

International Vaccination Records

50.037 Blockchain Technology DApp project report

Team members

| Name | Student ID |
|---------------------------|------------|
| Alex Wang Wei Jie | 1003474 |
| Jose Johnson Emerson Raja | 1003651 |

Problem background

The COVID-19 pandemic has brought many industries and countries to a screeching halt. Due to the high transmission rate of the virus, large gatherings and overseas travel have been prohibited in hopes of containing the virus.

As countries aim to open up their borders and citizens learn to adapt to the new normal, it is obvious that widespread vaccination will play a major role in expediting this process. Vaccinated individuals will be able to travel more freely and attend events with larger capacities.

However, vaccinations alone will not propel us towards the new normal. We also require efficient ways to verify if a person has been vaccinated.

We believe an **international vaccination record** will be able to accomplish this.

Solution criteria

Given the background of the problem, an international vaccination record must have the following two features.

- **Global accessibility and availability**
 - The information on the record needs to be accessible by any country, and not constrained to a specific geographical location.
 - The information should also be available to any country or entity that wishes to verify a person's vaccination record.
- **Secure storage**
 - Only trusted users should be able to store information on the record.
 - Malicious users should not be able to tamper with records that have already been uploaded to the international vaccination record.

As a bonus, given the two features above, the international vaccination record also gives a unique opportunity for **accurate analytics** of vaccination efforts around the world. A use case for this could be when a traveller can use these analytics to decide which country would be the safest for a vacation.

Decentralized Application implementation

Decentralized applications (DApp) provide a unique avenue for international vaccination records to be implemented. The rationale for a decentralized application implementation is summarised below.

- **Immutability**

- Since all information in a DApp is stored on a blockchain, malicious attackers are not able to tamper with data that is already part of the blockchain. This also means that any record of a person's vaccination cannot be tampered once added to the blockchain. This helps to ensure *secure storage*.

- **Decentralization**

- Since there is no single entity that maintains the blockchain, there is no possibility for a single point of failure.
- Due to the distributed nature of a DApp, the constant *global availability and accessibility* of the information on the record can be guaranteed. [^1]

- **Transparency**

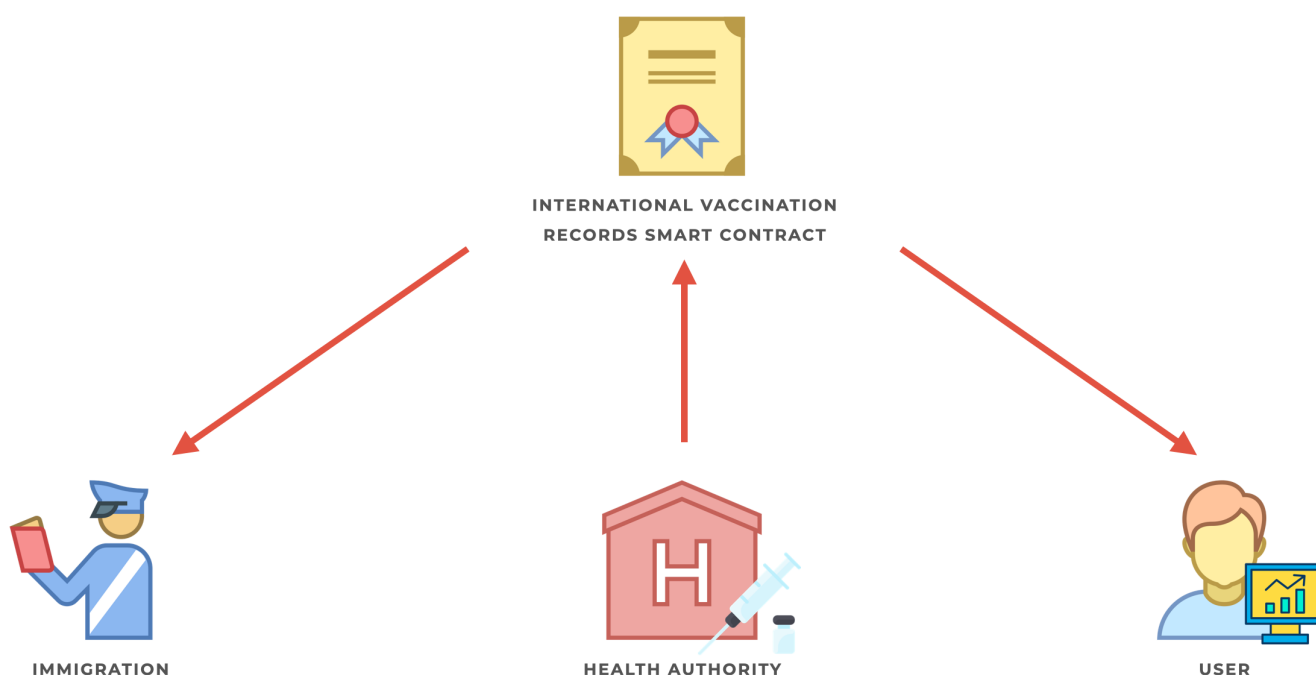
- Since all information on the DApp is publicly available, this allows for any country to access the records of a specific person. This helps to ensure *global accessibility and availability*.

- **Security**

- Since any information that is added to the blockchain is agreed upon by distributed consensus protocols, it is extremely difficult for a malicious user to add a new record to the international vaccination record. This helps to ensure *secure storage*.

Application Mechanism

Having established that a DApp is a suitable platform to develop the international vaccination record, we propose the mechanism below as a framework to deploy the solution.



We would deploy the international vaccine record smart contract on the Ethereum blockchain. This smart contract would be responsible for managing all the information about the vaccination records that have been uploaded. It would also act as the endpoint from where results can be made available to users and organisations.

