Cryptography based Online Voting System (Encrypted Database + OCR)

Nikhil Raj (B19CSE059), Om Solanki (B19CSE061), Raunak Gandhi (B19CSE117)

Indian Institute of Technology, Jodhpur
Department of Computer Science and Engineering

Abstract: This project aims to develop a secure e-voting system to create anytime anywhere voting possible. This is an android based application. The concern of trust deficit of people in online voting is addressed by providing security and confidentiality through Cryptography. Voter's privacy and individual verifiability are at the heart of our project.

Index Terms: Database, Table, Encryption, Application, OCR

1. Idea

Any democracy is built on the foundation of voting. The voting process must be secure and reliable in order to keep democracy healthy. It is not feasible to vote from anywhere at any time with the traditional voting system. You must physically go to the polling place, stand in line, and then cast your vote. There is a lack of vote-cast verifiability, resulting in a loss of voter faith. Low voter turnout is a result of this unappealing and time-consuming process. We can get rid of this issue with the use of online voting system. Voting will be considerably easier with an online voting system. People can vote from any location. There will be no need to wait in line for your turn, so the procedure will be quick. This will encourage voters to participate in large numbers. Even when an ongoing situation occurs like coronavirus, online voting systems are spot-on solutions. Traditional voting systems are also costlier than e-voting. Voting using ballot paper or using an EVM machine also has an inherent cost of physical security included. In using these systems, human resource is used extensively. Also due to the participation of humans, there is a chance of mistakes occurring during vote tallying.
There are institutes and colleges, where Democracy exists. Thus our vision is to provide them with an online e-voting system that can help them to make required decisions and take off the overhead of arrangement of elections. The e-voting system that we are proposing will help in cutting the election cost. As everything is automated, the odds of making a mistake are none. Further, we are going to use a cryptographic method to make a voting system secure and reliable. Privacy of voters and their vote is given utmost preference. Our system also focuses on voter verifiability through the use of vote acknowledgement.

2. Aims and Objectives

- Create a full stack android application.
- Make voting feasible at any time and from anywhere.
- Provide a service to create, conduct, and announce the results of an election.
- Keeping voters' faith in the voting process while making voting more convenient.

3. Implementation

3.1. Technical Stack

- Google Firebase Firestore was used as the database for this system.
- The entire application was developed using the Android Software Development Kit.
- We used XML for the frontend and Java JDK for the backend.
- We used Google's Million Image trained Tesseract model for Optical Character Recognition (OCR) (Not an API and hence works offline).

3.2. Implementation Stages

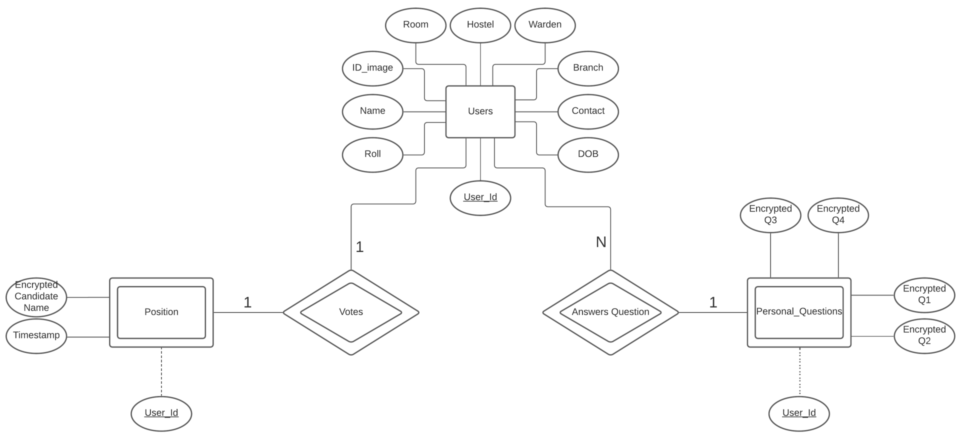
- **Wireframing**: On Whimsical, we designed the whole layout from the ground up, based on team discussions. The application's whole flow was completed at this point.
- **ER Diagram**: The database structure was developed at this point, taking into account all foreign keys, primary keys, and other aspects of each tuple.
- **Normalization and Query Optimization**: Following the creation of the ER diagram, all queries for each use case were identified, along with proper encryption and decryption for each tuple, and the required table structure was defined. As a result, normalization was done for each table, the specifics of which are provided in the next section of the report. In addition, each query's query optimization was accomplished by developing the Relational Algebra.
- **Application Programming**: Using Java as our programming language, we built the complete application from the ground up, keeping the wireframe design and query optimization in mind.
- **Testing and Debugging**: After the implementation was completed, it was given to coworkers and reviews were conducted in order to enhance overall performance. This was the only time that debugging was done.
- **Finalization**: Finally, the application was offered for demonstration.

