BLOCKCHAIN BASED SOLUTION FOR ELECTRONIC HEALTH RECORD INTEGRITY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF APPLIED MATHEMATICS
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

KAAN ÇELİK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
CRYPTOGRAPHY

SEPTEMBER 2022

Approval of the thesis:

BLOCKCHAIN BASED SOLUTION FOR ELECTRONIC HEALTH RECORD INTEGRITY

submitted by **KAAN ÇELİK** in partial fulfillment of the requirements for the degree of **Master of Science in Cryptography Department, Middle East Technical University** by,

ABSTRACT

BLOCKCHAIN BASED SOLUTION FOR ELECTRONIC HEALTH RECORD INTEGRITY

Çelik, Kaan
M.S., Department of Cryptography
Supervisor : Assoc. Prof. Dr. Oğuz Yayla

September 2022, 46 pages

With the development of technology, great developments have occurred in the healthcare area, as in every field. Over time, many solutions have been proposed for the processing of electronic health data. As a fact, there are critical factors that should be considered under these developments. This information in the field of electronic health is demanded both by some harmful organizations and people. In addition, there is an extensive market for these informations. Therefore, in electronic health data systems, the privacy of the patient, the executability of the system, and its protection against attacks are milestone requirements. One of the methods offered to provide these security measures is the blockchain technology. Many theoretical and practical blockchain-based studies have been achieved for preserving electronic health data. In this thesis, a blockchain based medical survey system is proposed to achieve the data integrity. The system is implemented on the Algorand blockchain and its steps are given. We also do the benchmark of the proposed system in terms of time and memory usage.

Keywords: Blockchain, Health Record, Cryptography

ÖZ

ELEKTRONİK SAĞLIK VERİLERİNİN BÜTÜNLÜĞÜ İÇİN BLOKZİNCİR TABANLI ÇÖZÜM

Çelik, Kaan
Yüksek Lisans, Kriptografi Bölümü
Tez Yöneticisi : Doç. Dr. Oğuz Yayla

Eylül 2022, 46 sayfa

Teknolojinin gelişmesiyle birlikte her alanda olduğu gibi sağlık alanında da büyük gelişmeler meydana gelmiştir. Bu süreçte, elektronik sağlık verilerinin işlenmesi için birçok çözüm yöntemi önerilmiştir. Bu gelişmelerin altında dikkate alınması gereken kritik faktörler olduğu bir gerçektir. Elektronik sağlık alanında ki bu bilgiler bazı kötü amaçlı kuruluşlar ve kişiler tarafından rağbet görmektedir. Ek olarak, bu sağlık verileri için geniş bir pazar bulunmaktadır. Bu nedenle elektronik sağlık veri sistemlerinde hastanın mahremiyeti, sistemin çalıştırılabilirliği ve saldırılara karşı korunabilmesi temel gereksinimlerdir. Bu güvenlik önlemlerini sağlamak için sunulan yöntemlerden birisi de blokzincir teknolojisidir. Elektronik sağlık verilerinin korunması için hem pratik hem de teorik olmak üzere bir çok çalışma yapılmıştır. Bu tezde verilerin bütünlüğünü sağlamak için blok zinciri tabanlı tıbbi anket sistemi önerildi. Sistem, Algorand blok zinciri üzerinde gerçekleştirilmiş ve adımları verilmiştir. Aynı zamanda, hız ve bellek kullanımı açısından performans testleri gerçekleştirildi.

Anahtar Kelimeler: Blokzincir, Tıp, Sağlık, Kriptografi

ACKNOWLEDGMENTS

I would like to express my very great appreciation to my thesis supervisor Assoc. Prof. Dr. Oğuz Yayla for his patient guidance, enthusiastic encouragement and valuable advices during the development and preparation of this thesis. His willingness to give his time and to share his experiences has brightened my path.

I also would like to express my gratitude and appreciation to Assoc. Prof. Dr. Ali Doğanaksoy for introducing me to cryptography.

I specially thank my family for their patience, support and mental motivation during thesis term.

Finally, I also would like to thank my colleagues for their technical support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

$\mathcal{A}$	Asset
`an`	Asset Name
`apaa`	Appliaction Arguments/Parameters
`apan`	Appliaction Call Type
`apap`	Appliaction Approval Program
`apap`	Appliaction Asset
`apgs`	Appliaction Global State
`apid`	Application ID
`apls`	Appliaction Local State
AVM	Algorand Virtual Machine
B^r	It denotes as constructed block-chain block in specified round r.
$\mathcal{C}$	Smart Contract
c	Challenge Value
`ctr`	Iteration Count
`dc`	Asset Decimal Digit Count
`df`	Default Frozen Value
$\mathrm{EPK}_i^{r,s}$, $\mathrm{ESK}_i^{r,s}$	Ephemeral public key, private key of user i in round r, step s
$\mathrm{ESIG}_i^{r,s}$	Ephemeral signature of user i in round r, step s
`fee`	Transaction Fee
`fv`	First Valid Round
`gen`	Genesis Hash
H	Hash value
$H^{\mathtt{i}}$,$\mathrm{SK}^{\mathtt{i}*}$	`i` implies last `i` bytes, `i`$*$ implies first `i` bytes of base value.
k	Nonce value
l	In the round r, it denotes as l^r i.e leader of round-r.
`lv`	Last Valid Round
m	Message, Data
$\mathtt{man}_{\mathtt{addr}}$	Asset Manager Address

MST	Medical Security Token
PAY^r	This expression implies a payment set in round block B^r.
PK	Public key
PK$_i^r$	The public key of user i in round-r.
PK^r	The set of public keys of all users in round-r.
PPK,PSK	Participation public key, participation secret key
Q^r	A unique value for specified round-r.
r	It implies that that current index of round in block-chain where $r \geq 0$.
s	It implies that that which step of certain round in block-chain where $s \geq 1$.
`snd`	Sender
SK	Private key / Secret key
SV^{r,s}	It implies that verifiers of block-chain round r, step s.
`tl`	Asset Total
TEAL	Transaction Execution Approval Language
`typ`	Transaction Type
`Tx`	Transaction
$\texttt{Tx}_{\texttt{SIG}}$	Signed Transaction
`un`	Unit Name
VRF	Verifiable Random Function
W	Word list

CHAPTER 1

INTRODUCTION

Technology is developing rapidly day by day. As a result of this development, many works that are done physically in real life are now transferring to the digital workflow. For instance, banking, commerce, multimedia, and advertising are affected by technology. At the same time, another area affected by these technological developments is the health field. Especially with the spread of the internet, the strengthening of computer technologies, and the learning of technology by people working in the field, the majority of health-related data transactions began to store in digital areas. During these transitions, databases were used as the first solution for storing health data. As a result of this transition, the number of data stored electronically has increased day by day. Due to both speed and memory requirements, distributed database structures were established. These accomplishments and transitions have been very beneficial for healthcare and many other sectors. However, it is an undeniable fact that there are parts of technology that cause difficulties in the field of health, as in every field. The most concerning subject among these challenges is the security of healthcare data. The medical operations and their results may contain sensitive or private data that is not desired to be captured or exposed by others. Therefore, ensuring the security of healthcare data is significantly necessary. Due to the necessity of providing security, many studies have been achieved about e-healthcare data protection, since the beginning of healthcare data processing in digital areas. The solution to the security concerns relies on the fundamentals of cryptography as confidentiality, integrity, and authentication. Many techniques have already been developed to provide these cryptographic fundamentals. These security problems can be eliminated with the combination of cryptographic techniques such as authorization control, encryption, masking, anonymization etc . As a result of the research and developed attacks in the literature, the problems that occur when sufficient security measures are not taken have been observed. One of the problems is that attacks made on a single point cause gain access to an adversary on the whole system since the centralized systems have a singular authority. At the same time, the fact that the system management is in a single authority is convenient for manipulating the data. In the medical sector, forging these results may lead to changing the treatment to be given to patients and all their characteristics, such as the duration and amount of these treatments. This action taken is likely to have very serious and undesirable consequences. Therefore blockchain structure has also been proposed among these measures which is used in many studies recently and their numbers are increasing day by day. In 2008, Satoshi Nakamoto released Bitcoin [16] officially as the first blockchain system. After that in 2013, Vitalik Buterin invented the Ethereum [5] and in 2014 he developed the Ethereum [6] with the smart contracts. With the interest in these two blockchains and the preservation of their place in the cryptocurrency market, the studies in this field have increased even more. One benefit of this abundance of blockchain systems is that we have many options when developing applications. Each one of them has different and similar attributes to the others. These features can be compared in terms of speed, security factors, cryptographic primitives, consensus mechanisms, block structures, account management, off-chain activities, accessibility, scalability, transaction cost, etc. We will mention the studies which are using the blockchain in Section 1.2. Algorand [9] is one of the developed blockchain systems in the literature. Micali [9] et.al proposed the Algorand as a Proof of Stake (PoS) based blockchain. We can list the features that caused us to prefer Algorand as follows. First of all, The PoS-based blockchain provides less energy consumption for transactions and block generations than PoW based blockchain. Moreover, Algorand's Byzantine agreement protocol [8] can handle the arbitrary number of malicious users as long as honest nodes hold the majority of the system. In addition, the consensus mechanism runs with the user replaceability principle and it provides preventing node-based attacks. One of the other attributes is that Algorand is a permissionless blockchain which means everyone can join Algorand as a node without any restriction. It cannot be forked at its origin layer. The next one is that it has 0.001 Algos fee which is an insignificant amount. Once we research the performance value of Algorand, we obtained the following results;

block proposal time is 0.5 seconds, block finalization time is 4.5 seconds and amount of transactions per second is 1,000.

1.1 Contribution

As we mentioned in previous section, the data preservation and the data immutability are the main requirements for the electronic health record systems. In this study, we design a system that aims to save the e-health record data and their results as evidence. Since proof immutability is the main requirement of these kinds of systems, an immutable ledger mechanism is an ideal solution for this requirement. Therefore, we have designed a system that can work in sync with the blockchain. We aim to provide a scalable, fast, and secure healthcare-proof system to users. While designing this system, we used cryptographic algorithms that comply with the standards. In addition, we chose to use parameters according to the specified standards for these algorithms. After this design, we developed and implemented a software that has an interface and works synchronously with the preferred blockchain structure. Due to the benefits of Algorand's technology, its internal mechanism, its unique structures, we preferred Algorand as the blockchain and developed the application compatible with Algorand. In addition, since, Algorand is a public blockchain, we have added a control mechanism for preventing the random access to our application context. For providing the access control mechanism, we inserted an Algorand-based token system to our software. The source code of the project is available in Github (https://github.com/kaan-celik/Blockchain-Based-Solution-For-Electronic-Health-Record-Integrity) [7].

1.2 Related Works

We will briefly discuss the studies about blockchain-based e-healthcare solutions in the literature.

Azaria et.al in [3] offered a blockchain-based user access control system named *MedRec*. They used the Ethereum blockchain and its smart contract mechanism. They used nodes with two roles as patient and provider nodes. In MedRec's solution, there are three smart contract designs. The registrar contract ensures the relation between the Ethereum address and the system user. The summary contract provides record history for each patient, and the patient-provider contract provides access control between patients and related providers.

Rajput et.al [19] proposed a blockchain-based control management system for the patient healthcare data. In Rajput's system, Hyperledger Fabric [1] was used for the blockchain mechanism. The business logic was implemented with the smart contract such as registering and retrieving data operations. In addition, they used API connection between the application and blockchain side. Moreover, they determined access control rules according to the roles of the users in the system for preventing unauthorized user activities on patients, doctors, or staff data.

