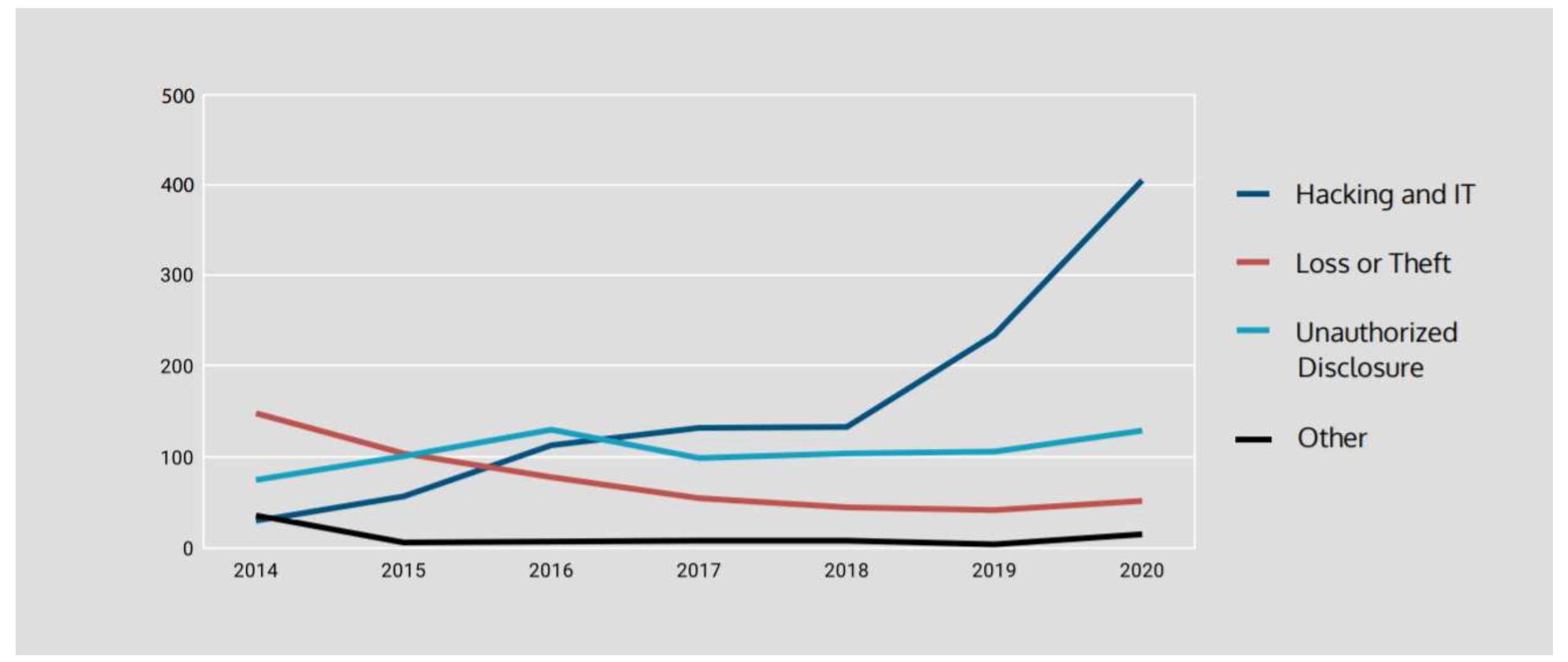
Improving Medical Record Security and Access with Blockchain

Tyler Lin
W Bruce Cottle
JHU APL AOS

Background

- The vast majority of healthcare organizations use electronic health records (EHRs) to store private patient health information
- As healthcare organizations continue to transition to the cloud, EHRs are becoming an increasing target of cyber attackers who hope to compromise sensitive data for monetary gain



