Real World Treasure Hunt Game with Secured Location Info

CHEN BO-YU[1], CHANG SHENG-YUAN[1]

[1]Department of Computer Science and Information Engineering, National Taiwan University

Abstract

The study to encrypt personal location information in homomorphic encryption allows the client and the server to knowledge if the client is inside a certain geofence without the need to share the precise location information. Client may keep their location information secured, and the server may keep the geofence size and position both secured. The practice of encrypting longitude and latitude information in homomorphic encryption and send it back and forth between the client and the server allows the server to determine a yes or no problem, and the client only needs to decrypt the additional calculations adding to the homomorphic encryption by the server without knowing the actual meaning of the calculations. The method maintains that client and server have no precise location information shared, and protects the server from having geofence being cracked by the client.

Introduction

A treasure hunting game requires both client and server acknowledgement of both the actual position of the treasure and the actual position where the client locates. This may cause manipulations since that the server may continuously collect location information from the client and utilize the data with risk of harming the client. Also, the client may obtain the location information of the geofence from the server which some of the geofence is meaningful and valuable for specific purposes. Therefore, we introduce a method to transform and validate the location information in homomorphic encryption to protect valuable information from both the client and the server.

Homomorphic encryption by the definition from Webster's dictionary is as "a mapping of a mathematical set into another set or itself in such a way that the result obtained by applying the operations to element of the first set is mapped onto the result obtained by applying the corresponding operations to their respective images in the second set". Which by the words, it means that homomorphic encryption is a method that one can directly process calculations in an encrypted manner without having to decrypt the data back to plaintext, solve the problem, and then encrypt the data back.

Geofence is a geographical grid containing certain information. In this case, the geofence is a location grid that contains the treasure allowing the client to find treasure in such location, but in other cases, the geofence can contain other information such as water restriction area, tsunami warning area, bombard area and so on. These geofence may contain information not allowed to be access by the client, hence, the homomorphic encryption is intergrated in with geofence to protect the geofence not being acknowledged.

Solutions Adopted in the System

Crypto-Geofence

Crypto-Geofence [1] is a method that client sends a homomorphic encrypted longitude and latitude to the server, then the server will apply calculation to it with geofence information and send back to client. Finally, the client will be able to determine if him or herself is inside the geofence by checking the decrypted data is positive or negative.

In more precise, the server will calculate the offset to the north, south, east, and west edge of the geofence. By letting the positive value means that the location is bounded by the edge, client can determine whether if they are inside the geofence or not simply by checking the positive or negative values. However, the manititude of the offset values are vulnerable for server that it may easily calculate the geofence size and locations. Therefore, the server will have to multiply the offset by a positive random number to prevent client inverse calculation to determine the geofence size and locations.

If we apply Crypto-Geofence to our problem, we may find out that since server has to know if the client is inside the geofence or not, the client who received the positive or negative values can manipulate the results of the calculation and lie to the server. Hence, for this case study, protection against geofence leak is guaranteed, whereas, the validity of response if client is inside the geofence is not assured.

Privacy-Preserving Proof-of-Location using Homomorphic Encryption

This solution[2] is using homomorphic encryption to proof whether the location is inside the polygon or not. The polygon is seen as different straight lines that closes each other together. If a point is inside the polygon, the point will be inside the positive region of lines not equal to that in the negative regions of lines. Then if the point is outside of the polygon, the point will be inside both positive and negative regions with the same number of lines.

Thus, to explain more explicitly, the edge is seen as line equations. When we enter the x and y into the line equation, the result will show the point is in a positive or negative region. For example as Figure 1, if the point is at the left of three line equations, the result will be two positives and one negatives, which we can state that the point is inside the edges. What if the point is out of the line equations, the result will be two positives and two negatives showing that the point is out of the polygon.

The algorithm of the computation is in Algorithm 1, which shows the red part is code executed in client side, and the blue part is code executed in the server side. First the client sends homomorphic encrypted location to the server. Then the server computes the positive and negative values of the location with the edges and send back to the client. When the server calculates the result, a mask is applied to the result to the client that the client may not know the genuine result is. After that, the client may decrypt the intermediate result, mask the results, and send back to the server. Finally, the server de-masks the result and may believe that the client is inside the polygon if the result if wn is not zero.

By the algorithm, the client is able to modify the results and send it back to the server making the final result leads to wn is not zero, which is tested and proved locally. Also, the algorithm of mask only contains $2^{2*edges}$ possible combinations, the client can easily brutal force out the geofence location, causing the server leaks the geofence location data.

