**位置隐私量化研究设计文档**

**目 录**

[**一. 需求分析 2**](#_Toc4323989)

[**二. 概要设计 3**](#_Toc4323990)

[**三. 详细设计 6**](#_Toc4323991)

[**暴露处理算法 6**](#_Toc4323992)

[**位置保护算法 7**](#_Toc4323993)

[**知识构造算法 9**](#_Toc4323994)

[**攻击算法 10**](#_Toc4323995)

[**度量算法 11**](#_Toc4323996)

[**位置还原 15**](#_Toc4323997)

[**四. 测试分析 16**](#_Toc4323998)

[**五. 总结 33**](#_Toc4323999)

# 一. 需求分析

因数据挖掘以及移动端各种技术的发展，大量保存在用户端、服务端的用户隐私信息很有可能会被窃取，人们开始担心自己的信息会泄露。而且近5年来信息泄露的事件时有发生。比如，2014年黑客入侵多家存在漏洞的快递网站，导致网上大量个人信息以图片的形式泄露出去，上面详细显示了姓名、电话号码、住址等个人信息；又比如，2016年，网络黑客窃取了美国有线电视公司时代华纳，公司旗下近32万用户的邮件及密码信息。这些种种的信息泄露事件引起了各方的重视，而信息泄露的途径有很多，近些年来兴起的基于位置的服务方便了人们的生活，但也伴随着位置信息等个人信息泄露的危险。

随着智能手机和平板电脑等移动设备的普及，基于位置的服务(LBS)变得越来越流行，人们通过网络进行查询时虽然能够使用服务获取自己想要的信息，但却将自己的位置信息暴露给了LBS提供商。个人通信设备的使用尽管为其所有者提供了便利，但仍然留下了不可擦除的痕迹，并且这些痕迹不仅是地图上的一组位置，根据它们追踪到的背景信息可以说明个人的习惯、兴趣、活动和关系等。它还可以揭示他们的个人或公司机密。这就可能使用户遭受不必要的广告和基于位置的垃圾邮件或诈骗，造成社会声誉或经济上的损失，并使他们成为勒索甚至身体暴力的受害者。此外，信息披露打破了知情实体和披露此信息的实体之间的力量平衡。

因此LBS需要有较好的保护措施来维护用户的隐私，然而用户却不知道自己的隐私被保护到了什么样的程度，位置隐私保护这一模块基本上对用户是透明的，而提供LBS的供应商不可能将自己的保护算法公开出来，否则就起不到保护作用了。因此，我们需要一个准确的测定方法来测量用户被保护的隐私程度。

不同的用户对自己的隐私保密程度的要求各不相同，我们的测定方法能够让用户知道自己的隐私被保护到何种程度，用户也无需拥有太多位置保护这方面的知识就能大体地知道自己的隐私保密程度，这也是我们设计该位置隐私量化程序的目的。

# 二. 概要设计

我们设计的程序模拟一些应用在使用过程中位置轨迹的暴露情况，对轨迹进行一定程度的暴露处理，然后将轨迹交给可信第三方，运行算法对轨迹进行保护，生成经过保护算法处理后的轨迹以及密钥文件（二进制文件），再让攻击方针对经过保护的轨迹进行攻击并生成攻击后的轨迹（在攻击之前需要事先构造先验知识），然后再根据实际轨迹和攻击后生成的轨迹对用户的位置隐私水平进行一定程度的度量。另外，我们的程序还模拟了服务提供方收到保护后的轨迹，根据密钥文件对用户原来的轨迹进行还原。

**整个位置隐私量化度量框架由6个部分组成：**

① 攻击方学习轨迹和用户实际轨迹的生成；

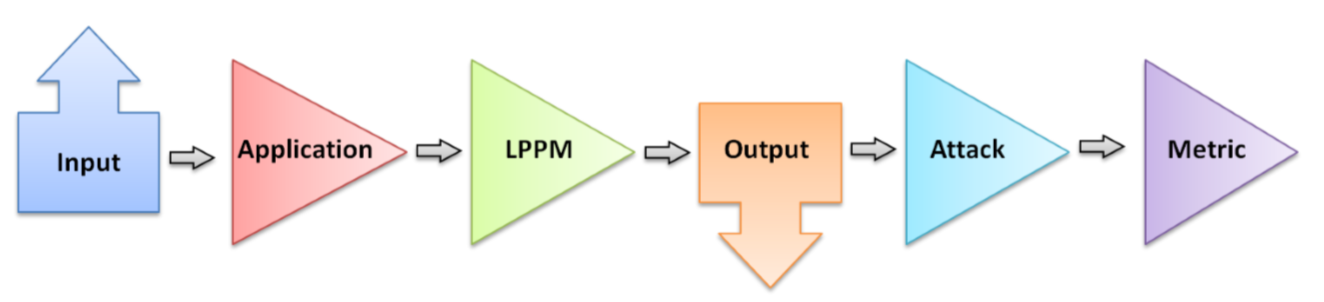
② 攻击方的知识构造；

③ 用户实际轨迹暴露处理；

④ 对经过暴露后的轨迹进行保护处理：

⑤ 攻击方针对经过保护后的轨迹进行攻击：

⑥ 根据攻击方生成的预测轨迹和用户的实际轨迹进行度量

图2-1

具体步骤如图2-1所示，我们采用马尔可夫链模型，知识构造使用的是概率转移矩阵。

**文件格式：**

关于文件格式，我们有两种轨迹文件，一种是只有3列的，一种是有4列的：3列的轨迹文件（actual.trace和exposed.trace）从左到右分别代表：用户戳、时间戳、位置戳；4列的轨迹文件（其他轨迹文件）从左到右分别代表：用户戳、时间戳、位置戳、暴露情况（0为未暴露，1为已暴露）

PS：用户一定要设计好下面这4个范围参数，否则程序运行可能出错

两种轨迹文件第一行均为4个范围参数：

[最小用户戳，最大用户戳]，[最小时间戳，最大时间戳]，[最小位置戳，最大位置戳]，[区域行数，区域列数]

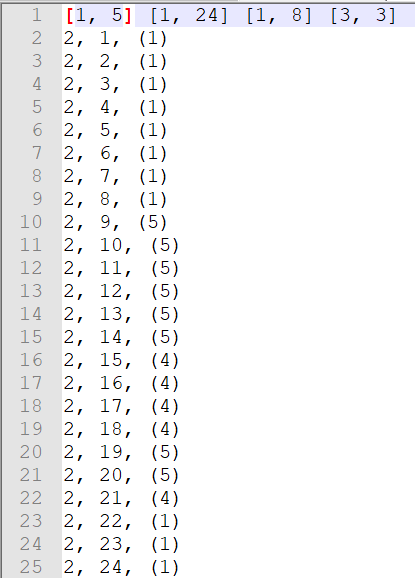
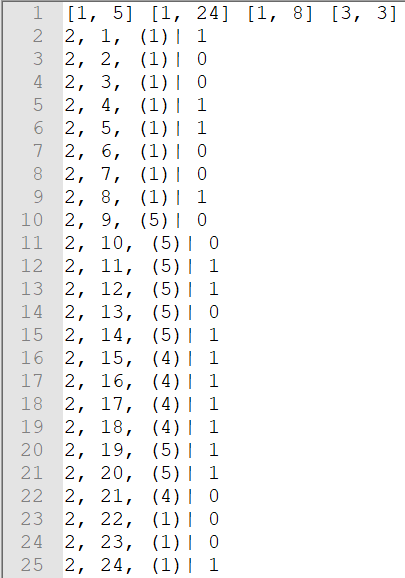
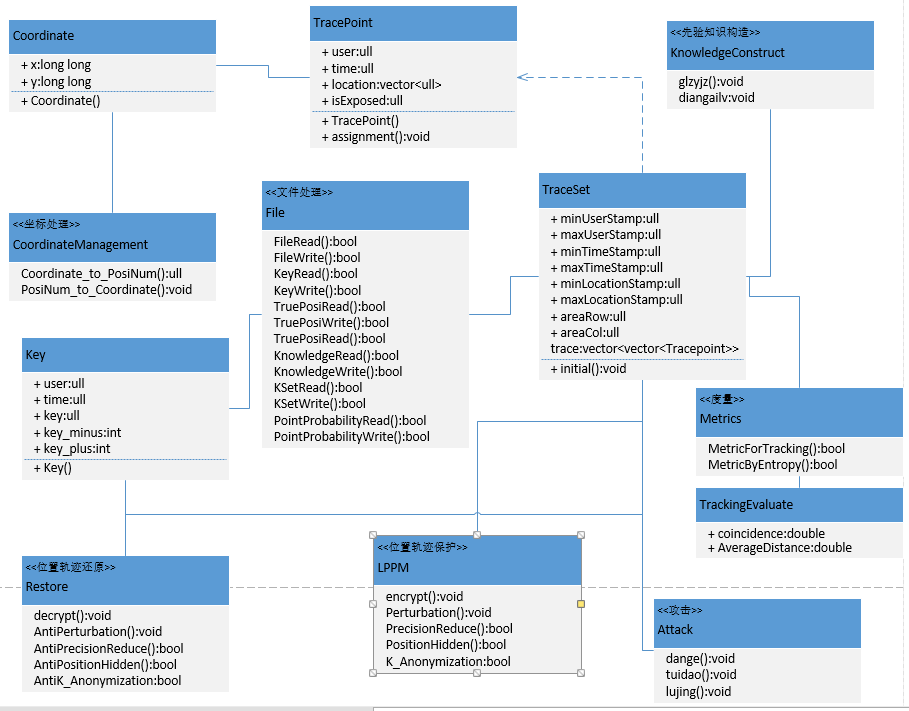
图1-1 图1-2