Shahnaz et al. [20] developed a pure blockchain system with role-based authorization for ensuring the privacy of healthcare data. In this system, they used Ethereum [5] and its smart contract mechanism Solidity. In their system, there are two types of the smart contract. The first one is the patient record contract, and the other one is for roles. The first one contains all create, read, update and delete (CRUD) functions such as patient record saving, viewing, grant or revoke access controls. The role contract is predefined via *OpenZeppelin* [2] library. Their user definitions are based on only Ethereum users. Therefore, the blockchain provides interaction between smart contracts and user.

Xu et.al [23] studied on blockchain-based approach about IoT based healthcare system named Healthchain. They constructed two related chains. The first one is a public blockchain named Userchain and the second one is a consortium blockchain named Docchain. Userchain stores the user information data in *Ublock*. For the confidentiality of the user's IoT data, they have used AES [10] symmetric encryption. In addition, for storing the encryption key and encrypted user's IoT data separately, they have used two different transactions. Doc-chain, on the other hand, stores the diagnostics of the related users in their block named *Dblock*. As a consensus mechanism, Userchain has a PoW mechanism. However, they prefer the PoS-based Byzantine Fault Tolerance for the Docchain. Furthermore, the encrypted data are stored on IPFS in storage nodes.

Fan et.al [11] suggested another blockchain-based electronic medical data sharing solution named MedBlock. In this solution, they constructed a private blockchain. In parallel, they use a hybrid consensus mechanism for reducing resource wasting and improving the network speed. The data security is provided by a signature-based access control protocol. Accordingly, if the user signature is not among the signature collection in their system, user access is blocked. The studies we have mentioned so far are mostly blockchain studies based on providing access control.

Li et.al [14] suggested a blockchain-based medical data preservation system with Ethereum. They developed three layer application as many solutions have been proposed. These layers are the user layer, the application layer, and the blockchain layer. Briefly, they retrieve the data from the user layer, process, read or update it at the application layer and submit the encrypted and hashed data to the blockchain layer. In this process, they used algorithms such as AES and SHA-256 to preserve data confidentiality and integrity.

Pavel et.al [18] has specified the problem as medical data transferring and proposed a blockchain-based solution to their PoS-based blockchain structure. They used signature-based authorization for the image transfer, retrieving, or viewing.

1.3 Overview of Thesis

In Chapter 2, we introduce the Algorand blockchain and its properties. It covers key generation methods, consensus mechanism, and the additional structures of Algorand. The key usage and its cryptographic attributes are mentioned in Section 2.1. Execution of the consensus and Byzantine Agreements are detailed in Section 2.2. Finally, the provided contract and asset technologies are mentioned briefly in Section 2.3. In Chapter 3, the core components of our design and its mechanism are introduced. Then, we give some applications, blockchain integration, and technical properties of the proposed system. Implementation of the token and smart contract structures is explained in Section 3.1. The background of the application, user processes, and the cryptographic background operations of these processes are explained in Section 3.2. Moreover, we mention software technologies used in the development of the proposed system. In the last chapter, we summarize the general outlines of the system we have created. In addition, we talk about the stages that we will do and aim to do in future studies.

CHAPTER 2

ALGORAND

2.1 Key Generations

Algorand uses specific keys for the transaction operations at the user side. There are 4 core keys in Algorand. These are public-private key pair, public address, mnemonic key, and participation key. Except for the participation keys, all of these keys are generated with private key SK and public key PK. In the next sections, basic keys (private key, public key), user keys (user address, mnemonic key) and participant key generation has been mentioned.

2.1.1 Public Key & Private Key Generation

Algorand uses the elliptic curve Ed25519 [4] for the key generation as in the following statement:

$$(\mathrm{PK}, \mathrm{SK}) = \mathrm{Ed25519}(\mathit{seed}), \tag{2.1}$$

where *seed* is a random initial value.

2.1.2 Address Generation

Since the users or participants can not use directly the PK, SK key pair, the high-level key requires for the transaction. And it called account address. The generated PK, SK pair is low (Algorand)-level keys and it is used in the background work of Algorand. Since the users or participants cannot use directly the PK, SK key pair, the high-level key requires for the transaction. And it called account address. The address generation is given in the following statement:

$$H_{\text{PK}} = \text{SHA-256/512}(\text{PK})$$
$$\texttt{Address} \xleftarrow[Base32]{encode} \text{PK} \mid\mid H^4_{\text{PK}}$$

Firstly, the hash value of the generated public key H_{PK} is obtained by SHA-256/512. In the second step, the four-byte checksum of the hash value of the public key H^4_{PK} and the public key PK is combined and the byte array is obtained. Finally the formed byte array is encoded with base32 and Algorand address is generated.

2.1.3 Mnemonic Key Generation

As mentioned in the previous section, the users cannot use directly the PK, SK key pair. For the public key usage at the user level, Algorand has an address structure. As with the public key, such a user-level structure, is required for the private key. This structure is named mnemonic key. The users can use this 25 words key as a password in their account. Besides, for user operations, it is more memorable than the byte string of a private key. The generation of the mnemonic key is given in Figure 2.1:

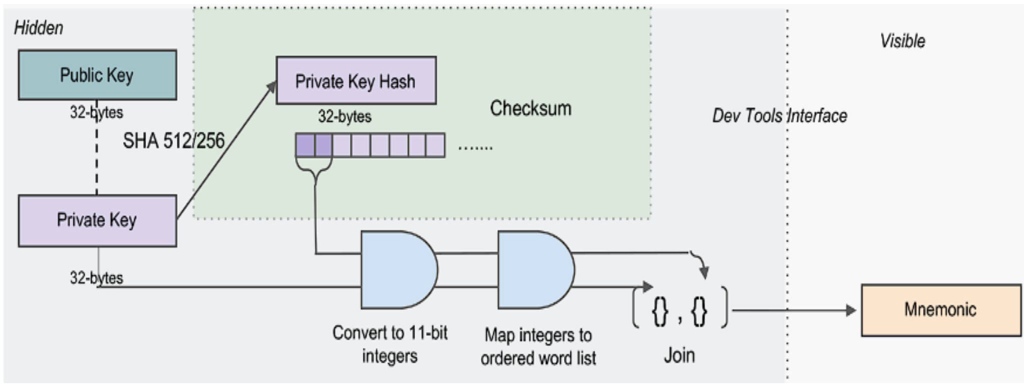

Figure 2.1: Mnemonic Key Generation
Reference: https://developer.algorand.org/docs/get-details/accounts/

$$f : \text{SK} \rightarrow W \mid \text{SK} \in \{0,1\}^{256}, W \in \{0,1\}^{11}$$
$$g : H^{2*}_{\text{SK}} \rightarrow W \mid H^{2*}_{\text{SK}} \in \{0,1\}^{16}, W \in \{0,1\}^{11}$$

First of all, hash value of the private key H_{SK} is generated with SHA-256/512. After that, the private key is mapped with the function f and 24 integers are obtained. Each one is 11-bit integer. Next, the integers are mapped one-to-one with the BIP-39 English word list and string of private key SK is obtained. The BIP-39 list contains 2048 words and it is equal 2^{11} Then the last 2 bytes checksum of the hash of the private key H^{*2}_{SK} is mapping with the function g and string of the hash value is generated. Finally string of private key and generated hash string is concatenated. As a result, we obtain 25 words mnemonic key.

2.1.4 Participation Keys

In general, the account owner uses generated PK, SK key pair for signing transactions. However, In consensus protocol, the user uses the participation key during the committee selection process. Once a user wishes to participate in the consensus mechanism, the user must have the following two key pairs. The first key pair is called *participation keys*. And the other one is ephemeral key. As mentioned in the previous section, these keys are generated with Ed25519 elliptic curve. It can be explained in the following statements:

Participation Keys: Participation public key PPK and participation private key PSK are generated for signing the generated ephemeral keys given below
Ephemeral Keys: For each user $i \in \text{SV}^r$, round $r \in r_0, \dots, r_n$ and step $s \in 1, \dots, s_n$, the corresponding ephemeral keys are generated as given below:

$$(\text{EPK}_i^{r_0,1}, (\text{ESK}_i^{r_0,1}), \dots, (\text{EPK}_i^{r_n,s_n}, (\text{ESK}_i^{r_n,s_n}) \tag{2.2}$$

The user i signs all his ephemeral public keys EPK as given in (2.2) with his participant secret key PSK. Then he deletes his PSK. In the selection process, he uses his ephemeral secret keys $\text{ESK}_i^{r,s}$ for signing and obtains the signature $\text{ESIG}_i^{r,s}$. After each signing, he deletes his ephemeral secret key $\text{ESK}_i^{r,s}$ for round r and step s. Furthermore, the user i uses his public keys $\text{EPK}_i^{r,s}$ and PPKs for the generated signature verification and propagates them at each round r and step s. This process prevents key corruption in the self-selection process.

2.2 Consensus Protocol

Algorand blockchain has Pure Proof of Stake consensus mechanism. It is based on the Byzantine Agreement protocol and the stake amount of the user. This protocol tolerates malicious users if the majority of the stake amount of honest players is greater than $\frac{2}{3}$. Algorand block generation is ensured by the voting process. Every user in Algorand checks if he is selected for the proposal. For the controls of the selection, the verifiable random function (VRF) is used by each node. After that, each selected node sends its block offer and the VRF proof to the other nodes. Next step, every participant executes the VRF again and checks if he is selected for the committee of verifiers. After that, the participants start voting for the proposed block until the amount of the proposed block reaches one. After the block voting is over, a new committee is rebuilt to check the selected block proposal for problems such as overspending, and double-spending. If all conditions are satisfied, the new block becomes active and inserted into the blockchain. The principle of consensus mechanism is given in Figures 2.2 and 2.3. During the user selection, any user doesn't know he is a verifier until he is selected. These attributes provide security against adversaries. The adversary cannot determine any committee member until after they include consensus. At this time, even if he can control and see the user's vote, the user has been propagate his message already. Therefore the adversary cannot manipulate the selection process. We will mention the details of selection and proposal operations in the next sections.