Once a person has been vaccinated, the health authority that has vaccinated the individual will upload the vaccination details to the blockchain via the smart contract, using the user's passport number as the index.

Now, users or authorities will be able to view these records by querying the blockchain via the smart contract. Based on the results from the query, the users or authorities can decide on what action to take. For instance, an

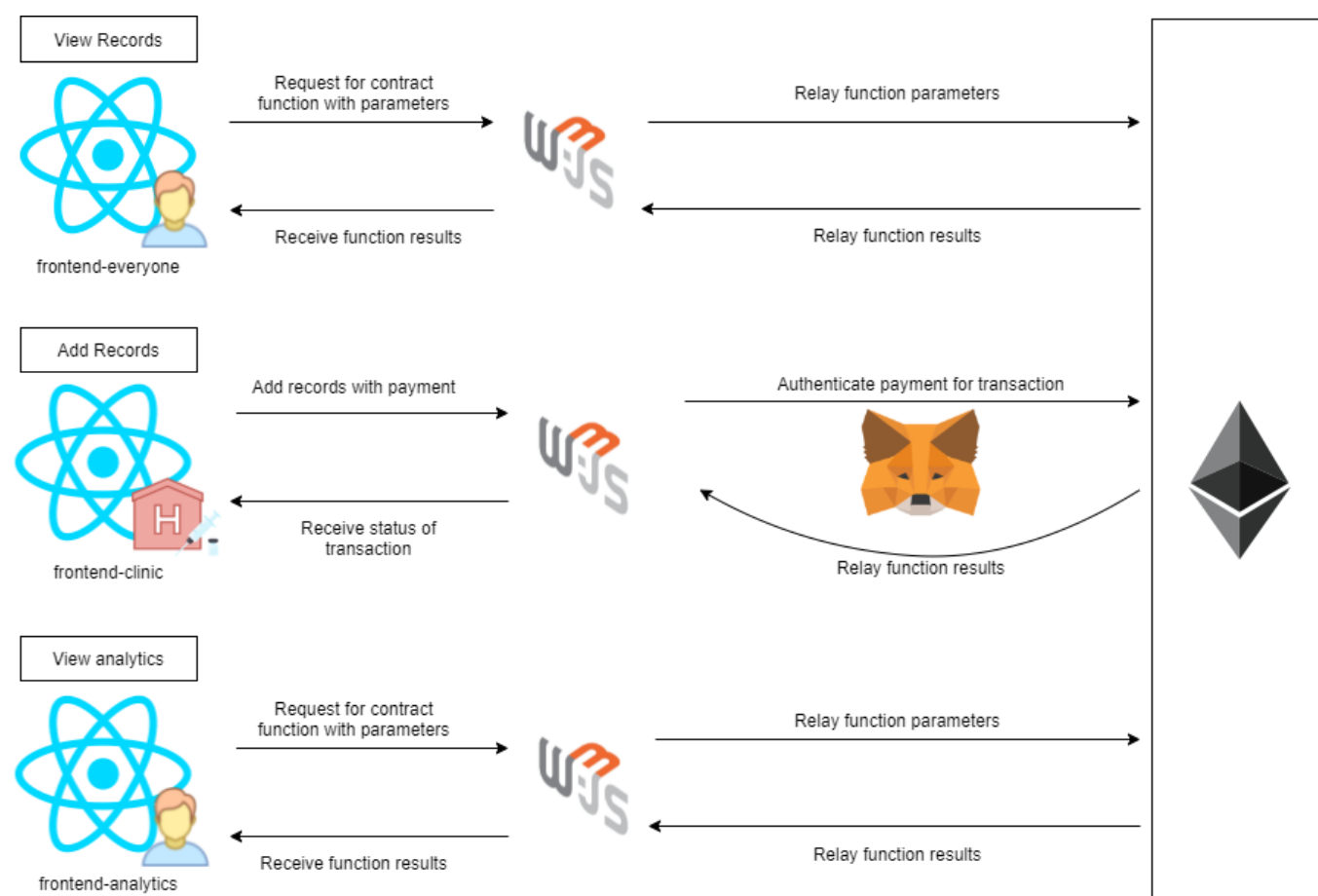
immigration officer can verify if a person has been fully vaccinated, before being allowed entry into a country.

And because all the information on the blockchain is transparent, the smart contract can also provide some form of analytics. For instance, users can consult these analytics on vaccination efforts in a specific country to decide if they would consider visiting the country.

Prototype

With a potential mechanism in mind, we were able to implement a prototype that aims to demonstrate the efficacy of an international vaccination record.

System Architecture



For our prototype, we have deployed the smart contract on the Rinkeby testnet blockchain network. It is accessible at the following address: [0x7152192Ec75ADA9cf90ca0e158f387026269Aad7](#)

Our prototype consists of three frontend webpages, as shown by the React icons on the architecture above.

The first and last webpages are used for viewing the state of the smart contract and therefore does not change the state. These webpages can be used by anyone in public.

The second webpage however involves a payment and changing the state of the contract. This is where we use Metamask to authorise and pay for the gas fees of the transaction.

Code structure

```
struct Record {
    uint256[] passportNumber;
    string vaccineManufacturer;
    string country;
    uint256 timestamp;
}

struct CountryDetails {
    string name;
    uint256 population;
    uint256 totalVaccinated;
}

struct RsaKey {
    uint256 key;
    uint256 modulus;
}

mapping(bytes32 => Record) records;
bytes32[] passportNumbers;
mapping(string => RsaKey) countryToPublicKeyMap;

mapping(string => CountryDetails) analytics;
string[] countryNames;
```

The smart contract that is deployed on the Rinkeby testnet uses these data structures to maintain all the information.

The **Record** struct defines the template of information that the health authority sends to the contract.

