

# **BLOCKCHAIN FOR SECURELY SHARING MEDICAL INFORMATION**

By Prabha Pamula

D17B / 51



# Introduction

In the healthcare sector, the secure sharing of patient data is not just a necessity but a fundamental requirement for delivering high-quality care.

Healthcare, as we know it, relies heavily on the ability to access and exchange medical information efficiently and securely.

This data encompasses a patient's medical history, treatment plans, test results, medications, and more. This data is important for effective healthcare delivery.



# Challenges of existing systems

## **Data Breaches:**

- ◆ Data breaches are a significant challenge in healthcare.
- ◆ Breaches can occur due to various reasons, including cyberattacks, insider threats, and accidental exposures.

## **Lack of Interoperability:**

- ◆ Healthcare providers often use a variety of EHR systems with different data formats and standards.
- ◆ Lack of interoperability leads to data barriers, making it difficult for healthcare professionals to access complete patient histories and coordinate care.

## **Fragmented Data Storage:**

- ◆ Patient data is often distributed among hospitals and other healthcare entities.
- ◆ Healthcare providers may struggle to obtain a complete picture of a patient's medical history, which can impact the quality and timeliness of care.

# Blockchain

Blockchain technology is a decentralized and immutable digital ledger system that records transactions across multiple computers, ensuring security and transparency. In a blockchain system, information is grouped into "blocks," linked chronologically to form a "chain." Every block in the chain contains a set of transactions linked to the previous block using cryptography, making it difficult to change data in any block without changing all the subsequent blocks.

Blockchain is a healthcare data security game-changer. It offers decentralization, eliminating central control. Immutability prevents data alteration, and transparency, coupled with cryptographic techniques, ensures robust data security

# Benefits of using Blockchain for sharing medical records

## ◆ **Enhanced Security :**

Blockchain employs robust cryptographic techniques to secure healthcare data. These techniques ensure that patient information is protected from unauthorized access, tampering, and breaches.

## ◆ **Decentralization :**

Blockchain's decentralized architecture that allows multiple participants to access the same dataset. This eliminates the need for a central authority to store data and reduces the risk of a single point of failure, thus enhancing data availability.

### ◆ **Immutability :**

Immutability in blockchain ensures that once data is recorded, it cannot be altered or deleted. This feature prevents data tampering and unauthorized changes to patient records.

### ◆ **Transparency :**

Blockchain provides transparency by recording all data transactions on an immutable ledger. The ledger is accessible to authorized parties for viewing historical data transactions.

Blockchain creates a secure ledger of patients' health records that allows the smooth sharing of information across different healthcare providers while maintaining patient consent and control. This reduces the need for patients to repeatedly share their medical records and brings more speed and efficiency to the healthcare system.

# Technologies

## ◆ **Ethereum :**

Ethereum plays a crucial role in healthcare applications.

The smart contracts, are self-executing agreements with predefined rules. These contracts automate and enforce agreements related to patient consent, data sharing, and billing.

It can be employed in applications ranging from Electronic Health Records (EHRs) to supply chain management for pharmaceuticals.

## ◆ **Hyperledger Fabric :**

Hyperledger Fabric is an open-source platform for building distributed ledger solutions.

Healthcare institutions demand robust security and compliance. Hyperledger Fabric provides the tools to create private, permissioned blockchains that adhere to these requirements. It allows for the creation of private, consortium blockchains, ensuring data confidentiality.

# Examples

## ◆ **MediBloc :**

The MediBloc is using blockchain technology to enable patients to control their own medical data and allow access to healthcare providers. The platform also facilitates interoperability among healthcare institutions by enabling seamless data exchange between different healthcare providers.

## ◆ **BurstIQ :**


BurstIQ provides a platform to help healthcare companies manage their patient data safely and securely. They leverage blockchain technology to offer a secure, decentralized environment for storing, sharing, and managing sensitive health data. By utilizing blockchain's immutability and transparency, BurstIQ ensures the integrity and privacy of patient information and empowers patients to have greater control over their data. This ensures that only authorized healthcare providers can access the data.



# Conclusion

The transformative potential of blockchain for healthcare is undeniable. By leveraging the power of blockchain, healthcare organizations can protect sensitive medical data, aid faster data exchange, streamline operations, and provide better healthcare services to patients. Applying blockchain can also save significant time and money for healthcare providers and patients.