图1-1为3列的轨迹文件，图1-2为4列的轨迹文件

**类图：**

# 三. 详细设计

## 暴露处理算法

暴露算法需要用到两个函数，一个是随机数生成函数，另外一个就是取随机数确定对特定记录是否进行暴露处理的函数，具体实现代码如下：

1. //从0到num随机生成数字，并放入ranlist向量的不同位置
2. void Random\_Num(vector<int> &ranlist, int num)
3. {
4. srand(time(NULL));
5. int \*d = new int[num];
7. for (int i = 0; i < num; i++)  //数组d用来置数,数组ranlist用来取不重复的随机数，特别的我想更随机一些，所以将5的倍数和6的倍数的数字再生成随机数
8. {
9. d[i] = i;
10. }
11. for (int i = 0; i < num; i++)
12. {
13. int index = rand() % (num - i);
14. ranlist[i] = d[index];
15. d[index] = d[num - 1 - i];
16. if (i % 5 == 0 || i % 6 == 0) ranlist[i] = rand() % (num \* 5);
17. }
18. }
19. //通过ranlist中的数取模，判断是否暴露，暴露取1，没暴露取0
20. void Get\_isExposed(vector<int> ranlist, TraceSet &a,double percent)
21. {
22. srand(time(NULL));
23. int target=a.trace[0].size()\*percent;
24. //先全部置0
25. for (int i = 0; i < a.trace.size(); i++)
26. {
27. for (int j = 0; j < a.trace[i].size(); j++)
28. {
29. a.trace[i][j].isExposed = 0;
30. }
31. }
32. //然后统计isExposed为1的记录数
33. for (int i = 0; i < a.trace.size(); i++)
34. {
35. int count=0;
36. for (int j = 0; count<target; j++)//确保每个用户的暴露记录数一定
37. {
38. if (ranlist[rand() % ranlist.size()] % 3 == 0 || ranlist[rand() % ranlist.size()] % 5 == 0)
39. {
40. if(a.trace[i][j%a.trace[i].size()].isExposed==0) count++;
41. a.trace[i][j%a.trace[i].size()].isExposed = 1;
42. }
43. }
44. }
45. }

我们利用Random\_Num生成一定范围内不重复的随机数，然后在生成出来的随机数里选取一些，看是否能取模为0，由此决定某条记录是否会暴露，这里我们会让用户输入暴露记录比（即可能会暴露百分之多少的轨迹记录），因此暴露出来的记录数是有固定比例的。

## 位置保护算法

**1. 位置扰动：**将暴露的位置进行一系列加密数学运算，生成一个虚假的位置暴露给攻击方，另生成一个解密文件，供服务方对加密后的轨迹进行解密，以便提供服务，这里我只给出加密的具体代码如下：

1. void encrypt(TraceSet &a, ull &n, Key &kk)
2. {
3. n ^= kk.key;//异或一下
4. while (n < a.minLocationStamp || n > a.maxLocationStamp)
5. {
6. if (n > a.maxLocationStamp)
7. {
8. n -= a.maxLocationStamp;
9. kk.key\_minus++;
10. }
11. else if (n < a.minLocationStamp)
12. {
13. n += a.minLocationStamp;
14. kk.key\_plus++;
15. }
16. }
17. }

这里的位置密钥文件是一种结构，这个结构记录的是位置戳进行异或以后需要进行加减minLocationStamp和maxLocationStamp的次数

**2. 假位置注入（降低精确度）：**对真实的轨迹文件加入噪音，将真实位置周围的一些位置点加入到轨迹文件中，扰乱攻击方所能观察到的轨迹记录，并生成相应的解密文件，供服务方解密提供服务使用，伪代码如下：

1. //降低精确度（虚假位置注入）：距离原坐标横向纵向坐标值在Grade之内的坐标点都会加进已经暴露的记录的location里面
2. //比如原位置为(1,5)，Grade为1，则(0,4)(0,5)(0,6)等(1,5)周围的8个点会选择部分点加进location里面，当然要考虑到边界点的情况
3. bool PrecisionReduce(char \*filename, TraceSet &a, ull Grade)
4. {
5. //首先和位置扰动一样，先找到已经暴露的记录
6. //先把原来的信息记录下来，等一下会输出成TruePosi文件，以便以后对数据进行还原
7. //进行坐标转换，获取原位置的坐标（我们原来的位置都是用位置戳表示的）
8. //将多个记录原来的位置读出后清空掉location这个向量
9. //现在构造该条记录location周围的点，为了方便构造我们以(x-Grade,y+Grade)即方阵的左上角为起点，构造出一个 2\*Grade + 1 的方阵，这些点就是我们要添加的假位置
10. //如果遇到边界点，我们就不要把它超出区域范围的点存入vector里面
11. //比如(0,2)，周围只有5个点左边3个点(-1,3)(-1,2)(-1,1)是不合法的
12. //开始注入虚假位置，要求随机选择出的虚假位置在位置范围之内否则舍弃
13. //获取到周围的有效坐标位置以后，我们要把他们转换成PosiNum，并且存入到
14. //相应记录的location向量里面，但是全部如果每个点都全部存入，会让攻击方
15. //推算到哪些被保护的点的位置是相同的，所以我们随机取出部分点存进去
16. //把TruePosition这个数组写到相应的路径filename中
17. return true;
18. }

**3. 位置隐匿：**将预测到要暴露出去的轨迹进行隐藏处理，同时生成解密文件，供服务方解密使用，伪代码如下：

1. bool PositionHidden(char \*filename, TraceSet& a)
2. {
3. //位置隐匿:将暴露出来的信息全部存起来，然后清空相应用户的location向量
4. //然后将存储隐匿的信息的输出成文件，存到相应的filename目录下
5. return true;
6. }

**4. K-匿名保护：**生成虚假轨迹，使得最终轨迹有k条（包含真实轨迹），并对用户与不同的轨迹的对应关系进行修改，从而扰乱攻击方对特定用户轨迹的预测，生成解密文件共服务方解密提供服务用，伪代码如下：

1. //K-匿名保护算法，伪造出k条和原记录相似的轨迹出来并且加入到文件里面
2. //然后混淆用户名，将用户放到其他人的轨迹下面，由于我们的轨迹文件比较
3. //简单，所以没有什么可以作为准标识符的，我对其进行一些改进，直接构造出
4. //虚假的轨迹，这样，攻击方对不同用户的攻击难度就会增加上升
5. //解密文件格式：总数+多个<user,数组下标>键值对(map<int,int>)
6. bool K\_Anonymization(char \*filename, TraceSet& a,int k)
7. {
8. //先记录用户对应哪条轨迹（等下生成map对会用到）
9. //然后作标记，将原来a的轨迹的第一个值改成一个不可能出现的位置戳的值（这里我用a.maxLocationStamp+user作为该条记录的标记)
10. //找出暴露轨迹文件中所有的用户有哪些因为等一下我们生成虚假轨迹要生成虚假的用户戳，不能和原有的用户戳相同
11. //接下来判断真实的用户轨迹够不够k条，如果不够就开始构造虚假轨迹和虚假用户戳
12. //如果足够，就直接混淆不同轨迹的用户名
13. //在构造虚假轨迹的时候因为想要伪造的轨迹想要真实一点，很多时候用户会定在一个位置不动，所以我们用一种随机的方式生成并记录一条轨迹里面一段时间内用户会多久定着不动
14. //因为轨迹不够k条，所以用户号也不够，要进行随机生成
15. //最后记录用户和其原轨迹所在位置的键值对，然后将键值对map写入文件
16. return true;
17. }

## 知识构造算法

**知识构造的大致思路：**

从学习轨迹文件中通过位置转移的概率计算，得到位置点的概率转移矩阵，以及通过每个位置点出现的概率，来构造出每个点出现概率的一维向量。

具体实现：

先遍历一次整个位置轨迹文件，用一个数组记录有多少个不同的位置点出现。接着用一个三位向量来记录用户的id，每个用户的每个点出现的总次数(排除最后一个位置点)以及每个位置转移的次数，向量的一维记录用户的id，二维记录每个点出现的次数，并且出现该点后后面还会出现新的位置点，三位记录点与点之间转移的次数。最后再用一个二维向量来存概率转移矩阵，每个转移概率取了用户的平均值，假如有的点从来没有出现过，就令它到其他任何点的概率都相同。每个点出现的概率直接用点出现的次数除以所有点出现的总次数来求得概率，最终将这两个概率文件作为知识构造文件。

## 攻击算法

**攻击大致思路：**

通过读取暴露的点，判断暴露的点是否符合一些基本条件，如果不符合，换一个合适的点作为暴露点的新位置点，接下来通过统计预测的方法，通过看到的点以及之前的先验文件来推测出未暴露出来的点。

具体实现：

先从点，再到局部，再到全部。用的是马尔可夫链模型以及后向推测算法。暴露的点与点之间的推导分为三种情况：

一. 用户从刚开始移动到第一个暴露的点，要推测之间的点，左边是没有暴露点的，而右边有暴露的点。

二. 用户最后一个暴露的点到用户运动结束，要推测之间的点，左边是有暴露点的，而右边是没有暴露点的。

三. 两个相邻暴露点之间的点的推测，左边是有暴露点的，右边也是有暴露点的。接下来讲一些如何实现这三种情况的未暴露点的还原。第一种情况：由于左边没有暴露的点，我需要从左边往右推导，看与右边暴露的点相邻的那个点是哪个位置点出现的概率最大，概率是通过如下的方式计算的：

先计算左边第一个时间点每个位置点出现的概率，用每个点出现的概率的文件实现，再推导第二个时间点每个位置点出现的概率，用第一个时间点出现的概率，乘以转移概率，加起来的和作为第二个时间点每个位置点出现的概率，以此类推，推导到倒数第二个时间点每个点出现的概率，最后再计算到右边暴露的那个位置点的概率，选取最大的概率点作为右边暴露点相邻左边的位置点。

第二种情况则是先求局部第二个时间点每个位置点出现的概率，以此类推，推导到最后一个时间点每个点出现的概率，选取最大概率的位置点作为最后一个点的位置点。第三种情况：考虑局部第一个时间点的位置点与最后一个时间点的位置点是已知的，中间过程同上。最后通过局部推导，将推导的局部路径连接到一起，就还原出了用户的完整路径。

关于反保护措施，因为暴露的数据是通过保护后的数据，所以需要去判断真实数据大概是多少，通过建立函数，去判断该时间点出现在该位置点是否合理，不合理的话推测出一个合理的点作为新的点，还要判断前后暴露的点是否符合实际，比如用户不可能在该时间段从这个位置点到另外一个位置点，那么需要推测一个合理的位置点作为新的位置点来作为攻击的数据。

## 度量算法

1. 基于位置信息失真的度量方法：由于我们的框架度量是针对轨迹攻击的，所以可以通过计算攻击轨迹和实际用户轨迹的吻合程度以及每一时刻两个轨迹的位置距离来评估位置隐私水平；