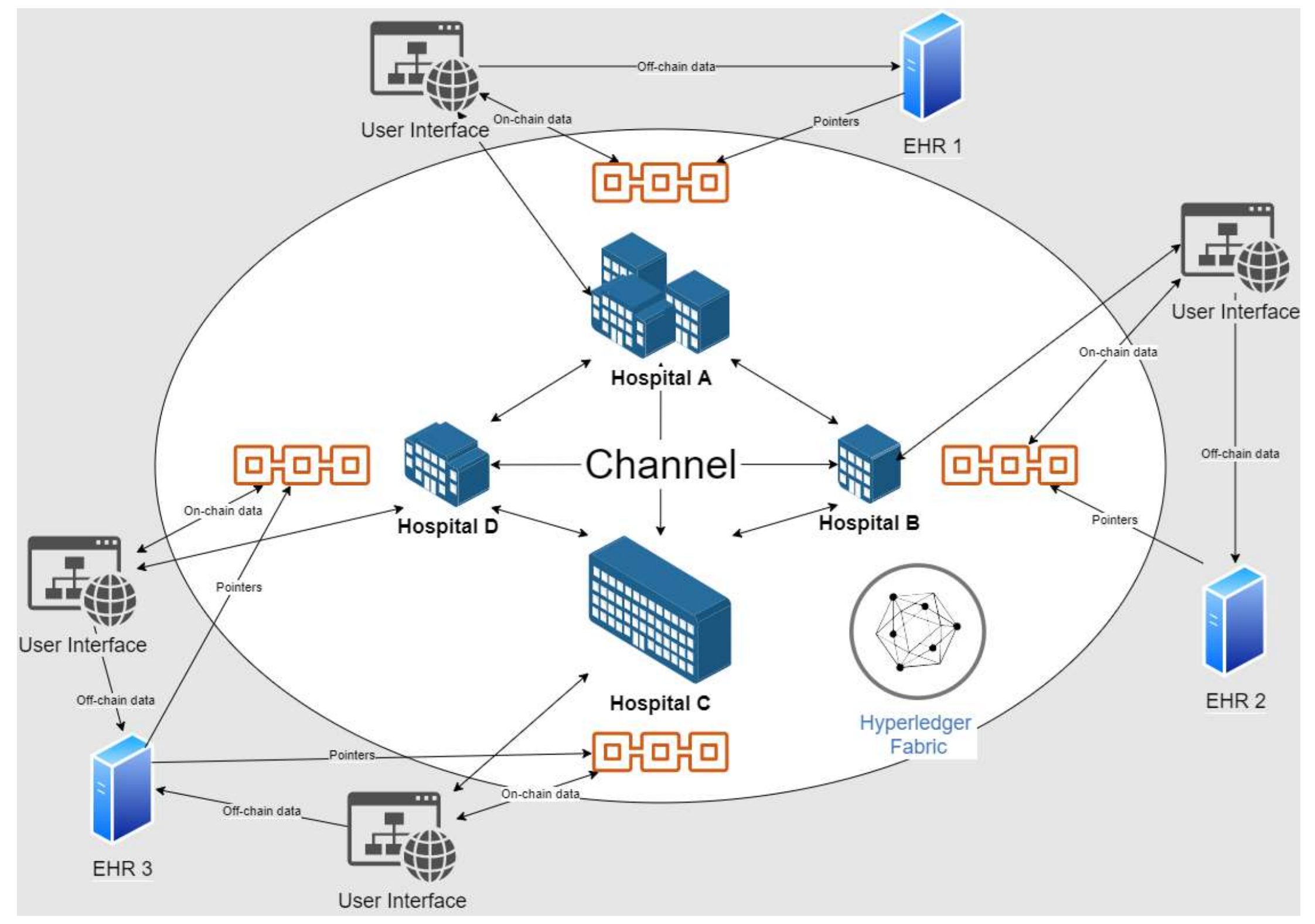
(Data from the U.S. Department of Health and Human Services)

- 2020: 599 total EHR breaches, 55% more than 2019
- Breaches by hackers led to 24.1 out of 26.4 million total breached records in 2020

Methodology

I employed blockchain technology as a layer of communication between hospitals and their EHRs because of its traceability and security.

- Hospital A uses a custom web interface to interact with a patient's records
- Simple data like demographics and medication history are stored directly on the blockchain (on-chain)
- More advanced data like images and scans are stored in the EHR and pointed to by the blockchain (off-chain)
- One cannot view or edit records without going through the blockchain

