert	pass	revert
endTender	revert	revert	pass
getWinner	revert	revert	pass

Table 6.1: Time constraints for functions

	Owner	Applicant
makeBid	revert	pass
updateBid	revert	pass
validateBid	revert	pass
endTender	pass	revert
getWinner	pass	pass

Table 6.2: Address constraints for functions

Solidity code is compiled for EVM. The tendering contract can be deployed to any EVM compatible network. The contract was created and tested contract on Remix JavaScript EVM's, Ethereum Ropsten testnet and Binance Smart Chain testnet. Testnet is a free-to-use blockchain

network with lower difficulty building blocks. Final tests are run on BSC testnet since BSC testnet is EVM-compatible [14] and have an useful block explorer. For testing purposes, days are replaced with minutes and hours. After successful tests, tests to cover all cases are done using the deployed verified contract in BSC testnet with 1 hour of bidding and 1 hour of evaluation phase.

Since *getWinner* is view function, it does not need to spend gas to call and it cannot modify the contract data. Tests are applied to the first four functions in Table 6.2, since they may modify contract storage. The final test that covers all constraints for these 4 functions is done with 4 accounts. Accounts created by Metamask and transactions are created using Ethereum Remix connected to Metamask. The contract is tested with function calls against constraints with different execution scenarios covering each situation. Table 6.3 shows results for an example scenario. The deployed contract address and transaction history of the example in the table is reachable in BSC testnet block explorer with detailed results in GitHub³.

Order	Account	Phase	Operation	Result
1	A	Bidding	constructor	Success
2	A	Bidding	makeBid (Revert: disabled for owner)	Reverted
3	B	Bidding	makeBid	Success
4	C	Bidding	updateBid (Revert: makeBid required)	Reverted
5	C	Bidding	makeBid	Success
6	C	Bidding	updateBid	Success
7	A	Bidding	endTender (Revert: not in post-tendering phase)	Reverted
8	B	Bidding	validateBid (Revert: not in evaluation phase)	Reverted
9	D	Evaluation	validateBid (Revert: makeBid required)	Reverted
10	B	Evaluation	validateBid	Success
11	D	Evaluation	makeBid (Revert: not in bidding phase)	Reverted
12	C	Evaluation	validateBid (Revert: invalid input)	Reverted
13	C	Evaluation	validateBid	Success
14	A	Evaluation	endTender (Revert: not in post-tendering phase)	Reverted
15	A	Post-Tendering	endTender (Revert: ownership required)	Reverted
16	A	Post-Tendering	endTender	Success

Table 6.3: An example scenario and test results

All functions are called using Ethereum Remix with injected Web3. Ethereum Remix warned against all failed transactions before submitting. Interacting contracts using trusted 3rd party tools is easier and warnings prevent users from transacting disallowed functions. Using warnings before transact saves gas money spent on a function that is reverted. It is more user friendly and secure than returning false.

³<https://github.com/urtuba/open-tendering-in-ethereum>

6.2 Coverage

The outcome of this project is a Solidity code consisting of 4 contracts, and a python script to show how the hashes are created. Tender contracts are deployed to EVM to run. The contract has three different states called phases. Bidding phase is for applicants to make bids. In the second phase called evaluation, bidders reveal their actual bids by providing inputs of committed hash values. Bid updates are not allowed in this state. The best bid is calculated during the evaluation phase. The final state is the post-tendering phase. Applicants cannot perform any operations and the tender owner (organizer) reveals constraints as defined in TPPL. If the best bid is within the boundaries, the owner of the bid is declared as winner. The tender is considered as finished after the owner calls *endTender* function. When tender is finished, *getWinner* function starts to return the winner if exists. This implementation covers the mathematical and logical rules of an OPT.

The aim of the project is to solve tendering, especially OPT, with smart contracts. In the implemented system, the mathematical background of the tendering is covered. However, tenders have other constraints, such as application documents, fees, specifications etc. So the tendering contract must be supported by conventional methods to run. Agencies may announce specifications and the contract address related to the tender. Also, there is no Ether transfer functionality in the contract. The tender contract is not capable of collecting fees or payments from applicants. Ether is not a replacement for conventional currencies. Winners may register with an official document to comply with the classic procurement procedure by proving ownership in relation to EOA that is the winner.