1. //针对跟踪攻击和相遇披露攻击，基于误差的度量，具体来说就是以一定的误差距离，判断两条轨迹之间是否有一致的点，然后计算两条轨迹一致的程度，其中a是用户的实际轨迹，b是攻击方的轨迹，u是要查询的用户，devition是误差允许范围（比如说2，那么攻击方推算出该点半径为2的范围内的点都算攻击成功）,timeBegin~timeEnd是要估算的时间范围。这里我设计了两个衡量标准，一个是吻合的位置的数目，另一个是对相应的攻击轨迹和实际轨迹的点求距离差，然后再取平均值，由这两者共同来衡量隐私保护程度
2. bool MetricForTracking(TraceSet &a, TraceSet &b, ull timeBegin, ull timeEnd, ull u, ull deviation, TrackingEvalutate &eva)
3. {
4. //我们默认TraceSet内轨迹大小都是一样的
5. vector<Coordinate> actualTrace;
6. vector<Coordinate> attackTrace;
7. int exitFlag = 0;
8. if (timeEnd <= timeBegin) return false;
10. //首先要将位置戳转换为坐标点，并存起来
11. for (int i = 0; i < a.trace[i].size(); i++)
12. {
13. if (a.trace[i][0].user == u)
14. {
15. if (timeBegin<a.trace[i][0].time || timeEnd>a.trace[i][a.trace[i].size() - 1].time)
16. {
17. exitFlag = 1; break;
18. }
19. for (int j = 0; j < a.trace[i].size(); j++)
20. {
21. if (a.trace[i][j].time >= timeBegin && a.trace[i][j].time <= timeEnd)
22. {
23. Coordinate temp;
24. PosiNum\_to\_Coordinate(a.areaCol, a.trace[i][j].location[0], temp);
25. actualTrace.push\_back(temp);
26. }
27. }
28. break;
29. }
30. }
31. if (exitFlag == 1) return false;
33. //由于攻击方的文件的用户顺序和实际轨迹文件的用户顺序可能不一样，这里只能分开两个循环来写了
34. for (int i = 0; i < b.trace[i].size(); i++)
35. {
36. if (b.trace[i][0].user == u)
37. {
38. if (timeBegin<b.trace[i][0].time || timeEnd>b.trace[i][b.trace[i].size() - 1].time)
39. {
40. exitFlag = 1; break;
41. }
42. for (int j = 0; j < b.trace[i].size(); j++)
43. {
44. if (b.trace[i][j].time >= timeBegin && b.trace[i][j].time <= timeEnd)
45. {
46. Coordinate temp;
47. PosiNum\_to\_Coordinate(b.areaCol, b.trace[i][j].location[0], temp);
48. attackTrace.push\_back(temp);
49. }
50. }
51. break;
52. }
53. }
54. if (exitFlag == 1) return false;
56. vector<double> distance;
57. double match = 0;
58. //下面开始进行两条轨迹的吻合度判断，这里我想了一种模型，将每条记录的位置戳的距离记录下来，最后求和再取均值，这样能够作为衡量两条曲线吻合度的一种不太准确的衡量标准
59. for (int i = 0; i < actualTrace.size(); i++)
60. {
61. //计算某条记录的距离差
62. long double powx, powy;
63. powx = pow(abs(double(actualTrace[i].x - attackTrace[i].x)), 2);
64. powy = pow(abs(double(actualTrace[i].y - attackTrace[i].y)), 2);
65. distance.push\_back(sqrt(powx + powy));
66. if (distance[distance.size() - 1] <= deviation)
67. {
68. match++;
69. }
70. }
71. eva.coincidence = match / actualTrace.size();
72. eva.AverageDistance = 0;
73. for (int i = 0; i < distance.size(); i++)
74. {
75. eva.AverageDistance += distance[i];
76. }
77. eva.AverageDistance = eva.AverageDistance / distance.size();
78. return true;
79. }

**2. 基于不确定性（熵）的度量方法：**通过计算攻击方的背景知识（每一时刻的概率转移矩阵）的熵对位置隐私水平进行评测，不过因为我们的概率转移矩阵元素之和不为1，所以要进行归一化处理，再进行度量；

1. //基于熵的度量方法，先读取攻击方的Knowledge文件，获取转移概率矩阵，然后计算熵和最大熵值的比值并记录到result里面
2. void MetricByEntropy(vector<vector<double>> &a, double &result)
3. {
4. //先对矩阵进行百分比处理，让所有概率加起来在1以内
5. double sum = 0;
6. for (int i = 0; i < a.size(); i++)
7. {
8. for (int j = 0; j < a[i].size(); j++)
9. {
10. sum += a[i][j];
11. }
12. }
13. //然后进行归一化
14. for (int i = 0; i < a.size(); i++)
15. {
16. for (int j = 0; j < a[i].size(); j++)
17. {
18. a[i][j] /= sum;
19. }
20. }
21. double Hmax = -log2(1 / double(a.size()\*a[0].size()));
22. double H = 0;
23. for (int i = 0; i < a.size(); i++)
24. {
25. for (int j = 0; j < a[i].size(); j++)
26. {
27. if(a[i][j]!=0) H += -a[i][j] \* log2(a[i][j]);
28. }
30. }
31. result = H / Hmax;
32. }

## 位置还原

位置还原算法的思路比较简单，刚才在介绍保护算法的时候也提及过，在保护的同时，我们会针对每种不同的保护生成不同的加密文件，比如说位置扰动对应的就是Key文件，Key文件里面对应存放的是一组Key结构值，Key里面存的主要数据是进行加密的记录的用户戳、时间戳、以及位置戳进行了多少次加减minLocationStamp或maxLocationStamp的操作，因此只要读取相应的文件就可以知道应该如何进行解密。而对于降低精确度和位置隐匿保护，位置密钥文件存放的是原记录的位置，我们解密只需要读取这些记录就可以了。对于K匿名保护，密钥文件是一些键值对，分别对应用户戳和保护后该用户真实轨迹在数组中的位置，还原起来就是根据这个键值对到数组中寻找即可。

# 四. 测试分析

**轨迹暴露处理测试如图5-1~图5-3所示：**

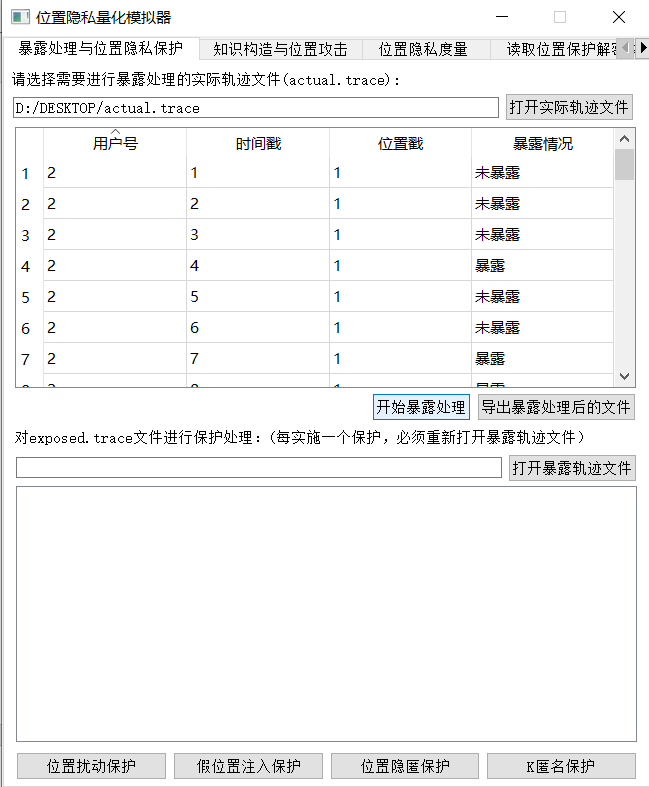
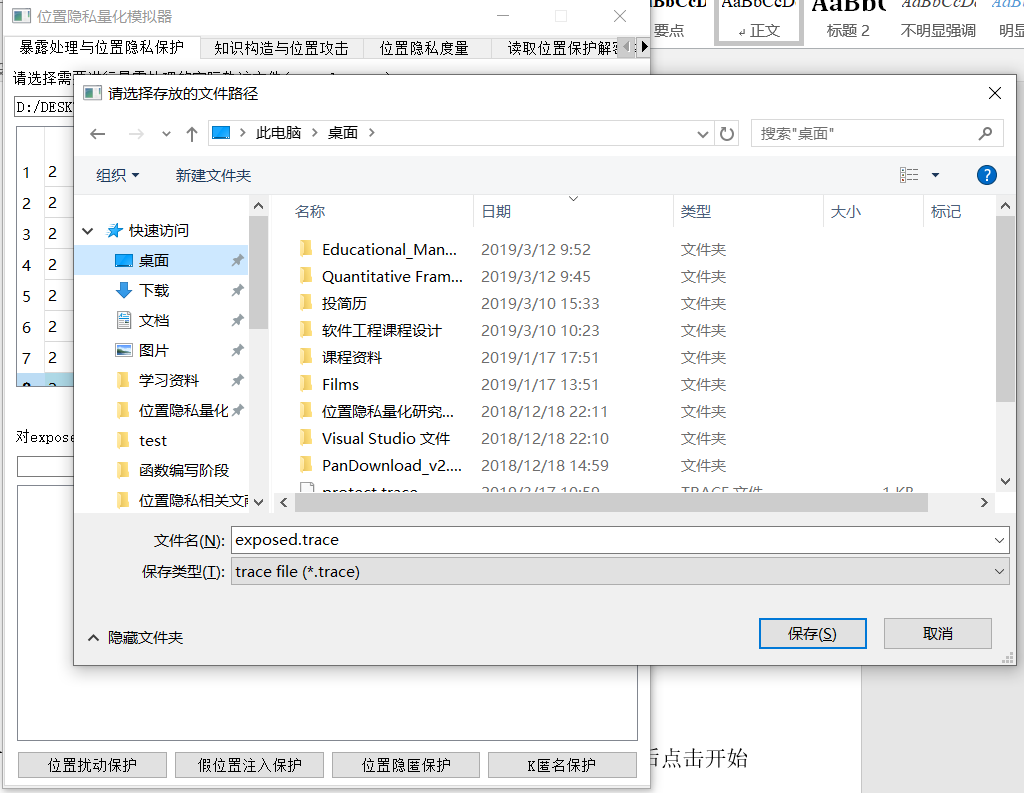
图5-1