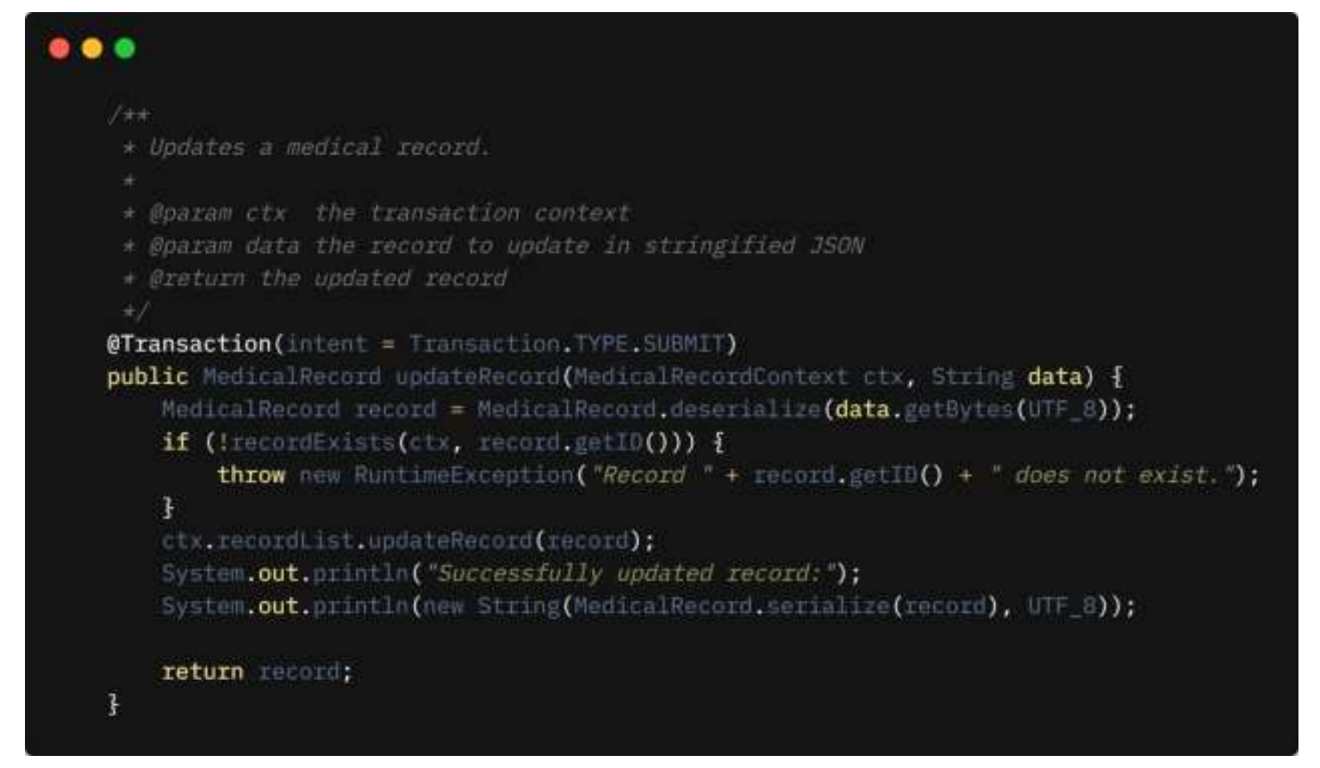


A conceptual design of my system. Arrows indicate the flow of data.

Blockchain Implementation

I implemented the blockchain using **Hyperledger Fabric**, a versatile distributed ledger framework hosted by the Linux Foundation.
In Fabric, a group of computers that share a blockchain is called a *channel*. Each channel features

- a blockchain transaction log
- a state database that stores the current value of the ledger
- a *smart contract*, or a program that executes business logic and updates the blockchain accordingly