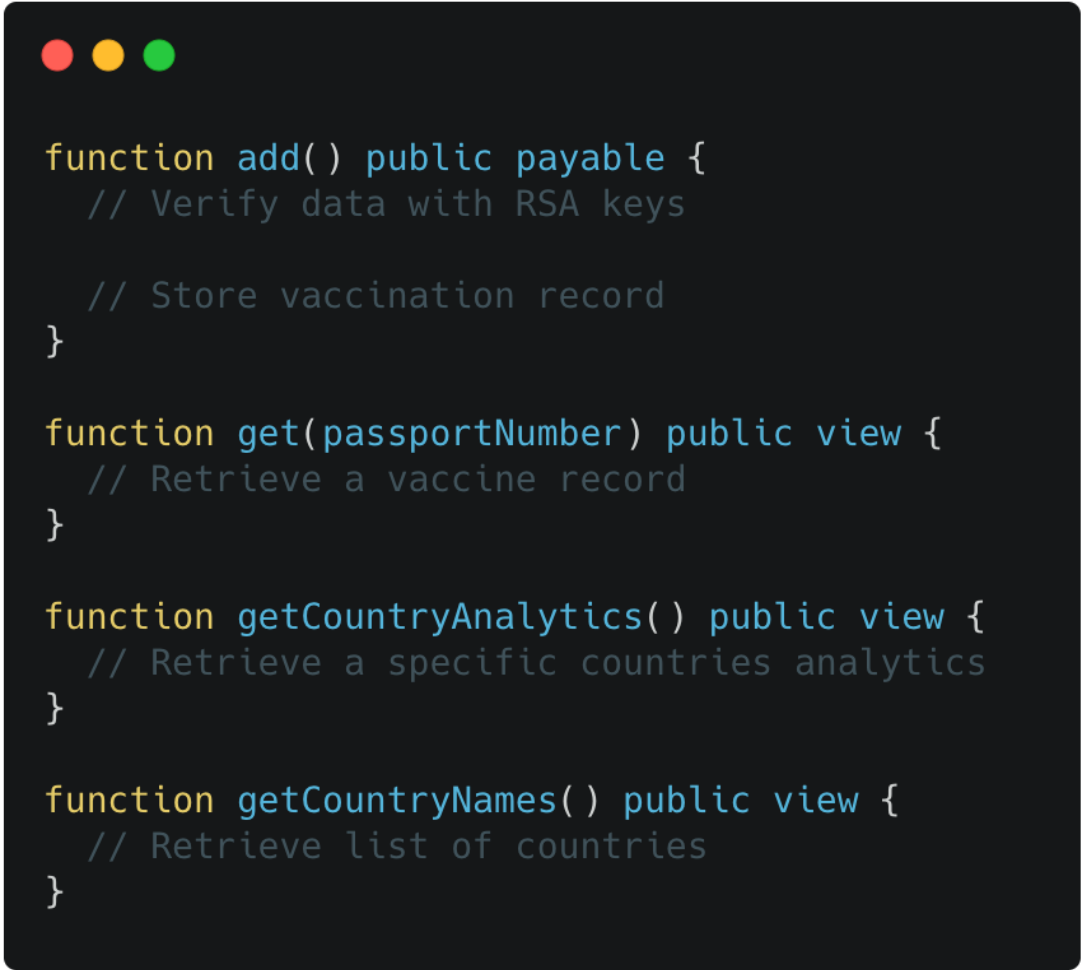
The contract then stores all the records in the **records** map. The map contains a mapping of the passport number hash to the corresponding record itself.

The contract also keeps track of all the passport number hashes saved in the bytes32 array called **passportNumbers**. This separate array is used to complement the **records** map for querying the total number of passportNumbers and checking if one has already been recorded.

The **CountryDetails** struct defines the template to store information about a specific country.

The contract then stores the information about the countries in the **analytics** map. This map contains a mapping of the name of the country to its corresponding details.

The **RsaKey** struct defines the template to store the RSA keys of a specific country. This struct is necessary for our public-key cryptography implementation, which we expand on later in the report. The mappings of the country to its key details are kept in the **countryToPublicKey** map.



```
function add() public payable {  
    // Verify data with RSA keys  
  
    // Store vaccination record  
}  
  
function get(passportNumber) public view {  
    // Retrieve a vaccine record  
}  
  
function getCountryAnalytics() public view {  
    // Retrieve a specific countries analytics  
}  
  
function getCountryNames() public view {  
    // Retrieve list of countries  
}
```

There are the 4 public functions that the users and health authorities use to interact with the smart contract. There are other helper functions, but these have a private modifier.