图5-2

图5-3

首先我们打开实际轨迹文件，然后表格中会显示轨迹文件的内容，然后点击开始暴露处理，即可生成出每条轨迹的暴露情况，我们可以点击表头来按某一列的数据进行排序，再按导出暴露处理后的文件即可完成exposed.trace的生成。

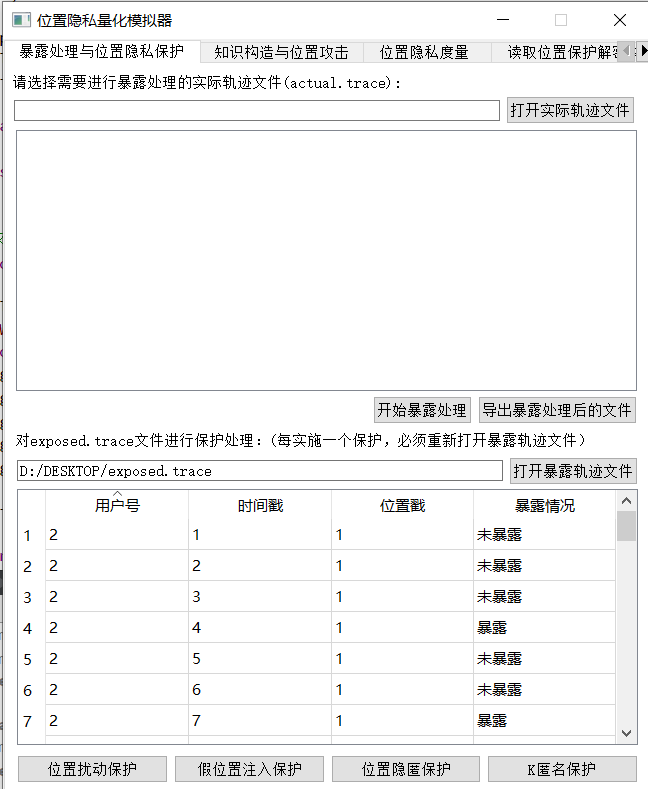
**位置扰动保护测试如图5-4~图5-7所示：**

图5-4

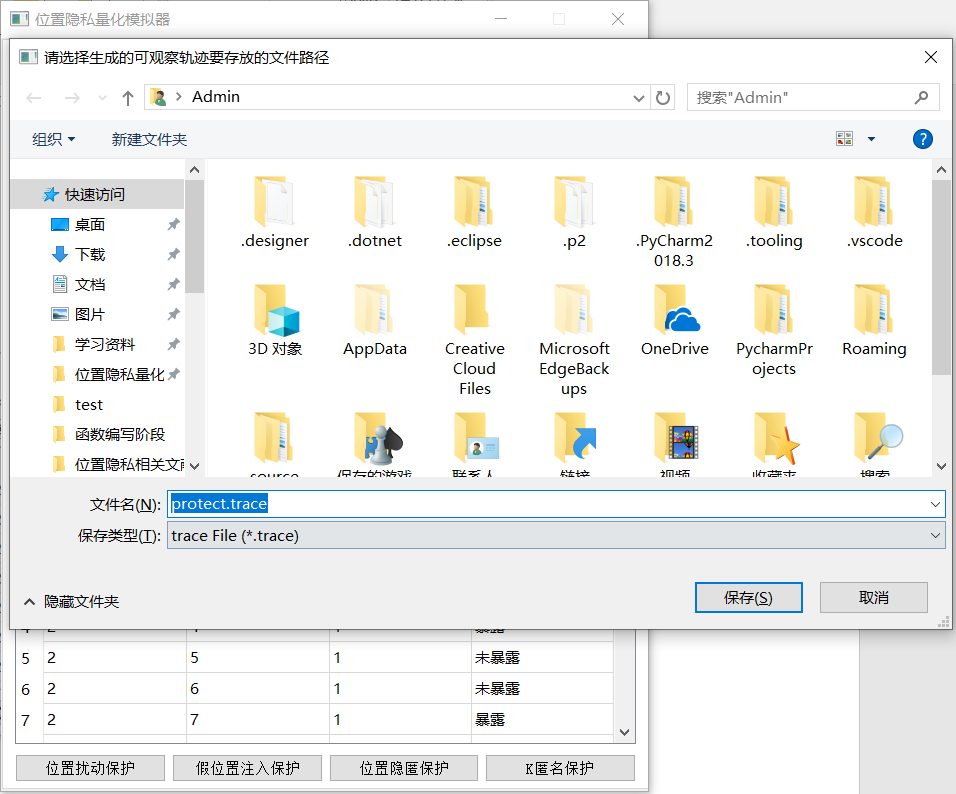
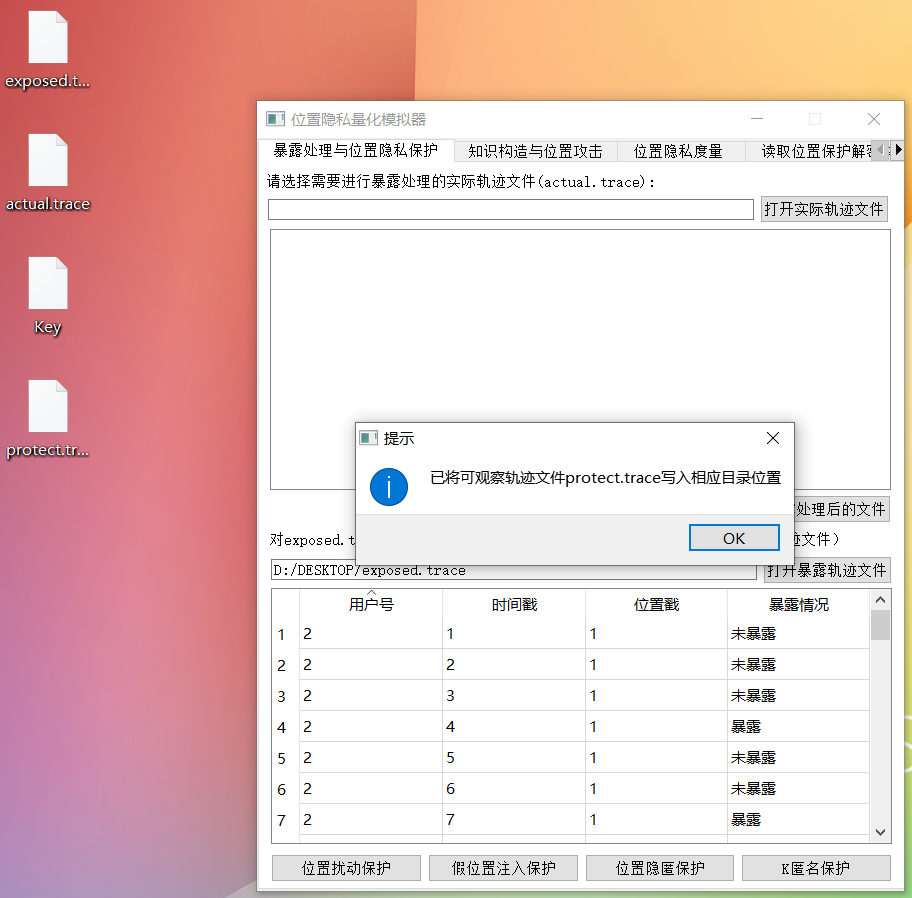
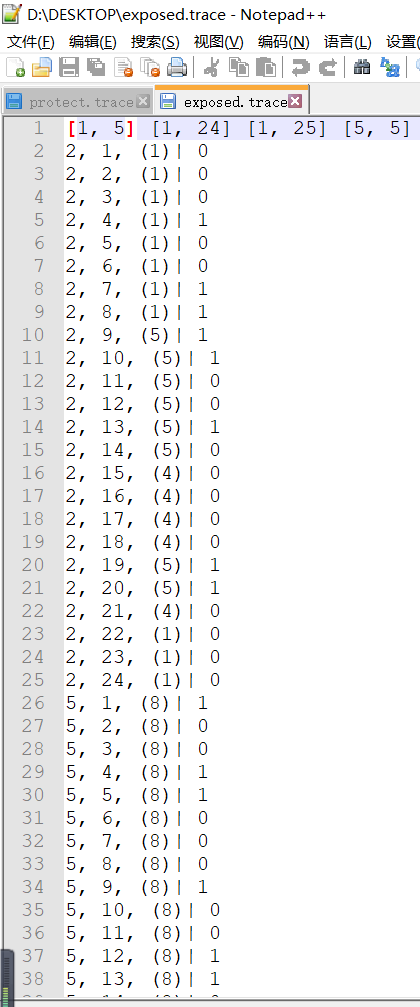
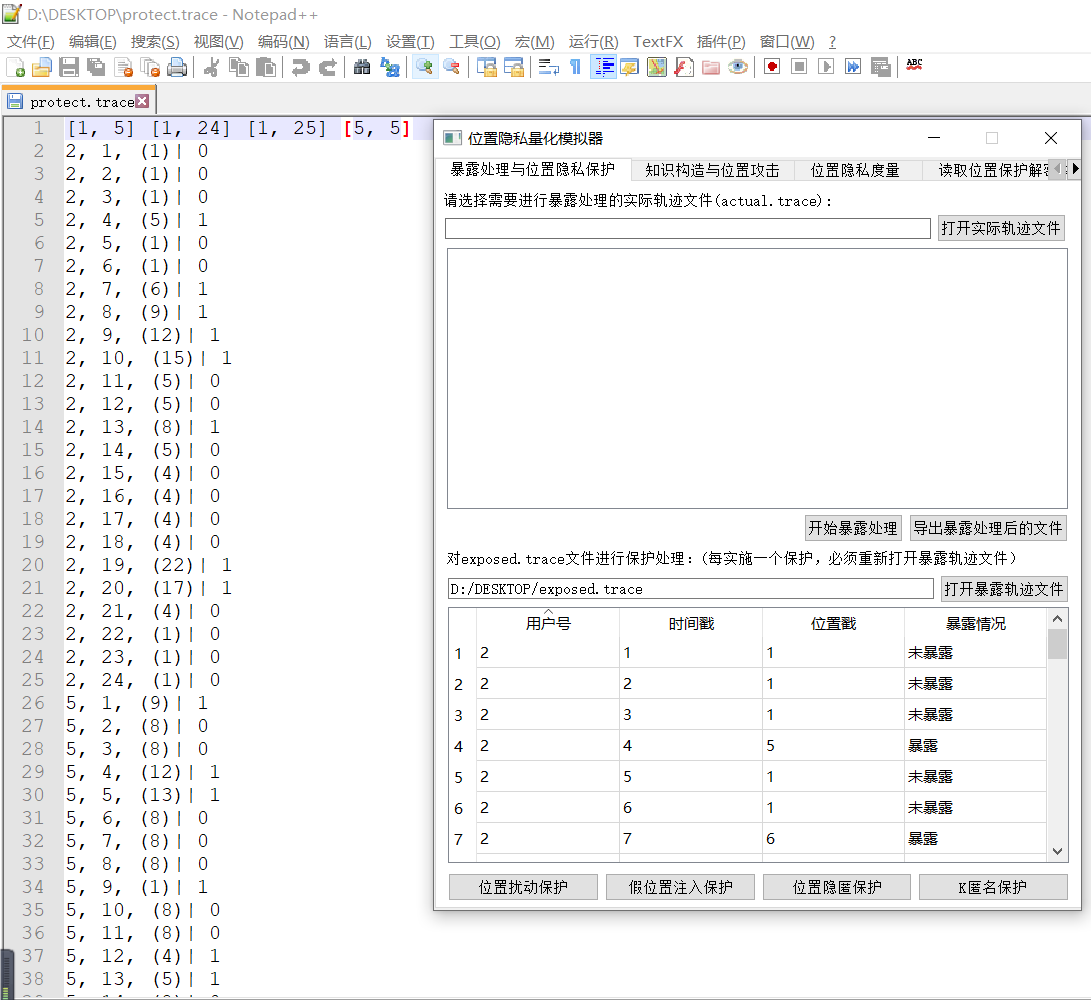
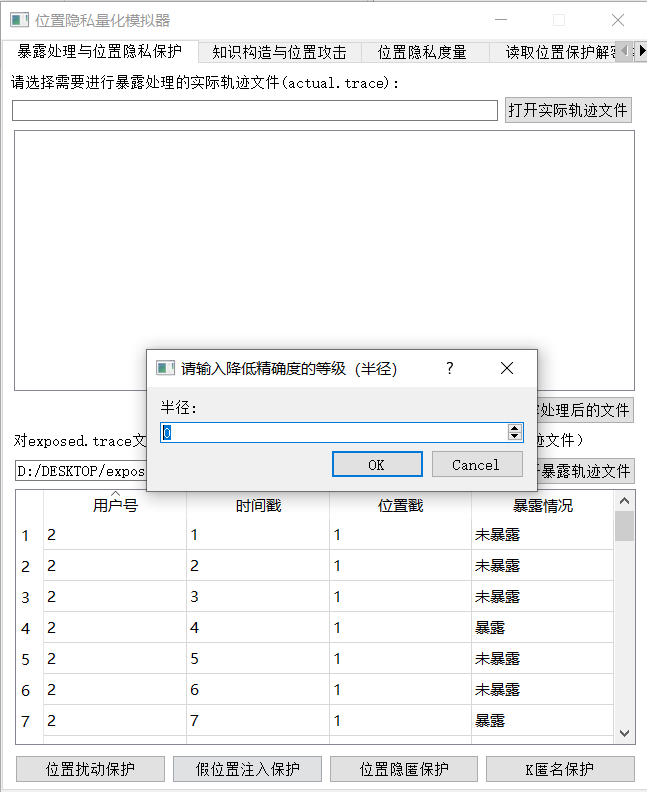
图5-5

