Professor: Massimo Villari

Project Advisor: Mario Colosi

Student: Arya Khosravirad

Matricola: 534 170

Università degli Studi di Messina

End-to-End Encrypted Real-Time Chat Application

Table Of Contents

Introduction

I built this chat application with a focus on security using End to End Encryption. All encryption and decryption happen only on the user's device (client-side) and not on the server. When a user sends a message it gets encrypted on their device using the recipient's public key. The encrypted message is then sent to the server to be stored(no encryption here). Only the recipient can decrypt the message on their device with their private key which is never shared with the server.

During user registration each user generates an RSA key pair directly on their device. The public key is sent to the server for sharing with other users while the private key stays on the user's device and will be downloaded as a .pem file.

Since all encryption and decryption happen on the client side the server never sees the private key or the original message. Even if the server is hacked the data remains secure and protected.

System Architecture

This chat application is built with three main parts:

client side: I used html, css, and javascript with the web crypto API handling encryption and decryption. Every user generates their RSA key pair (public and private keys) directly on their device during registration. The public key is sent to the server, while the private key is downloaded as a .pem file and stays securely on the user's device. When someone sends a message, it’s encrypted using the recipient’s public key before being sent to the server. Only the recipient can decrypt it on his device using their private key.

server side: I built the server with php and mysql it handles storing encrypted messages and public keys. It cannot decrypt messages because it never has access to private keys. The server’s role is limited to secure storage and message delivery between users.

Database: I kept the database simple. The users table stores user credentials and public keys while the messages table stores encrypted messages with the sender and recipient IDs. no plaintext messages or private keys are ever saved in the database.

Client A => Encryption => Server => Client B => Decryption

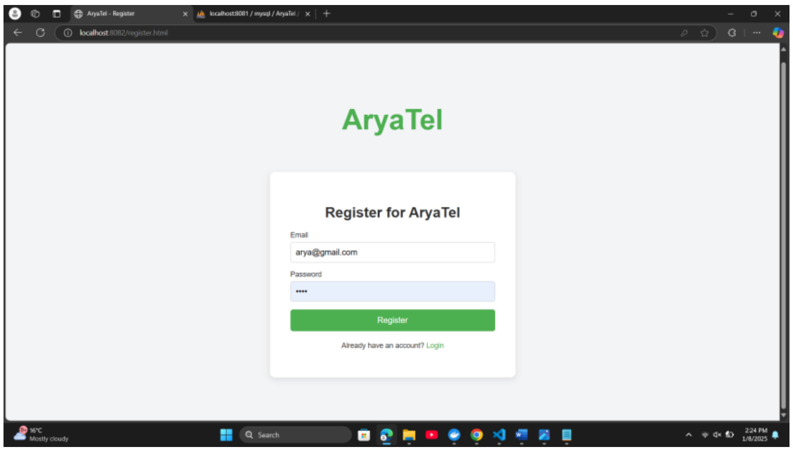
(server never does any encryption just client does encryption)

To make deployment easier I containerized the app with Docker.

Functionality:

1. User Registration and Key Generation:

First user register and key generation happens here because it ensure that every user has a unique rsa key pair for end to end encryption. This process happens entirely on the client-side because of security reasons.



AryaTel

Register for AryaTel

Email: arya@gmail.com

Password: ••••

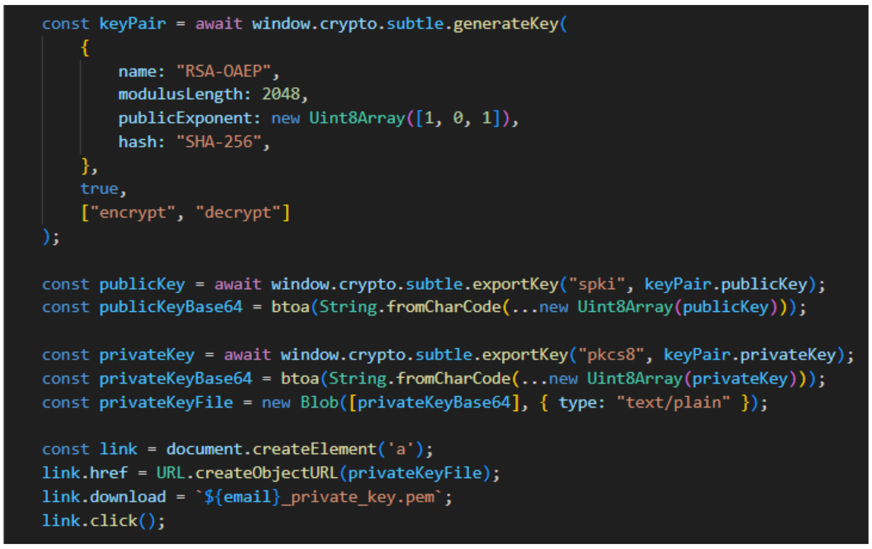
Register

Already have an account? Login

RSA Key Pair Generation (Client-Side):

when a user registers an rsa key pair is generated on their device using the web crypto API.

the public key is sent to the server to be stored in the database and private key is downloaded as a .pem file` and saved on the user's device to be used later.