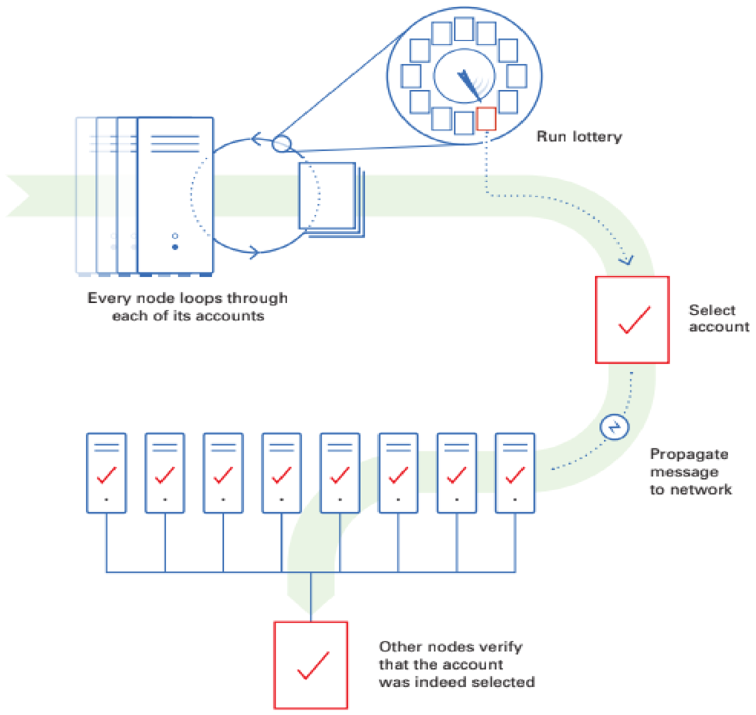
Run lottery
Every node loops through each of its accounts
Select account
Propagate message to network
Other nodes verify that the account was indeed selected

Figure 2.2: Block Proposal
Reference: https://developer.algorand.org/docs/get-details/algorand_consensus/

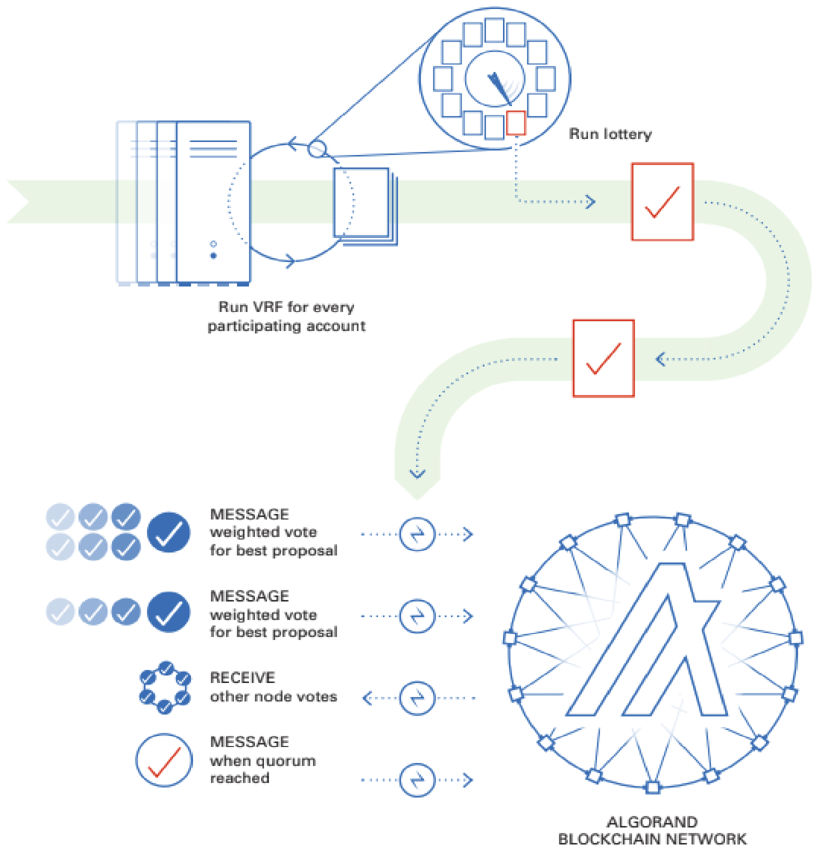
Run lottery
Run VRF for every participating account
MESSAGE weighted vote for best proposal
MESSAGE weighted vote for best proposal
RECEIVE other node votes
MESSAGE when quorum reached
ALGORAND BLOCKCHAIN NETWORK

Figure 2.3: Proposal Voting
Reference: https://developer.algorand.org/docs/get-details/algorand_consensus/

2.2.1 Verifiable Random Function

Verifiable Random Function (VRF) [12] is used for the voting and user selection in the rounds of the consensus mechanism. The user who has the private key can calculate the hash value that he wants to send. However, anyone with the public key can verify this hash. Once the users check if they selected for the voting, they use a new participant key instead of the standard spending private key. There are many variants of the VRF. These variants depend on cipher suits that are used in the protocol. Algorand prefers to ECVRF-EDWARDS25519-SHA512-ELL2 function. It is an elliptic curve based version of the protocol. It uses Ed25519 [4] as an elliptic curve and SHA-512 as a hash function. The VRF contains three main steps: Proof function, proof to hash function, verifying function.

Proof function: It takes input as private key SK and message m with the curve salt.

Where the SK is the private key and B is the generator of group, the calculation is in the following statements ;

1. Obtain the secret scalar value x from private key SK
2. Compute public key Y
$$Y = x \cdot B$$
3. With m and salt, obtain H and h:
$$H \xleftarrow[Ed25519]{encode} m$$
$$h \xleftarrow{string} H$$
4. Compute the point γ:
$$\gamma = x \cdot H$$
5. Compute the nonce value $k \in \mathbb{Z}^{+} : 1 \leq k \leq q - 1$
$$\begin{aligned} \text{H}' &= \text{H}'_0 \dots \text{H}'_{63} = \text{SHA-512(SK)} \\ \text{H}_k &= \text{SHA-512}(\text{H}'_{32} \dots \text{H}'_{63} \mid\mid h) \\ k &= \text{H}_k \mod q \end{aligned}$$
6. Compute the challenge value $c \in \mathbb{Z} : 0 \leq c \leq 2^{8L}$ where $L = 16$:
$$\begin{aligned} &P_1 = k \cdot B \\ &P_2 = k \cdot H \\ &\text{H}_c = \text{SHA-512}(3 \mid\mid 2 \mid\mid Y \mid\mid H \mid\mid \gamma \mid\mid 3 \mid\mid P_1 \mid\mid P2) \\ &\ c \leftarrow \text{H}_c \end{aligned}$$
7. Compute $s \in \mathbb{Z}_q$:
$$s = (k + c \cdot x) \mod q$$
8. Finally output string is obtained as in the following statement:
$$\text{output}_{proof} = \gamma \mid\mid c \mid\mid s \tag{2.3}$$

Proof to hash function: This function takes the output$_{proof}$ (2.3) as input. The algorithm steps are given below;

1. Invert the output$_{proof}$ string to $D = (\gamma, c, s)$.
    - If it could not invert to the D, it returns the invalid value and halts the function.
2. Otherwise, compute the output as:

$$\text{output}_{hash} = \text{SHA-512}(3 \mid\mid 3 \mid\mid 8 \cdot \gamma \mid\mid 0)$$

Verifying function: It takes inputs as public key string PK$_{str}$, output$_{proof}$, and the same message m and the salt in proof function mentioned above. The protocol steps are given below;

1. The point of public key Y is obtained with converting string PK to point.
    - If Y is not valid, it returns the invalid value and halts the function.
2. Otherwise, compute Y':

$$Y' = 8 \cdot Y$$

    - If Y' is the identity element of the curve, it returns the invalid value and halts the function.
3. Invert the output$_{proof}$ string to $D = (\gamma, c, s)$.
    - If it could not invert to the D, it returns invalid value and halts the function.
4. With m and salt, obtain H and h as:

$$H \xleftarrow[Ed25519]{encode} m$$
$$h \xleftarrow{string} H$$

5. Compute the challenge value $c' \in \mathbb{Z} : 0 \leq c' \leq 2^{8L}$ where $L = 16$:

$$U = s \cdot B - c \cdot Y$$
$$V = s \cdot H - c \cdot \gamma$$
$$\mathrm{H}_{c'} = \text{SHA-512}(3 \mid\mid 2 \mid\mid Y \mid\mid H \mid\mid \gamma \mid\mid 3 \mid\mid U \mid\mid V)$$
$$c' \leftarrow \mathrm{H}_{c'}$$

6. If $c = c'$ it returns $(\text{true}, \text{output}_{hash})$

According to [12], this protocol provides the security properties given below:

- Full Uniqueness: For public key PK, message m, and output_{proof_1}, output_{proof_2}

$$\text{VRF}_{verify}(\text{PK}, m, \text{output}_{proof_1}) = (\text{true, output}_{hash_1})$$
$$\text{VRF}_{verify}(\text{PK}, m, \text{output}_{proof_2}) = (\text{true, output}_{hash_2})$$
$$\text{output}_{hash_1} \neq \text{output}_{hash_2}$$

- Full Collision Resistance: For public key PK, different messages m_1, m_2, and output_{proof_1}, output_{proof_2}

$$\text{VRF}_{verify}(\text{PK}, m_1, \text{output}_{proof_1}) = (\text{true, output}_{hash_1})$$
$$\text{VRF}_{verify}(\text{PK}, m_2, \text{output}_{proof_2}) = (\text{true, output}_{hash_2})$$
$$\text{output}_{hash_1} \neq \text{output}_{hash_2}$$

- Pseudorandomness provides that, the output_{hash} is indistinguishable from a random value, when anyone knows an output_{hash} without a corresponding output_{proof} and secret key SK.

2.2.2 Selection of Committees:

Leader Selection: In the Algorand block proposition, A leader must exist to perform a round with the validators. The optimal block structure of the round-r is in the following statement:

$$B^r = (r, \text{PAY}^r, Q^r, H(B^{r-1}))$$

where r is round index, PAY^r is all valid pay-sets until round-r, Q^r is the round quantity, and $H(B^{r-1})$ is the hash value of the previous round block. Every online user $i \in$ PK^{r-k} calculates the value given below; Each user has common knowledge about status of chain $B^0, B^1, B^2, \ldots, B^{r-1}$ and set of public keys PK^1,PK$^2, \ldots,$PK^{r-1}, and each B^r contains Q^r.

- Each player has the signature SIG$_i(r, 1, Q^{r-1})$ which provides uniqueness.
- $H($SIG$_i(r, 1, Q^{r-1}))$ generates 32 byte unique value.
- $.H($SIG$_i(r, 1, Q^{r-1}))$ converts the hash value to decimal point form as $0.x$, where x is a numeric value of hash.

$$.H(\text{SIG}_i(r, 1, Q^{r-1})) \leq p$$

where p is close to high probability which can be denote as $1 - F$ and F is a negligible amount of probability such that 10^{-12}.

$$\{min(H(\text{SIG}_i(r, 1, Q^{r-1}))) \mid i = 0 \ldots n, H(\text{SIG}_i(r, 1, Q^{r-1})) \leq p\}$$

The potential round leader l^r is who has the minimum hash value among all of the players. However, the leader l^r must be inside the system for k rounds. The current leader learns that the player is a leader when this player is chosen by the system.

Validator Selection: The committee of validators is selected for the voting process. Every step of round-r is performed by a subset of validators SV^{r,s}, where step $s > 1$. As in leader selection part, the same calculation $H($SIG$_i(r, s, Q^{r-1}))$ is calculated by each user with the quantity of previous round Q^{r-1}. If the user is a validator then he must ensure the following condition similarly;

$$.H(\text{SIG}_i(r, s, Q^{r-1})) = p_{H_i} \leq p'$$

where $p_{H_i}, p' \in \mathbb{R} : [0, 1]$. The probability p' is selected as $1 - F$ and F is negligible amount of probability.

2.2.3 Byzantine Agreement Protocol:

Blockchain systems can be described as *Synchronous Networks* [22]. This type of network performs in the following way: At the time t, every sender i sends his message m simultaneous to the receiver j. This message can be shown as m^t_{ij}. And at next time $t+1$, the messages are received by receiver j's. In this type of network, the adversary can manipulate the message by controlling the malicious user. Therefore, the equality of sent message and received message is a major property.

Byzantine Agreement is the core property of Algorand blockchain. The agreement provides system protection against the adversary who can make malicious a user and control it to manipulate the message inputs or response outputs.

