橢圓加密聊天室

徐偉哲	胡珈華	王嘉笙	張凱博	梁耕銘
108502531	108502537	108502528	108502541	107502563

摘要

本文先對多人聊天室的原理進行探究，並基於隱私部分對訊息加密一些分析與探討，最後實作出一個帶有訊息加密的多人聊天室。

Abstract

This article first explores the principle of multi-person chat rooms, and analyzes and discusses the encryption of messages based on the privacy part, and finally makes a multi-person chat room with message encryption.

關鍵字：多人聊天室、ECC、socket、TCP

1 Introduction

現今，通訊軟體千百種，網路傳輸訊息已經成為現代人不可或缺的功能，然而，隨著網路訊息傳遞的普及，訊息內容的安全性，便是一個重要的問題，若在傳遞重要訊息時，遇上有心人士惡意破解，並竊取重要資料，將造成嚴重的損失，因此，在現今世代，聊天室的加密尤為重要。

甚麼是加密？若我們將平常和朋友的「聊天」，想像成一個「通道」，通道的兩端有兩個門，這兩個門的鑰匙，只有帳號的主人才擁有，也就是說，只有「聊天的雙方」，才能開啓通道，看裡面傳了些什麼。若沒有門的鑰匙，想要破解這個門是十分困難的。

本篇論文將探討的，是一個帶有訊息加密功能的聊天室，它可以保護使用者在使用聊天軟體通訊時的隱私權。

2 Related Work

想要建一個聊天室就必須牽涉到網路，說到網路就必須提到socket。

2.1 Socket [2]

Socket是讓開發者可以在抽象層面和外部系統溝通的一個工具，開發者可以不用去煩惱電腦(路由器)是如何讀取協定、IP和變成電子訊號傳遞到目的地。

開發者只要懂得自己熟練的程式語言如何透過Sokcet connect / write / read / close那就可以著手實踐網路程式。

如果以Java物件導向去思考，Socket就是一個有send/get的物件，可以設定protocal去決定他要用何種協定去傳送資料。

2.1.1 Stream Socket

我們建聊天室是使用Stream Socket。使用Stream Socket可以進行較可靠的傳送，在讀寫Socket資料前，接收方和傳送方都彼此做個連線確認。

2.1.2 TCP [4]

Stream Socket使用的協定是TCP。

TCP是一個可靠的傳輸協定，他是將所傳送的資料送達目的地，且資料本身是正確無誤，所以在資料傳出去後，必須由接收端確認無誤後，回應給傳送端，才能達到TCP的可靠性。以下是TCP的流程。

Step 1.端點主機A傳送資料(封包1)給端點主機B，並設定一個"計時器"(timer)開始計時，並等待B的確認封包回應。

Step 2.端點主機B接收到"封包1"時，驗證資料正確無誤後，隨即送出一個"確認"封包(確認1)給A，告訴A已正確收到"封包1"

Step 3.當端點主機A在"固定時間"內，收到B所回應的"確認1"的封包後，便繼續傳送下一個"封包2"，並重複Step1到Step3。

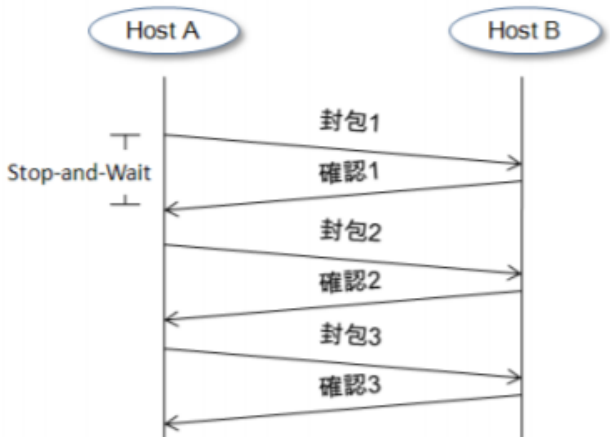

圖 1: TCP流程

2.2 橢圓曲線密碼學(ECC) [1]