图5-6

图5-7

这里我解释一下文件格式的问题，文件开头标明了用户戳、时间戳、位置戳的范围以及地区分割的行列数，然后后面每一条记录都是一个四元组，第一列是用户戳，第二列是时间戳，第三列则是位置戳，而第四列则是暴露情况，0表示没有暴露，1表示已经暴露。文件的详细介绍可以看我封装的项目文件的README帮助文档。

这里位置扰动从上面两图中，我们可以看到，被暴露的轨迹记录都是被处理过，将位置戳换成另外一个非真实的位置戳，说明算法没有问题。

**假位置注入保护测试如图5-8~图5-10所示：**

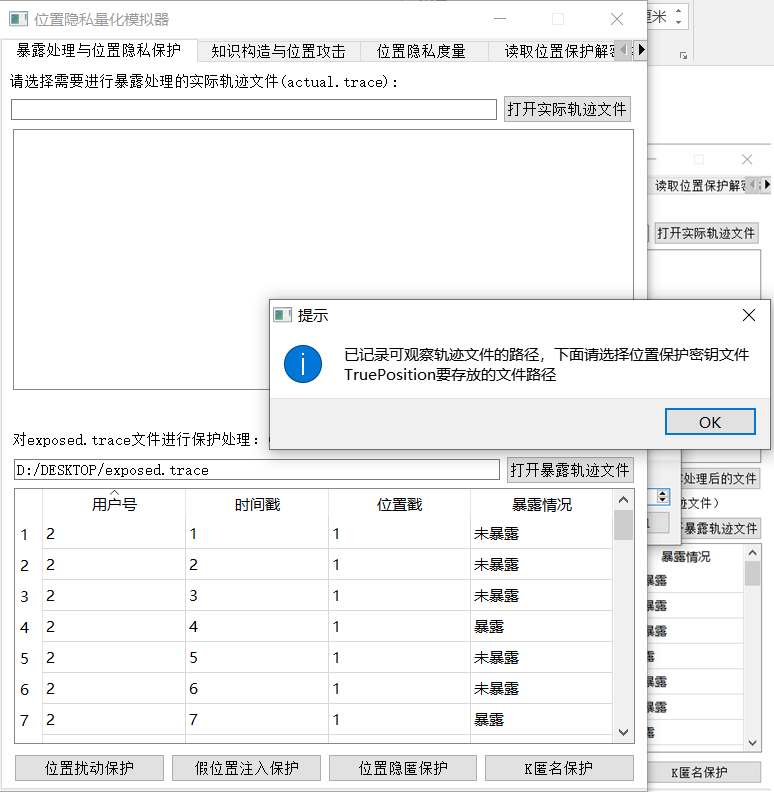
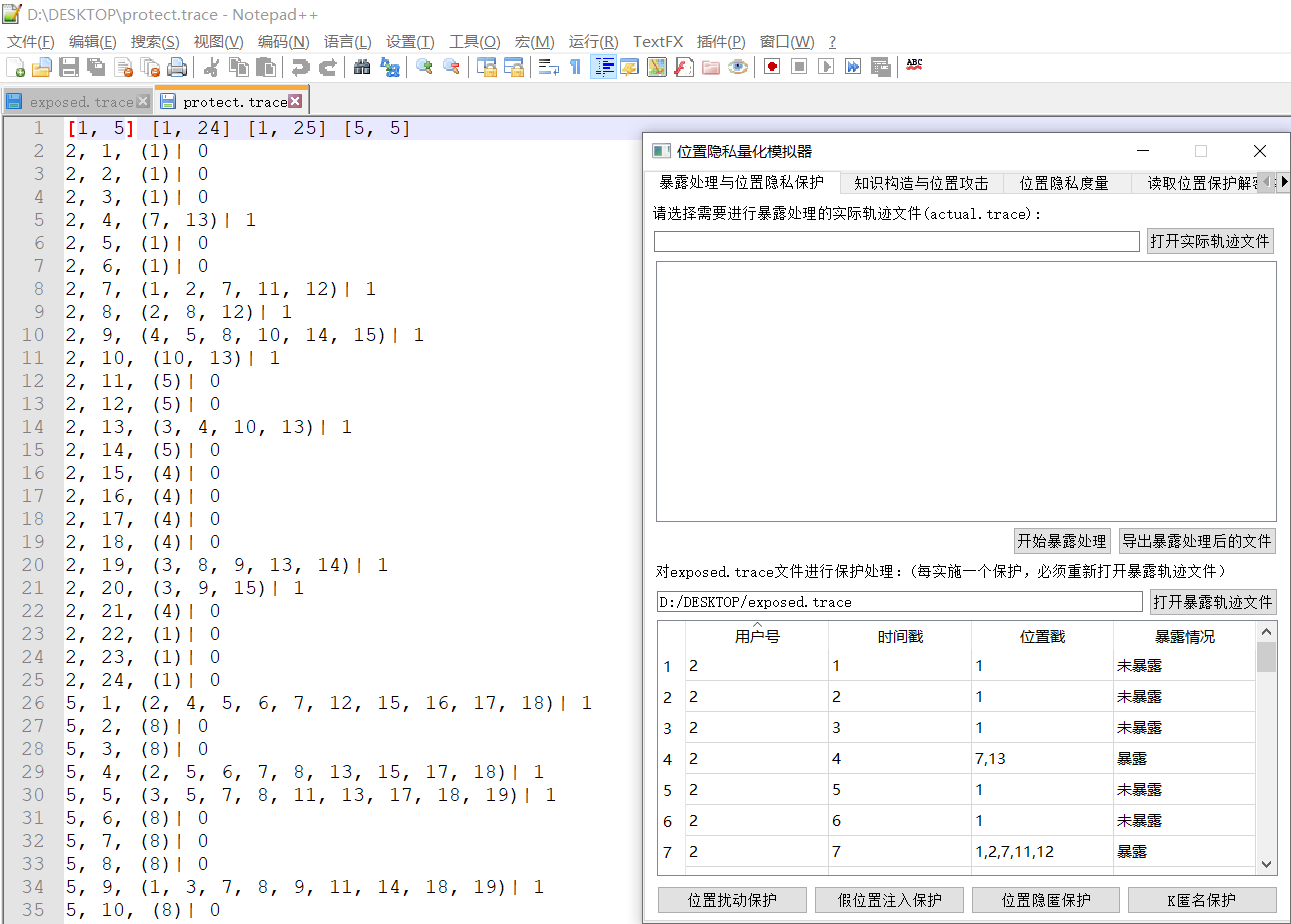
图5-8