Definition: *Let's n is the number of total users/players, and t is the number of malicious users/players, where;*

$$n \geq 2t + 1$$

Let V be a finite domain. A protocol P among every player $i \in SV^{r,s}$ holds an input value $v_i \in V$ and finally decides on an output value $o_i \in V$ achieves consensus (or is a consensus protocol) with respect to $SV^{r,s}$ and V if it satisfies the following conditions:

1. *Validity: If all honest players i hold the same input value $v_i = out$ then all honest players i decide on it, $v'_i = out$*
2. *Agreement: All honest players decide on the same output value, if user $i \in SV^{r,s}$ and user $i \in SV^{r,s}$ are correct then $v_i = v_j$.*

Binary BA Protocols (BBA) Protocol has a requirement as a random string str in setup step. The rest of the protocol consists of a 3-part loop. Briefly, the players send their values to others. During this process, some players may exit at the some part of loop since, they do not satisfy the required conditions. And remaining players continue the value sharing.

Protocol Steps:

- γ_i is the counter the of player i's loop execution.

- $n \geq 3t + 1$
- lsb(x) implies least significant bit of value x.
- $\#_i^s(v)$ number of player, has received value v in step s from player i.
- If player i exits, he sends a special value $0*$ or $1*$.

1. Each player i sends b_i
   (a) If $\#_i^1(0) \geq 2t + 1$, then $b_i = 0$, send special value $0*$, $d_i = 0$, EXIT
   (b) If $\#_i^1(1) \geq 2t + 1$,then $b_i = 1$
   (c) Else $b_i = 0$
2. Each player i sends b_i
   (a) If $\#_i^2(1) \geq 2t + 1$, then $b_i = 1$, send special value $1*$, $d_i = 1$, EXIT
   (b) If $\#_i^2(0) \geq 2t + 1$,then $b_i = 0$
   (c) Else $b_i = 1$
3. Each player i sends b_i and his own signature as $SIG_i(\gamma_i, r)$
   (a) If $\#_i^3(0) \geq 2t + 1$, then $b_i = 0$
   (b) If $\#_i^3(1) \geq 2t + 1$,then $b_i = 1$
   (c) Else $b_i =$ lsb(*min*$(H($SIG$_i(\gamma_i, r))))$, increase γ_i and go to first step.

Note: min is the minimum of hash value of a player's signature value.

Graded Consensus Protocol (GC): As in Byzantine Agreement , n is the number of players/users and t is the number of malicious players/users. Each player has the input v_i' and the output pair (v_i, g_i), where $g \in G\{0, 1, 2\}$. And any user must provide the conditions given below;

- Each honest user i, j, outputs should be as $\mid g_i - g_j \mid \leq 1$.
- Each honest user i, j, if $g_i, g_j > 0$ then $v_i = v_j$.
- For each honest user i, input values $v_i = v$ value, then $g_i = 2$ for each user i.

Protocol Steps:

1. The each user $i \in \text{SV}^{r,2}$ sets $v'_i = H(B^r_l)$ if he confirms that he has the leader message m^r_l. Then he prepares $m^{r,2}_i = (\text{ESIG}_i(v'_i), \sigma^{r,2}_i)$ and propagates his $m^{r,2}_i$.

2. After each user $i \in \text{SV}^{r,3}$ receives the $m^{r,2}_i$, If $\#^2_i(m^{r,2}_i) \geq 2t + 1$,then each player prepares their $m^{r,3}_i = (\text{ESIG}_i(v'_i), \sigma^{r,3}_i)$ value and propagates it.

3. Decision Step: Computation of outputs pair as;

   - If $\#^3_i(m^{r,3}_i) \geq 2t + 1$, $v'_i = v_i$ and $g_i = 2$
   - If $\#^3_i(m^{r,3}_i) \geq t + 1$, $v'_i = v_i$ and $g_i = 1$
   - Else $v_i = H(B^r_\epsilon)$ and $g_i = 0$

   Then the user $i \in \text{SV}^{r,4}$ calculates the b_i, the input of BBA as $b_i = 0$ if $g_i = 2$, otherwise he set as $b_i = 1$. Finally each user i prepares their $m^{r,4}_i$ value and propagates it in the following statement:

$$m^{r,4}_i = (\text{ESIG}_i(b_i), \text{ESIG}_i(v'_i), \sigma^{r,4}_i)$$

BA Protocol: The BA protocol [9] is the actual protocol used inside Algorand. It consists of a combination of the steps of the protocols mentioned in Sections 3.2.2 and 3.2.3. The protocol steps are given below.

Protocol Steps:

- Every user $i \in \text{SV}^{r,s}$ performs Graded Consensus (GC) with own input vote v_i to calculate their output pair (v_i, g_i).

- Then all $i \in \text{SV}^{r,s+2}$ perform the Binary Byzantine Agreement (BBA) protocol, where (v_i, g_i) output pair, if $g > 0$ then the first inputs are equal to zero, otherwise the users calculate the output as following conditions.

  – For all user $i \in \text{SV}^{r,s+5}$, if output$_i = 0$, output$p_i = v_i$, else it equals to special symbol.

The BA protocol with the steps given above is one of the key mechanisms of Algorand's consensus protocol. To increase the benefits of this protocol, Algorand has developed the *faster and partition resilient Byzantine Agreement* [8] protocol with some performance tweaks.

2.2.4 Block Generation:

As mentioned in previous sections, the block structure of round-r is

$$B^r = (r, \mathrm{PAY}^r, Q^r, H(B^{r-1})).$$

Each user $i \in \mathrm{PK}^{r-k}$ computes the $.H(\mathrm{SIG}_i(r, 1, Q^{r-1}))$ and he checks the result of the computation is less than probability p. If the condition is satisfied than the users include the committee of potential leader $\mathrm{SV}^{r,1}$. After that, each user $i \in \mathrm{SV}^{r,1}$ generates his block B^r which includes set of all pay-sets $\mathrm{PAY}^{r-k}, \ldots, \mathrm{PAY}^r$, his credential $\sigma_i^{r,1} = \mathrm{SIG}_i(r, 1, Q^{r-1})$, and hash of the previous block B^{r-1}. After the generation of block, the user prepares his propagation message m_i^r as in following statement:

$$m_i^r = (B_i^r, \mathrm{SIG}_i(H(B_i^r)), \sigma_i^{r,1})$$

The other components of m_i^r are used as proof of the potential leader attributes. Finally the proposed blocks are propagated by each user $i \in \mathrm{SV}^{r,1}$. After that second step begins. The second step aims to decrease the number of proposed blocks to one. The each user $j \in \mathrm{SV}^{r,2}$, receives the propagated message m_i^r. Then, each user calculates the

$$\{min(H(\sigma_i^{r,1})) \mid i \in \{0, \ldots, n\}, H(\sigma_i^{r,1}) \leq p\}$$

for the $\sigma_i^{r,1}$ component of each received m_i^r. Then the user can find the credential $\sigma_l^{r,1}$ that implies the credential of the round leader l^r. As the last step, each user prepares his leader votes $H(B_{l^r}^r)_j$. And he checks the result of the computation is less than the probability p. After the leader selection, the leader's block is inserted into the ledger.

***Q* parameter:** The quantity of the rth block Q^r is constructed uniquely for each block as in the following statement:

$$Q^r = H(\mathrm{SIG}_i^{l^r}(Q^{r-1}), r)$$

2.3 Algorand Specific Structures

As we know, there are numerous blockchain systems. Among the blockchain systems, some of them have their smart contract system and some of them use the other blockchain's smart contract system. Algorand is one of the private smart contract owner. This contract named *Transaction Execution Approval Language (TEAL)*. In addition, Algorand has another private structure. It is named *Algorand Standart Asset*. It is exchangeable entity that using in the blockchain. The details of these structures are given in next section.

2.3.1 TEAL

TEAL is opcode-based language. It works with stack mechanism and push-pop operations. In Algorand, the smart contract is executed in *Algorand Virtual Machines (AVM)*. Once the node is configured, the AVMs automatically stand by as ready in the node machine. There are two types of smart contract programs for construction. The one is *Approval Program* and the other is *Clear Program*. The approval program provides deployment of all types of application calls. The clear program provides execution of all clear-remove calls. There are 6 types of application-call transactions. These are *NoOp, OptIn, DeleteApplication, UpdateApplication, CloseOut, ClearState* calls. Algorand smart contracts can also store values on the blockchain. The data can be stored as both global and local. Local storage refers to storing values in an account balance record if that account participates in the contract. When programs are called to the storage of the data, these data are given as argument parameters and processed by the defined program.

2.3.2 Algorand Standard Asset

Nowadays, token objects are widely used in many applications. Generally, tokens appear in applications as points, credits, or any spendable, usable, and exchangeable elements. It can also be used to define some rights or authorizations to users within the application. *Algorand Standart Asset* is a specially designed token structure. Besides, Algorand allows the users to create their assets provided they have sufficient balance in their account. The Algorand Standart Asset structure has some properties such as asset name, unit name, amounts, URL, etc. In addition, it has also management attributes. They are significant properties since the manager account, freezing authority account, refund, and reserve address are determined in this field. Moreover, all transactions that require authorization are sent from these accounts determined within the asset.

CHAPTER 3

DESIGN

In this study, we developed a healthcare survey application. This application consists of three main parts. We can summarize these three parts as follows: The first part is the web application of the project. The web server provides the connection with the database and constructs all transaction data with the Algorand SDK and Algorand API. In addition, the survey operations are creation, filling, and consent processed by the web application. During these operations, all request body data are converted to designed back-end class objects. The other one is Algorand blockchain side. On the Algorand side, all types of transaction data sent by the web server via Algorand API. Then the validity of the transactions is checked by Algorand [9]. In this case, if the sent data is valid then the data is committed to the blockchain. Moreover, on the blockchain side, we use the smart contract and asset technology provided by Algorand. We will mention the details of these technologies in the next sections. The last part is the database module. For each generated survey, we create a new executable decentralized application with a unique ID. With this unique ID, we create a new register in the database and map this register with the smart contract ID. In addition, the survey data such as questions, options, descriptions, etc. are stored in the database. Most of time, the database operations are quicker than backend data operations. Thanks to the cryptographic library methods, database encryption is faster than back-end encryption. Hence, we use a database for sending survey data more quickly to the patient side instead of back-end operations. The overview of general structure is given in Figure 3.1. We will discuss the parts we have mentioned in more detail in the next sections.