6.3 Confidentiality And Security

Bids are hashed with randomly generated string to achieve confidentiality and bids are not revealed before the evaluation phase. The implemented python script proposes 64-length strings for hashing tender data. This string is concatenated from the minimum, maximum, estimated bid values and pseudo-random string. Bids are proposed to be hashed from 32-length strings concatenated from a bid and a random string. An attacker trying to predict the price for bids while attacking to get random string is unlikely to succeed, since any length of random string can be used.

The method used to secure bids is a simple commitment scheme. Commitment schemas are used to commit values while keeping them hidden, in order to reveal later [15]. In the commitment schema used in tendering contracts, the bids are hidden to be revealed later. Since EVM has a built-in Keccak hash function to create SHA-3 hashes, the Keccak function of Ethereum web3.py is used. It is a secure hashing algorithm awarded by NIST as SHA-3 competition winner [16]. An applicant commits a hash of the bid extended by pseudo-random string. Requested 32 and 64 for string lengths are examples to show how python scripts are implemented to create hashes. Applicants can select arbitrary-length pseudo-random strings to use the full

potential of SHA-3.

The security of the data is provided by the community of Ethereum or a compatible blockchain where the contract is deployed. Attackers are not able to change the contract data, assuming Ethereum is secured by a consensus protocol such as proof-of-work. The only way to change contract data is to create Ethereum blockchain faster than trustworthy miners. Ethereum incentives trustworthy miners to prevent 51% attack.

7 Conclusion and Future Work

7.1 The Project

The tendering smart contract for OPT is developed by this study. Since tendering is a legal procedure, the laws were decisive for the design. The project covered complete basics of tendering such as making and updating bids and revealing results to the public. Violating conditions reverts the function instead of returning false, so end-users may be alerted by web3 applications that before paying the gas price for failed transactions.

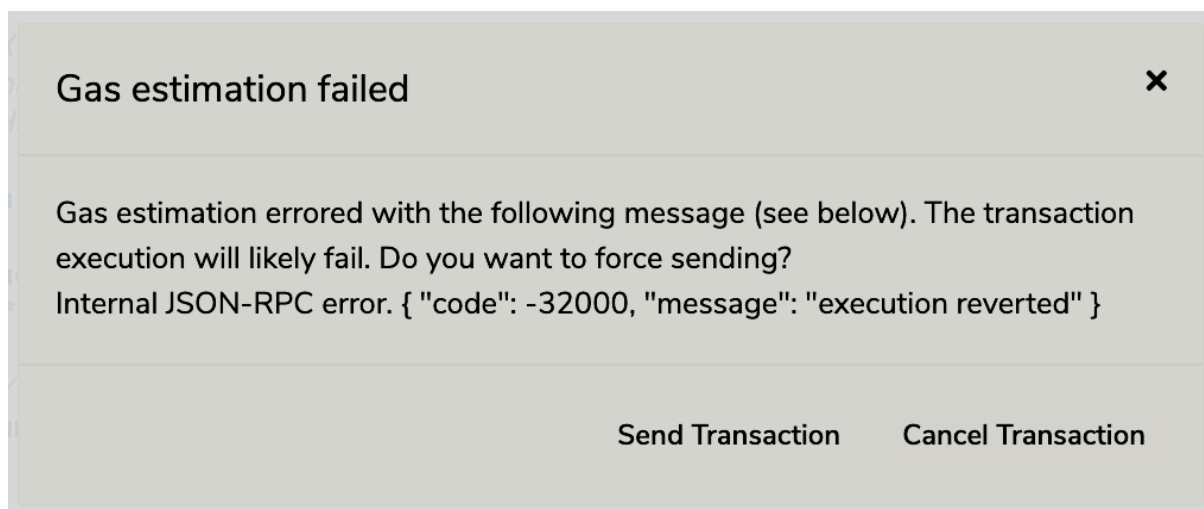


Figure 7.1: Remix warns that transaction is going to fail

Each tender is hosted at its own designated address in the blockchain. All interactions changing contract data and status are recorded. Retrospective investigations are easier than any conventional system. Deployed source code is verified for openness. Any citizen can investigate the code and historical transactions of any tender. It only provides essential rules of tendering, it is a step towards open governance. Guidance for users is crucial, since society is not experienced enough with blockchain technology.