图5-9

图5-10

假位置注入保护中我们可以看到，暴露了的轨迹记录会有复数个位置戳，其中这些位置戳是环绕原位置半径为r的位置戳中的一部分，这样能够降低轨迹暴露的辨识度。

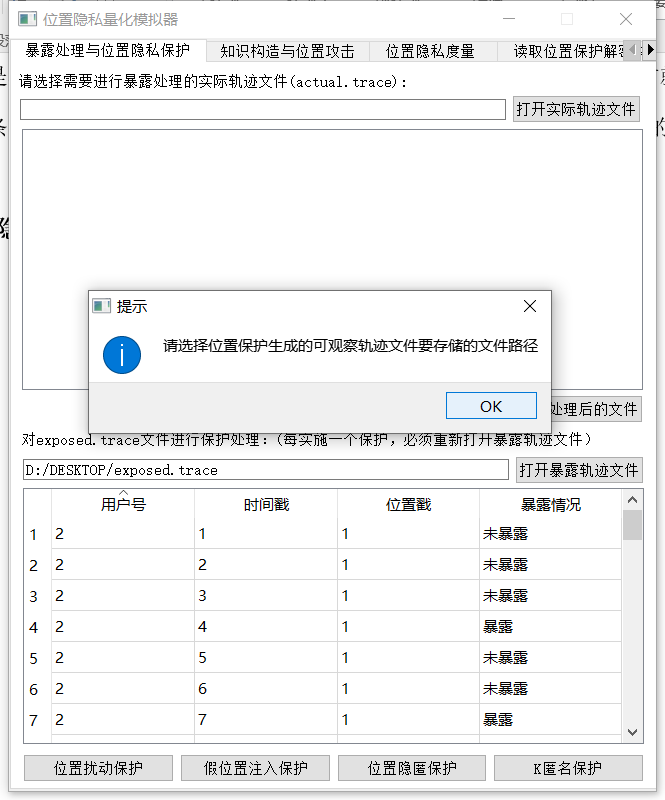
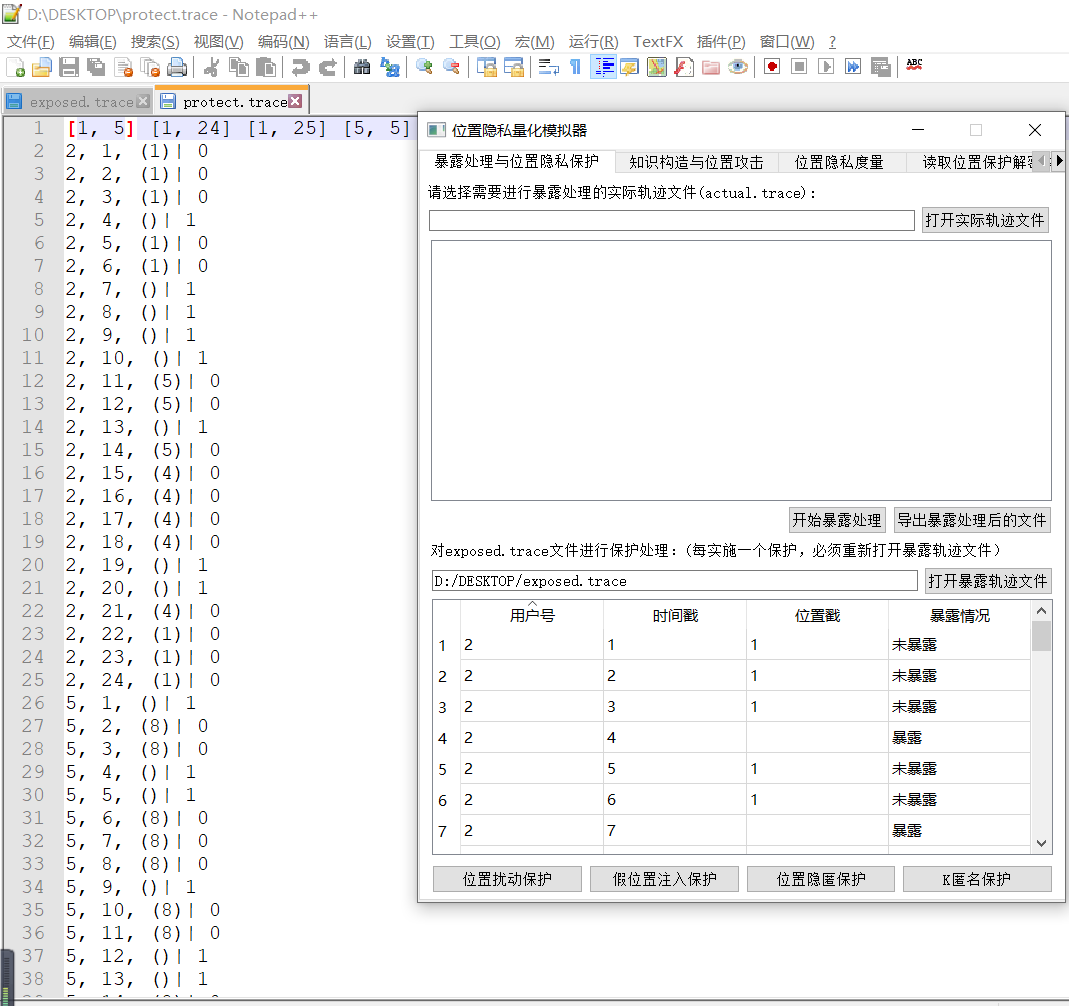
**位置隐匿保护如图5-11~图5-12所示：**

图5-11

图5-12

对于位置隐匿保护，我们可以看到暴露的轨迹记录的位置戳全部被置为空，所以能够起到预期的保护效果。

**K-匿名保护如图5-13~5-17所示：**

对于K匿名保护，就是创建虚假轨迹，然后混淆用户轨迹之间的匹配关系，具体效果如下：

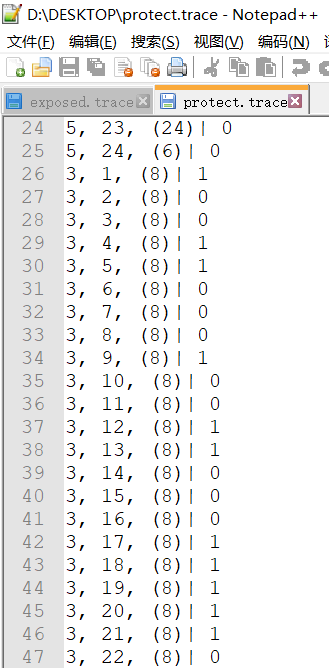
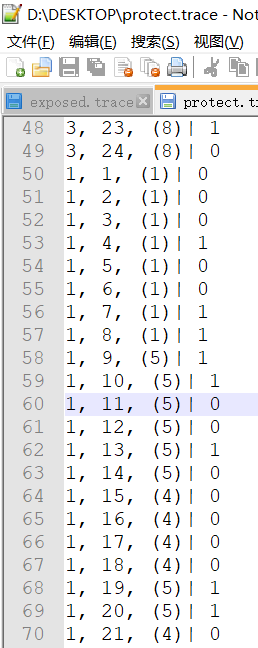
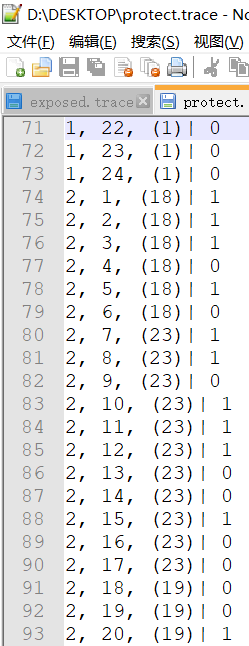
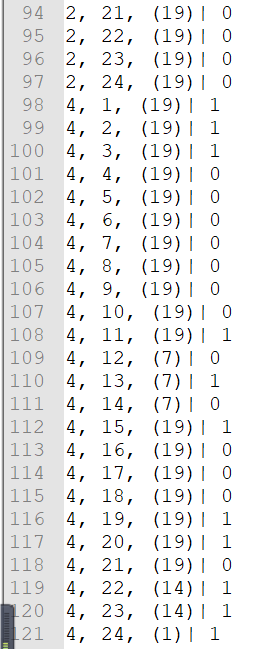
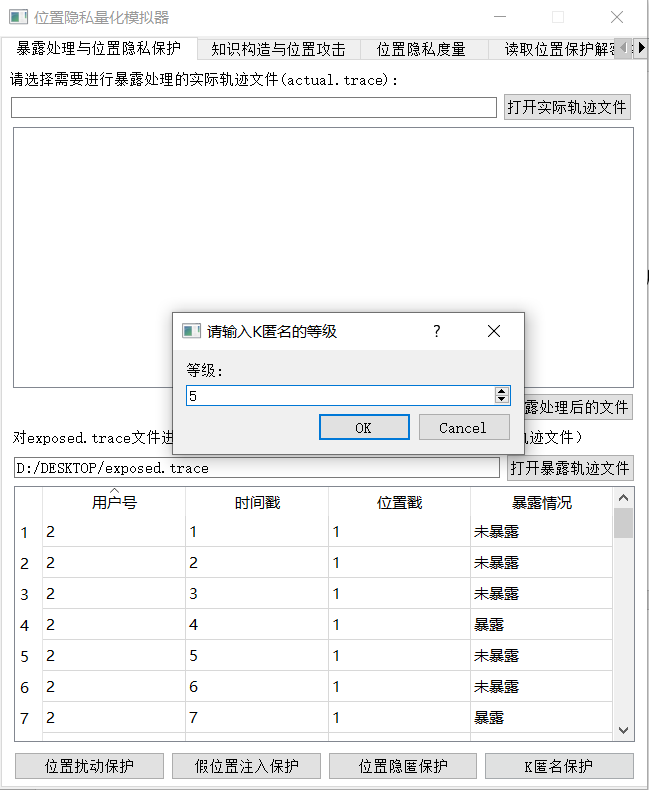
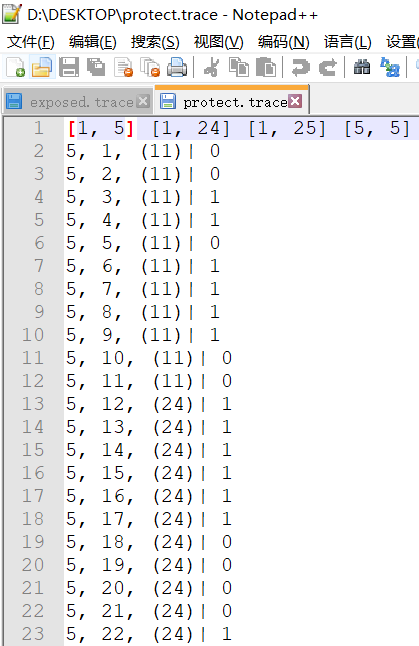
图5-12 图5-13