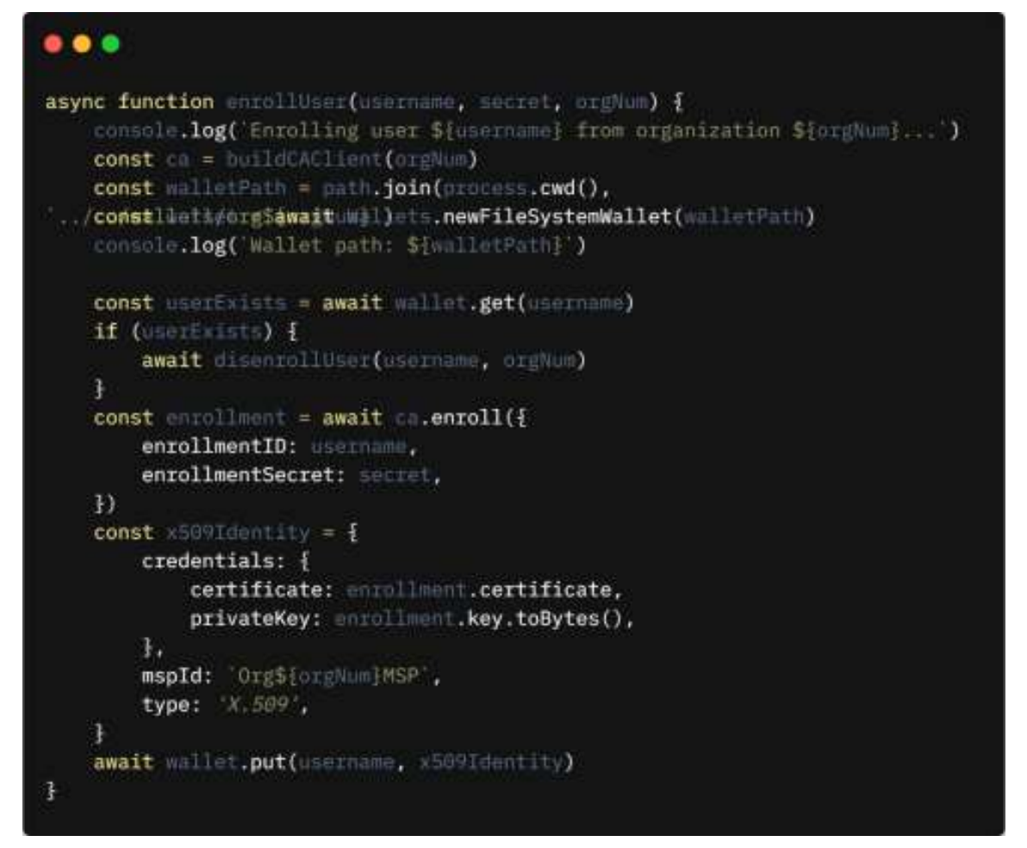
```java
/**
 * Updates a medical record.
 *
 * @param ctx  the transaction context
 * @param data the record to update in stringified JSON
 * @return the updated record
 */
@Transaction(intent = Transaction.TYPE.SUBMIT)
public MedicalRecord updateRecord(MedicalRecordContext ctx, String data) {
    MedicalRecord record = MedicalRecord.deserialize(data.getBytes(UTF_8));
    if (!recordExists(ctx, record.getID())) {
        throw new RuntimeException("Record " + record.getID() + " does not exist.");
    }
    ctx.recordList.updateRecord(record);
    System.out.println("Successfully updated record:");
    System.out.println(new String(MedicalRecord.serialize(record), UTF_8));

    return record;
}
```

- I programmed my smart contract in Java to handle medical records
- Transactions (add a record, edit a record, search records, etc.) are each handled with their own method
- My `MedicalRecord` class contains patient ID, name, immunizations, and medication

Authentication

I handled user enrollment and authentication through *certificate authorities*.

- Users register with a username and secret (password)
- The web application generates a public-private key pair for the user
  - Anyone can encrypt a message with a user's public key, but only the user themself can decrypt the message with their private key
- The web app signs a certificate signing request (CSR) with the private key and sends it to the CA
- The CA issues an X.509 certificate binding the public key to the user, allowing for trusted access to the blockchain

