Private GPS trail intersection

Abstract:

Exposure notification in GPS based contact tracing platform requires performing intersection of at least two GPS trails, however, these GPS trails are sensitive information about a user and hence requires performing intersection in a private manner. Low entropy in the GPS trails makes this problem even more challenging as brute force becomes feasible. In this paper, we propose methods and system design for performing private GPS trail intersection using secure cryptographic methods. We also highlight potential attacks which can be performed on the proposed techniques.

Introduction:

- Problem Statement:
    - Every App user records their GPS data every five minutes and stores it in the following format (latitude, longitude, ts) where latitude and longitude correspond to the geo-coordinates at timestamp ts. In this format, every day only 288 points are stored in the local database.
      A set of users Q among all the users get tested positive and upload their dataset of location trails. Rest of the remaining users P want to compute the intersection to check whether they have been in the same location as an infected user. However, Q and P do not want to leak their privacy by submitting the data in raw format hence they cryptographically share this data and perform some additional computations in order to compute the intersection of their trajectories in a secure manner.
    - A match is found between User i and User j IF -
        - *dist((latitude_i, longitude_i), (latitude_j, longithude_j))* < *dist_threshold* and
          (time_i - time_j) < time_threshold
          To avoid the exact computation, we can discretize the latitude and longitude values into bins and the extent of discretization can be chosen based on the *dist_threshold*.
    - Desirable properties:
        - Q can go offline after securely uploading their data
        - Server should not know the outcome of the results
- The problem can be cast as a set intersection protocol [3-7] in which some of the parties go offline once they upload their encrypted sets. In our solutions we require 2 non-colluding servers where the second server allows us to achieve a solution in which the infected people can go offline.
- There are further attacks which do target the privacy of users but instead maliciously inject wrong data. For example - An attacker who acts as an infected person and spoofs their GPS and sends coordinates of a different place to create panic. Proposed method can be extended to circumvent some of these malicious adversaries and will be part of our next version.

Method:

Apart from the cryptographic method based security, we add another layer of defense by adding rate limiting for all communication between server to client and server to server. This is to make sure that brute force attacks are infeasible. While the network communication latency already makes some of the brute force attacks infeasible, we still keep the rate limiting as a measure to strictly enforce the necessary query limit.

Notations:

P - healthy person, Q - Infected person, S1 - Server 1, S2 - Server 2

|P| - number of healthy person, |Q| - number of infected person

X - location data of healthy person, x - for a single entry in the location data

Y - location data of infected person, y - for a single entry in the location data

|Y| - number of entries in P's dataset, |X| - number of entries in Q's dataset

Registration Phase - Q goes offline after communicating and registering its data to the servers and doesn't interact at all with any server or client during the online phase.

Request Phase - P requests for data/computation to be performed in order to obtain the result and get exposure notification

Attacker taxonomy -

i) Malicious User - An attacker could be posing as a healthy person with the intent to leak an infected person's trail using GPS. There are potentially different attack targets for this user. For example - they might be targeting a single person to know whether they are infected or not. They might try to decrypt the location information of all individuals they receive.

ii) Malicious Server - One of the servers S1 or S2 could also act as an attacker and try to take sensitive information out of the **encrypted data** or try to interpret **encrypted results**. The goal of the rogue server could be to decrypt location information or obtain the results of the intersection.

iii) Collusion between the two servers - Both Server 1 and Server 2 could try to collude in order to leak information about either a healthy or infected person.

iv) Collusion between an App and one of the servers - The Malicious App described in i) could team up with a malicious server in ii) to target user's privacy. Another way of manifestation of this attacker is that the server also acts as an App provider. Another potential attack is where the server can deliberately inject a few points as an infected person to obtain the Healthy person's information.

v) Timestamp Attack - In this attack scheme, the attacker has the knowledge about potential interest points and they want to see whether a target person was present there or not. This is quite practical in the instances where the attacker knows the set of points in one which the target user will be present at any given point of time. Since we have imposed a limit on the number of queries, the attacker uses the same timestamp for all the location instances they want to query.

