The Freedom Layer

Introduction

The Freedom Layer is a secure messaging app built with SwiftUI that uses end-to-end encryption to protect your messages. It lets you communicate freely and safely, ensuring only the intended recipient can read your conversations.

Community Impact & Values

This project shows a real impact on my community by giving people a safe and trustworthy way to express themselves. The app protects messages so that everyone can share their thoughts without fear of their information being stolen or misused. It demonstrates compassion by keeping users safe, inclusivity by allowing everyone to join in, integrity by using strong encryption to guard privacy, and creativity by turning complex technology into an easy-to-use tool that lets every voice be heard.

UN Sustainable Goal

This project is related to UN Goal 16 because it supports strong and transparent institutions by protecting freedom of speech and user privacy. The secure messaging app gives people a safe way to share their ideas and express their opinions without fear of censorship or surveillance. This helps build trust in government and other institutions while allowing every person to participate in open and fair discussions. The project makes an impact by ensuring that everyone has the right to speak freely and keep their personal information private, which is essential for a healthy and just society.

Problem Statement & Objectives

What’s the problem?

Traditional messaging platforms often compromise user privacy and expose sensitive data to unauthorized access, limiting free speech and trust.

Project Objectives

- Develop a secure messaging app with state-of-the-art end-to-end encryption.
- Leverage elliptic curve cryptography and authenticated ciphers for uncompromised data integrity.
- Establish a self-hosted backend that enforces strict security policies while promoting community trust.

High-Level Architecture

- iOS app developed in SwiftUI
- Utilizes Apple’s CryptoKit for advanced ECC and cipher operations
- Private keys securely stored using the iOS Keychain
- Supabase managing authentication, a PostgreSQL database (profiles and messages), and secure, encrypted file storage
- Uses CoreImage and VisionKit for seamless public key sharing

How the app works?

1.

Key generation and secure storage on the client.

2.

End-to-end encryption of messages before transmission.

3.

Secure decryption at the recipient’s end with immediate message deletion post-delivery.

Advanced Cryptography

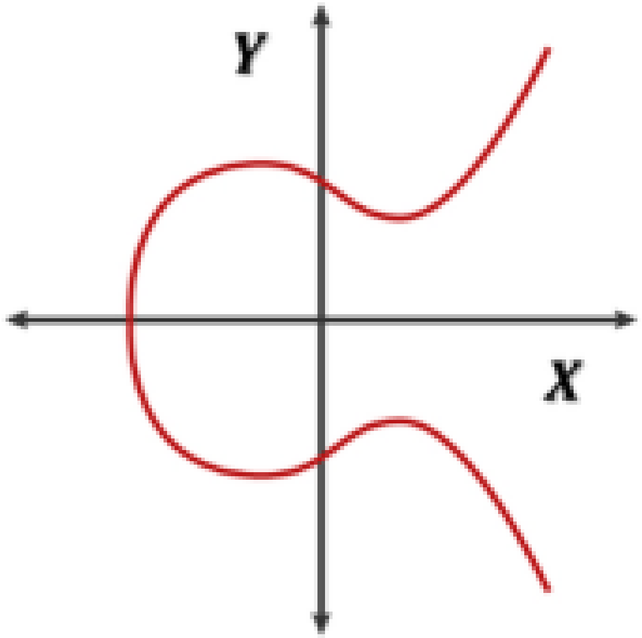
Elliptic Curve Cryptography (ECC)

- The project employs Curve25519, a modern elliptic curve known for its efficiency and high security. Its compact key sizes and excellent performance make it ideal for mobile environments.
- Using ECC enables Elliptic Curve Diffie-Hellman (ECDH), through which two parties derive a shared secret.

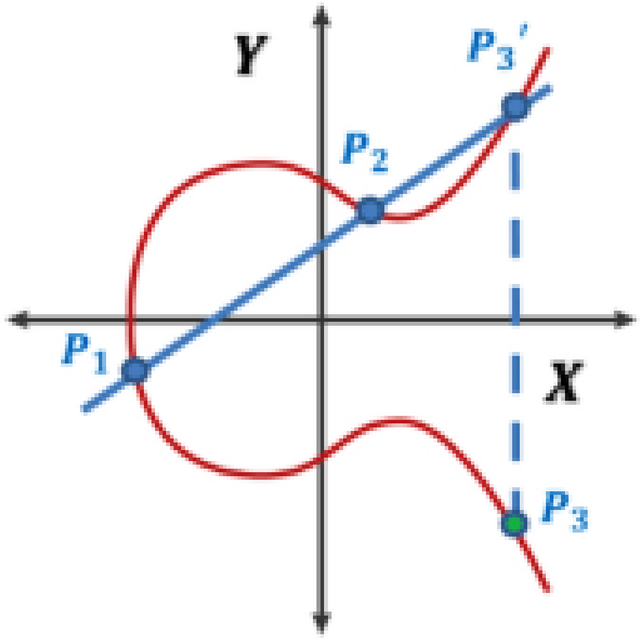
```swift
// Generate key pair using Curve25519
let privateKey = Curve25519a.KeyAgreement.PrivateKey()
let publicKey = privateKey.publicKey

// Derive shared secret using ECDH
let sharedSecret = try privateKey.sharedSecretFromKeyAgreement(with: contactPublicKey)
```