橢圓曲線密碼學（英語：Elliptic Curve Cryptography，縮寫：ECC）是一種基於橢圓曲線數學的非對稱性加密演算法。

橢圓曲線是由以下形式的方程式通式定義的平面曲線

$y^2 = ax^3 + bx^2 + cx + d$

其中a和b是實數。這類稱為Weierstrass方程式

2.2.1 定義橢圓曲線的運算規則

群

首先要先介紹群的概念。群是一種代數結構，由一個集合以及一個二元運算所組成。已知集合和運算(G,*)如果是群則必須滿足如下要求:

- 封閉性: $\forall a, b \in G, a * b \in G$
- 結合性: $\forall a, b, c \in G, (a * b) * c = a * (b * c)$
- 單位元: $\exists e \in G, \forall a \in G, e * a = a * e = a$
- 逆元: $\forall a \in G, \exists b \in G$ 使得$a * b = b * a = e$

另外，有一種特殊的群叫阿貝爾群，它除了上面的性質還滿足交換律公理$a * b = b * a$

加法

過曲線上的兩點P、Q畫一條直線，找到直線與橢圓曲線的交點-R 交點關於x軸對稱位置的點，定義為P Q，即為加法。如下圖所示：PQ = R。

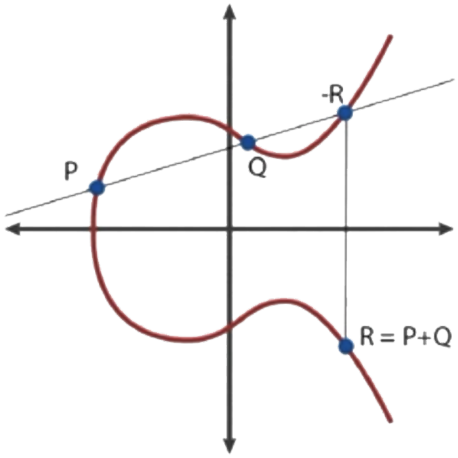

圖 2: 加法

乘法定義(兩倍運算)

上述方法無法解釋PP，即兩點重合的情況。因此在這種情況下，將橢圓曲線在P 點的切線，與橢圓曲線的交點，交點關於水平對稱軸對稱位置的點，定義為PP，即2P，即為二倍運算。

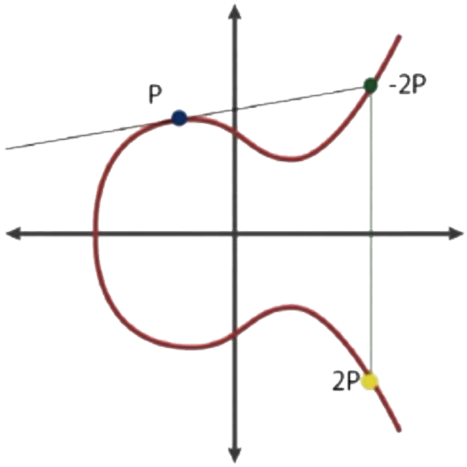

圖 3: 乘法

無窮遠點

如果將A與-A相加，過A與-A的直線平行於y軸，可以認為直線與橢圓曲線相交於無窮遠點。

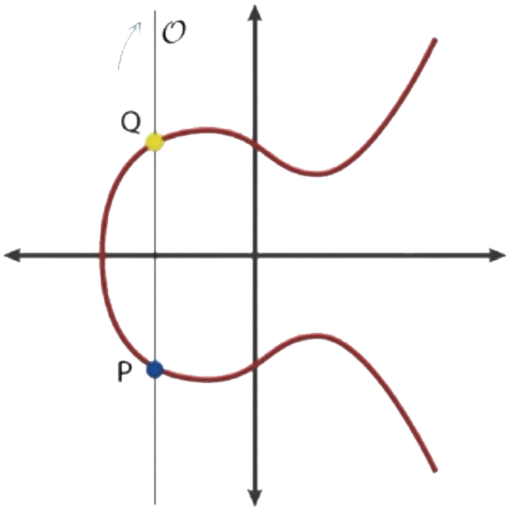

圖 4: 無窮遠點

2.2.2 公式

$$m \equiv \begin{cases} \frac{3X_G1^2+a}{2G1_y}(mod\beta), & G1 = G2 \\ \frac{G2_y-G1_y}{G2_x-G1_x}(mod\beta), & G1 \neq G2 \end{cases}$$