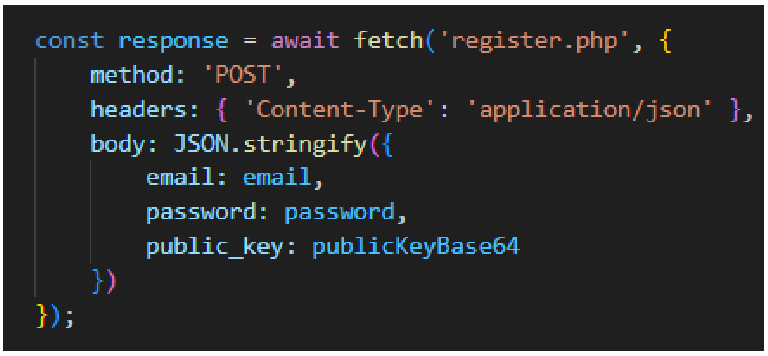


```
const keyPair = await window.crypto.subtle.generateKey(
    {
        name: "RSA-OAEP",
        modulusLength: 2048,
        publicExponent: new Uint8Array([1, 0, 1]),
        hash: "SHA-256",
    },
    true,
    ["encrypt", "decrypt"]
);

const publicKey = await window.crypto.subtle.exportKey("spki", keyPair.publicKey);
const publicKeyBase64 = btoa(String.fromCharCode(...new Uint8Array(publicKey)));

const privateKey = await window.crypto.subtle.exportKey("pkcs8", keyPair.privateKey);
const privateKeyBase64 = btoa(String.fromCharCode(...new Uint8Array(privateKey)));
const privateKeyFile = new Blob([privateKeyBase64], { type: "text/plain" });

const link = document.createElement('a');
link.href = URL.createObjectURL(privateKeyFile);
link.download = `${email}_private_key.pem`;
link.click();
```

Secure Transmission to Server:

Along with the public key the user's email and hashed password are sent to the server during registration.

Passwords are hashed before being saved in the database to prevent unauthorized access.

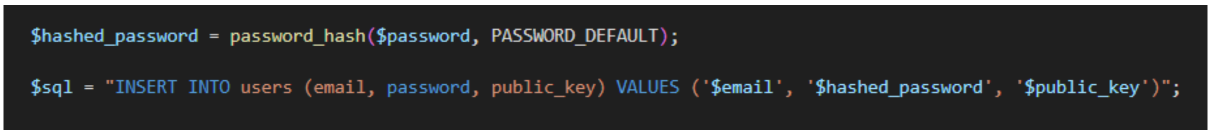


```javascript
const response = await fetch('register.php', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        email: email,
        password: password,
        public_key: publicKeyBase64
    })
});
```

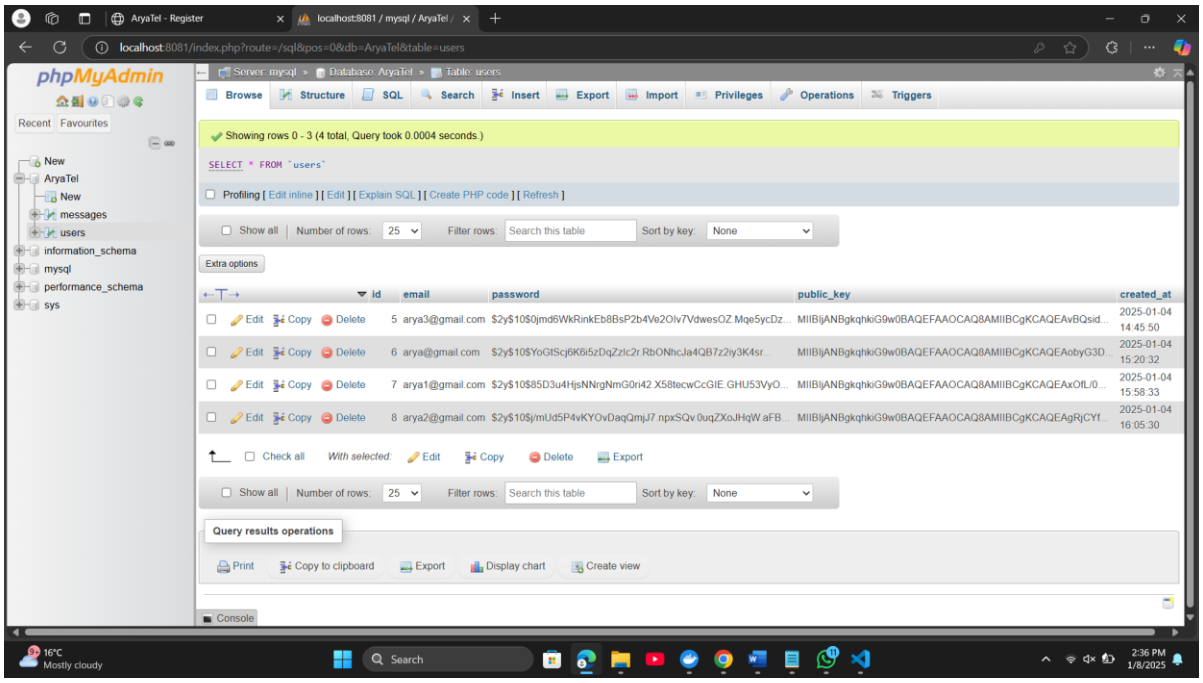
Server-Side Storage:

On the server the public key and hashed password are securely stored in the users table of the database.