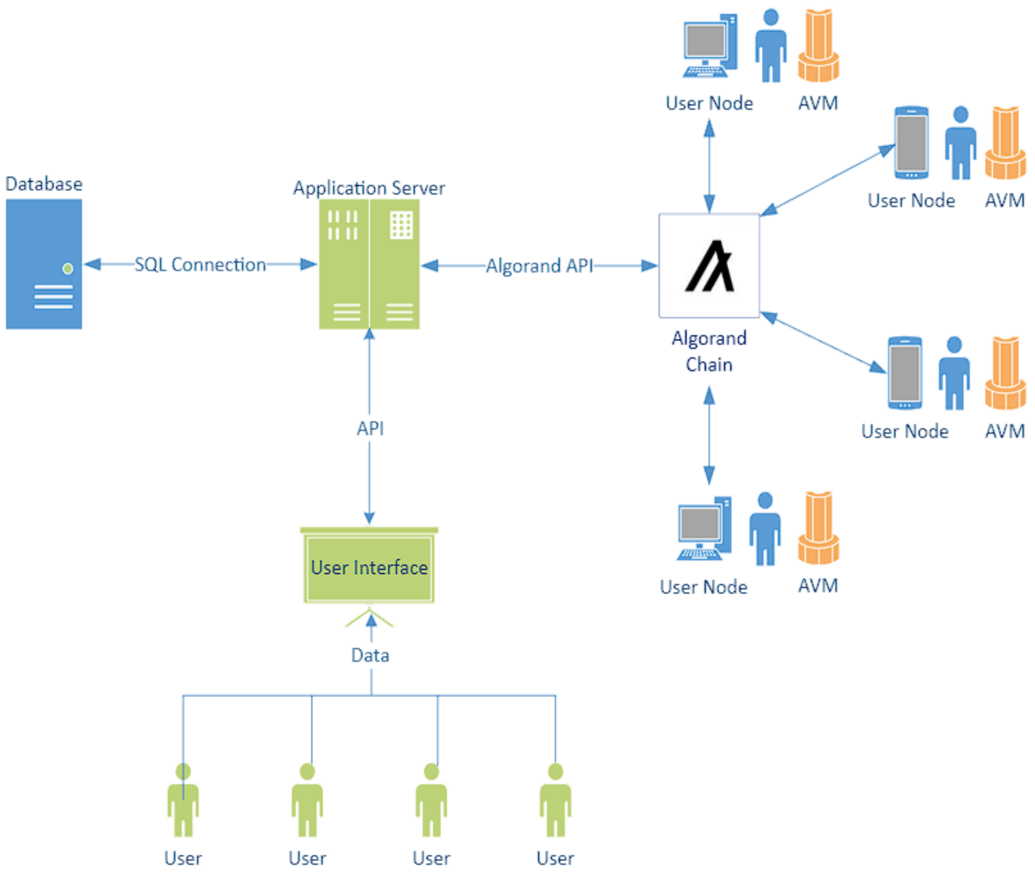

Figure 3.1: General Structure

3.1 Blockchain Side

In our design, we use the Algorand API for providing the connection between the application and the Algorand blockchain. There are two main structures of Algorand that we use in our solution. The first one is the smart contract structure. The other one is Algorand Standard Asset. The details of these structures are mentioned in Chapter 2. We will below mention the details of techniques in our design.

3.1.1 Components

All the structures mentioned above can be expressed as transaction structures. These transactions consist of several components. These components are used singularly or in a combination. There are three main components of Algorand in used in our application. Transaction component is common component of our asset and smart contract transaction. According to the usage situation, we use either the asset or smart contract component together with the transaction component. The structures of these components are given below:

Transaction Structure:

$$\texttt{Tx} = (\texttt{snd}, \texttt{gen}^*, \texttt{fv}^*, \texttt{lv}^*, \text{typ}, \text{fee}^*) \tag{3.1}$$

Asset Structure:

$$\mathcal{A} = (\texttt{tl}, \texttt{dc}, \texttt{un}, \texttt{an}, \texttt{df}, \texttt{man}_{\texttt{addr}}) \tag{3.2}$$

Smart Contract Structure:

$$\mathcal{C} = (\texttt{apid}, \texttt{apap}, \texttt{apas}, \texttt{apgs}, \texttt{apls}, \texttt{apaa}) \tag{3.3}$$

Equation 3.1 is common transaction structure. Equations (3.2) and (3.3) are asset and smart contract structures respectively. The empty or unused parameters are not displayed in the given equations above. The parameters that are marked with (*) are filled as default by Algorand. The generalized smart contract and asset transaction structure can be represented respectively as in the following statements:

$$\texttt{Tx}_{\mathcal{C}} = (\texttt{Tx}, \mathcal{C})$$
$$\texttt{Tx}_{\mathcal{A}} = (\texttt{Tx}, \mathcal{A})$$

The input values of the transactions depends to usage. They take different types of values according to their use cases and their behavior changes accordingly. These values are mentioned in the following sections.

3.1.2 Smart Contract

In our application, we generate a TEAL code for our smart contract. Then we use the smart contract in survey operations such as survey filling, survey creating, and consent form approval. In order to use and create smart contracts, we generate the transaction structure with the contract-specific parameters and send these transactions to the blockchain. The principle of the smart contract mechanism is given as in Figure 3.2:

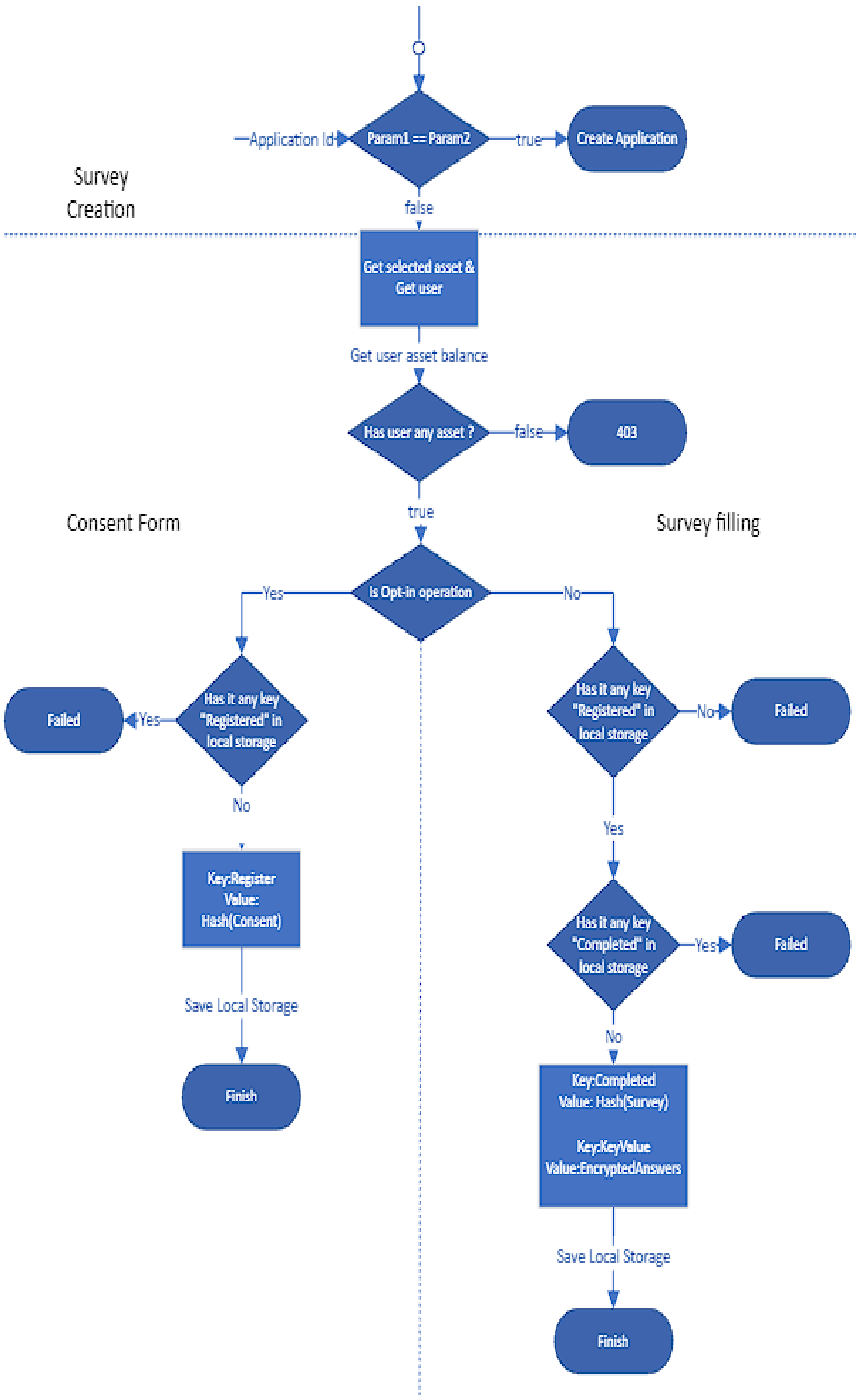
Survey Creation
Application Id
Param1 == Param2
true
Create Application
false
Get selected asset & Get user
Get user asset balance
Has user any asset ?
false
403
true
Consent Form
Survey filling
Is Opt-in operation
Yes
No
Has it any key "Registered" in local storage
Yes
Failed
No
Key:Register Value: Hash(Consent)
Save Local Storage
Finish
Has it any key "Registered" in local storage
No
Failed
Yes
Has it any key "Completed" in local storage
Yes
Failed
No
Key:Completed Value: Hash(Survey)
Key:KeyValue Value:EncryptedAnswers
Save Local Storage
Finish

Figure 3.2: Smart Contract Mechanism

Once the smart contract is executed by any application transaction, firstly the smart contract checks the application transaction type. If application id equals zero it implies the new application will be generated. This control is used for the survey creation step. If it is not equal to zero then there is an application already. After that controls, the smart contract checks if the user has authorization. To achieve this control, it checks the MST balance of the user. It is successful if the user has not zero balance of MST. Otherwise, the smart contract finishes itself for this user. This control is used in both consent forms and survey filling operations. At this point, the smart contract is divided into two branches. The one is for the consent form approval. The other one is for survey filling. When the consent form is filled, the smart contract is called. And once more, the previous steps are checked. If all conditions are satisfied, first, the application examines the type of smart contract invoked. If it is an opt-in type, it provides a check on whether or not it has registered before. If provided, it takes the argument sent for the consent form and stores it as key-value pairs in the user's local storage of the smart contract. Otherwise, the smart contract broke the agreement. As a result of the canceling, the transaction fails. In the survey filling case, the smart contract is called again and rechecks the properties that must satisfy. If all conditions are valid then the smart contract checks two parameters. The first is checking whether the user is registered or not. For the second, the existence of previously recorded survey data is checked. If the contract couldn't find any data about registration or it finds any stored survey data, it breaks the agreement and the transaction will fail again. Otherwise, the contract takes the argument sent for the hashed survey data and encrypted survey answers, then stores them as key-value pairs in the user's local storage of the smart contract as in the consent form process.

3.1.3 Medical Security Token

As mentioned in Section 2.3.2, Algorand has a primitive transaction unit called Algorand Standard Asset, which is similar to its cryptocurrency. In addition, This asset structure can be used for many reasons as units such as points, credits, assets, coins, or tokens. Therefore, we have benefited from this Algorand asset feature for the authorization among the nodes. Unregulated smart contracts can be called and used by the users who know the generated application id. Due to this unauthorized structure, public participation may pose a problem for security-based applications. Therefore, we attempted to prevent this public structure with the token system. This security token is named Medical Security Token *MST*.