1) Hashing scheme:

b) Method

i) <u>Registration phase:</u>

ii) Q sends h(y)^r to S1 for a random r where h() is a collision resistance hash function. (for all y in Y and similarly for all x in X, in the following steps as well)

iii) S1 returns (h(y)^r)^k for a radom k

iv) Q computes ((h(y)^r)^k)^s, such that s*r = 1 and sends the result to Q (h(y)^k) to S2

v) Q goes offline

vi) <u>Request phase of a healthy person:</u>

vii) P sends h(x)^r for another random r to S1.

viii) S1 returns (h(x)^r)^k

ix) P computes ((h(x)^r)^k)^s, where s*r = 1

x) P downloads all h(y)^k from the server 2 and compares it with its own h(x)^k

c) Attacks

i) Malicious App: A malicious app, pretending to be a generic P can obtain h(y)^k but in order to know the y it has to know r which is known only by Q. The other attack which a malicious app can perform is to bruteforce all possible values for h(x)^k and comparing it with h(y)^k, however this requires sending multiple queries on the server 1 (since P doesnt

know k), however, we have kept the condition that only one query (for a limited bucket of locations) can be made in a day, so brute force is also not possible.

ii) Malicious Server: A malicious server can try to estimate X or Y from h(X)^r or h(Y)^r respectively, but again the random masks r are only known to P and Q.

iii) Collusion between the two servers: S1 can send k inverse for all users and S2 can apply this k inverse. The security of the scheme breaks in this scenario.

iv) Collusion between a healthy person's App and S2: The scheme breaks in this case as S2 can pose as an app user P and brute force all the points over Y.

v) Collusion between a healthy person's App and S1: The scheme also breaks if S1 can also pose as an app user P, since he knows k and can do a brute force by comparing all the h(y)^k downloaded from S2

vi) Collusion between an infected person's App and servers: Nothing is revealed

vii) Time stamp attack: Not secure against this attack

Computation Efficiency

viii) For Q - Q needs to raise every point in their dataset to the power r and then raise to the power s. Hence it is O(|Y|) for a constant time exponentiation operation.

ix) For P - P also needs to perform the same operation as Q. In addition, P also performs |Q|*|Y| equality operations for each element in |X|. Hence it is O(|X| * |Q|* |Y|)

x) For Server 1 - Server 1 needs to perform exponentiation operation for all elements in every set, which is O(|P| * |X| + |Q| * |Y|)

xi) For Server 2 - Server 2 doesn't perform any computation, it just hosts the datasets coming from different Qs and passes it to Ps.

d) Communication Efficiency

i) For Q - It queries the Server 1 with ita whole set Y once and then uploads it to Server 2. This is O(|Y|).

ii) For P - P uploads and downloads its data and also downloads Y's encrypted data O(|X| + |Y|)

iii) For Server 1 - O(|P| * |X| + |Q| * |Y|)

iv) For Server 2 - O(|P| * |X| + |Q| * |Y|)

2) Hashing scheme with Server Communication:

a) Method

i) Registration phase:

ii) Q sends h(y)^r to S1 for a random r where h() is a hash function. (for all y in Y and similarly for all x in X, in the following steps as well)

iii) S1 sends (h(y)^r)^{k_1} for a radom k_1 to S2

iv) S2 sends ((h(y)^r)^{k_1})^{k_2} for a random k_2 to Q

v) Q computes ((h(y)^r)^{k_1 k _2})^s, such that s*r = 1 and sends the result to Q (h(y)^{k_1 k_2}) to S2

vi) Q goes offline

vii) Request phase of a healthy person:

viii) P sends h(x)^r for another random r to S1.

ix) S1 sends (h(x)^r)^{k_1} to S2

x) S2 sends (h(x)^{r})^{k_1 k_2} to P

xi) P computes ((h(x)^r)^{k_1 k _2})^s, where s*r = 1

xii) P downloads all h(y)^{k_1 k_2} from the server 2 and compares it with its own h(x)^{k_1 k_2}

b) Attacks

i) Malicious App: A malicious app, pretending to be a generic P can obtain h(y)^k but in order to know the y it has to know r which is known only by Q. The other attack which a

malicious app can perform is to bruteforce all possible values for h(x)^k and comparing it with h(y)^k, however this requires sending multiple queries on the server 1 (since P doesnt know k), however, we have kept the condition that only one query (for a limited bucket of locations) can be made in a day, so brute force is also not possible.

ii) Malicious Server: A malicious server can try to estimate X or Y from h(X)^r or h(Y)^r respectively, but again the random masks r are only known to P and Q.

iii) Collusion between the two servers: S1 can send k inverse for all users and S2 can apply this k inverse. The security of the scheme breaks in this scenario.

iv) Collusion between a healthy person's App and S2: The scheme breaks in this case as S2 can pose as an app user P and brute force all the points over Y.

v) Collusion between a healthy person's App and S1: The scheme also breaks if S1 can also pose as an app user P, since he knows k and can do a brute force by comparing all the h(y)^k downloaded from S2

vi) Collusion between an infected person's App and servers: Nothing is revealed

vii) Time stamp attack: Not secure against this attack

Computation Efficiency

viii) For Q - Q needs to raise every point in their dataset to the power r and then raise to the power s. Hence it is O(|Y|) for a constant time exponentiation operation.