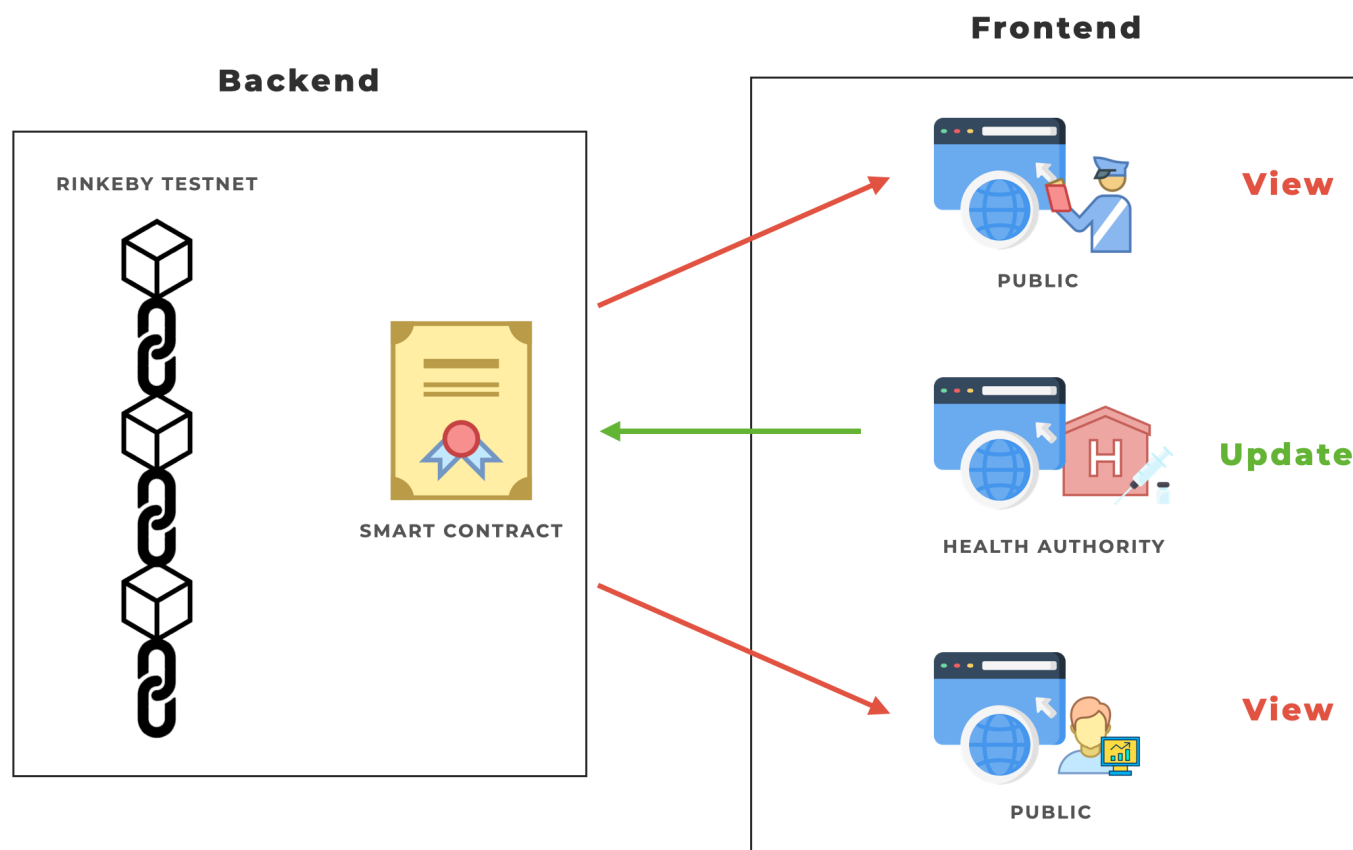
The **add** function is used to add a record to the testnet, after it verifies the data received with the country's corresponding RSA keys.

The **get** function retrieves a vaccination record given a passport number hash.

And the **getCountryAnalytics** and **getCountryNames** functions provide the analytics information to the frontend.

Use Case diagram

Below is a use case diagram that summarises the features of the prototype.



Our prototype has 4 main features which are detailed below.

1. Store a person's vaccination record.
2. Secure data transfer between the health authority and the smart contract by using RSA public-key cryptography.
3. Perform quick analytics regarding vaccination efforts in a specific country.
4. Search for a person's vaccination records

RSA Cryptography, Digital Certificate

RSA Basics

To support only authorising approved vaccination clinics to be able to add records, we decided to use RSA public key cryptography. However, we could not find an openssl equivalent library in solidity to support the functions we needed. Hence, we created our own RSA system by starting from the basics of RSA.

In traditional RSA, the following steps are used to generate public and private keys:

1. Choose 2 prime numbers, p, q
2. Compute
 - $n = pq$
 - $z = (p-1)(q-1)$
3. Choose e , such that $e < n$, and e, z are relatively prime
4. Choose d , where $(ed-1) \% z = 0$
5. Private key = (e, n)
6. Public key = (d, n)

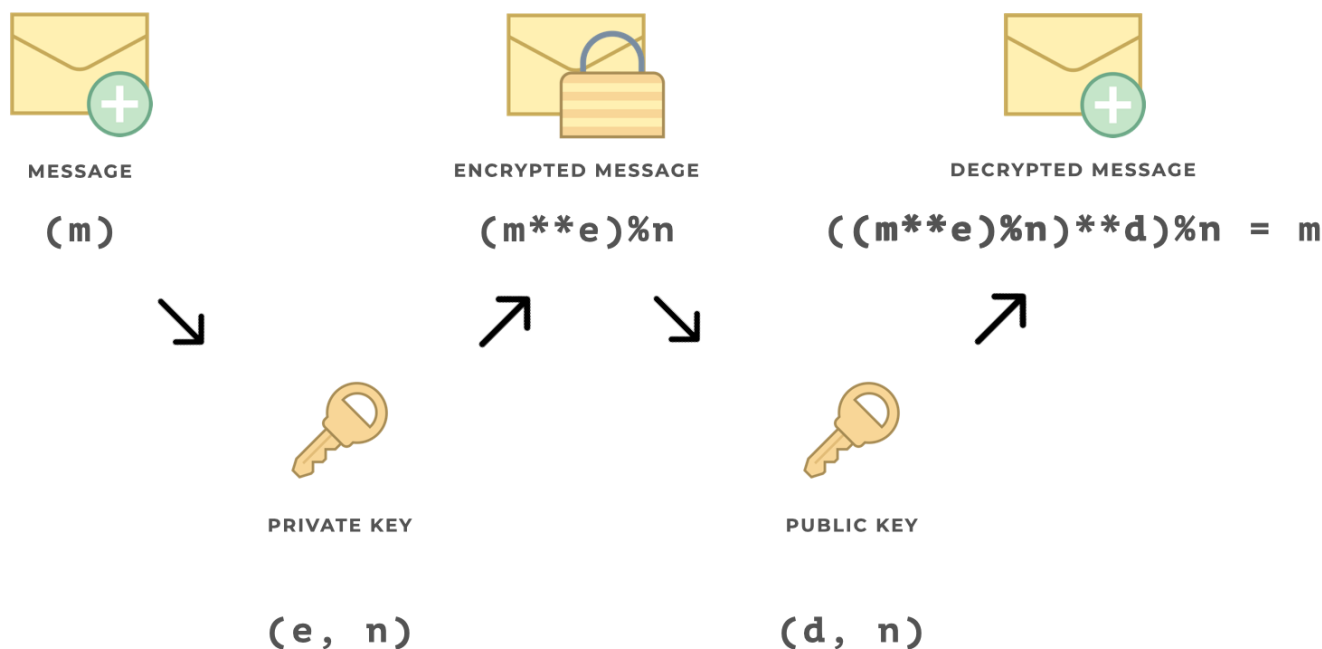
7. For encryption, with message m :