In the use case, we assume that our participant patients are determined by the authority. Firstly we already have generated the asset with the given parameters in (3.2) and (3.1) as in the following statements. In this process (3.4), we decide the management accounts PKs and pass the address value to the asset structure in (3.2). The owner of the manager account can be a doctor, a system administrator, or both of them in different fields. In addition, we specified the total amount, freeze, unit, name and decimal properties as 1000, false (0), *MST*, *Medical Security Token* and 0 respectively.

$$\mathcal{A}_{MST} = (1000, 0, \text{MST}, \text{Medical Security Token}, *, \text{PK}) \tag{3.4}$$

Then, in (3.5), we specified the transaction sender and transaction type as authority's address PK and type enumeration index of Asset Configuration Transaction, respectively. The remaining $\texttt{Tx}$ parameters are filled by Algorand.

$$\texttt{Tx} = (\texttt{snd}, \text{typ}, \texttt{gen}^*, \texttt{fv}^*, \texttt{lv}^*, \text{fee}^*) = (\text{PK}, 3, *, *, *, *) \tag{3.5}$$

After that, we generate the transaction

$$\texttt{Tx}_{asset} = (\texttt{Tx}, \mathcal{A}_{MST}).$$

Finally we sign the $\texttt{Tx}_{asset}$ by EdDSA and we obtain signed transaction:

$$\texttt{Tx}_{\texttt{SIG}} = (\text{SIG}(\texttt{Tx}_{asset}), \texttt{Tx}_{asset})$$

After the asset is generated, we obtained an asset id $\mathcal{A}_{id}$ and we will use this id for all asset transactions. Next step, we send 1 of these assets to each Algorand account declared as patients. We pass the transaction sender, and type parameters as doctor's address PK and type enumeration index of Asset Transfer Transaction, respectively as $\text{Tx} = (\text{PK}, 4, *, *, *, *)$. The remaining parameters are filled in a similar way. Furthermore,asset id, asset receiver and asset amount with the generated transaction are given below:

$$\texttt{Tx}_{axfer} = (\mathcal{A}_{id}, \texttt{arcv}, \texttt{aamt}, \texttt{Tx})$$

Finally we sign the transaction with EdDSA and send it to the specified asset receiver as in the following form:

$$\mathtt{Tx}_{\mathtt{SIG}} = (\mathrm{SIG}(\mathtt{Tx}_{axfer}), \mathtt{Tx}_{axfer})$$

After the token transfer, we immediately freeze the balances of relevant asset that associated with the their accounts. Except for the transaction type parameter, we pass the same parameter to the asset transfer transaction as in (3.6).

$$\mathtt{Tx} = (\mathtt{snd}, \mathrm{typ}, \mathtt{gen}^*, \mathtt{fv}^*, \mathtt{lv}^*, \mathrm{fee}^*) = (\mathrm{PK}, 5, *, *, *, *) \tag{3.6}$$

We specified asset id, asset receiver and asset amount as in transfer transaction. In addition, we set the freeze status and give as parameter for the freezing operation as in

$$\mathtt{Tx}_{afrz} = (\mathcal{A}_{id}, \mathtt{arcv}, \mathtt{aamt}, \mathtt{afrz}, \mathtt{Tx}).$$

As in the previous operations, we sign the transaction again with EdDSA and send it to the specified asset receiver as follows:

$$\mathtt{Tx}_{\mathtt{SIG}} = (\mathrm{SIG}(\mathtt{Tx}_{axfer}), \mathtt{Tx}_{axfer})$$

The reason for the freezing process is preventing the right to fill out this survey defined for the patients from being sent to others or used for other purposes. For the user side, once a user request to reach a survey, the smart contract checks the asset balance and sends a response about whether the user has authentication or not. According to the contract response, the participant's authority to participate in the survey is determined. As a result, the access control mechanism is satisfied with the token mechanism. The principle of token operations in given Figure 3.3.

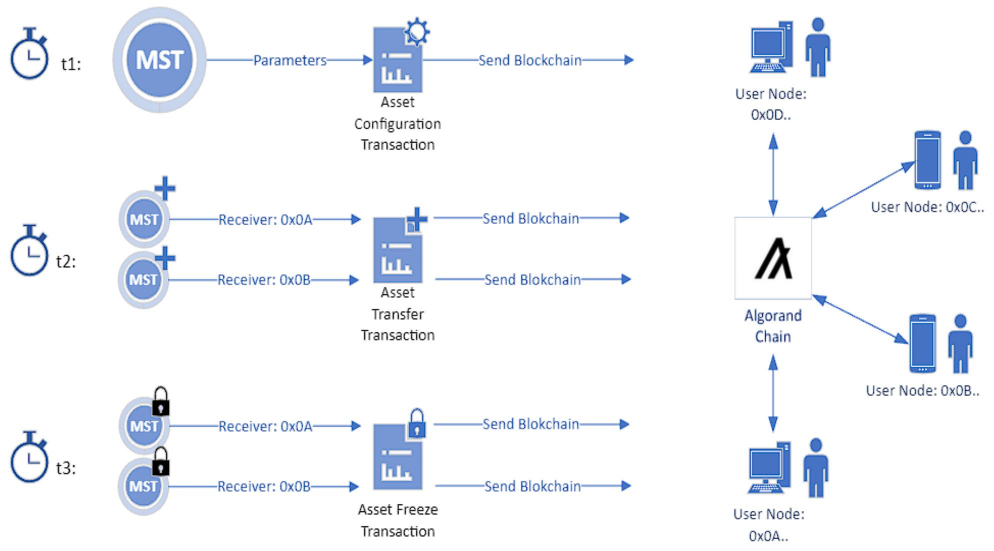

Figure 3.3: Overview of Token Transactions

3.2 Application Side

In our design, this part contains the application operations and methods. The users interact with the application side via the designed web page. The users construct their requests and the server side catches the incoming request and then processes it. The application side processes three main requests survey creation, consent form, and survey filling. We will explain the details of these operations in next sections.

3.2.1 Survey Creation

Once we start creating a survey, both application and blockchain sides work in sync. The most significant element of this step is the construction of smart contracts and making them an acceptable and suitable agreement for the blockchain. As we introduced at the beginning of this section, each survey creation triggers the new decentralized application with a unique ID in the Algorand blockchain. The details of the survey creation step as in the following statements;

In our study, we assume that doctor is the creator of the survey. Therefore he prepares

Algorithm 1 Survey Creation

```
    Input: Survey JSON data,mnemonic
 1: data ← FormatToSurvey(Survey JSON data)
 2: $PK$ ← GetAccount(mnemonic)
 3: $\texttt{Tx}_{create}$ ← UnsignedContractTransaction($PK$)
 4: $\texttt{Tx}_{\texttt{SIG}}$ ← (EdDSA($\texttt{Tx}_{create}$),$\texttt{Tx}_{create}$)
 5: isSuccess, dAppId ← SendToBlockchain($\texttt{Tx}_{\texttt{SIG}}$)
 6: if isSuccess then
 7:     SaveSurvey(dAppId,Encryption(data))
 8: else
 9:     return err
10: end if
```

the survey and sends it to the application side with his mnemonic key. In this step, the system requires survey data in JSON format. Once the survey data reaches to application, it is adapted to the designed object class for processing. Then, the contract transaction creation process is performed. This transaction is named Application Call Transaction. It is denoted as $\texttt{Tx}_{create}$. The sender's account information is retrieved with the mnemonic key. After that, the user address(public key) PK and its information are obtained by the retrieved account. TEAL smart contract program is set to `apap` parameter. Our MST token id is set to `apas`. The global storage and local storage values are set to `apgs`, `apls` respectively. The smart contract component generate as in (3.3). Finally, the contract transaction $\texttt{Tx}_{create}$ is generated as in the following statement:

$$\texttt{Tx}_{create} = (\texttt{Tx}, \mathcal{C})$$

Once the contract transaction is generated, it is signed with EdDSA signature algorithm [13] on the elliptic curve Ed25519 [4] as in the following statement:

$$\texttt{Tx}_{\texttt{SIG}} = (\texttt{SIG}(\texttt{Tx}_{create}), \texttt{Tx}_{create}).$$

The signed contract transaction $\texttt{Tx}_{\texttt{SIG}}$ is sent to the blockchain. After validation of the blockchain, if there is not an invalid parameter or operation request, the transaction is committed to Algorand. And our smart contract is executed in the AVM. Otherwise, the transaction fails. In the successful case, we adapt the survey data to the data transfer object. After that, we save the survey information to our database. During registration in the database, we encrypt the survey data with the AES algorithm with the CBC mode. In addition, PKCS is used for padding operations. Then we store the encrypted data in the database. Due to the efficiency of database insert and encryption functions, we prefer database-side encryption for data confidentiality. The flow chart of survey creation is given in Figure 3.4.

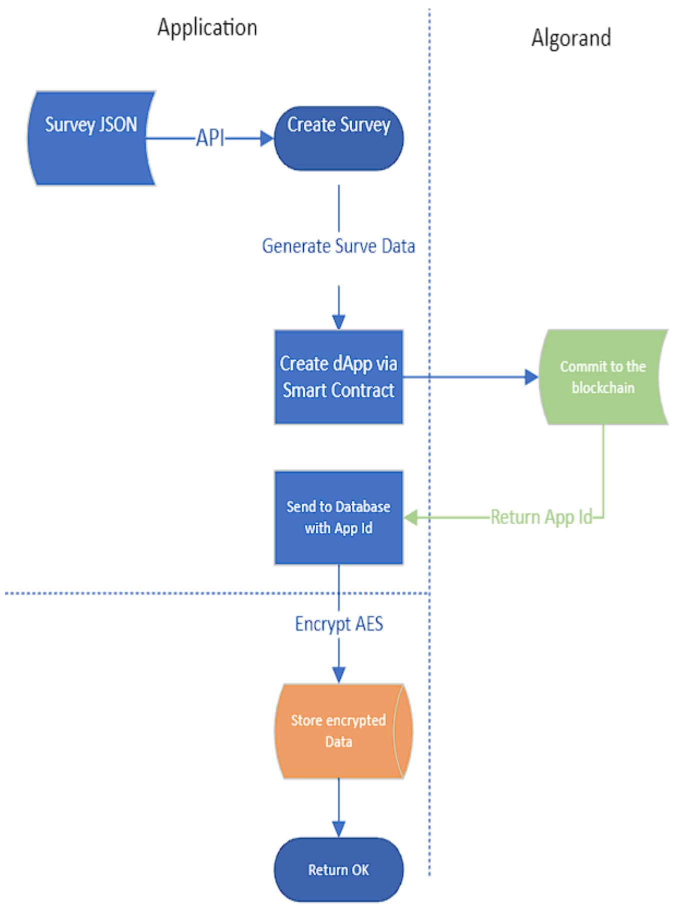

Figure 3.4: Survey Creation Steps

3.2.2 Consent Form

Medical surveys are widely used in many areas of healthcare. These surveys can be done incrementally, once, continuously, or both before and after some medical operations. The answers or the results of the survey might contain sensitive data. Due to the principle of doctor-patient confidentiality, both doctors and patients avoid the disclosure of this data. Moreover, some tests might have unexpected consequences. Therefore the consent form is filled out by the patients for the data confidentiality agreements and a disclaimer for unpredictable results. As a consequence of these requirements in the medical surveys, we construct a consent form structure. Since the consent form is an agreement between the doctor and the patient, we store this consent form on the blockchain side for immutable proof of agreement. Algorithm 2 explains the generation of transactions and the execution of the smart contract mechanism for the consent form.