The server never stores or sees the private key so it cannot decrypt any messages.



```php
$hashed_password = password_hash($password, PASSWORD_DEFAULT);

$sql = "INSERT INTO users (email, password, public_key) VALUES ('$email', '$hashed_password', '$public_key')";
```



Showing rows 0 - 3 (4 total, Query took 0.0004 seconds.)

```
SELECT * FROM `users`
```

id	email	password	public_key	created_at
5	arya3@gmail.com	$2y$10$0jmd6WkRinkEb8BsP2b4Ve2Olv7VdwesOZ.Mqe5ycDz...	MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAvBQsid...	2025-01-04 14:45:50
6	arya@gmail.com	$2y$10$YoGtScj6K6i5zDqZzIc2r.RbONhcJa4QB7z2iy3K4sr...	MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAobyG3D...	2025-01-04 15:20:32
7	arya1@gmail.com	$2y$10$85D3u4HjsNNrgNmG0ri42.X58tecwCcGIE.GHU53VyO...	MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAxOfL/0...	2025-01-04 15:58:33
8	arya2@gmail.com	$2y$10$j/mUd5P4vKYOvDaqQmjJ7.npxSQv.0uqZXoJHqW.aFB...	MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAgRjCYf...	2025-01-04 16:05:30

2. End-to-End Encryption (E2EE) Workflow:

I ensured that the server acts just as a middleman because real end to end encryption should be like that

This is achieved by leveraging RSA asymmetric encryption with client-side encryption and decryption.

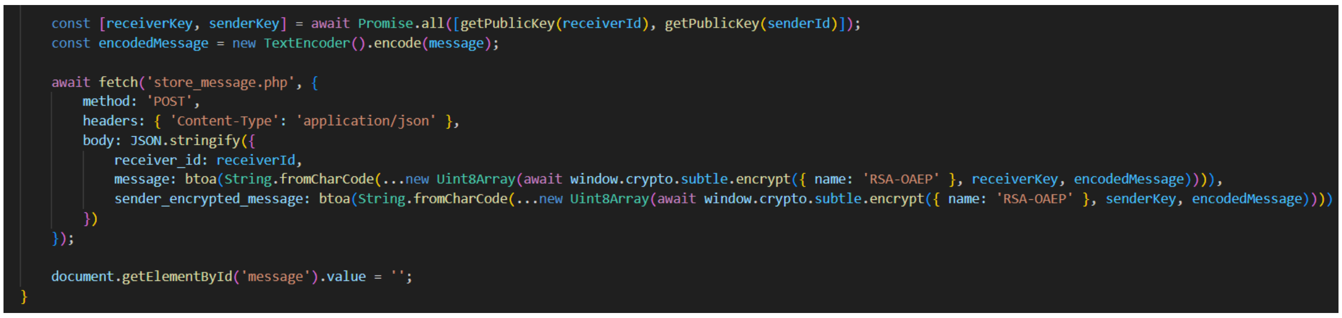
Fetching the Recipient's Public Key:

When a user wants to send a message their browser fetches the public key of the recipient from the server using the get_public_key.php API.

```php
$receiver_id = intval($_GET['receiver_id']);
$sql = "SELECT public_key FROM users WHERE id = ?";
$stmt = $conn->prepare($sql);
$stmt->bind_param("i", $receiver_id);
$stmt->execute();
$result = $stmt->get_result();
```

Encrypting Messages on the Client-Side:

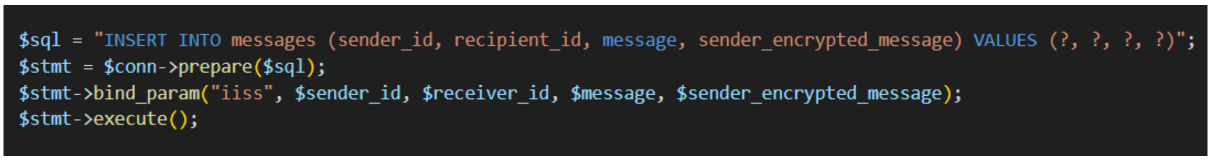
The sender encrypts the message using the recipient's public key before sending it to the server. the message is self encrypted also(explained in part 4).



```
    const [receiverKey, senderKey] = await Promise.all([getPublicKey(receiverId), getPublicKey(senderId)]);
    const encodedMessage = new TextEncoder().encode(message);

    await fetch('store_message.php', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
            receiver_id: receiverId,
            message: btoa(String.fromCharCode(...new Uint8Array(await window.crypto.subtle.encrypt({ name: 'RSA-OAEP' }, receiverKey, encodedMessage)))),
            sender_encrypted_message: btoa(String.fromCharCode(...new Uint8Array(await window.crypto.subtle.encrypt({ name: 'RSA-OAEP' }, senderKey, encodedMessage)))
        })
    });

    document.getElementById('message').value = '';
}
```