7.2 Future Work

There are points to improving blockchain-based e-tendering. Tenders need documents and prerequisites that can be validated by blockchain. They may be part of this decentralized app, also if attackers sabotage the contracts with very high and very low orders. Government agencies demand application fees depending on the size of tenders. It can be included in contract, *makeBid* function may demand application fee in Ether or a token. The fee also makes attacking tenders illogical. Deploying the same contract for each tender may be inefficient and

potentially an error point. Proxy contracts may be used to refer to valid implementation as a logic contract, but store its own data. Using the address of the logic contract, everybody can be sure the correct version of the contract is used.

The project's scope is narrow, but it points to a potential starting field for open governance. Some countries and communities are interested in blockchain-based open governance models, even using cryptoassets as national currency. Using crypto currency as a national currency has its own risks and wide-spread macroeconomic effects to evaluate [17]. However, smart contracts, such as Blockchain-based e-Tendering Systems, are capable of working better when more governmental services use blockchain and cryptoassets. Transitions to blockchain are likely to change existing laws on finance and governance.

References

- [1] M. J. M. Chowdhury, A. Colman, M. A. Kabir, J. Han, and P. Sarda, “Blockchain versus database: A critical analysis,” in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. IEEE, 2018, pp. 1348–1353.
- [2] Y. Sezer, “Kamu ihale kanunu: Şeffaflık ve rekabet,” *Amme İdaresi Dergisi*, vol. 35, no. 4, pp. 57–82, 2002.
- [3] M. Nofer, P. Gombler, O. Hinz, and D. Schiereck, “Blockchain,” *Business & Information Systems Engineering*, vol. 59, no. 3, pp. 183–187, 2017.
- [4] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” *bitcoin.org*, 2008. [Online]. Available: <https://nakamotoinstitute.org/bitcoin/>
- [5] V. Buterin, “A next-generation smart contract and decentralized application platform,” *white paper*, vol. 3, no. 37, 2014. [Online]. Available: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf
- [6] “Kamu ihale kanunu (public procurement law),” *Resmi Gazete*, vol. 22.01.2002, no. 24648, 2002.
- [7] R. Yang, T. Murray, P. Rimba, and U. Parampalli, “Empirically analyzing ethereum’s gas mechanism,” in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2019, pp. 310–319.
- [8] C. Dannen, *Introducing Ethereum and solidity*. Springer, 2017, vol. 318.
- [9] M. Wohrer and U. Zdun, “Smart contracts: security patterns in the ethereum ecosystem and solidity,” in *2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE, 2018, pp. 2–8.
- [10] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, “Making smart contracts smarter,” in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 254–269.
- [11] F. S. Hardwick, R. N. Akram, and K. Markantonakis, “Fair and transparent blockchain based tendering framework - a step towards open governance,” pp. 1342–1347, 2018.
- [12] D. Mali, D. Mogaveera, P. Kitawat, and M. Jawwad, “Blockchain-based e-tendering system,” pp. 357–362, 2020.
- [13] E. Foundation, “Web3.py,” <https://github.com/ethereum/web3.py>, 2021.
- [14] B. C. Community. Binance chain documentation. [Online]. Available: <http://web.archive.org/web/20210813193411/https://docs.binance.org/>
- [15] O. Goldreich, *Foundations of cryptography: volume 1, basic tools*. Cambridge university press, 2007.

- [16] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, “The making of keccak,” *Cryptologia*, vol. 38, no. 1, pp. 26–60, 2014.
- [17] T. Limba, A. Stankevičius, and A. Andrulevičius, “Towards sustainable cryptocurrency: Risk mitigations from a perspective of national security,” 2019.