图5-14 ~ 5-17

我们可以看到，原来的轨迹只有两条，那么当我们输入K匿名等级为5时，就说明我们要虚拟构建出3条的轨迹，而且用户和轨迹的对应关系也和原先不一样了，原来的估计对应应该是用户5一直保持在原地，即在位置戳为8的位置不动，但是现在用户3占有了这条轨迹，反观用户5的轨迹是虚构出来的，能够起到保护作用。

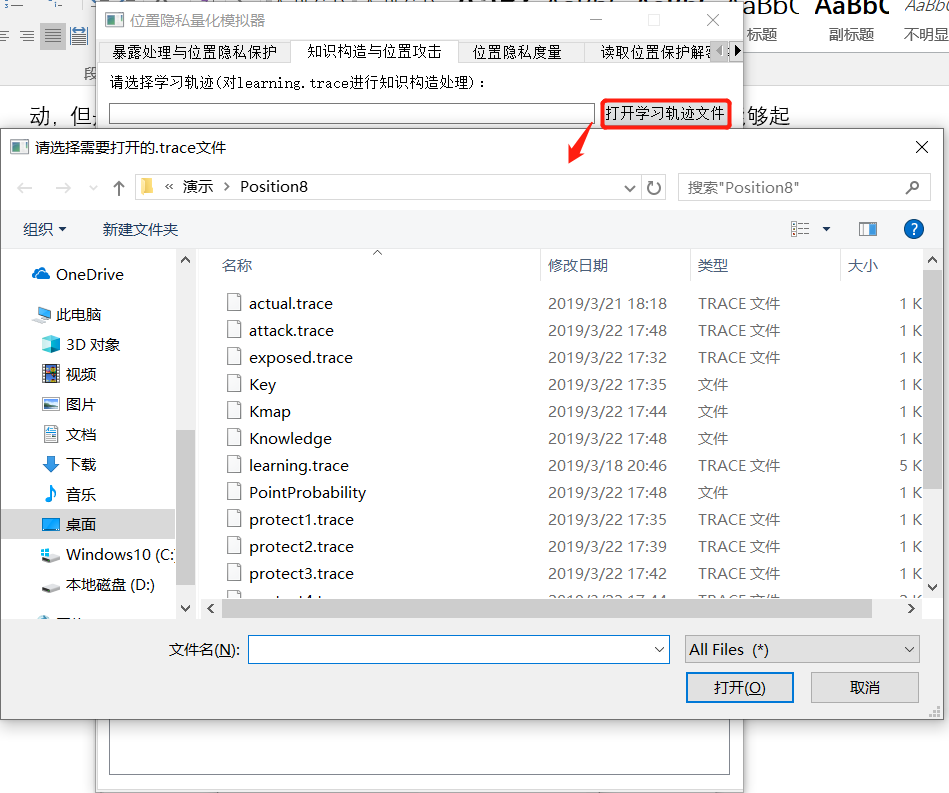
**知识构造和攻击测试如图5-18~图5-22所示：**

图5-18

图5-19

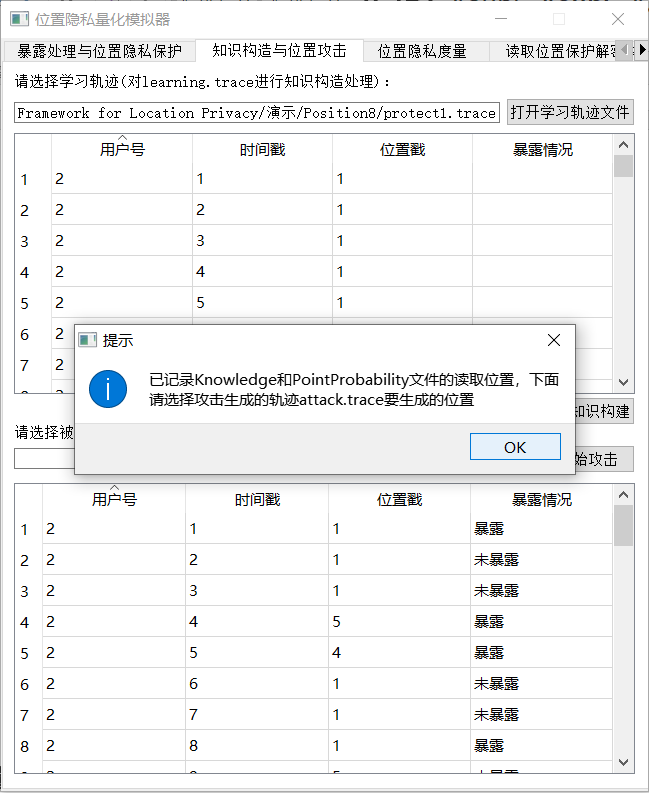
图5-20

图5-21

图5-22

我们可以看到，攻击功能的实现是完整的，但是精确度可能不会太高，原因是攻击的时候，我们没有进行反保护的操作，这样我们暴露出去的轨迹记录由于经过了保护算法对位置戳的处理，反而会对攻击方的预测造成不利的影响，当然攻击方的算法还有待改善。

**度量部分：**

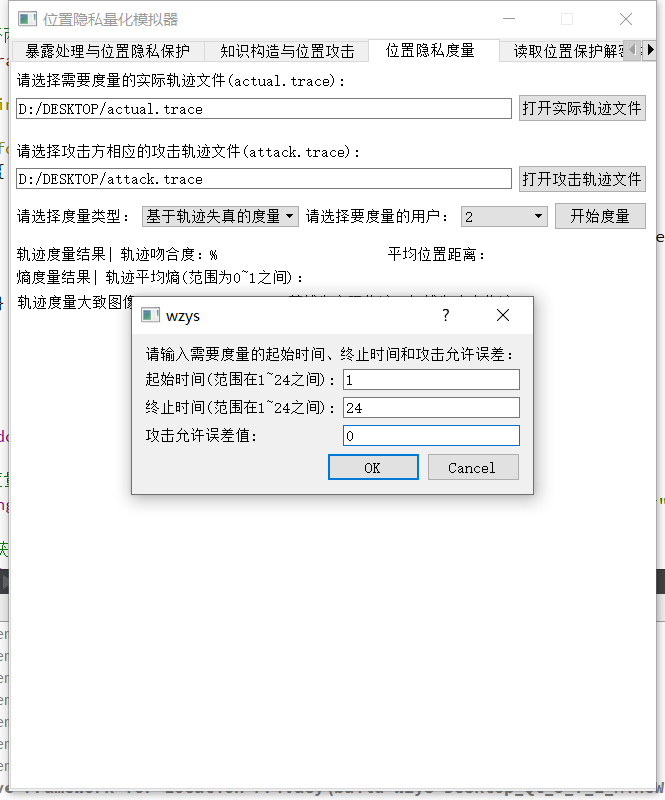
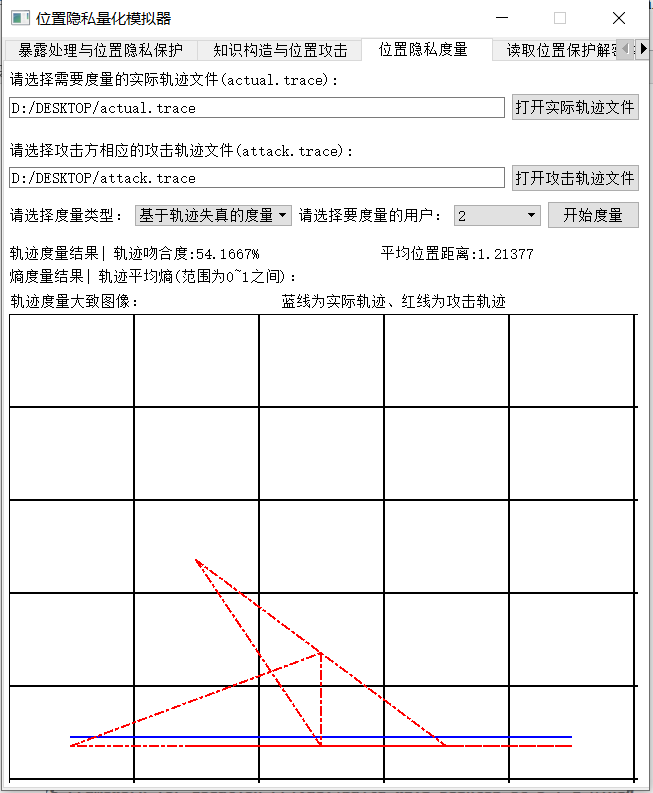
**基于轨迹失真的度量测试如图5-23~图5-24所示：**

图5-23

图5-24

首先我们在度量之前要先打开实际轨迹和攻击轨迹，然后选择度量类型，选择要度量的用户，最后按开始度量。弹出的一个框让用户是让用户来选择要度量的时间范围还有误差允许范围，比如图中，我填写的误差允许范围为0，那就说明一条轨迹只有当攻击轨迹和实际轨迹的位置戳完全相同时，这条轨迹才算作攻击成功。我们可以看到图中描述的用户轨迹是比较模糊的，这个轨迹图单纯是让用户看到被度量的轨迹轮廓大概是怎么样的。另外结果也显示在了屏幕中，其含义在算法设计中有详细介绍，这里我就不一一解释了。