ix) For P - P also needs to perform the same operation as Q. In addition, P also performs |Q|*|Y| equality operations for each element in |X|. Hence it is O(|X| * |Q|* |Y|)

x) For Server 1 - Server 1 needs to perform exponentiation operation for all elements in every set, which is O(|P| * |X| + |Q| * |Y|)

xi) For Server 2 - Server 2 doesn't perform any computation, it just hosts the datasets coming from different Qs and passes it to Ps.

c) Communication Efficiency

i) For Q - It queries the Server 1 with ita whole set Y once and then uploads it to Server 2. This is O(|Y|).

ii) For P - P uploads and downloads its data and also downloads Y's encrypted data O(|X| + |Y|)

iii) For Server 1 - O(|P| * |X| + |Q| * |Y|)

iv) For Server 2 - O(|P| * |X| + |Q| * |Y|)

v)

3) ElGamal Encryption Scheme :

a) Method:

i) Registration phase:

ii) S2 chooses a key pair for ElGamal encryption (pk_Q,sk_Q) and sends pk_Q to Q

iii) Q encrypts each location in Y using ElGamal Encryption C_q = (c1,c2) = Enc(pk_Q, y)^t for a random mask t (use different t if we don't want to leak the ratio of locations) and sends t to S1 and C_q to S2. [c1=g^yh^r,c2=g^r]

iv) Q goes offline

Request phase of a healthy person:

v) P receives t from S1

vi) P chooses (pk_P,sk_P) and encrypts his location C_p = Enc(pk_P, - x)^t and sends C_p and pk_P to Server 2

vii) Server 2 decrypts C_q using sk_Q, to get g^{y*t} and encrypts it again under the key pk_P:

(1) C_q'= Enc(pk_P, Dec(sk_Q, C_q))

(2) ct_i = (C_p * C_q')^s for a random s and sends it to P

viii) P decrypts ct_i with sk_P to get g^{s*t(y-x)}

(1) Dec (sk_P, ct_i) = Dec (sk_P, (C_p + C_q')^s)

(2) = Dec (sk_P, (d * Enc(pk_P, Dec(sk_Q, C_q))) ^s)

(3) = g^{s*t(y-x)}

ix) The answer is 1 if the location is the same, otherwise the answer is a non-zero element which reveals nothing about Y.

Note that communication channels are encrypted

b) Attacks

i) Malicious App: A malicious app pretending to be a generic P can try to bruteforce over all possible sets of coordinates, however, the aforementioned rate limiting defense makes sure that only a limited set of points can be queried.

ii) Malicious Server: If server 2 acts as a malicious entity then the only knowledge they have is about the key pairs (pk_Q, sk_Q) and C_q and ct_i. Since the server 2 does not have the access to t they can not obtain any y from C_q. This again maps back to breaking ElGamal encryption. If server 1 acts as a malicious entity then the only knowledge they have is about the t and not ciphertext.

iii) Collusion between the two servers: If the 2 servers collude then Server 1 will know the sk and then security breaks.

iv) Collusion between a healthy person's App and S1: Nothing is leaked

v) Collusion between a healthy person's App and S2: healthy learns t and this means that S2 can approximate the locations of Q.

vi) Collusion between an infected person's App and one of the server: Nothing is leaked

vii) Time stamp attack: Only if we introduce a 1 round communication between S1 and S2

c) Computation Efficiency per Q

i) For Q - |Y| encryptions

ii) For P - |Y| encryptions and |Y| decryptions

iii) For Server 1 - no computation

iv) For Server 2 - |Y| encryptions and |Y| decryptions per P

d) Communication Efficiency per Q

i) For Q - |Y| ciphertexts

ii) For P - |Y| encryptions

iii) For Server 1 - |P| messages

iv) For Server 2 - |Y| encryptions per P

4) Using leveled and low depth FHE (PALISIDE library of [CRPR19])

a) Method: Server 1 here is used for FHE computation and for storing the encrypted locations by the infected user. Server 2 does the job of generating keys and performing partial decryption of the results and returning it to P.

i) Sketch of the solution:

(1) Every user keeps tracks of their 100x100 km tile

(2) S2 keeps pk, sk for each tile

(3) Each phone uploads (tile number, Enc(pk, X), Enc(pk, r))

(4) If phone is infected, then (tile number, Enc(pk, Y), is used

(5) S1 concatenates all infected locations into CT (below)