Storing Encrypted Messages:

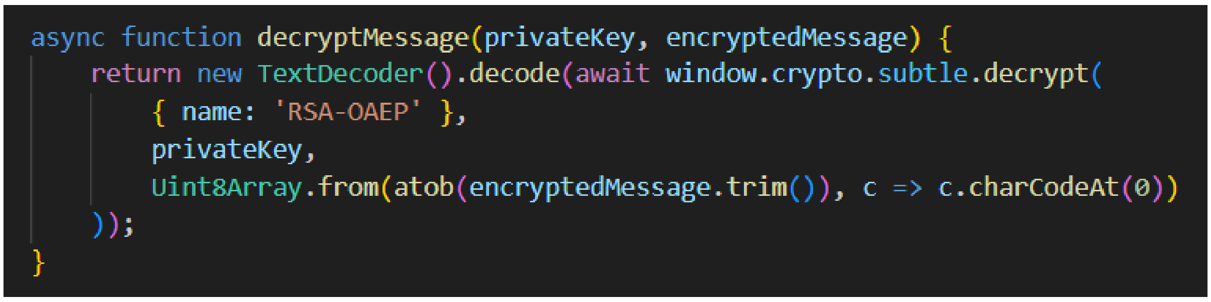
The server stores only the encrypted message and cannot decrypt it.



```
$sql = "INSERT INTO messages (sender_id, recipient_id, message, sender_encrypted_message) VALUES (?, ?, ?, ?)";
$stmt = $conn->prepare($sql);
$stmt->bind_param("iiss", $sender_id, $receiver_id, $message, $sender_encrypted_message);
$stmt->execute();
```

Decrypting Messages on the Client-Side:

The recipient fetches encrypted messages from the server. And decrypts it locally on their device with their private key.

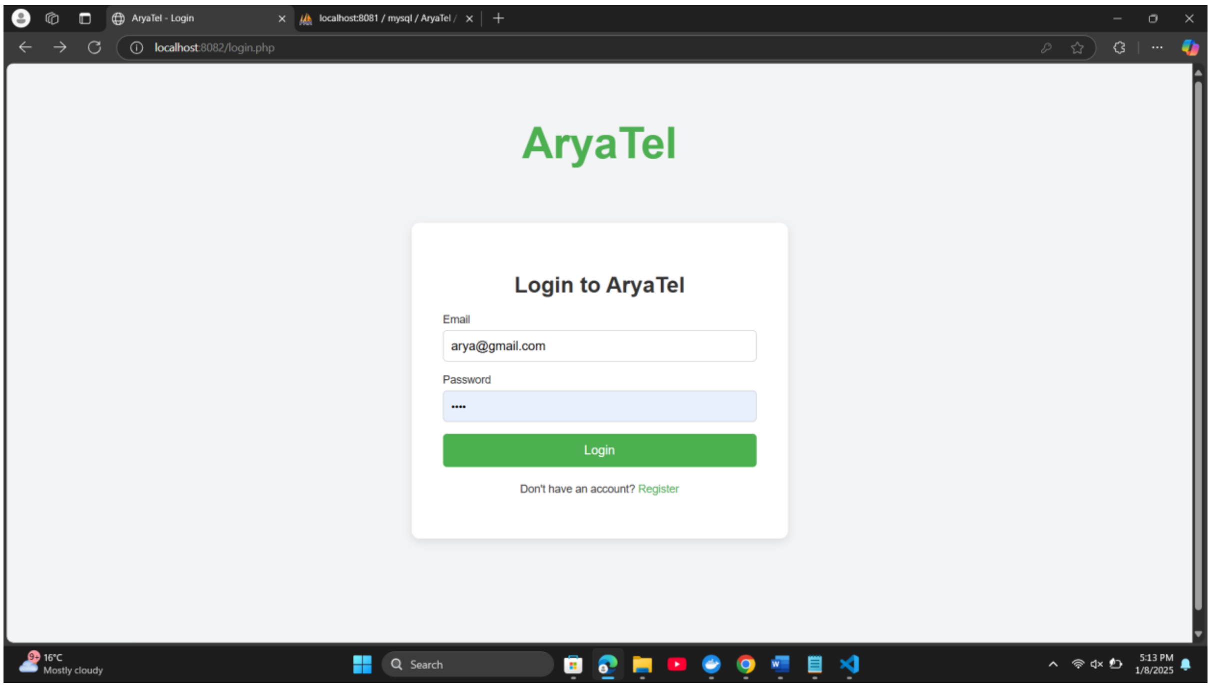


```
async function decryptMessage(privateKey, encryptedMessage) {
    return new TextDecoder().decode(await window.crypto.subtle.decrypt(
        { name: 'RSA-OAEP' },
        privateKey,
        Uint8Array.from(atob(encryptedMessage.trim()), c => c.charCodeAt(0))
    ));
}
```