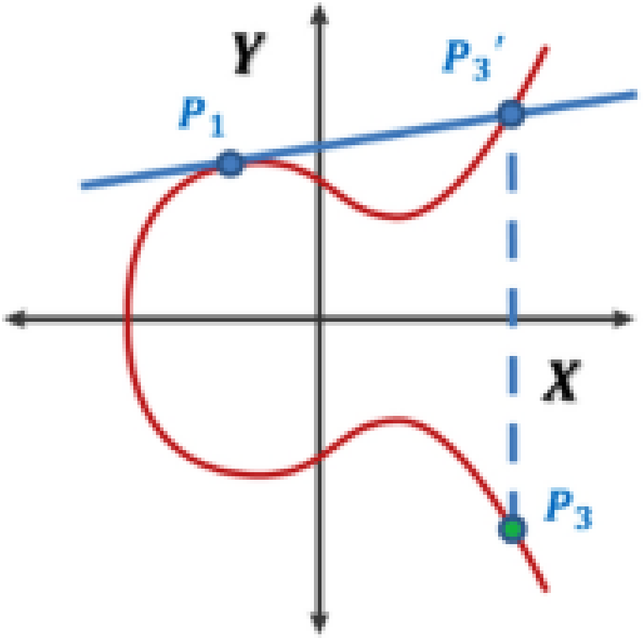
$y^2 = x^3 + Ax + B$



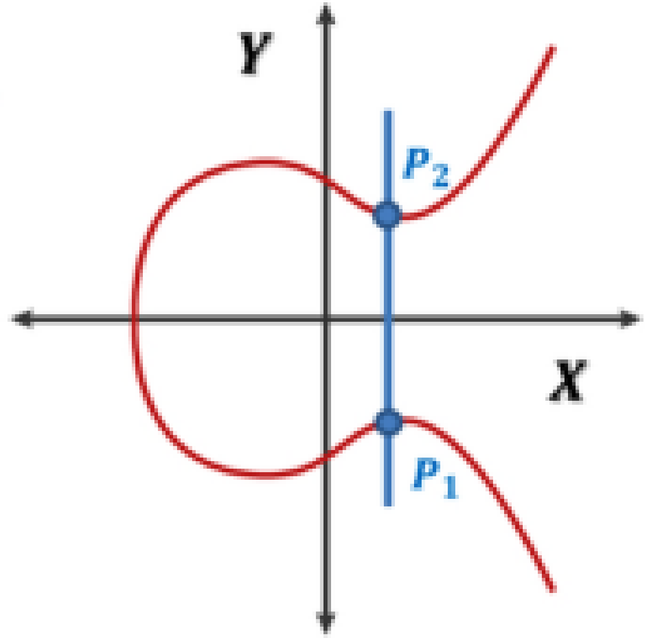
$P_3 = P_1 + P_2,\ x_1 \neq x_2$



$P_3 = 2P_1,\ y_1 \neq 0$



$P_3 = O, x_1 = x_2, y_1 \neq y_2$



Sophisticated Cipher Suite & Key Derivation

Modern Cipher Implementation

- I’ve used ChaCha20-Poly1305, a state-of-the-art authenticated encryption algorithm, for securing message content. This cipher is optimized for performance on mobile processors while providing robust security guarantees.
- The shared secret from ECDH is processed with HKDF (HMAC-based Key Derivation Function) to derive a symmetric key, using a salt (e.g., “FreedomLayerSalt”) to ensure uniqueness and randomness.

```swift
// Derive symmetric key from shared secret using HKDF
let saltData = "FreedomLayerSalt".data(using: .utf8)!
let symmetricKey = sharedSecret.hkdfDerivedSymmetricKey(using: SHA256.self,
                                                        salt: saltData,
                                                        sharedInfo: Data(),
                                                        outputByteCount: 32)
// Encrypt a message using ChaChaPoly
let messageData = "Confidential Message".data(using: .utf8)!
let sealedBox = try ChaChaPoly.seal(messageData, using: symmetricKey)
let cipherText = sealedBox.combined.base64EncodedString()
```

Design & User Interface

User Interface

- This app is made with SwiftUI, a native solution for making beautiful designs on Apple devices.
- I made it to be easy to use and simplistic.
- UI design made in Figma.

Supabase Backend & Database Schema

Backend Architecture

- Authentication is managed through Supabase, which provides secure email-based sign-up and login.
- Profiles Table: Stores user identifiers, usernames, and public keys with enforced uniqueness
- Messages Table: Contains encrypted message content, sender and receiver IDs, timestamps, and any attachment references.

```sql
CREATE TABLE profiles (
    id UUID PRIMARY KEY REFERENCES auth.users(id) ON DELETE CASCADE,
    username TEXT UNIQUE NOT NULL,
    public_key TEXT NOT NULL,
    created_at TIMESTAMPTZ DEFAULT now()
);
```

QR Code Functionality & Contact Discovery

QR Code Generation

- Public keys are Base64 encoded and transformed into QR codes using CoreImage, facilitating easy sharing.

```swift
func generateQRCode(from string: String) -> UIImage? {
    let data = Data(string.utf8)
    guard let filter = CIFilter(name: "CIQRCodeGenerator") else { return nil }
    filter.setValue(data, forKey: "inputMessage")
    filter.setValue("Q", forKey: "inputCorrectionLevel")
    guard let ciImage = filter.outputImage else { return nil }
    let transform = CGAffineTransform(scaleX: 10, y: 10)
    return UIImage(ciImage: ciImage.transformed(by: transform))
}
```

QR Code Scanning

- Uses VisionKit’s DataScanner to capture QR codes efficiently, enabling rapid onboarding of new contacts.

The End