- encrypted message = $(m**e)\%n$

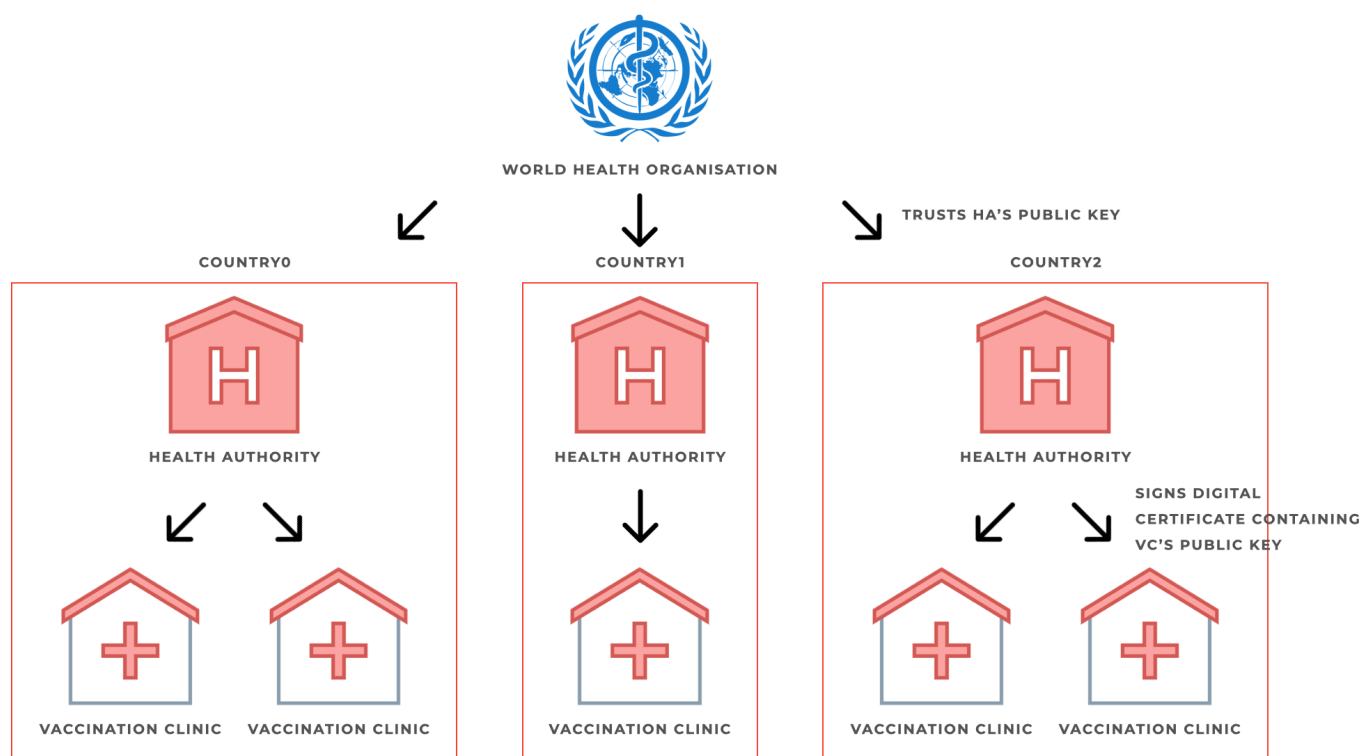
8. For decryption, apply the encrypted message m with public key

- $m = ((m**e)\%n)**d)\%n$

The encryption and decryption process is summarised in the diagram below:



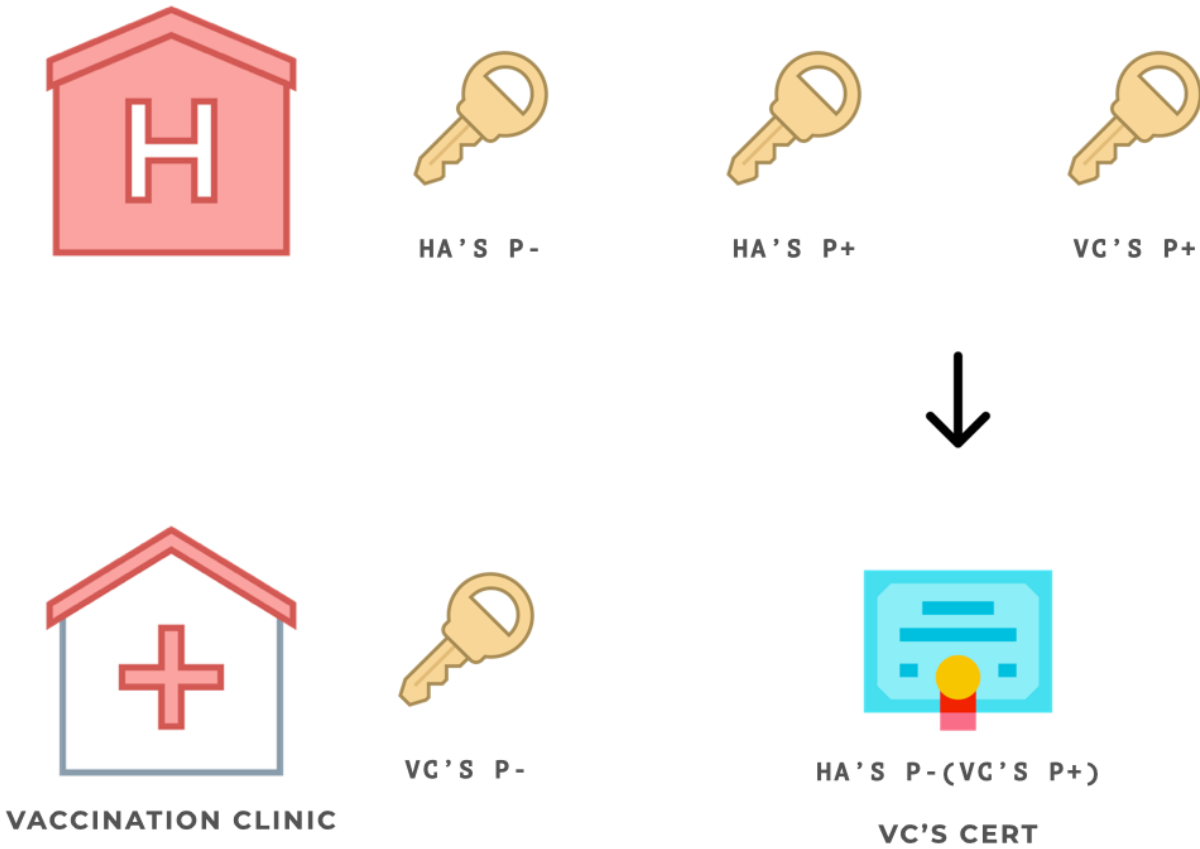
RSA Use Case



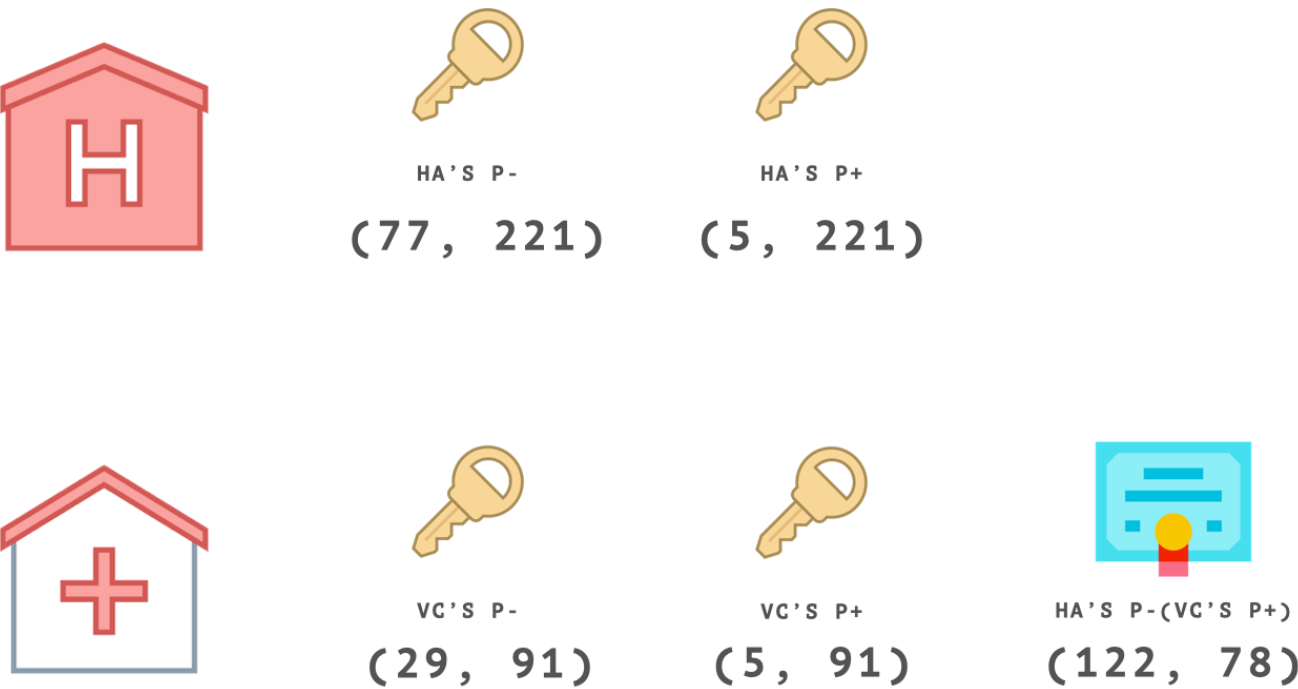
To emulate traditional Certification Authorities (CA) and individuals who apply for certificates for their public keys, the World Health Organisation acts as the root CAs that 'trusts' various countries' health authority (HA)'s public keys. Each countries' health authority in turn certifies various vaccination clinics' (VC) public keys.