$$G3_x \equiv m^2 - G1_x - G2_x(mod\beta) \tag{1}$$

$$G3_y \equiv G1_p + m(G3_x - G1_x)(mod\beta) \tag{2}$$

$$G3_y \equiv G2_p + m(G3_x - G2_x)(mod\beta) \tag{3}$$

2.2.3 定義橢圓曲線的運算規則

建立基於橢圓曲線的加密機制，需要找到類似RSA質因子分解或其他求離散對數這樣的難題。而橢圓曲線上的已知G和xG求x，是非常困難的，此即為橢圓曲線上的的離散對數問題。此處x即為私鑰，xG即為公鑰。

橢圓曲線加密演算法原理如下：因此給定橢圓曲線的某一點G，當給定G點時，已知x，求xG點並不困難。反之，已知xG點，求x則非常困難。此即為橢圓曲線加密演算法背後的數學原理。

設私鑰、公鑰分別為k、K，即K = p k*G，其中G為G點。

公鑰加密：

選擇隨機數r，將訊息M生成密文C，該密文是一個點對，即：

C = rG, M rK，其中K為公鑰

私鑰解密：

M rK – k(rG) = M r(kG) – k(rG) = M

其中k、K分別為私鑰、公鑰。

2.2.4 橢圓曲線V.S. RSA

表 1: 橢圓曲線V.S. RSA

橢圓曲線密碼長度	112-bits	163-bits	224-bits
RSA 密碼長度	512-bits	1024-bits	2018-bits
金鑰長度比	1 : 5	1 : 6	9 : 1
橢圓曲線密碼長度	256-bits	384-bits	512-bits
RSA 密碼長度	3072-bits	7680-bits	15360-bits
金鑰長度比	1 : 12	1 : 20	1 : 30

3 Method

3.1 程式碼架構及重點 [3]

3.1.1 Server

1. clients的陣列會存取用戶IP跟名字用的。

2. ServerSocketc會建立伺服器，並且設定埠號。

3. ServerThread是一個自己寫的class，用來建立Client與Server的連線用的。

```
clients = new ArrayList<ClientThread>();
serverSocket = new ServerSocket(port);
serverThread = new ServerThread(serverSocket, max);
serverThread.start();
isStart = true;
```

圖 5: ServerStart

4. ServerThread這個Class是用來創建Client和Server連線用的。

```
class ServerThread extends Thread {
    private ServerSocket serverSocket;
    private int max;// 人數上限

    // 伺服器執行緒的構造方法
    public ServerThread(ServerSocket serverSocket, int max) {
        this.serverSocket = serverSocket;
        this.max = max;
    }
```

圖 6: ServerThread

5. run()是ServerThread裡面的函式，用來建立連線、執行讀取與發送資料。

6. Socket socket = serverSocket.accept();是用來等待Client與Server連線的，若連線成功則會繼續執行程式法，否則會停在這。

7. BufferedReader會讀取收到的訊息。收到的訊息會由InputStreamReader取得。

8. PrintWriter會發送訊息。因為是要發送訊息，所以要用getOutputStream。

```
public void run() {
    while (true) {// 不停的等待客戶端的連結
        try {
            Socket socket = serverSocket.accept();
            if (clients.size() == max) {// 如果已達人數上限
                BufferedReader r = new BufferedReader(
                        new InputStreamReader(socket.getInputStream()));
                PrintWriter w = new PrintWriter(socket
                        .getOutputStream());
```