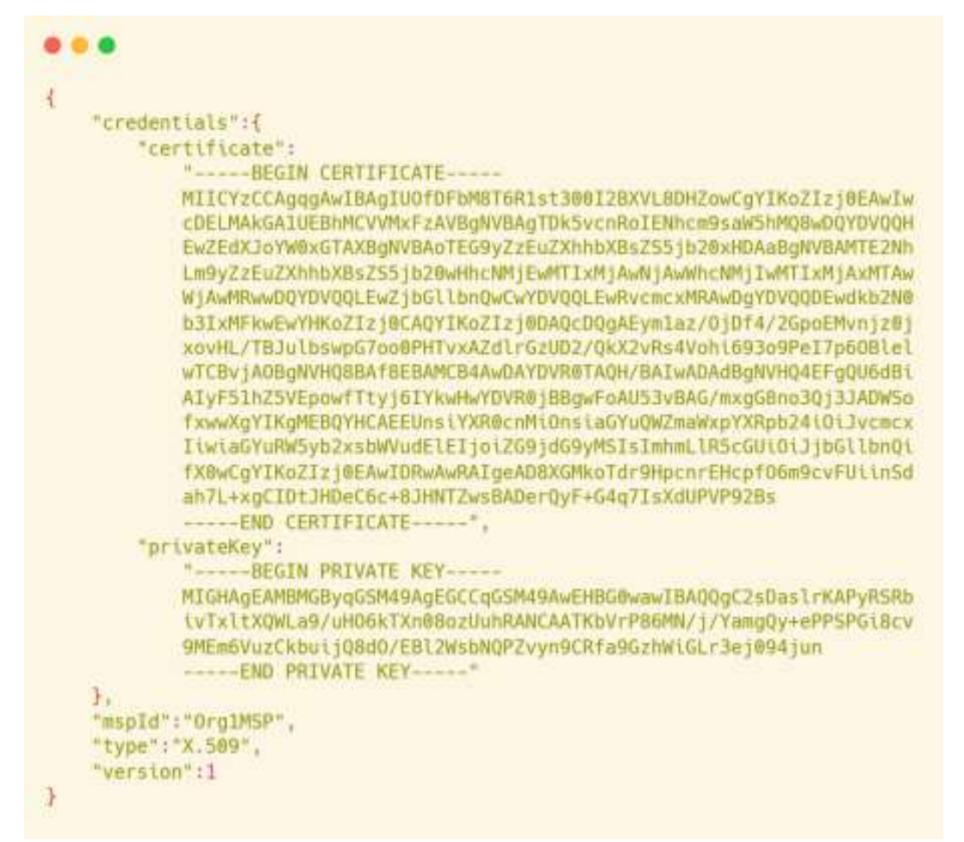


```javascript
async function enrollUser(username, secret, orgNum) {
    console.log(`Enrolling user ${username} from organization ${orgNum}...`)
    const ca = buildCAClient(orgNum)
    const walletPath = path.join(process.cwd(),
'../wallets/org${orgNum}')
    const wallet = await Wallets.newFileSystemWallet(walletPath)
    console.log(`Wallet path: ${walletPath}`)

    const userExists = await wallet.get(username)
    if (userExists) {
        await disenrollUser(username, orgNum)
    }
    const enrollment = await ca.enroll({
        enrollmentID: username,
        enrollmentSecret: secret,
    })
    const x509Identity = {
        credentials: {
            certificate: enrollment.certificate,
            privateKey: enrollment.key.toBytes(),
        },
        mspId: `Org${orgNum}MSP`,
        type: 'X.509',
    }
    await wallet.put(username, x509Identity)
}
```

< User enrollment code (Node.js)