至于基于熵的位置度量我就不截图了，就是直接生成一个文件，里面会写着用户在各个时刻的隐私水平（熵/最大熵值）。

**还原轨迹部分：**

**位置扰动保护轨迹的还原测试如图5-25~图5-28所示：**

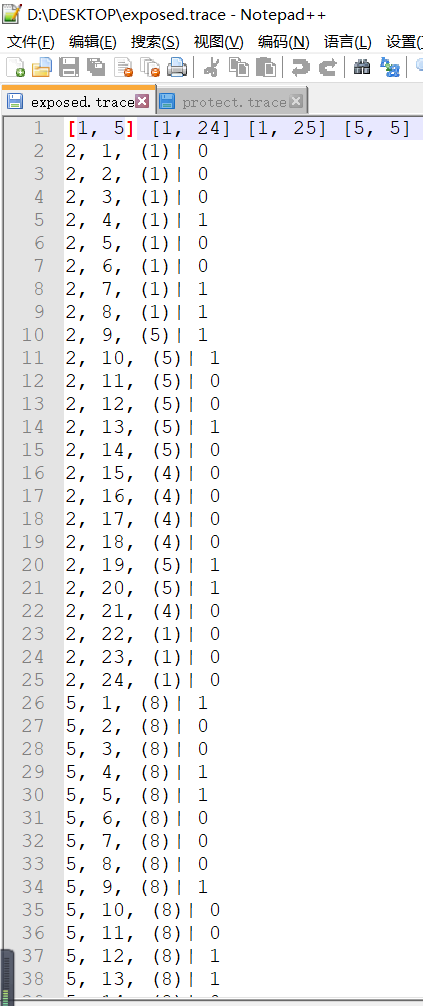
图5-25

图5- 26 图5-27

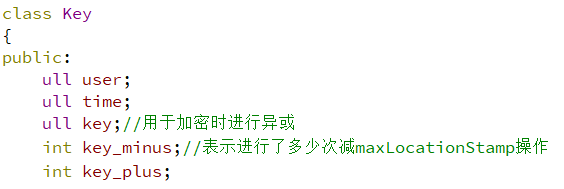
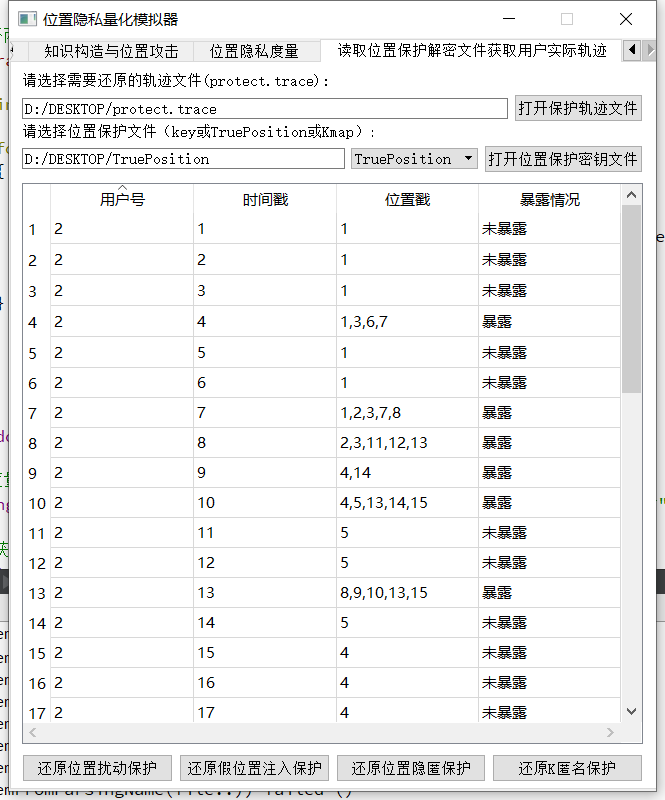
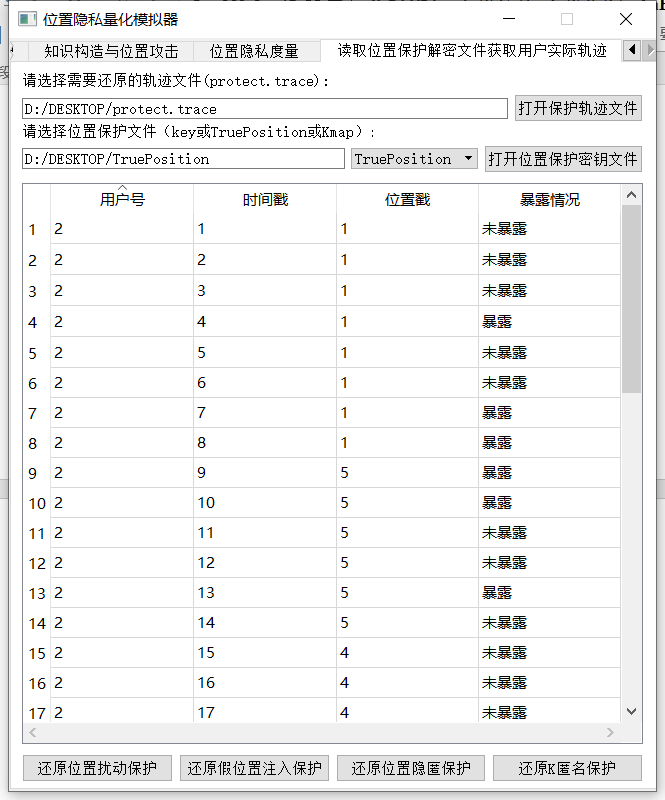
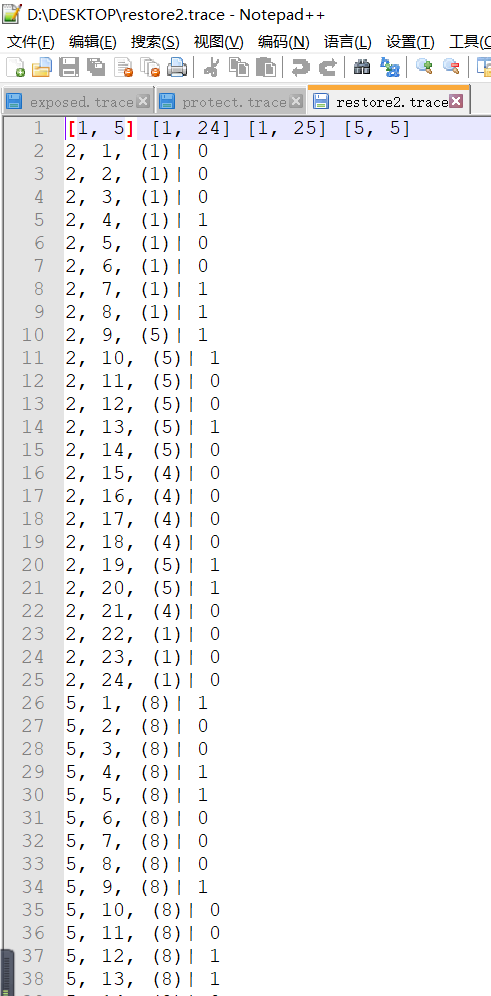
可以看到，还原出来的轨迹确实和原来的轨迹一模一样，说明还原成功，具体算法其实就是扰动的逆过程，key中存放的其实就是一个Key结构：

图5-28

**假位置注入轨迹的还原测试如图5-29~图5-30所示：**

图5-29

图5-30

还原出来的轨迹和原来的实际轨迹一样，说明确实还原成功了，算法思路就是将TruePosition里面存放的原来的轨迹读出来，没有什么算法可言，这里就不介绍了。

**位置隐匿保护轨迹的还原：**

位置隐匿的还原原理和假位置注入的还原原理相同，我就不截图了。

**K-匿名保护轨迹的还原测试如图5-31~图5-32所示：**

图5-31

图5-32

可以看到还原成功，并且也生成了相应的文件，这里就不截图了。具体原理其实就是在构造好虚假轨迹集合时，先记录下用户与其轨迹的对应关系，还原的时候只需要读出这个关系，以及轨迹集合就能够知道用户对应的真是轨迹是什么了。

# 五. 总结

总的来说，我们这次的程序用到的保护算法和度量算法都是比较简单的，我们简化了轨迹攻击和保护模型，没有对位置转移可行性进行假设（比如现实中，由于距离原因有的位置不可能在一个小时内到达，有的位置可能会有障碍）。

暴露处理方面，我们也没有针对具体的App进行分析，可能不同的App对位置暴露的情况是不一样的，有的App可能并不需要获取到整条轨迹就可以提供服务，而我们现在假设的是按照一定的百分比，对轨迹进行暴露处理，然后用一些算法进行保护。

知识构造和攻击方面，由于我们实验用的数据都是用路网位置生成器生成出来的位置轨迹数据，所以每个用户的轨迹是有一定的规律概率的。先用位置生成器生成一组学习轨迹数据，利用学习数据来生成位置转移概率矩阵，用来后面的概率推导是合理的，因为现实生活中每个用户每天的位置轨迹都是有一定的规律的，比如说一个老师，周一到周五白天都会去学校，这就是一个用户的位置概率，攻击方通过观测到的点，来推测最大概率出现的点。攻击的话是把用户的轨迹看成马尔科夫链，因为下一步只与前面的一步有关，首先在攻击的时候，判断暴露的点是否符合实际，不符合时间的话推测出一个符合实际的点作为新的推测点。最后通过后向算法从后往前推每个点出现的最大概率位置点作为该点的位置。

保护方面，同样地，在编写的时候我们没有考虑到现实中的场地限制，比如位置扰动保护，如果我们不进行场地限制的考虑，那么攻击方可能很容易就能推断出保护方提供的位置戳是假的，进而根据先验知识进行攻击，使得用户的隐私水平降低。又比如K匿名，我们可以进行结合聚类操作来进行改良，将相似的轨迹聚集起来然后再运用K匿名可以更好地保护用户的位置隐私。还有一些根据知识构造文件对轨迹进行保护的算法，也是有待实现的。

知识构造的算法的话概率准确度是挺高的，只要学习文件是完整的，效果就不会很差。但是在实际情况中学习文件是有缺失的，所以还是直接读取文件来算概率的话准确度就会比较低，需要用一些采样方式来采样才能提高准确度，这也是接下来需要继续做的事情。攻击算法的话针对没有做保护处理的数据的还原程度会比较高，普适性比较好，但是针对数据经过保护处理后，我需要知道用户是用了哪种保护算法才能更好的去解析保护算法，如果不知道用户用的哪种保护算法，那么我的攻击效果就会比较差。

关于后续工作，我们可能会对知识构造做进一步的改进，利用采样方法去采样不完整的学习轨迹。在攻击方面会去思考更好的算法，去攻击不同保护算法的数据，而不需要事先知道用户用了哪种保护算法。

度量方面，我们用的基于失真度的度量和基于熵的度量其实都是有缺点的，比如基于熵的度量考虑的对象太过于单一了，仅仅考虑攻击方的先验知识文件，而且如果是在未知攻击方采用什么形式构造知识文件的情况下，这种度量方法的度量结果其实并不够准确。

还原轨迹方面，我们的密钥文件都是存成二进制文件模式的，我们不能确定攻击方能不能将这些文件破解出来，这是需要进一步进行研究的。