圖 7: ServerThread中的run()

9. ClientThread會建立Client與Server的連線。

10. Socket是Client與Server之間連線的通道

11. 這裡的BufferReader和PrintWriter與ServerThread中的功能一樣，會讀取與發送訊息。

```
class ClientThread extends Thread {
    private Socket socket;
    private BufferedReader reader;
    private PrintWriter writer;
```

圖 8: ClientThread

3.1.2 Client

1. send是Client裡的函式，只要點擊GUI中的傳送按鈕，就會將textField中的發出。

```
// 執行傳送
public void send() {
    if (!isConnected) {
        JOptionPane.showMessageDialog(frame, "還沒有連線伺服器,無法傳送訊息!", "錯誤",
                JOptionPane.ERROR_MESSAGE);
        return;
    }
    String message = textField.getText().trim();
    if (message == null || message.equals("")) {
        JOptionPane.showMessageDialog(frame, "訊息不能為空!", "錯誤",
                JOptionPane.ERROR_MESSAGE);
        return;
    }
```

圖 9: Client中的send()

2. connectServer是Client中的一個函式，用來與Server建立連線。

3. socket = new Socket(hospIp, port);會與Server的Ip位址以及服務器的port進行連線，如此一來，就能夠成功地將Client與Server連線了。

4. BufferReader與PrintWriter一樣會讀取與發送訊息。

5. messageThread是自己創建的Class，會不斷的接收訊息，並將收到的訊息貼到textArea上。

```java
public class ECC {
    // $y^2 \equiv x^3 + ax + b \mod mod$
    private int a, b, mod;
    public ECC() {
        a = b = mod = 0;
    }
    public ECC(int ia, int ib, int imod) {
        a = ia;
        b = ib;
        mod = imod;
    }
    public int get_a() {
        return this.a;
    }
    public int get_b() {
        return this.b;
    }
    public int get_mod() {
        return this.mod;
    }
}
```

圖 12: 設定橢圓曲線的Class

```java
public boolean connectServer(int port, String hostIp, String name) {
    // 連線伺服器
    try {
        socket = new Socket(hostIp, port);// 根據埠號和伺服器ip建立連線
        writer = new PrintWriter(socket.getOutputStream());
        reader = new BufferedReader(new InputStreamReader(socket
                .getInputStream()));
        // 傳送客戶端使用者基本資訊(使用者名稱和ip地址)
        sendMessage(name + "@" + socket.getLocalAddress().toString());
        // 開啟接收訊息的執行緒
        messageThread = new MessageThread(reader, textArea);
        messageThread.start();
```

圖 10: connectServer

6. run是messageThread中的函式，會不斷地接收訊息。

```java
public void run() {
    String message = "";
    while (true) {
        try {
            message = reader.readLine();
```

圖 11: messageThread中的run()

3.1.3 加密

2. 橢圓曲線的Class
2. 橢圓曲線上的整數點以及其運算

3.2 Demo

以下三張圖片是實際Demo的結果:

```java
public class Node {
    private int x, y;
    public Node() {
        x = y = 0;
    }
    public Node(int i, int j) {
        x = i;
        y = j;
    }
```

圖 13: Node 初始化

```java
public static int extend_modulus(int dividend, int divisor, int mod) {
    if (dividend < 0) dividend = dividend%mod + mod;
    if (divisor < 0) divisor = divisor%mod + mod;
    int tmp = divisor;
    while(tmp%mod != 1) {
        tmp += divisor;
    }
    tmp = tmp/divisor;
    int ans = (dividend*tmp)%mod;
    if (ans < 0) ans += mod;
    return ans;
}
```