(6) S1 finds all phones P that are in same tile_number. And computes CT+R

(7) S1 sends that to S2

(8) S2 decrypts with sk for that tile

ii) Registration phase:
iii) Server 2 chooses a key pair for packed FHE encryption (pk_Q,sk_Q) and sends pk_Q to Q
iv) Q encrypts each location y_i, c_i= Enc(pk, y_i) and sends the cipher text to Server 1 (Q can go completely offline).
v) Request phase:
vi) P receives pk_Q from Server 2
vii) P encrypts his location d= Enc(pk,x) and R= Enc(pk, r) for random r and sends (d,R) to Server 1
viii) Server 1 computes ct_i = c_i - d =Enc(pk, y_i) - Enc(pk,x) and CT= Enc(pk, s) *ct_1 * ct_2*…*ct_|Y| + R = Enc (s((x-y_1)*...*(x-y_|Y|)) + R) for random s and sends CT to server 2
(CT is the sum of ALL the infected locations).
ix) Server 2 decrypts dec(CT, sk_Q) and gets either 0 + R or random + R and sends the answer to P
x) P checks if solution = R

Note that communication channels are encrypted

b) Attacks
   i) Malicious App: No information is leaked.
   ii) Malicious Server : Either of the servers learn nothing
   iii) Collusion between the two servers: If the 2 servers collude then Server 1 will know the sk and then security breaks
   iv) Collusion between a healthy person's App and S1: No information is leaked
   v) Collusion between a healthy person's App and S2: No information is leaked
   vi) Collusion between an infected person's App and one of the servers: No information is leaked
   vii) Time stamp attack: secure as described above
c) Computation Efficiency
   i) For Q - |Y| encryptions
   ii) For P - 1 encryption per location of P or O(log |Y|) encryptions with the depth optimization
   iii) For Server 1 - circuit of depth O(log |Y|) or log(log(|Y|/n)) with depth optimization and n buckets
   iv) For Server 2 - 1 decryption or n if we use n buckets
d) Communication Efficiency
   i) For Q - |Y| ciphertexts
   ii) For P - 1 ciphertext per location of P or O(log |Y|) ciphertexts with the depth optimization or O(n log(|Q|/n)) if we use n buckets
   iii) For Server 1 - 1 ciphertext or n if we use n buckets per P
   iv) For Server 2 - 1 message or n if we use n buckets per P

Multiplicative depth of FHE:

Server 1 homomorphically evaluates the polynomial $F(x, Y) = s \prod_{y \in Y}(x-y)$ of degree |Y|. Naive homomorphic computation of this polynomial would require depth O(log |Y|). We can reduce the depth as follows:

- P sends the encrypted coefficients of the polynomial based on x: d_j = Enc(x^{2^{j}}) for $0 \le j \le \log |Y|$, i.e., encryptions of x, x^2,x^4 etc...

- Server 1 computes the encrypted polynomial coefficients which are based on Y using c_i received in step iv)* and it also computes the encryptions of all the powers of x (i.e., encryptions of x^3,x^5, x^6 etc...) using the binary representation of the exponents and d_j
- Server 1 evaluates the polynomial using d_j and c_i

The depth of the FHE is reduced to log(log|Y|). Moreover, if we batch the locations of Q in buckets of size n then the depth is log(log(|Y|/n)).

Conclusion:

In this paper we proposed several schemes for privately computing intersections of GPS trails. We elucidate all three key aspects of these methods - Privacy, Communication and Computation efficiency. In view of higher privacy guarantee, simplicity and efficient communication, we recommend the adoption of Method 4.

Other references

1. [CRPR19]: David B. Cousins, Gerard Ryan, Yuriy Polyakov, and Kurt Rohloff. PALISADE. https://gitlab.com/palisade, 2019.
2. C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In 1986 IEEE Symposium on Security and Privacy, pages 134–134, April 1986.
3. Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Proc. of the 1st ACM Conference on Electronic Commerce, pages 78–86. ACM Press, 1999.
4. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Proc. of EUROCRYPT 2004, pages 1–19. Springer, 2004.
5. Ronald Fagin, Moni Naor, Peter Winkler. Comparing Information Without Leaking It. Commun. ACM 39(5): 77-85 (1996)
6. Moni Naor, Benny Pinkas. Oblivious Transfer with Adaptive Queries. CRYPTO 1999: pages 573-590. Springer, 2004.
7. Berke A, Bakker M, Vepakomma P, Larson K, Pentland A. Assessing Disease Exposure Risk with Location Data: AProposal for Cryptographic Preservation of Privacy. arXiv:2003.14412
8. Location context in Google Apple API for Exposure Notification. Raskar R, Singh A, Zimmerman S. https://github.com/PrivateKit/PrivacyDocuments/blob/master/LocationContext.pdf

Contributors

Abhishek Singh, MIT Media Lab
Antigoni Polychroniadou, JPM AI Research
Mayank Varia, BU
Prabhanjan Ananth, UCSB
Ramesh Raskar, MIT Media Lab