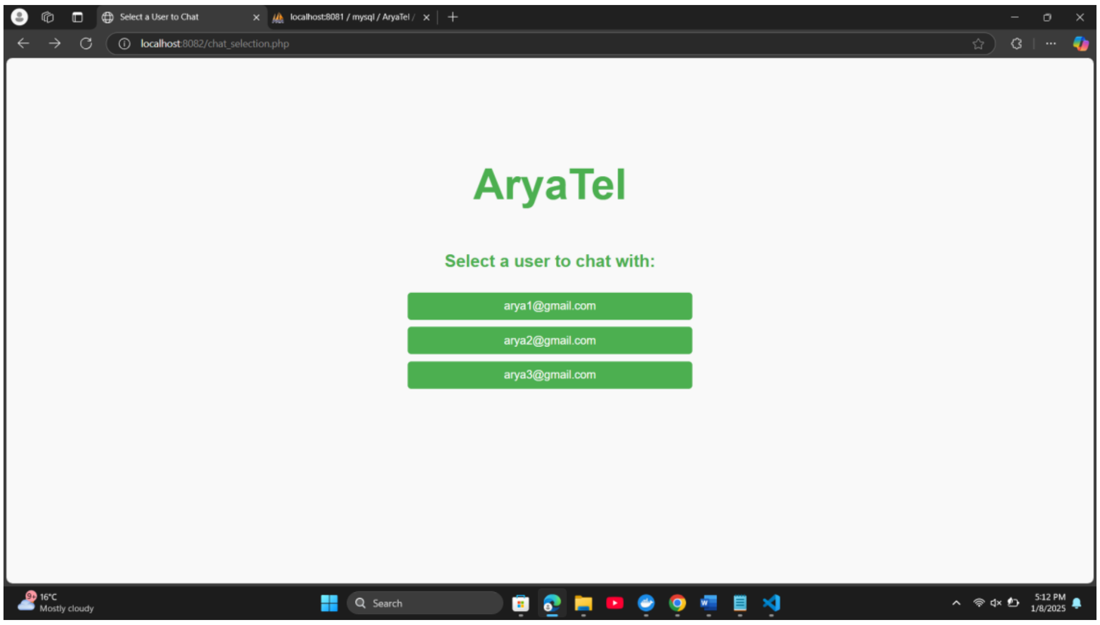
3. Secure Chat Session Setup

When a user logs into AryaTel, a secure chat session is established to ensure safe and encrypted communication. This process involves three main steps:

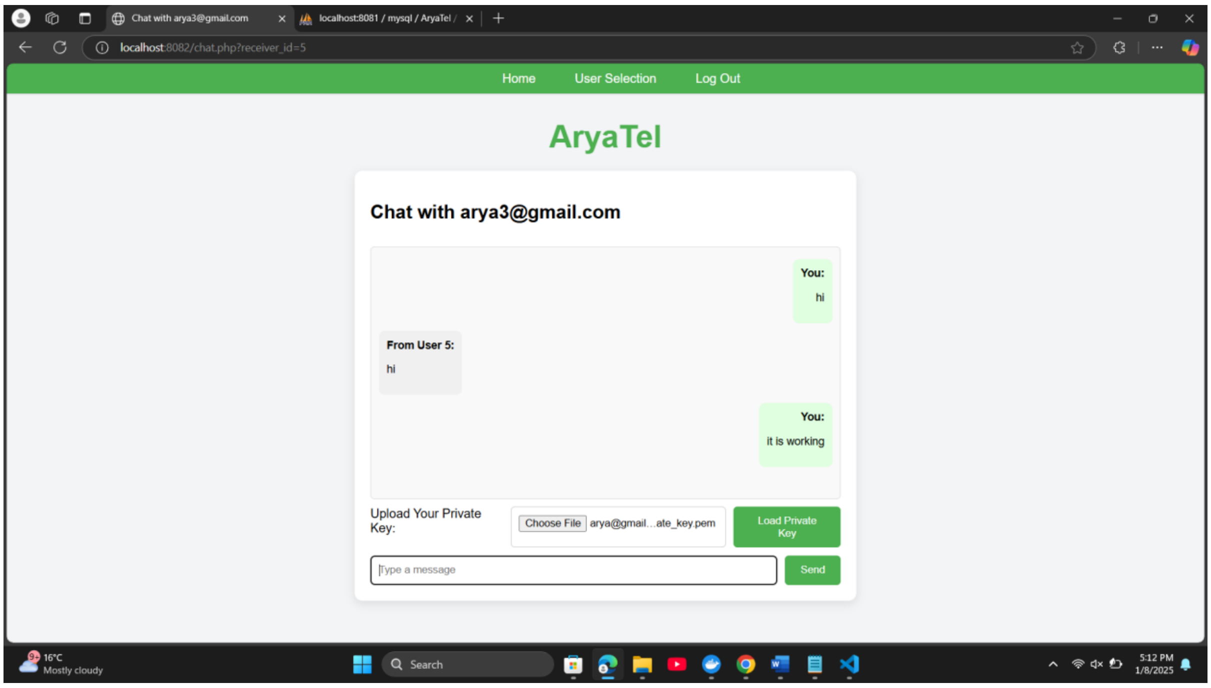
1. **User Login:**
   The user provides their email and password to securely access the platform.