Algorithm 2 Consent Form Approval

```
    Input: Consent data(C), dAppId
 1: if Accept then
 2:     $H(C) \leftarrow$ SHA-256(C)
 3:     $PK \leftarrow$ GetAccount(mnemonic)
 4:     $\mathtt{Tx}_{optin} \leftarrow$ UnsignedContractTransaction($PK$,dAppId,$H(C)$)
 5:     $\mathtt{Tx}_{\mathtt{SIG}} \leftarrow$ (EdDSA($\mathtt{Tx}_{optin}$),$\mathtt{Tx}_{optin}$)
 6:     isSuccess $\leftarrow$ SendToBlockchain($\mathtt{Tx}_{\mathtt{SIG}}$)
 7:     if isSuccess then
 8:         MST $\leftarrow$ Smart Contract: CheckMST($PK$)
 9:         if MST then
10:             Smart Contract: LocalStore($H(C)$)
11:         else
12:             Smart Contract: return 0
13:         end if
14:     else
15:         return err
16:     end if
17: else
18:     return 403
19: end if
```

Once the participant requests to fill out the determined survey, the consent form first appears to the patients before the survey has been sent. The participant reads the terms and conditions. If the patient declines the terms and conditions, he redirects to the main page, and the survey is canceled until he accepts the consent form. Once he confirms the consent form, the consent data is constructed with the server-side functions and generate the designed object class. Then the hash value of the consent data $H(C)$ is generated via the SHA-256 hash function. After this step, the contract transaction generation process begins. This contract transaction is named Application Opt-in Transaction. Additionally, in this step, we set the `apid` field with decentralized application ID and `apaa` field with $H(C)$. $\texttt{Tx}_{optin}$ constructed as in the following statement:

$$\texttt{Tx}_{optin} = (\texttt{Tx}, \mathcal{C}(\texttt{apid}, \texttt{apaa}))$$

It provides the registering of the blockchain users to the generated decentralized application. Also, it declares local storage access on the generated decentralized application. The opt-in transaction preparation is similar to the transaction preparations we mentioned in the survey creation. After that, the contract transaction is signed with the EdDSA[13] with the elliptic curve Ed25519[4] as in the following statement:

$$\texttt{Tx}_{\texttt{SIG}} = (\texttt{SIG}(\texttt{Tx}_{optin}), \texttt{Tx}_{optin})$$

Once, the transaction is sent to Algorand [9] blockchain, our smart contract is executed unless there is no violation in the transaction. The first thing the smart contract checks is the sender has access to register the generated decentralized application. Smart contract checks the sender has our security token MST. Due to satisfy this requirement, The smart contract retrieves the asset balance specified in the contract variable. If the sender has our security token asset, he can opt-in to the smart contract and saves the proof of consent data as fixed key-value pair in his local storage at generated application. However, once the smart contract could not find any security token asset in his balance, that implies the sender is not authorized by any authority on our side. In this case, it returns the 403 response to the sender. The flow chart of consent operation is given in Figure 3.5.

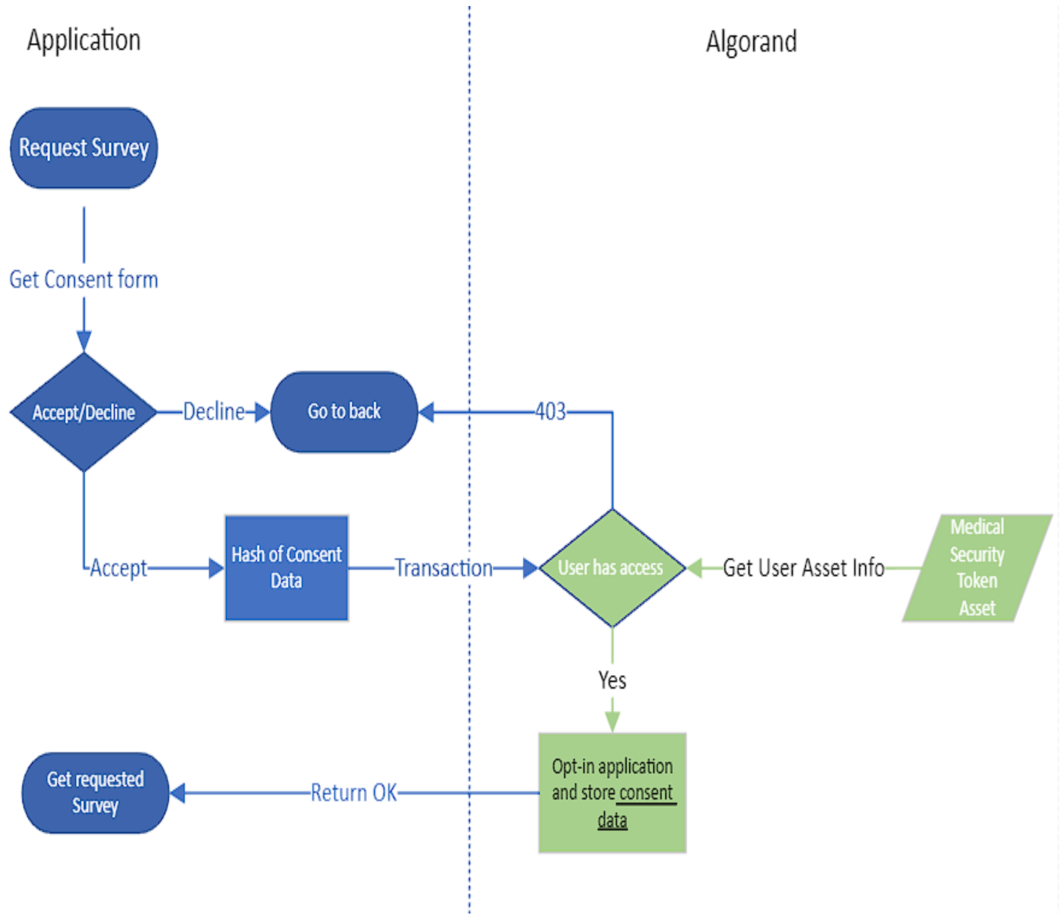

Figure 3.5: Consent Form Steps

3.2.3 Completion of the Survey

The survey completion is the key component of our solution. The participants who accepted the terms and conditions in the consent form has the authority to answer the questions in the survey.In the previous step, once the user requests the survey, he must confirm the consent form. After gaining access, he can attain the survey and starts filling the survey. The survey data is gathered from user answers then data is formatted for the specified survey object structure. At this part, the application is divided into two branches. The first one is taking the hash value of the survey data with the SHA-256 hash algorithm. The aim of this operation is in the following statements:

1. It provides a unique proof of user's answers.
2. It fixes the size of survey data since the size of the survey data depends on the

number of questions and options.

The other one is formatting the user's answers. Due to the size of the survey data can be expanded to a large amount, and formatting the survey answer is necessary for memory gain. We format each answer o for each question q as in the following statement:

$$q_i : o_j \mid\mid q_{i+1} : o_j \mid\mid \ldots \mid\mid q_n : o_j,$$

where n is amount of question, $\{n : n \in \mathbb{Z}^+\}$, i is index of the question, $\{i \mid i < n : i \in \mathbb{Z}^+\}$ and j is index of the selected answer of the question, $\{j \mid j < m : j, m \in \mathbb{Z}^+\}$. Each q_i and o_j is size of 1 byte. The index strings are concatenated with comma character $(,)$ and separated with colon character $(:)$. After the formatting, we encrypt the formatted data to ensure the confidentiality of user answers with the AES-256 algorithm with the CBC mode. Furthermore, we used the password-based key derivation function PBKDF2 [15] for a secret key generation with the following parameters:

$$\text{Derivated Key} = \text{PBKDF2(HMAC, SP, SHA-256}, s, \texttt{ctr})$$

where SP is secret password, `ctr` is iteration count. According to NIST's suggestion [21], the length of the salt should be 128 bit and the iteration count should be between 10^3 and 10^7. In addition, According to OWASP [17], the iteration count should be $310,000$ for the HMAC-SHA-256 usage. We determined the iteration count as $\texttt{ctr} = 2^{16}$.

The aim of this operation, the answer of completed survey data should store in the blockchain since the answers can be used for the survey evaluation later. Since Algorand is a public blockchain, the confidentiality of the answers data must be ensured. As in previous sections, we begin to generate transaction data with the parameters. After the formatting is finished, the transaction generation is started. We construct an Application Call Transaction. Additionally, in this part, we set the application argument field `apaa` with the encrypted and hashed values. Then we set the application id `apid` field with the generated contract id. $\texttt{Tx}_{call}$ is generated as the following statement:

$$\texttt{Tx}_{call} = (\texttt{Tx}, \mathcal{C}(\texttt{apid}, \texttt{apaa}))$$

This contract transaction is the default usage of the generated smart contract. After that, the contract transaction is signed with the EdDSA [13] on the elliptic curve Ed25519 [4] as in survey creation step.

$$\mathtt{Tx}_{\mathtt{SIG}} = (\mathtt{SIG}(\mathtt{Tx}_{call}), \mathtt{Tx}_{call})$$

In the final step, the smart contract is executed unless there is no violation of the transaction structure. The mechanism of the contract is given as in the following statement:

1. It checks user has any MST for the access
2. If he has any MST, then it checks user registry status
3. If it is satisfied then it stores the hash and encrypted data as key-value pair separately.

After the smart contract finishes itself, the whole process is completed. The overview of the survey filling operation is given in Figure 3.6:

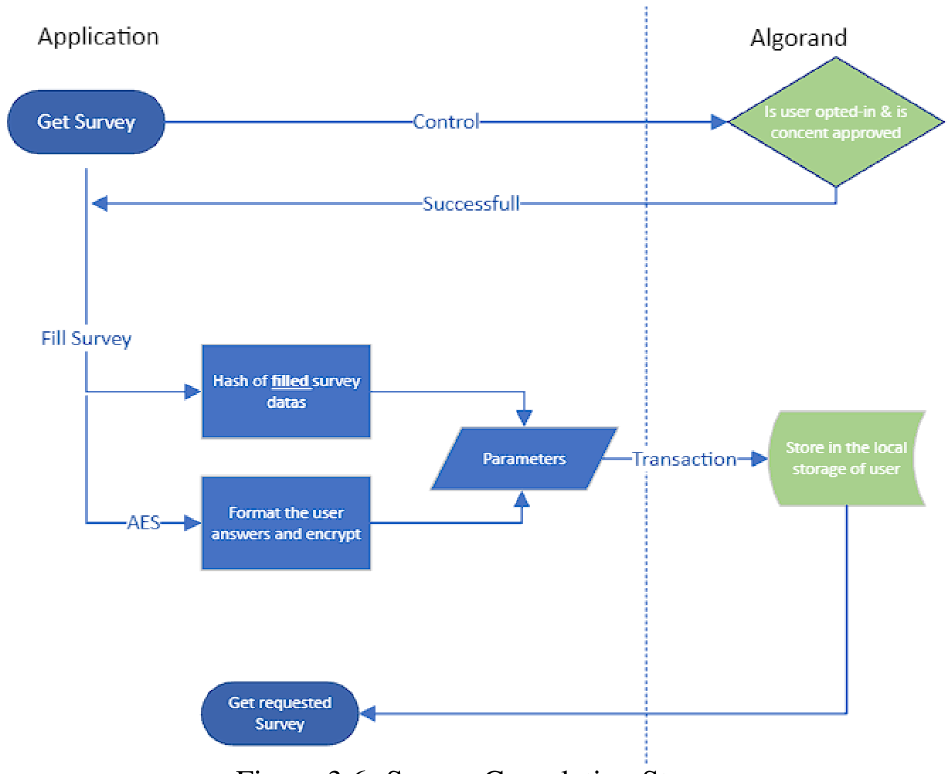
Application
Algorand
Get Survey
Control
Is user opted-in & is concent approved
Successfull
Fill Survey
Hash of filled survey datas
Parameters
Transaction
Store in the local storage of user
AES
Format the user answers and encrypt
Get requested Survey

Figure 3.6: Survey Completion Steps

Algorithm 3 Save Survey

```
Input: Answered Survey Data(D), dAppId
1: $S \leftarrow$ adapt(D)
2: $H(S) \leftarrow$ SHA-256($S$)
3: $S_f \leftarrow$ format($S$)
4: $ENC(S_f) \leftarrow$ Encryption($S_f$)
5: $PK \leftarrow$ GetAccount(mnemonic)
6: $\texttt{Tx}_{call} \leftarrow$ UnsignedContractTransaction($PK$, dAppId, $H(S)$, $ENC(S_f)$ )
7: $\texttt{Tx}_{\texttt{SIG}} \leftarrow$ (EdDSA($\texttt{Tx}_{call}$),$\texttt{Tx}_{call}$)
8: isSuccess $\leftarrow$ SendToBlockchain($\texttt{Tx}_{\texttt{SIG}}$)
9: if isSuccess then
10:     MST $\leftarrow$ Smart Contract: checkMST($PK$)
11:     if MST then
12:         isRegistered $\leftarrow$ Smart Contract: checkRegistry($PK$)
13:         if isRegistered then
14:             Smart Contract: LocalStore($H(S)$, $ENC(S_f)$)
15:         else
16:             Smart Contract: return 0
17:         end if
18:     else
19:         Smart Contract: return 0
20:     end if
21: else
22:     return err
23: end if
```