For simplified version of the RSA process involving 1 HA and 1 VC:

- 1. Both generate their own pair of private and public keys:
 - HA's p^- , p^+
 - VC's p^- , p^+
- 2. The VC gives the p^+ to HA
- 3. HA checks that the VC is legitimate, and certifies the p^+ by encrypting it with HA's p^-



For the purpose of the prototype, we generated keys for 1 HA and 1 VC as follows:



RSA Walkthrough

1. VC encrypts person's passport number with VC's p^-

[International_Vaccination_Records/frontend-clinic/index.html](#)

```
var passportNumberEncrypted = [];
passportNumber.split('').map((e) => {
  passportNumberEncrypted.push(
    parseInt(
      BigInt(e) ** BigInt(clinicPrivateKey) % BigInt(clinicModulus)
    )
  );
});
```

2. VC uploads encrypted passport number, the passport number hash, and the certificate to the smart contract by making a `send()` transaction [International_Vaccination_Records/frontend-clinic/index.html](#)

```
contract.methods
  .add(
    passportNumberEncrypted,
    passportNumberHash,
    vaccineManufacturer,
    country,
    clinicKeyCert,
    clinicModulusCert
  )
  .send({ from: accountSelected, gas: 1000000 })
  .then((output) => {
    console.log(output);
    showStatus(status);
  });
```

In this step, we make sure that no `passportNumber` is sent in plaintext to prevent network sniffing, and having it recorded in plaintext as input data in the blockchain.

3. When the smart contract receives the data, it first looks up the HA's p^+ key based on the country the VC is in, then use the HA's p^+ to obtain VC's p^+ , from the certificate the VC uploaded

[International_Vaccination_Records/backend/Records.sol](#)

```
RsaKey memory countryPublicKey = countryToPublicKeyMap[country];
uint256 clinicKey = computeRsa(
  clinicKeyCert,
  countryPublicKey.key,
  countryPublicKey.modulus
);
uint256 clinicModulus = computeRsa(
```

```

        clinicModulusCert,
        countryPublicKey.key,
        countryPublicKey.modulus
    );
    RsaKey memory clinicPublicKey = RsaKey(clinicKey, clinicModulus);

```

4. With **VC's p+**, the smart contract uses it to decrypt the passport number

[International_Vaccination_Records/backend/Records.sol](#)

```

uint256[] memory passportNumberDecrypted = new uint256[](
    passportNumberEncrypted.length
);

for (uint256 i = 0; i < passportNumberEncrypted.length; i++) {
    passportNumberDecrypted[i] = computeRsa(
        passportNumberEncrypted[i],
        clinicPublicKey.key,
        clinicPublicKey.modulus
    );
}

```

5. In the end, the smart contract computes its own hash based on the decrypted passport number, and checks that it matches the one that is uploaded by the VC

smart contract's hash

[International_Vaccination_Records/backend/Records.sol](#)

```

function verifyHash(uint256[] memory input, bytes32 hash)
    private
    view
    returns (bool)
{
    return keccak256(abi.encodePacked(input)) == hash;
}

```

frontend VC's hash

[International_Vaccination_Records/frontend-clinic/index.html](#)

```

var passportNumberHash = web3.utils.soliditySha3({
    type: "uint256[]",
    value: passportNumber.split("").map((e) => parseInt(e)),
});

```

If they match, the smart contract can be certain that this VC is indeed certified by their country's health authority, which is trusted by the smart contract itself. This is also known as "chain of trust" in conventional digital certificates.

Test Data

We followed the above steps and inserted the following passportNumbers in the following transactions:

| passportNumber | Tx Hash |
|----------------|--|
| 281721231 | 0xca65a2bce66997fb6658a5b5417bed7aad37e9b6692d1196635d39deb5b9965e |
| 3495871231 | 0x43c8b48cb4e1b4a0f00454cbe8984808e6d79e34c94ea0393f4407adec8d2f53 |
| 283712389 | 0x8f35869ed0d242d98a6a9e351be9cc7584e71881c9ae87725fa4f18081ad9b84 |

Usage manual

Since the smart contract has already been deployed to the Rinkeby testnet, all you have to do is interact with it through our web pages.

Make sure to have your Metamask extension installed and connected to the Rinkeby testnet network.

In our source code, you will find three directories with a `frontend-` prefix. These are the webpages that interact with the smart contract.

`frontend-clinic` and `frontend-everyone` are basic web pages. This means you need to serve the respective `index.html` with a local webserver and view the pages in a Chrome browser with the Metamask extension.

`frontend-analytics` on the other hand is a React project. Therefore, you would need to use `npm start` while in the directory to run the React server. The corresponding webpage will be deployed on `localhost:3000`.

Performance analysis

As each block will be mined in ~15 seconds, there will be a delay whenever `add()` function is called. To reduce the delay, we daisy chained `.call()` and `.send()` in `frontend-clinic` in the following manner:

`International_Vaccination_Records/frontend-clinic/index.html`

```
var status = contract.methods
  .add(
    passportNumberEncrypted,
    passportNumberHash,
    vaccineManufacturer,
    country,
    clinicKeyCert,
    clinicModulusCert
  )
  .call({ gas: 1000000 })
  .then(
    (output) => {
```

```

        console.log(output);
        status = output;

        if (status == "Record added successfully.") {
            contract.methods
                .add(
                    passportNumberEncrypted,
                    passportNumberHash,
                    vaccineManufacturer,
                    country,
                    clinicKeyCert,
                    clinicModulusCert
                )
                .send({ from: accountSelected, gas: 1000000 })
                .then((output) => {
                    console.log(output);
                    console.log(status);
                    showStatus(status);
                });
        } else {
            showStatus(status);
        }
    },
    (e) => showStatus(e)
);

```

The initial `call()` emulates the subsequent `send()` function, but without altering the blockchain's state, this allows the frontend to determine whether the request will be successful. If the return data from the smart contract matches the string "Record added successfully.", the frontend then continues to do a `send()` transaction. Otherwise, if any errors are encountered, they will be reflected to the user quickly, without waiting for the next block to be mined, reducing the latency and improving usability.