2. **Recipient Selection:**
   After login, the user selects a recipient from the list of registered users.



3. **Chat Initialization:**
   Once a recipient is selected the system fetches the recipient's public key and prepares the chat interface. The user's private key is loaded locally for decrypting incoming messages ensuring that all encryption and decryption happen on the client side.



Chat with arya3@gmail.com

You:
hi

From User 5:
hi

You:
it is working

Upload Your Private Key: arya@gmail...ate_key.pem

4. Message Retrieval Problem and Solution

The Problem:

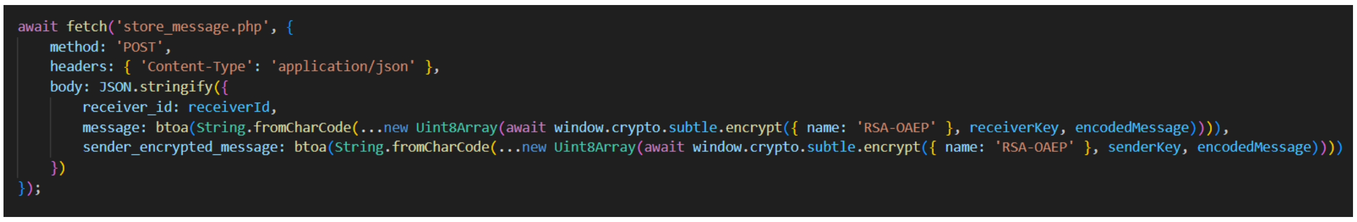
If messages were only encrypted using the recipient's public key and saved in the database then how could the sender decrypt their own messages later??
they could not because they don't have the recipient's private key.

This means the sender would lose access to their own messages. :(

The Solution: Dual Encryption

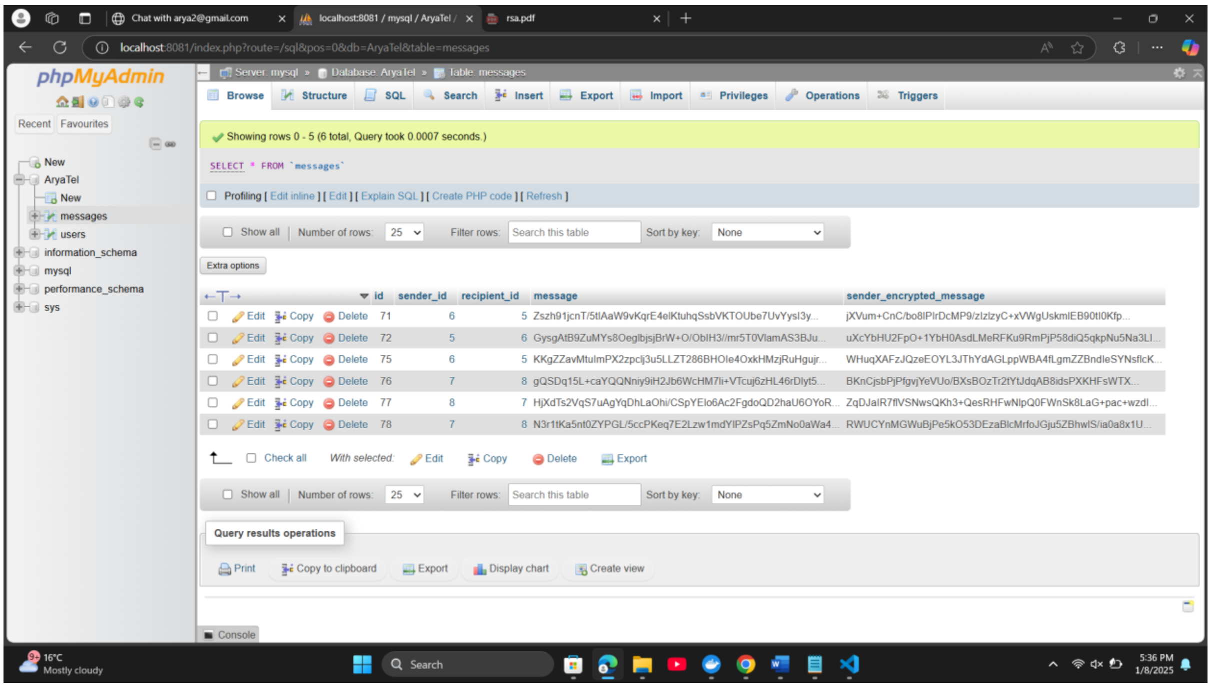
We know that the message is encrypted using the recipient's public key to ensure only the recipient can decrypt it.