圖 14: 擴展模除

```java
public Node add(Node p, Node q, ECC ecc) {
    Node node = new Node();
    int dividend, divisor, m;
    if (p.get_x() == q.get_x() && p.get_y() == q.get_y()) {
        dividend = 3*(p.get_x()*p.get_x()) + ecc.get_a();
        divisor = 2*p.get_y();
    } else {
        dividend = (p.get_y() - q.get_y());
        divisor = (p.get_x() - q.get_x());
    }
    m = extend_modulus(dividend, divisor, ecc.get_mod());
    node.set_x(extend_modulus((m*m - p.get_x() - q.get_x()), 1, ecc.get_mod()));
    node.set_y(extend_modulus(m*(p.get_x() - node.get_x()) - p.get_y(), 1, ecc.get_mod()));
    return node;
}
```

圖 15: Add 實作

第一張是client1、client2與server進行連線後，由server端發送訊息。

第二張是client1接收到的server訊息，因為client1有鑰匙所以可以將已經加密過的server訊息進行解密得到正確訊息。

第三張是client2接收到的server訊息，因為client2沒鑰匙無法解開加密過的server訊息所以得到錯誤訊息。

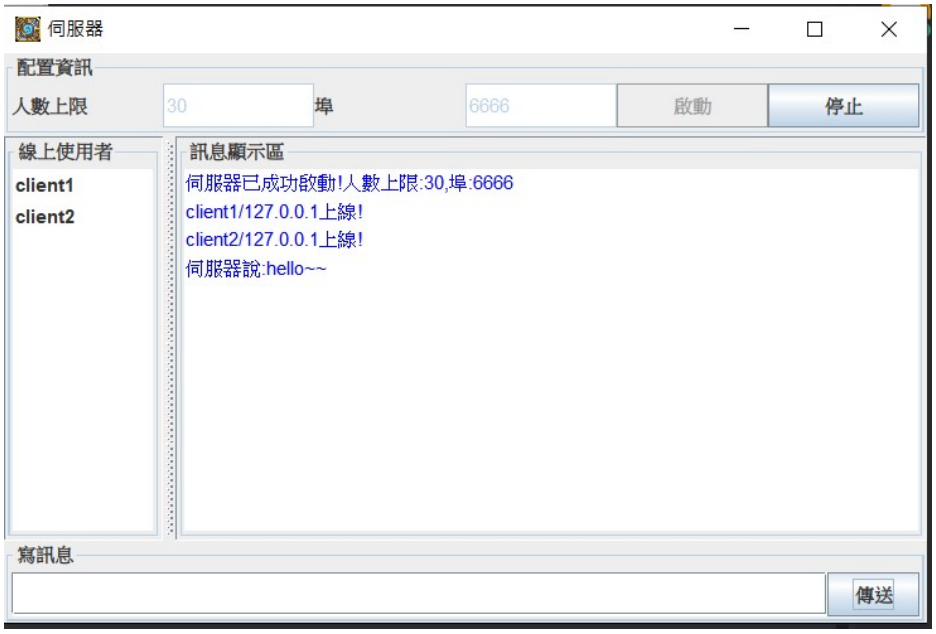

圖 16: server傳送訊息

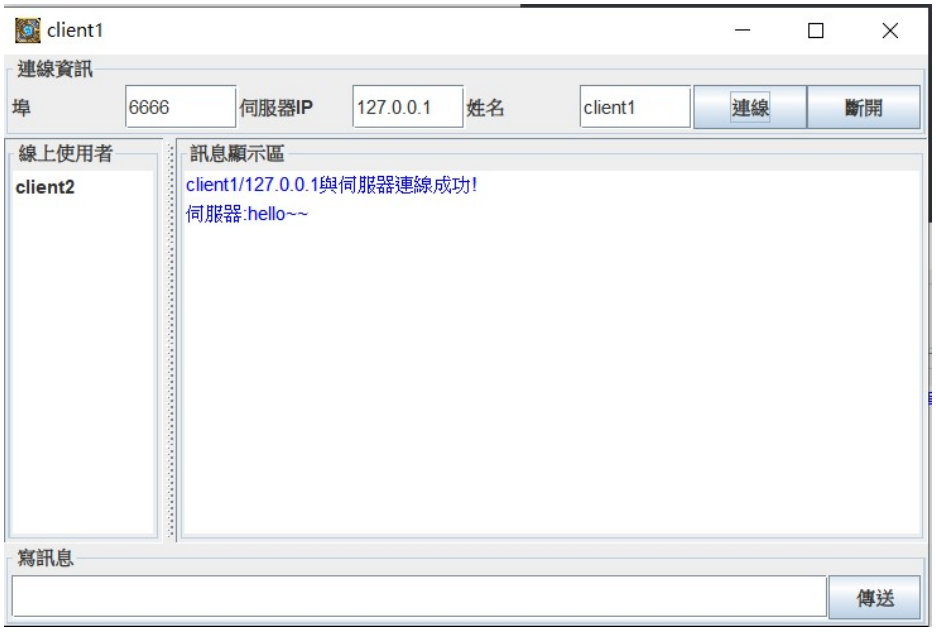

圖 17: client1(有鑰匙)收到的訊息

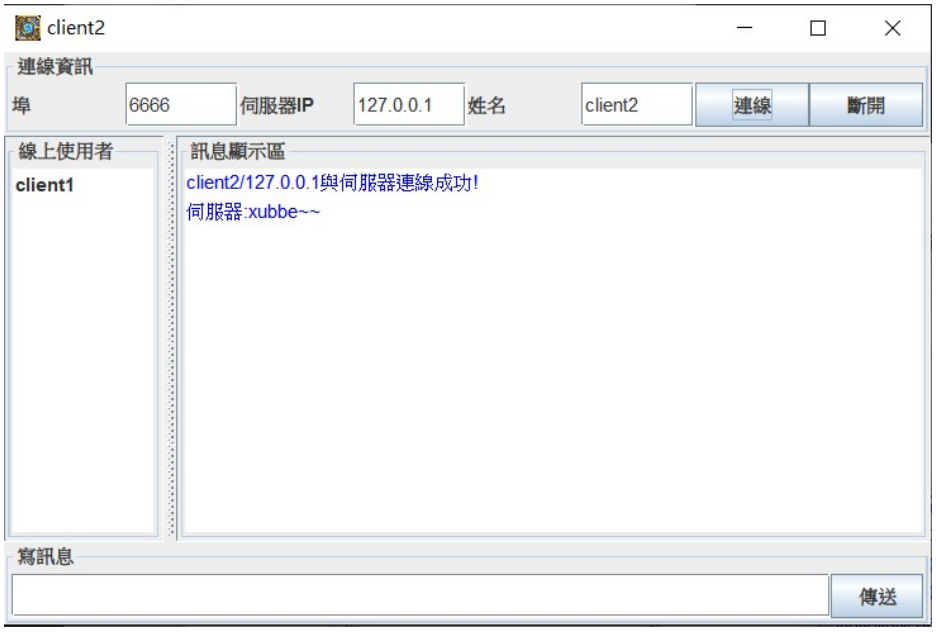

圖 18: client2(沒鑰匙)收到的訊息

4 Conclusion

在這個資訊傳播廣泛的時代，享受訊息交換的便利之時，加密方式的選擇也是十分重要的，除了安全性外，運算速度，也是判別聊天室好壞的重要環節，若一味追求安全性而不斷提高密鑰長度，或許會導致訊息傳輸緩慢・・・等問題。如何兼顧兩者才是我們該探討的目標。

References

[1] Ecc 簡介. https://ieeexplore.ieee.org/document/5931464.

[2] Java socket. https://medium.com/bucketing/java-%E7%B6%B2%E8%B7%AF%E9%96%8B%E7%99%BC-%E4%B8%80-socket%E6%98%AF%E7%94%9A%E9%BA%BC-dc34173e7bc5.

[3] Java socket code 參考. https://oblivious9.pixnet.net/blog/post/74178285.

[4] Tcp 介紹. http://dns2.asia.edu.tw/~wzyang/slides/info_net/info_B/CH10TCP.pdf.