3.3. Implementation Schedule

- On August 12th, 2021, the project was approved.
- On August 22nd, 2021, the project was discussed with the instructor.
- The Implementation (Wireframing, Normalization and Query Optimisation, Application Programming) began on September 1, 2021.
- Testing and Debugging were place from November 6th to November 12th, 2021.
- On November 15, 2021, the Finalization was completed.

4. Database Management System

4.1. ER Diagram



The following is a thorough description of the ER Diagram shown above.

- **Symbol Description**:

Symbol	Meaning
Single Rectangle	Strong Entity
Double Rectangle	Weak Entity
Single Oval	Attribute
Double Diamonds	Weak Relationship
Dashed Line	Foreign Key
Underlined Attribute	Primary Key
1 1	One-One Relationship
N 1	Many-One Relationship

- **Entities**:
  1) **Users**: It is a strong entity as it is not dependent on any other entity.
  2) **Personal_Questions**: It is a weak entity since it will rely on the User's entity, as users will select four questions from a list of seven upon registration.
  3) **Position**: It is a weak entity since the data stored there will be reliant on the users' votes when they cast their votes for a candidate.

  NOTE: There can be several positions for voting, thus we can have multiple entities for that.
- **Attributes**:
  1) **Users**:
     a) **User_Id**: It is the primary key, which is why it is underlined in the ER Diagram.
     b) **Branch**
     c) **Contact**
     d) **DOB**
     e) **Hostel**
     f) **ID_image**: If the user has provided an ID card image, it is saved.
     g) **Name**
     h) **Roll**
     i) **Room**
     j) **Warden**
  2) **Personal_Questions**:
     a) **User_Id**: It is the Foreign key from the User's table, and so a dotted line connects them.
     b) **Encrypted Q1**
     c) **Encrypted Q2**
     d) **Encrypted Q3**
     e) **Encrypted Q4**
  3) **Position**:
     a) **User_Id**: It is the Foreign key from the User's table, and so a dotted line connects

them.

b) **Encrypted Candidate Name**: The encrypted version of the user-selected candidate name.

c) **Timestamp**: The time when the user casts his or her vote for a candidate.

- **Relations**:

  1) **Votes**: It's a weak relationship between the User entity and the Position entity since it's a weak relationship between a strong entity and a weak entity. It's a one-to-one relationship since each user may only vote for one position at a time.
  2) **Answers Question**: It is a relationship between the User entity and the Personal_Questions entity, and it is a weak relationship since it is a relationship between a strong entity and a weak entity. Because a set of questions might be the same for several users, it's a many-one relationship.

4.2. Normalization

- **Schema**:
  Users (User_Id, Branch, Contact, DOB, Hostel, ID_image, Name, Warden, Room, Roll)
  Personal_Questions (User_Id, (Encrypted) Q1, (Encrypted) Q2, (Encrypted) Q3, (Encrypted) Q4)
  Position (User_Id, (Encrypted) Candidate Name, Timestamp)

  1) **1NF**: The provided Schema is in 1NF since it only includes atomic values and the table has no attributes that may hold multiple values.
  2) **2NF**: All non-key attributes (such as Branch, Contact, DOB, Hostel, and so on) are totally functionally dependent on the primary key (User_Id) in the provided Schema.

     Closure for Users Table:
     User_Id += {User_Id, Branch, Contact, DOB, Hostel, ID_image, Name, Warden, Room, Roll}

     Closure for Personal_Questions Table:
     User_Id += {User_Id, (Encrypted) Q1, (Encrypted) Q2, (Encrypted) Q3, (Encrypted) Q4}

     Closure for Position Table:
     User_Id += {User_Id, (Encrypted) Candidate Name, Timestamp}

     Here, User_Id is primary key.
     Subset of User_Id = $\{\phi\}$
     Since, the subset of User_Id is not a primary key so it is a candidate key.
     Furthermore, because our candidate key is atomic (i.e. prime attributes of candidate key = $\{\phi\}$), partial dependencies are not possible. As a result, the non-prime attribute is not functionally dependent on a candidate key component.
  3) **3NF**: The provided Schema is in 3NF and does not contain transitive partial dependencies, as it is in 2NF. X is a super key or Y is a prime attribute in any X $\rightarrow$ Y relationship. Our X will always be User_Id, and we know it's a super key. As a result, the first requirement, that X is a super key, is satisfied. As a result, no transitive partial dependencies exist.
  4) **BCNF**: The provided Schema is in BCNF, just like it is in 3NF, and for every functional dependence X $\rightarrow$ Y, X should always be the table's super key, and as we saw before in 3NF, it already follows the condition, thus it is likewise in BCNF form.