But I also encrypted the message using the sender's public key before being sent to the server. This allows the sender to decrypt their own message later using their private key.



```
await fetch('store_message.php', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
        receiver_id: receiverId,
        message: btoa(String.fromCharCode(...new Uint8Array(await window.crypto.subtle.encrypt({ name: 'RSA-OAEP' }, receiverKey, encodedMessage)))),
        sender_encrypted_message: btoa(String.fromCharCode(...new Uint8Array(await window.crypto.subtle.encrypt({ name: 'RSA-OAEP' }, senderKey, encodedMessage))))
    })
});
```

The message column holds the recipient encrypted version of the message too which is in sender_encrypted_message column



Showing rows 0 - 5 (6 total, Query took 0.0007 seconds.)

```
SELECT * FROM `messages`
```

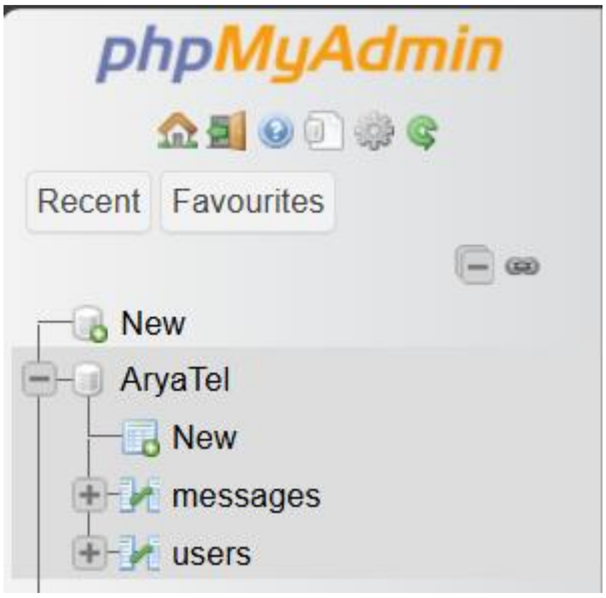
id	sender_id	recipient_id	message	sender_encrypted_message
71	6	5	Zszh91jcnT/5tlAaW9vKqrE4elKtuhqSsbVKTOUbe7UvYysI3y...	jXVum+CnC/bo8lPIrDcMP9/zlzlzyC+xVWgUskmlEB90tl0Kfp...
72	5	6	GysgAtB9ZuMYs8OeglbjsjBrW+O/ObIH3//mr5T0VlamAS3BJu...	uXcYbHU2FpO+1YbH0AsdLMeRFKu9RmPjP58diQ5qkpNu5Na3LI...
75	6	5	KKgZZavMtuImPX2zpclj3u5LLZT286BHOle4OxkHMzjRuHgujr...	WHuqXAFzJQzeEOYL3JThYdAGLppWBA4fLgmZZBndleSYNsflcK...
76	7	8	gQSDq15L+caYQQNniy9iH2Jb6WcHM7li+VTcuj6zHL46rDlyt5...	BKnCjsbPjPfgvjYeVUo/BXsBOzTr2tYtJdqAB8idsPXKHFsWTX...
77	8	7	HjXdTs2VqS7uAgYqDhLaOhi/CSpYElo6Ac2FgdoQD2haU6OYoR...	ZqDJalR7flVSNwsQKh3+QesRHFwNlpQ0FWnSk8LaG+pac+wzdl...
78	7	8	N3r1tKa5nt0ZYPGL/5ccPKeq7E2Lzw1mdYlPZsPq5ZmNo0aWa4...	RWUCYnMGWuBjPe5kO53DEzaBlcMrfoJGju5ZBhwlS/ia0a8x1U...

This approach ensures maintaining true end-to-end encryption (E2EE) (:

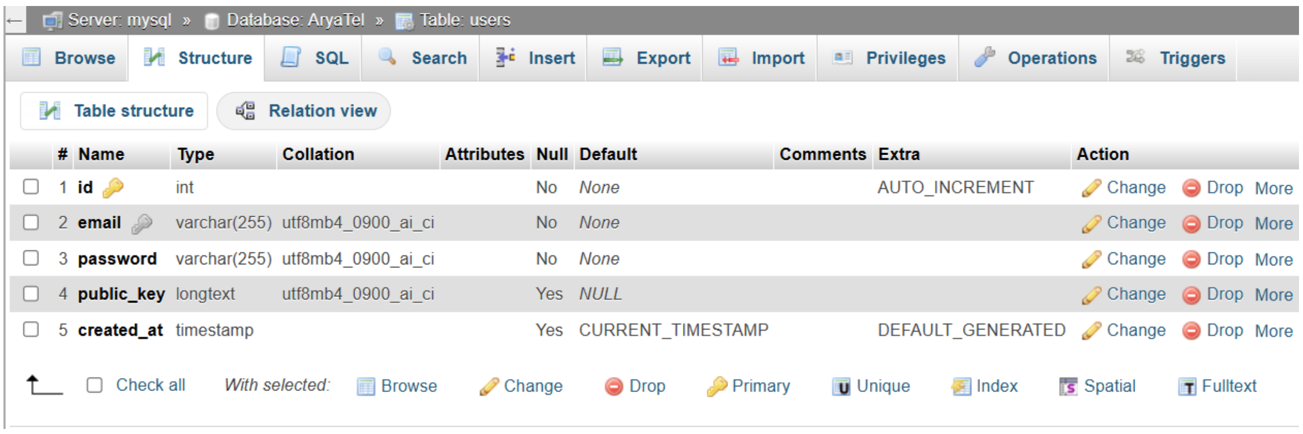
Database Design

I used docker and phpmyadmin to set up the database for my chat application.