# Experiment 6 : Smart Contract 1

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.7.0 <0.9.0;
4
5  contract PatientRecord {
6      address public owner;
7      string public patientName;
8      string public medicalRecord;
9
10     mapping(address => bool) public authorizedProviders;
11
12     event MedicalRecordUpdated(string newMedicalRecord);
13
14     constructor(string memory _patientName) {  infinite gas 727400 gas
15         owner = msg.sender;
16         patientName = _patientName;
17         authorizedProviders[msg.sender] = true; // The owner (patient) is initially authorized
18     }
19
20     modifier onlyOwner() {
21         require(msg.sender == owner, "Only the owner can perform this action");
22         _;
23     }
24
25     modifier onlyAuthorized() {
26         require(authorizedProviders[msg.sender], "Not authorized to perform this action");
27         _;
```

# Experiment 6 : Smart Contract 2

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.7.0 <0.9.0;
4
5  import "./PatientRecord.sol";
6
7  contract MedicalProvider {
8      address public owner;
9      string public providerName;
10
11
12      event MedicalRecordUpdateAttempt(address patientContract, string newRecord);
13
14      constructor(string memory _providerName) {  ⚠ infinite gas 1643800 gas
15          owner = msg.sender;
16          providerName = _providerName;
17      }
18
19      modifier onlyOwner() {
20          require(msg.sender == owner, "Only the owner can perform this action");
21          _;
22      }
23
24      function createPatientRecord(string memory patientName) public returns (address) {  ⚠ infinite gas
25          PatientRecord newPatientRecord = new PatientRecord(patientName);
26          return address(newPatientRecord);
27      }
28
29      function updateRecord(address patientContract, string memory newRecord) public {  ⚠ infinite gas
30          PatientRecord patient = PatientRecord(patientContract);
31          require(patient.authorizedProviders(address(this)), "Not authorized to update this patient's record");
32          patient.updateMedicalRecord(newRecord);
33          emit MedicalRecordUpdateAttempt(patientContract, newRecord); // Log the attempt event
34      }
35
36      function viewRecord(address patientContract) public view returns (string memory) {  ⚠ infinite gas
37          PatientRecord patient = PatientRecord(patientContract);
38          require(patient.authorizedProviders(address(this)), "Not authorized to view this patient's record");
```

# Experiment 7 : Integration with Metamask

```
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.7.0 <0.9.0;
4
5  import "./PatientRecord.sol";
6
7  contract MedicalProvider {
8      address public owner;
9      string public providerName;
10
11
12      event MedicalRecordUpdateAttempt(address patientContract, string newRecord);
13
14      constructor(string memory _providerName) {  ⚠ infinite gas 1643800 gas
15          owner = msg.sender;
16          providerName = _providerName;
17      }
18
19      modifier onlyOwner() {
20          require(msg.sender == owner, "Only the owner can perform this action");
21          _;
22      }
23
24      function createPatientRecord(string memory patientName) public returns (address) {  ⚠ infinite gas
25          PatientRecord newPatientRecord = new PatientRecord(patientName);
26          return address(newPatientRecord);
27      }
28
29      function updateRecord(address patientContract, string memory newRecord) public {  ⚠ infinite gas
30          PatientRecord patient = PatientRecord(patientContract);
31          require(patient.authorizedProviders(address(this)), "Not authorized to update this patient's record");
32          patient.updateMedicalRecord(newRecord);
33          emit MedicalRecordUpdateAttempt(patientContract, newRecord); // Log the attempt event
34      }
35
36      function viewRecord(address patientContract) public view returns (string memory) {  ⚠ infinite gas
37          PatientRecord patient = PatientRecord(patientContract);
38          require(patient.authorizedProviders(address(this)), "Not authorized to view this patient's record");
```

# Experiment 7 : Integration with Metamask

The screenshot displays the Remix IDE interface for deploying a Solidity contract. The browser address bar shows the URL: `remix.ethereum.org/#lang=en&optimize=false&runs=200&evmVersion=null&version=soljson-v0.8.18+commit.87f61d96.js`.

**Left Sidebar (DEPLOY & RUN TRANSACTIONS):**

- ENVIRONMENT:** Injected Provider - MetaMask
- Custom (33) network**
- ACCOUNT:** 0xc15...2C3Dc (0.05 ether)
- GAS LIMIT:** 3000000
- VALUE:** 0 Wei
- CONTRACT:** PatientRecord - PatientRecord.sol
- evm version:** paris
- Deploy:** Button with dropdown menu showing "john"
- Publish to IPFS:** ☐
- At Address:** Load contract from Address