4.3. Queries for Different Use Cases

1) **Choosing Eligible Candidate**:

SQL Query:

```
SELECT User_Id FROM Users;
WHERE Hostel = ‘Hostel Name’;
```

Relational Algebra:

$$\Pi_{User\_Id}(\sigma_{Hostel=HostelName}(Users))$$

2) **Fetching Questions during Voting**:

SQL Query:

```
Decrypt(
SELECT Questions
FROM Personal_Questions,Users
WHERE Users.User_Id = Personal_Questions.User_Id)
```

Relational Algebra:

$$D(\Pi_{Questions}(\sigma_{Users.User\_Id=Personal\_Questions.User\_Id}(UsersXPersonal\_Questions)))$$

3) **Finding Winner**:

SQL Query:

```
CREATE TABLE Temp(
  User_ID varchar(50) Key varchar(4)
) INSERT INTO Temp (
  SELECT
    User_Id,
    Make_Key (
      Decrypt(Personal_Questions.Question1),
      Decrypt(Personal_Questions.Question2),
      Decrypt(Personal_Questions.Question3),
      Decrypt(Personal_Questions.Question4)
    )
  FROM
    Position,
    Personal_Questions
  WHERE
    Position.User_Id = Personal_Questions.User_Id
)
```

Now that we have a table with the User IDs and associated keys, we can decrypt the votes of the users and determine the winner.

```
FROM Position,Temp
WHERE Position.User_Id=Temp.User_Id
CASE
    WHEN Decrypt_2 (Position.Candidate,Key) = “Candidate1” THEN C1++
    WHEN Decrypt_2 (Position.Candidate,Key) = “Candidate2” THEN C2++
    WHEN Decrypt_2 (Position.Candidate,Key) = “Candidte3” THEN C3++

END

CREATE TABLE Result(
    Name varchar(20);
    Votes int;
);
INSERT INTO Result VALUES ((“Candidate1”,C1),(“Candidate2”,C2
    ),(“Candidate3”,C3));
SELECT * FROM Result ORDER BY Votes DESC;
```

4) **Verifying Answers during Voting**:
Store the answers in Personal_Questions table.
Make a temporary table to store the user’s answer during the verification process. Let say table T1.

SQL Query:

```
FROM Personal_Questions,T1
CASE
WHEN Personal_Questions.Question1 = T1.Question1 AND
    Personal_Questions.Question2 = T1.Question2 AND
    Personal_Questions.Question3 = T1.Question3 AND
    Personal_Questions.Question4 = T1.Question4 THEN ‘Thanks You for
    your Time’

ELSE ‘Incorrect’
END
```

5) **Voting Once by User**:

SQL Query:

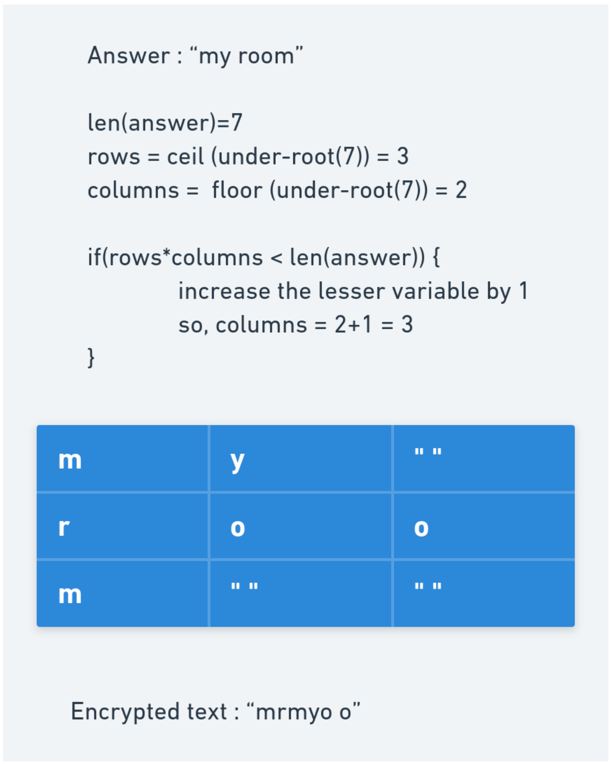
```
FROM Users,Position
CASE
WHEN Users.User_Id EXISTS IN Position.User_Id THEN DISABLE
    (Vote_Button)

ELSE ‘Thanks You for your Time’
END
```