AryaTel's database consists of two main tables:



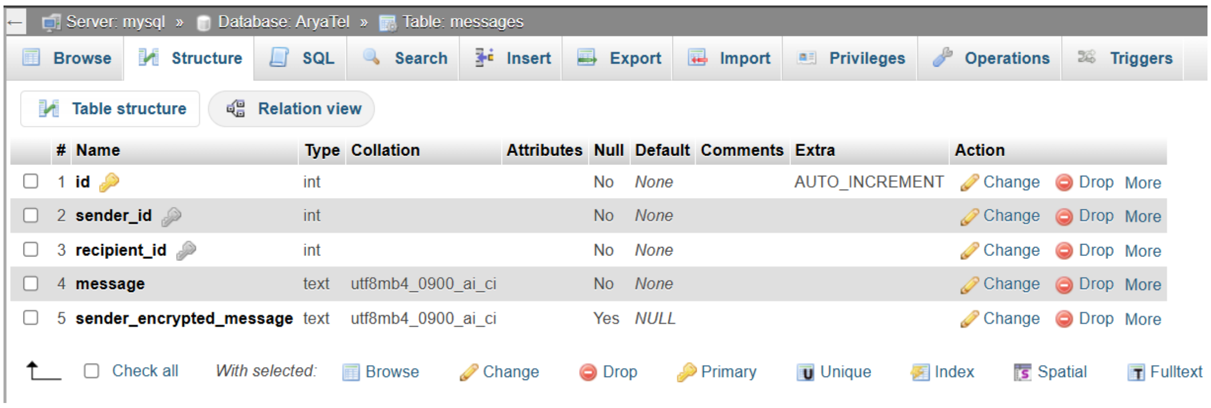
users: Each user has a unique entry in this table including their public key ensuring that only authorized users with corresponding private keys can decrypt



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int			No	None		AUTO_INCREMENT	Change Drop More
2	email	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
3	password	varchar(255)	utf8mb4_0900_ai_ci		No	None			Change Drop More
4	public_key	longtext	utf8mb4_0900_ai_ci		Yes	NULL			Change Drop More
5	created_at	timestamp			Yes	CURRENT_TIMESTAMP		DEFAULT_GENERATED	Change Drop More

messages: Stores encrypted messages exchanged between users.

Messages are encrypted twice once with the recipient's public key to ensure only they can decrypt the message. And another time with the sender's public key to allow the sender to retrieve and verify their sent messages.


Server: mysql » Database: AryaTel » Table: messages
Browse Structure SQL Search Insert Export Import Privileges Operations Triggers
Table structure Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	int			No	*None*		AUTO_INCREMENT	Change Drop More
2	sender_id	int			No	*None*			Change Drop More
3	recipient_id	int			No	*None*			Change Drop More
4	message	text	utf8mb4_0900_ai_ci		No	*None*			Change Drop More
5	sender_encrypted_message	text	utf8mb4_0900_ai_ci		Yes	*NULL*			Change Drop More

Check all *With selected:* Browse Change Drop Primary Unique Index Spatial Fulltext

6. Security Measures

I’ve implemented several security measures to ensure that messages and user data remain safe.

1. End-to-End Encryption (E2EE)

Every message is encrypted twice before being sent to the server. First it’s encrypted using the recipient's public key so only they can read it. Second it’s encrypted using the sender's public key so I can also retrieve and read my messages later. The server only stores these encrypted messages, and it cannot decrypt them.

2. Secure Password Storage

User passwords are never stored as plain text. Instead they are encrypted using a secure hashing method before being saved in the database. This ensures that even if someone accesses the database they can’t see the original passwords.

3. Client Side Decryption

Messages are always decrypted on the user's device using their private key. The server never sees the original content of the messages keeping all conversations fully private.

5. SQL Injection Prevention

To protect the database from malicious attacks I used prepared statements and proper query validation. This prevents hackers from injecting harmful SQL code into the database.

6. User Authentication and Session Management

Users need to log in securely with their email and password. Sessions are managed ensuring only authenticated users can access their accounts and data.

7. Conclusion

In this project I built a secure chat system where encryption and decryption happen only on the client side. Each user generates their RSA key pair on their own device and the private key never leaves their device. Only the public key is sent to the server for secure key exchange.

The key exchange process works well allowing users to share public keys and encrypt messages securely. I used dual encryption so both sender and recipient can decrypt and verify messages without relying on the server.

The server only stores encrypted messages and cannot decrypt them keeping the communication private so I successfully implemented end-to-end encryption with client-side security and secure key exchange, making the chat system safe and reliable.

REFERENCES

1. Web Cryptography API Documentation
2. PHP Official Documentation
3. MySQL Documentation
4. RSA Encryption Algorithm
5. JavaScript Fetch API
6. Best Practices for End-to-End Encryption (E2EE)
7. Stack Overflow Discussions
8. Official PHP PDO Documentation