**Center Panel (Contract Code):**

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.7.0 <0.9.0;
4
5 contract PatientRecord {
6     address public owner;
7     string public patientName;
8     string public medicalRecord;
9
10    mapping(address => bool) public authorizedProviders;
11
12    event MedicalRecordUpdated(string newMedicalRecord);
13
14    constructor(string memory _patientName) {
15        owner = msg.sender;
16        patientName = _patientName;
17        authorizedProviders[msg.sender] = true; // The owner (p
18    }
19
20    modifier onlyOwner() {
21        require(msg.sender == owner, "Only the owner can perform");
22    }
23 }
```

**Right Sidebar (Confirmation Modal):**

- Account 1** (with arrow icon) **New contract**
- https://remix.ethereum.org**
- CONTRACT DEPLOYMENT** (button)
- DETAILS** (selected) **DATA**
- Estimated gas fee** 0.00007563 **0.000076 TR-BTC**
- Site suggested Max fees: 0.00007563 TR-BTC
- Total** 0.00007563 **0.00007563 TR-BTC**
- Amount + gas fee Max amounts: 0.00007563 TR-BTC
- Reject** (button) **Confirm** (button)

**Bottom Panel:**

- listen on all transactions** ☐
- Search with transaction hash or address** (input field)
- creation of PatientRecord pending...**

# Experiment 8 : Deploying with Ganache Account

The screenshot displays the Remix IDE interface, which is used for developing and deploying smart contracts. The interface is divided into several sections:

- Left Sidebar (Deploy & Run Transactions):** This section contains options for the injected provider (MetaMask), the network (Custom (5777) network), the account (0x3D4...FFBA2), the gas limit (3000000), the value (0 Wei), and the contract to be deployed (PatientRecord - PatientRecord.sol). It also includes a "Deploy" button and a "Publish to IPFS" checkbox.
- Central Code Editor:** This area shows the Solidity code for the PatientRecord contract. The code includes a license identifier, pragma solidity statement, and the contract definition with variables for patientName and medicalRecord.
- Bottom Console:** This section displays the deployment transaction details, including the transaction hash, the contract address, and the deployment status (Confirmed).

The following Solidity code is shown in the central editor:

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.7.0 <0.9.0;

contract PatientRecord {
    address public owner;
    string public patientName;
    string public medicalRecord;

    mapping(address => bool) public authorizedProvider;
}
```

The console shows the following transaction details:

- Transaction hash: 0x608...00000
- Contract address: 0x413...F9C
- Deployment status: Confirmed

# Experiment 8 : Deploying with Ganache Account

Ganache

ACCOUNTS

BLOCKS

TRANSACTIONS

CONTRACTS

EVENTS

LOGS

SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK  
1

GAS PRICE  
20000000000

GAS LIMIT  
6721975

HARDFORK  
MERGE

NETWORK ID  
5777

RPC SERVER  
HTTP://127.0.0.1:7545

MINING STATUS  
AUTOMINING

WORKSPACE  
QUICKSTART

SAVE

SWITCH

BACK

BLOCK 1

GAS USED  
925853

GAS LIMIT  
6721975

MINED ON  
2023-10-09 13:51:21

BLOCK HASH  
0x06f10b1f6c3ed2c3decea2bfa7c0d066b32df03b5536531e3b036cdeaa430ebb

TX HASH  
0x41f2cbb6e6686c27af4bdbfae85d8fc19dd47b16b47b455d190b3a26888f5b8b

CONTRACT CREATION

FROM ADDRESS  
0x3D44dD13e2A69C06F05f254656c18116C5cFFBA2

CREATED CONTRACT ADDRESS  
0x41304791e5747dA1253f54dE988c4A18e36F90f1

GAS USED  
925853

VALUE  
0

# Experiment 8 : Deploying with Ganache Account

The screenshot displays the Remix IDE interface within a Firefox browser window. The main workspace is divided into three panels: a left sidebar for file management, a central editor for the Solidity code, and a right sidebar for transaction management and deployment details.

**Left Sidebar (Deployed Contracts):**

- Contract: PATIENTRECORD AT 0x413...F9f
- Balance: 0 ETH
- Functions: grantAccess, revokeAccess, updateMedicalRecord, authorizedPro...
- Variables: medicalRecord, owner, patientName
- Low level interactions: CALLDATA, Transact

**Central Editor (PatientRecord.sol):**

```
1 // SPDX-License-Identifier: GPL-3.0
2
3
4 pragma solidity >=0.7.0 <0.9.0;
5
6
7 contract PatientRecord {
8     address public owner;
9     string public patientName;
10    string public medicalRecord;
11
12
13    mapping(address => bool) public authorizedProviders;
14}
```

**Right Sidebar (Transaction Management):**

- Account 2: 0x413...90f1
- Details: DETAILS, DATA, HEX
- Network status: Network is busy. Gas prices are high and estimates are less accurate.
- Gas (estimated): 0.00021522 ETH
- Very likely in < 15 seconds
- Max fee: 0.00021522 ETH
- Total: 0.00021522 ETH
- Amount + gas fee: 0.00021522 ETH

**Bottom Panel (Transaction Log):**

- Transaction: [block:1 txIndex:0] from: 0x3D4...F8A2 to: PatientRecord.(construct...
- data: 0x608...00000 logs: 0 hash: 0x06f...30ebb
- transact to PatientRecord.updateMedicalRecord pending ...



# References

**MediBlocks: secure exchanging of electronic health records (EHRs) using trust-based blockchain network with privacy concerns**

**Secure decentralized electronic health records sharing system based on blockchains**

**Other References :**

<https://www.turing.com/resources/blockchain-for-healthcare>

<https://www.mosmedicalrecordreview.com/blog/blockchain-technology-ensure-secure-ehr-patient-data-sharing/>