Gas analysis

The only transaction that requires gas fees is the `add` function that is called by the Health Authorities to add a vaccination record to the blockchain.

From our analysis of the transactions on Etherscan, it seems that each add transaction uses around **0.0004081044 Ether** for an `add` transaction. The breakdown of the transaction fees is shown below.

| | |
|--------------------------|-------------------------------|
| Transaction Fee: | 0.0004081044 Ether (\$0.00) |
| Gas Price: | 0.0000000011 Ether (1.1 Gwei) |
| Txn Type: | 0 (Legacy) |
| Gas Limit: | 1,000,000 |
| Gas Used by Transaction: | 371,004 (37.1%) |

You can find this transaction at [this](#) link.

Since the rest of our smart contract functions are only for viewing the state of the contract, they are both fast and free.

Security analysis

Impersonating a Vaccination Clinic

It is possible that a rogue VC wishes to impersonate a legitimate VC to add records for people who are not vaccinated.

It is trivial to capture a VC's certificate through network sniffing or decoding the input data in each transaction.

However, we assume that a legitimate VC keeps their `p` securely. This way, despite the rogue VC having a legitimate VC's certificate, they are not able to encrypt the `passportNumber` in a way that the smart contract can decrypt the encrypted `passportNumber` and match it to an existing `passportNumber`.

We developed the smart contract to return an error message when such checks are not passed:

```
if (verifyHash(passportNumberDecrypted, passportNumberHash)) {
    passportNumbers.push(passportNumberHash);
    records[passportNumberHash] = Record({
        passportNumber: passportNumberDecrypted,
        vaccineManufacturer: vaccineManufacturer,
        country: country,
        timestamp: block.timestamp
    });

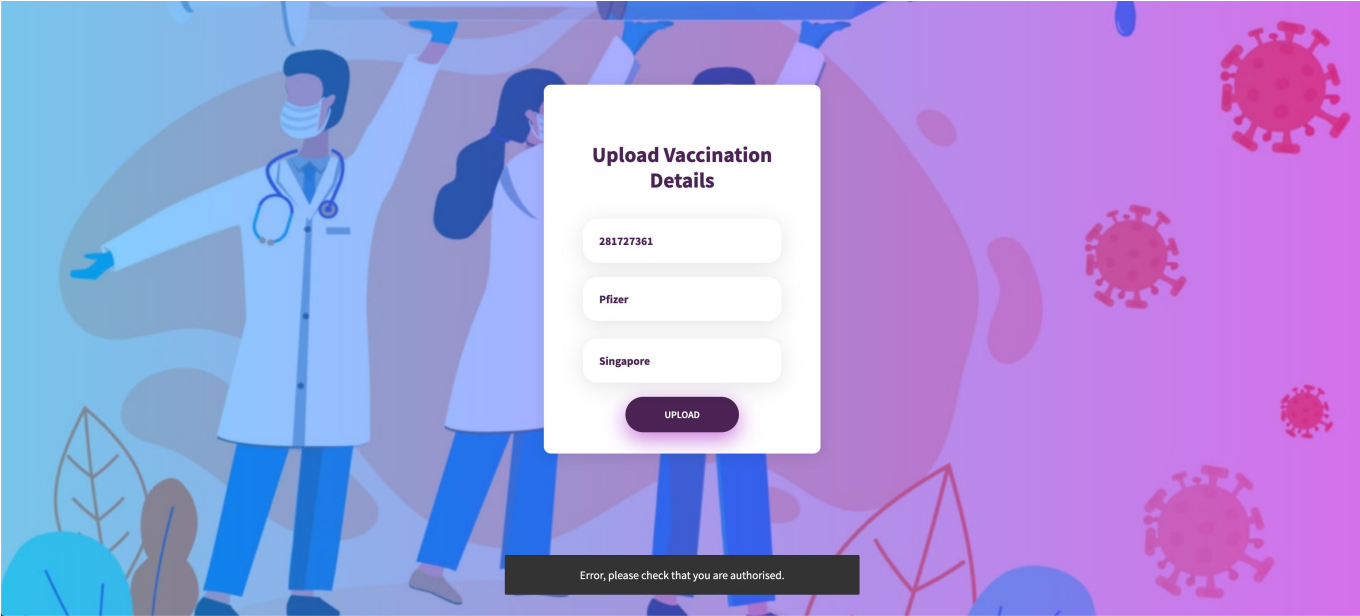
    analytics[country].totalVaccinated += 1;

    return "Record added successfully.";
} else {
    return "Error, please check that you are authorised.";
}
```

For demonstration purposes, we changed the credentials of the VC from the correct `p` of (29, 91) to (29, 90):

```
var clinicPrivateKey = 29;
var clinicModulus = 90;
```

With the incorrect p , the rogue VC quickly realises that it is not able to add any records to the smart contract:



Cryptographic complexity

We realise that the public and private keys used in the demonstration are trivial to bruteforce, compared to the typical 256 bit keys generated by openssl. This is the limitation of our particular implementation on the smart contract, as a bigger exponent will result in high gas cost, and potentially exceeding the gas limit.

Future work could include writing an efficient algorithm for calculation exponentiation, for example, using [square and multiply method](#).

Personal Contribution

| Work | Alex | Jose |
|--------------------------|------|------|
| Brainstorm | ✓ | ✓ |
| Smart contract analytics | | ✓ |
| Smart contract RSA | ✓ | |
| Frontend Clinic | ✓ | ✓ |
| Frontend Analytics | | ✓ |
| Frontend Everyone | ✓ | ✓ |
| Presentation | ✓ | ✓ |

[^1]: This accessibility and availability is contingent on the network connectivity of the user using the application.