produces

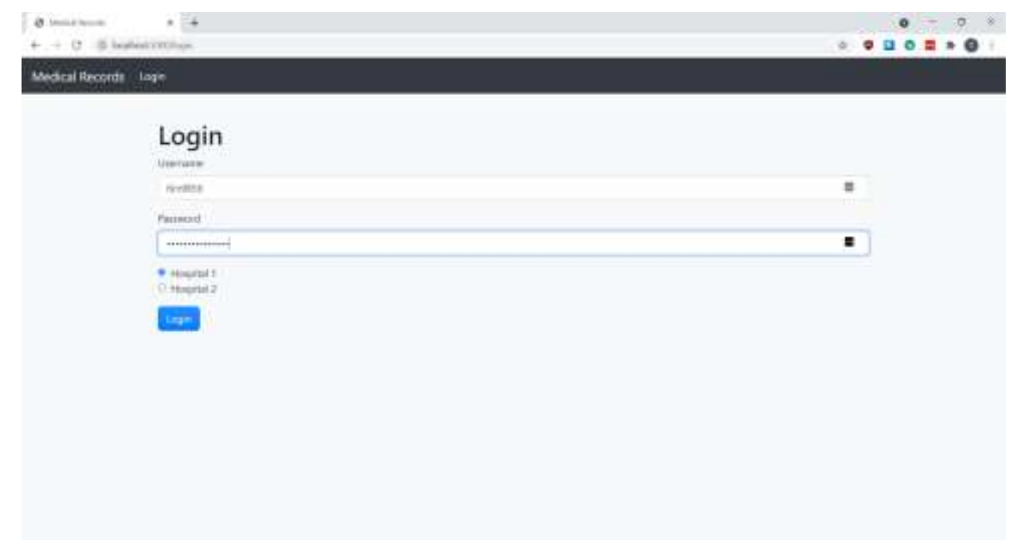
Enrollment certificate >

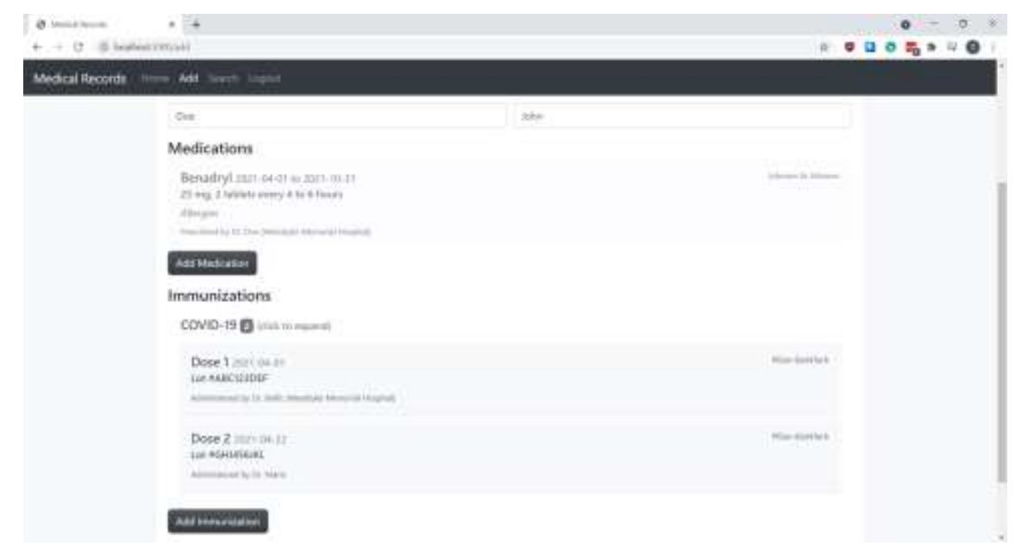


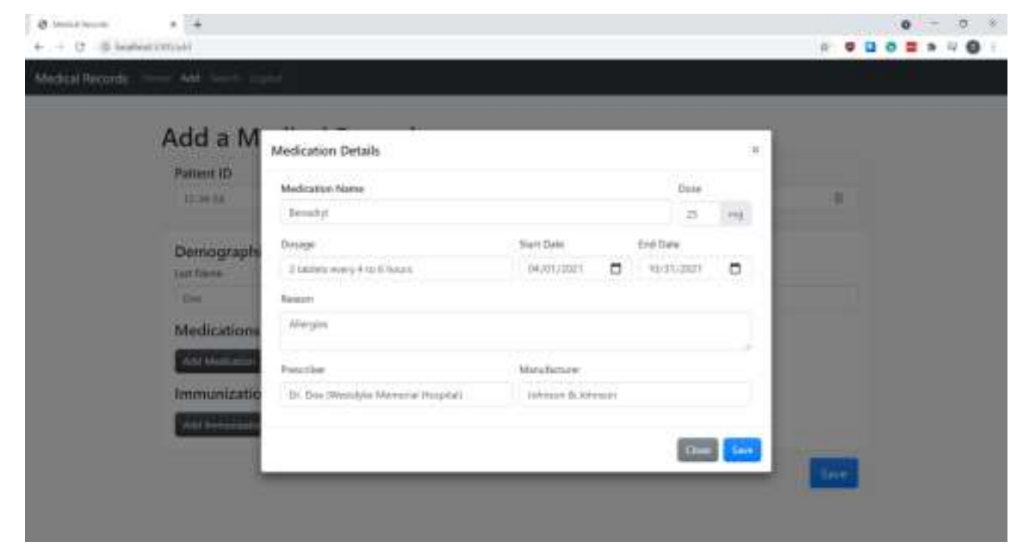
```
{
    "credentials":{
        "certificate":
            "-----BEGIN CERTIFICATE-----
            MIICYzCCAgqgAwIBAgIUOfDFbM8T6R1st300I2BXVL8DHZowCgYIKoZIzj0EAwIw
            cDELMAkGA1UEBhMCVVMxFzAVBgNVBAgTDk5vcnRoIENhcm9saW5hMQ8wDQYDVQQH
            EwZEdXJoYW0xGTAXBgNVBAoTEG9yZzEuZXhhbXBsZS5jb20xHDAaBgNVBAMTE2Nh
            Lm9yZzEuZXhhbXBsZS5jb20wHhcNMjEwMTIxMjAwNjAwWhcNMjIwMTIxMjAxMTAw
            WjAwMRwwDQYDVQQLEwZjbGllbnQwCwYDVQQLEwRvcmcxMRAwDgYDVQQDEwdkb2N0
            b3IxMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEym1az/OjDf4/2GpoEMvnjz0j
            xovHL/TBJulbswpG7oo0PHTvxAZdlrGzUD2/QkX2vRs4Vohi693o9PeI7p6OBlel
            wTCBvjAOBgNVHQ8BAf8EBAMCB4AwDAYDVR0TAQH/BAIwADAdBgNVHQ4EFgQU6dBi