3.3 Database Side

As mentioned in the previous section, we store the generated survey data in the database. These data contain only the survey name, survey description, questions, and options. In addition, we encrypt the mentioned survey data with the AES-CBC mode with cryptographic functions of the pgcrypt library. We designed our database structure to be simple in the first phase. The diagram of the database is given below:

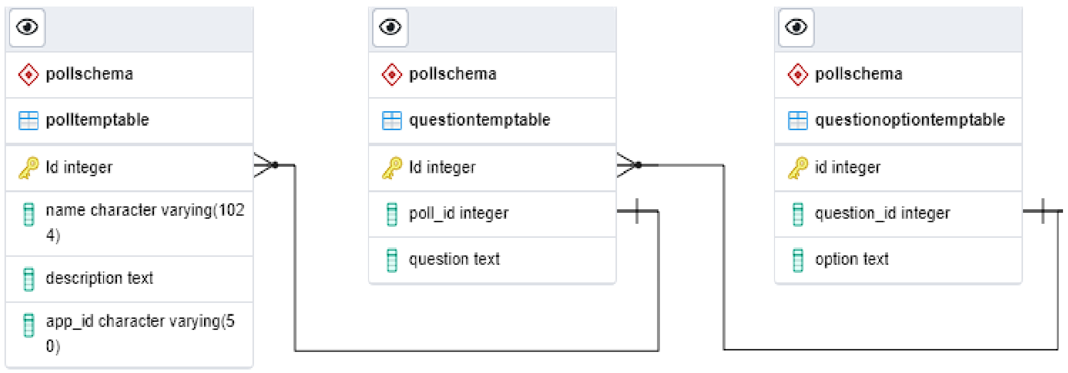

Figure 3.7: Database Diagram

According to Figure 3.7, the structure can be explained as the following statements: We store the generated survey application with its unique id, name, description, and related decentralized application id in *polltemptable*. Each question is stored with its unique id, text, and corresponding survey id in *questiontemptable*. In a similar way, each option is stored with a related question id in *questionoptiontemptable*.

3.4 Technical Properties

The technologies have been used in this project given below:

- **Frontend:** React.js
  - It is used for designing a primitive web page.
- **Backend:** Spring Boot / Java
  - It is used for providing interaction with the user and Algorand side via application API and converting the user data into transaction data.

- **Server:** Apache Tomcat
    - It is used for ensuring HTTP service for executing the application.
- **Database:** PostgreSQL
- **Blockchain:** Algorand Testnet
- **Other Technologies:**
    - **Pgcrypt:** PostgreSQL cryptographic plug-in
    - **TEAL:** Algorand smart contract language
    - **Algorand API:** The API ensures the connectivity between Algorand and Application.
    - **Algorand Standard Asset:** It is used for the security token mechanism

In a general scenario, the frontend side takes input and waits for any submission or request. In case of any request, the server-side API catches this request and processes the data according to the structure we have developed and it returns output data. After data processing, constructed data which is a smart contract transaction or asset transaction is sent to Algorand Testnet with the Algorand API service. Besides, after the transactions and data generation process, the constructed data is encrypted and stored in the PostgreSQL database.

3.5 Performance of Design

In this section, we shows the performance analysis of the used methods and key operations. Our system properties and experiment constants are given below:

- **Processor**: AMD Ryzen 5 3600 6-Core 3.6 GHz L1 Cache: 128 KB L2 Cache: 3 MB
- **RAM:** Memory: 16 GB Read: 3200 MHz Write: 1700 MHz
- **OS:** Windows 10 64-bit
- **Network:** Algorand Testnet

The sample survey has 10 questions and each question has 4 options. Therefore, the size of our formatted answer is 39 bytes. Database operations has fixed size. Therefore, all value columns given in the memory table have same result.

Table 3.1: Time Analysis

Functions	**Time**			
	Median	**Average**	**Min**	**Max**
PKDF2	17ms	17.87ms	15ms	78ms
AES-CBC	0.119ms	0.142 ms	0.084 ms	2.191 ms
SHA-256	0.096 ms	0.155 ms	0.085 ms	1.110 ms
Database Insert(w/AES)	0.031 ms	0.028 ms	0.009 ms	1.05 ms
Database Select(w/AES)	0.016 ms	0.018 ms	0.015 ms	0.156 ms
Signing Transaction	0.499 ms	0.968ms	0.410 ms	4.871 ms
Asset Transaction	0,082 s	0.083 s	0.076 s	0.088 s
Smart Contract Transaction	0.08656 s	0.08623 s	0.075 s	0.093 s

Table 3.2: Memory Analysis

Functions	**Time**			
	Median	**Average**	**Min**	**Max**
PKDF2	2.506 MB	3.085 MB	2.476 MB	4.946 MB
AES-CBC	1.4382 MB	1.4398 MB	1.4293 MB	1.5891 MB
SHA-256	0.9946 MB	0.9954 MB	0.9916 MB	1.0325 MB
Signing Transaction	26.31 MB	26 MB	24.87 MB	26.32 MB
Asset Transaction	26.35 MB	26.05 MB	24.91 MB	26.42 MB
Smart Contract Transaction	25.30 MB	26.53 MB	25.05 MB	28.82 MB

The difference between the maximum and minimum values is due to the fact that the data is kept in the application cache after a certain moment.

As we mention in the related works section, Rajput et.al [19] proposed a Hyperledger based system. In this system, average response time of the transaction is 5969 ms with the 319 B memory usage. And their total operation time is between 15 seconds and 18 seconds approximately. In addition, Li et.al [14] proposed an Ethereum based data preservation system. They have evaluated their performance with total operation time. They have used 50 B ,300 B sized texts and 5 MB, 30 MB sized files in their measurement. As a result, the lower bound is measured as less than 200 ms and the upper bound is measured as greater than 900 ms for the certain number of concurrent operation. In our system, the results are given in tables 3.1, 3.2. In our measurements, we haven’t included the blockchain conformation time.

CHAPTER 4

CONCLUSION

In this paper, we proposed a blockchain-based medical survey system as a solution for preventing data forging by the survey creators. While implementing this application, we used the Algorand blockchain and its smart contract language, TEAL. We designed it as a system where users can access the open surveys and participate in these surveys. Moreover, their answers cannot be changed by anonymous or 3rd party people with smart contract control. At the same time, to ensure the confidentiality of the patient's data, we kept the hashed value of the information we collected from the patient's survey data as evidence and the encrypted version so that data can be used for the evaluation, in the relevant storage units of the generated decentralized application.

4.1 Future Works

In this section, we mention the improvements we can try to put into practice in the future. Firstly, we insert a result evaluation page with the security and authorization mechanism. In addition, we provide pseudo-anonymity in the application. We will try to convert pseudo-anonymity to anonymity in the transaction. We may try zero-knowledge proof-based authentication for the general authorization and access control mechanism in the application and generated smart contract mechanism. As we mentioned in previous sections, we used Algorand blockchain for our solution. We will try to modify some basic cryptographic structures of the Algorand blockchain such as key generation and cryptographic signature algorithm or instead of that, we can try to construct our blockchain structure with a certain consensus mechanism and some basic cryptographic primitives.

REFERENCES

[1] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolic, S. W. Cocco, and J. Yellick, Hyperledger fabric: A distributed operating system for permissioned blockchains, CoRR, abs/1801.10228, 2018.

[2] M. Araoz, D. Brener, F. Giordano, S. Palladino, T. Paivinen, A. Gozzi, and F. Zeoli, ZeppelinOS: An open-source, decentralized platform of tools and services on top of the EVM to develop and manage smart contract applications securely, Technical Report 0.1.2, November 2017.

[3] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, Medrec: Using blockchain for medical data access and permission management, in *2016 2nd International Conference on Open and Big Data (OBD)*, pp. 25–30, 2016.

[4] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, High-speed high-security signatures, Journal of Cryptographic Engineering, 2(2), pp. 77–89, Sep 2012, ISSN 2190-8516.

[5] V. Buterin et al., Ethereum white paper, GitHub repository, 1, pp. 22–23, 2013.

[6] V. Buterin et al., A next-generation smart contract and decentralized application platform, white paper, 3(37), 2014.

[7] K. Celik and O. Yayla, Blockchain Based Solution For Electronic Health Record Integrity, 9 2022, https://github.com/kaan-celik/Blockchain-Based-Solution-For-Electronic-Health-Record-Integrity.

[8] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos, Algorand agreement: Super fast and partition resilient byzantine agreement, Cryptology ePrint Archive, Report 2018/377, 2018.

[9] J. Chen and S. Micali, Algorand, 2016, Algorand Whitepaper.

[10] M. Dworkin, E. Barker, J. Nechvatal, J. Foti, L. Bassham, E. Roback, and J. Dray, Advanced encryption standard (aes), 2001-11-26 2001.

[11] K. Fan, S. Wang, Y. Ren, H. Li, and Y. Yang, Medblock: Efficient and secure medical data sharing via blockchain, Journal of Medical Systems, 42(8), p. 136, Jun 2018, ISSN 1573-689X.

[12] S. Goldberg, L. Reyzin, D. Papadopoulos, and J. Včelák, Verifiable Random Functions (VRFs), Internet-Draft draft-irtf-cfrg-vrf-13, Internet Engineering Task Force, June 2022, work in Progress.

[13] S. Josefsson and I. Liusvaara, Edwards-Curve Digital Signature Algorithm (EdDSA), Internet-Draft draft-irtf-cfrg-vrf-13, January 2017.

[14] H. Li, L. Zhu, M. Shen, F. Gao, X. Tao, and S. Liu, Blockchain-based data preservation system for medical data, Journal of Medical Systems, 42(8), p. 141, Jun 2018, ISSN 1573-689X.

[15] K. Moriarty, B. Kaliski, and A. Rusch, PKCS #5: Password-Based Cryptography Specification Version 2.1, RFC 8018, January 2017.

[16] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, Decentralized Business Review, p. 21260, 2008.

[17] OWASP-CheatSheet-Series-Team, Password Storage Cheat Sheet, August 2021, Accessed on 2022-08-16.

[18] V. Patel, A framework for secure and decentralized sharing of medical imaging data via blockchain consensus, Health Informatics Journal, 25(4), pp. 1398–1411, 2019, pMID: 29692204.

[19] A. R. Rajput, Q. Li, M. Taleby Ahvanooey, and I. Masood, Eacms: Emergency access control management system for personal health record based on blockchain, IEEE Access, 7, pp. 84304–84317, 2019.

[20] A. Shahnaz, U. Qamar, and A. Khalid, Using blockchain for electronic health records, IEEE Access, 7, pp. 147782–147795, 2019.

[21] M. Sonmez, E. Barker, W. Burr, and L. Chen, Recommendation for password-based key derivation part 1: Storage applications, 2010-12-22 2010.

[22] G. Weiss, Synchronous networks, IRE Transactions on Automatic Control, 7(2), pp. 45–54, 1962.

[23] J. Xu, K. Xue, S. Li, H. Tian, J. Hong, P. Hong, and N. Yu, Healthchain: A blockchain-based privacy preserving scheme for large-scale health data, IEEE Internet of Things Journal, 6(5), pp. 8770–8781, 2019.