NOTE: We’ve developed our DISABLE() function, which accepts UI buttons as input and disables them when called.

5. Encrpytion Algorithm Used

Three separate encryption algorithms are used to ensure varying levels of security at each stage.

- **Encrypting question with Prime Numbers**: Because the user was given seven questions, each one was given a unique prime number as a label, and instead of saving the complete question in the database, the binary form of those prime numbers was saved. This guarantees that no one can tell which question the user has selected to answer by looking at the database.
- **Encrypting answers with 2-Dimensional Matrix**: The user's replies were encrypted using the matrix, which was filled with the response column wise and saved in the database by reading it row wise as the second degree of protection.
  This strategy is demonstrated in the figure below.



```
Answer : "my room"

len(answer)=7
rows = ceil (under-root(7)) = 3
columns =  floor (under-root(7)) = 2

if(rows*columns < len(answer)) {
            increase the lesser variable by 1
            so, columns = 2+1 = 3
}
```

m	y	" "
r	o	o
m	" "	" "

Encrypted text : "mrmyo o"

- **Encrypting candidate name with Vigenère Cipher**: We construct a key using the answers to the user's specified questions as the last degree of protection, and then use that key alone to decrypt the candidate's name.

6. Application Flow

The application flow for giving a vote in a step-by-step way is discussed in this section.

1) **Google Authentication**: The user will first authenticate using the institute's email address.
2) **OCR**: The user will take a picture of his or her institute's ID card, and all of the relevant data will be filled up immediately.
3) **Further details**: After OCR, the user will submit his or her hostel name, contact number, and other information before moving on to the next step.
4) **Personal Questions**: The user will select four of their favourite questions from a selection of seven and offer an answer to each of them. The chosen questions and their answers will be encrypted before being saved in the Personal_Questions table.

5) **Voting Room Selection**: After filling out all of the required information, the user will select the hostel post he or she want to vote for. One thing to keep in mind is that if a person does not belong to the hostel for which he or she is voting, they will not be permitted to visit the lobby. To gain access, the user must supply biometric data via their device hardware. They can enter the voting room after they have been validated.
6) **Voting Room**: The user will choose his or her preferred candidate to vote for and then press the vote button. After pressing the vote button, he or she must respond to all of the personal questions they completed throughout the registration procedure. If all of the answers are right, their response will be saved in the hostel table. It's worth noting that the user's vote for a candidate is encrypted using their own personal question-answer as the key. So, if someone tries to gain unauthorised access to the database in order to learn the name of the candidate for whom the user voted, he or she must first decrypt the answer from the Personal_ Questions table, then create a key using all possible combinations of answers, and then use that key to decrypt the candidate name. It's worth noting that the hacker is unaware that a key is required to decrypt the candidate's name.
7) **Result Declaration**: Finally, after the voting timer runs out on the voting room selection activity, the winner is announced, with his or her name and total number of votes displayed on the screen. Even still, the encryption and decryption processes described before must be carried out in order to complete this operation.

7. Conclusion

- The IITJ Voting System demonstrates that a secure voting service may be provided via the Internet.
- The IITJ Voting System has the potential to minimise election costs. This applies to both financial and human resources.
- This system's efficiency is demonstrated by its fair and limited use of resources.
- The IITJ Voting System provides simple and easy to use voting service.
- The scope of this system is limited to elections within an institute or college. Using this system in large elections like elections in state/country may cause scalability issues.

8. Future Scope

- To make the database more secure and reliable, we're considering using BlockChain technology.
- A fundamental challenge is the scalability of the IITJ Voting System. We'd like to increase the system's scalability so that it can be utilised in large-scale elections.
- We're considering using client-side computing to boost scalability.
- To make the data more safe, we're considering improving the encryption mechanism.
- More attributes and features can be added to this prototype.

Github Repository Link: IITJ Voting System