            AIyF51hZ5VEpowfTtyj6IYkwHwYDVR0jBBgwFoAU53vBAG/mxgG8no3Qj3JADWSo
            fxwwXgYIKgMEBQYHCAEEUnsiYXR0cnMiOnsiaGYuQWZmaWxpYXRpb24iOiJvcmcx
            IiwiaGYuRW5yb2xsbWVudElEIjoiZG9jdG9yMSIsImhmLlR5cGUiOiJjbGllbnQi
            fX0wCgYIKoZIzj0EAwIDRwAwRAIgeAD8XGMkoTdr9HpcnrEHcpf06m9cvFUiinSd
            ah7L+xgCIDtJHDeC6c+8JHNTZwsBADerQyF+G4q7IsXdUPVP92Bs
            -----END CERTIFICATE-----",
        "privateKey":
            "-----BEGIN PRIVATE KEY-----
            MIGHAgEAMBMGByqGSM49AgEGCCqGSM49AwEHBG0wawIBAQQgC2sDaslrKAPyRSRb
            ivTxltXQWLa9/uHO6kTXn08ozUuhRANCAATKbVrP86MN/j/YamgQy+ePPSPGi8cv
            9MEm6VuzCkbuijQ8dO/EBl2WsbNQPZvyn9CRfa9GzhWiGLr3ej094jun
            -----END PRIVATE KEY-----"
    },
    "mspId":"Org1MSP",
    "type":"X.509",
    "version":1
}
```

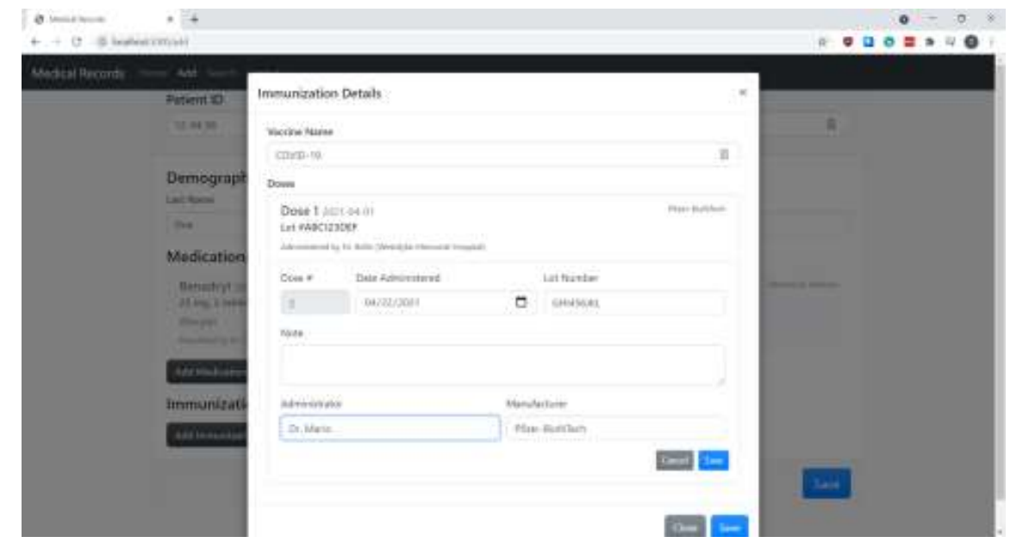
Web Application

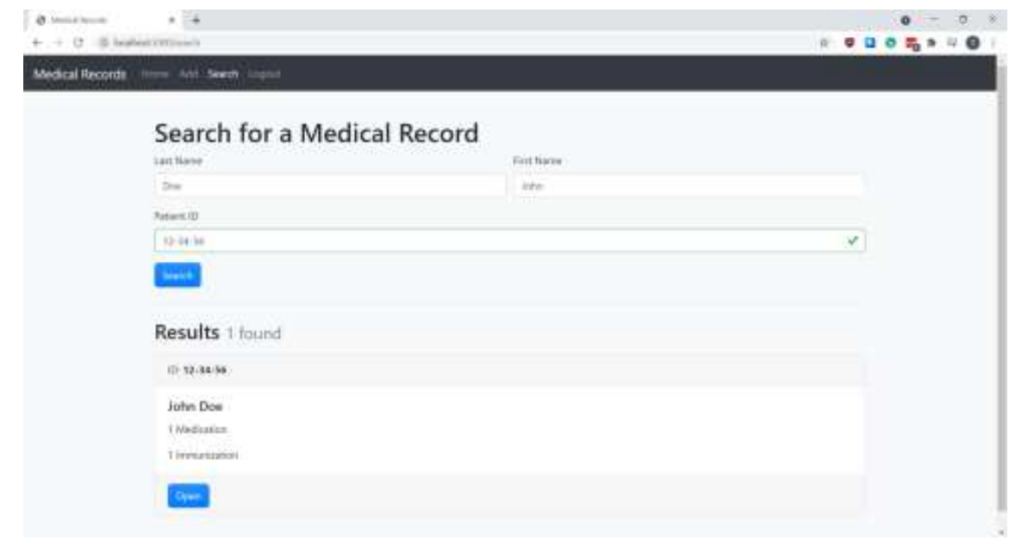
I engineered a user interface using Node.js and Bootstrap that attempts to emulate the user experience offered by actual EHRs. It allows the user to register or login and add, edit, delete, or search medical records while communicating with Fabric in the background.

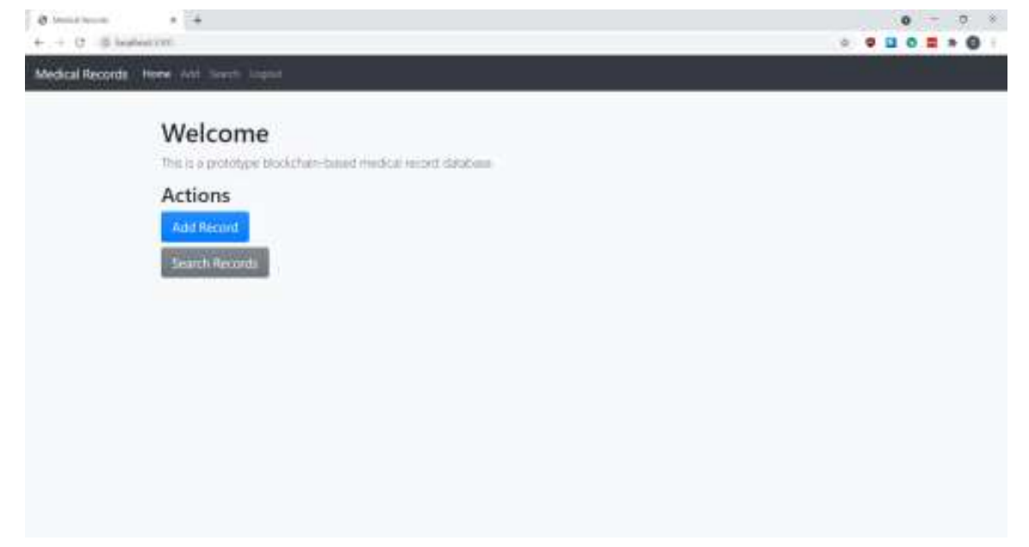












Conclusion

This project offers promise towards the idea of blockchain-based EHRs by proving that such systems may be feasible.

	Traditional EHRs	Blockchain-based EHRs
1	Vulnerable to cyber attacks	Inherent cryptographic security
2	Centralization creates a single point of failure	Decentralization limits the damage done by a single attacker
3	Patient identifiers must be synchronized across systems	Patients are identified by universal public and private keys
4	Varying data standards reduce interoperability	Shared data enables near real-time updates across the network
5	Inconsistent rules and permissions inhibit patient data access	Smart contracts allow for consistent, rule-based access
6	No easy way to log access and edit history for legal purposes	Blockchain of transactions creates an undeniable audit trail

References

Bitglass. (2021, February 17). *Healthcare Breach Report 2021*. https://pages.bitglass.com/rs/418-ZAL-815/images/CDFY21Q1HealthcareBreachReport2021.pdf
Deloitte. (2016, August). *Blockchain: Opportunities for Health Care*. https://www2.deloitte.com/content/dam/Deloitte/us/Documents/public-sector/us-blockchain-opportunities-for-health-care.pdf

Acknowledgements

I would like to thank my mentor, Bruce Cottle, for his invaluable support during this internship and project.