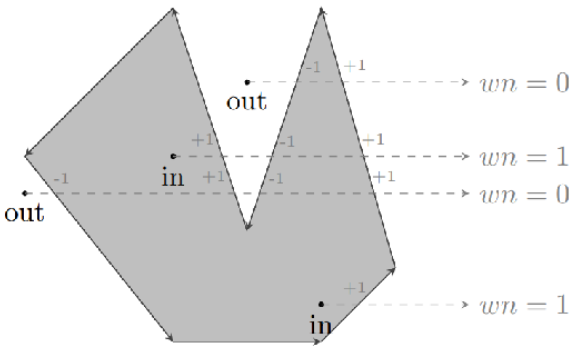

Figure 1: *Examples for points inside the polygon and outside the polygon counting positive numbers of edge passed.*
Source: `https://www.diva-portal.org/smash/record.jsf?dswid=-8729&pid=diva2%3A14458`

Algorithm 1 Vectorised computation with masking (red: client; blue: server)

1: wn = 0, point P
2: P = Enc(pk, P)
3: Send P
4: Vertices $\{V_i\}$, flags flag1, flag2.
5: Compute $\Delta x, \Delta y$:
$\Delta x = V.x - P.x, \Delta y = V.y - P.y$
6: Cyclically shift the arrays to the left by one element to obtain $\Delta x_{\text{next}}, \Delta y_{\text{next}}$
7: Compute the *is left* indicator array
$is\ left = \Delta x \cdot \Delta y_{\text{next}} - \Delta x_{\text{next}} \cdot \Delta y$
8: Masking with flags
$\Delta y = \Delta y \cdot \text{flag1}, is\ left = is\ left \cdot \text{flag2}$
9: Send Δy and *is left*
10: Decrypt intermediate results:
$\Delta y = \text{Dec}(sk, \Delta y), is\ left = \text{Dec}(sk, is\ left)$
11: Masking with the sgn function
$\Delta y = \text{sgn}(\Delta y), is\ left = \text{sgn}(is\ left)$
12: Send Δy and *is left*
13: De-masking with flags
$\Delta y = \Delta y \cdot \text{flag1}, is\ left = is\ left \cdot \text{flag2}$
14: Add the times when edges cross upward and P is left of edges
wn += $\sum(\Delta y \leq 0) \cdot (\Delta y_{\text{next}} > 0) \cdot is\ left$
15: Subtract the times when edges cross downward and P is right of edges
wn += $\sum(\Delta y > 0) \cdot (\Delta y_{\text{next}} \leq 0) \cdot is\ left$
16: **return** wn $\neq 0$

Methodology

Since both methods have flaws that allow the client to trick the server into believing that the given coordinate is inside the geofence, this paper aims to develop a scheme for the server to verify whether the client has faked the data.

First of all, the client should encrypt their longitude and latitude to the server in a homomorphic manner. This is same as the homomorphic geofence method.

Secondarily, server calculates the north, south, east, and west offset from the given homomorphic data, and returns client with message of those four offsets each multiplied by a random float in range $[0.1, 1] \cup [-0.1, -1]$ and a sequence of checksum values. The checksum values are the combinations of those four offsets multiplied by a random float in range $[0.1, 1] \cup [-0.1, -1]$. Since it is randomly applied, the client should not be able to know the permutation of the checksum to cheat on the server.

After that, the client should receive a message from the server containing the four offsets with a sequence of checksum values in a homomorphic manner. After the client decrypts the messages, it should send the server with a message of those four offsets and checksum by 1s and -1s, 1 for value that is positive after decryption, -1 otherwise.

Finally, the server checks if the checksum sign matches, meaning that the sign of the checksum should align with the sign of the random float times the contained offsets. If matched, the server may trust the client and check whether the client is in the geofence or not using the decrypted offsets client returned. Otherwise, the server can determine that the client has faked the data.

Possible Attack: significantly large input

If the client makes a certain input significantly large, the absolute value of the related checksum is also likely to be large, despite being multiplied by a random float. Thus, the client may infer that a checksum value contains one or both of these offset values if it is significantly large. For example, if the client sets the latitude input to 10000, the magnitudes of the north offset (A) and south offset (B) will be 10^5. If the checksum values are AB, BCD, and D, then their magnitudes are likely to be 10^{10}, 10^5, and 1, respectively. Thus, the client could infer that there is both A and B in the first checksum; there is either A or B in the second checksum; and neither A nor B is in the third checksum. This provides additional information about the checksum combinations, increasing the client's chance to fake the offset and checksum while keeping the server unaware.

However, when the client find out that the checksum may contain either A or B, the client has only 0.5 chance of guessing it correctly. If we have 128 additional checksum data, the probability of the client guessing all the checksum combinations correctly would be approximately zero.

Possible Attack: Multiple Requests

Since the client can change the position and send multiple requests to the server, the server should reply the client if the client is inside the geofence or not, the client could approximate the geofence location with the distribution of the offset with a large amount of request.

In order to reduce the accuracy and likelihood for client to approximate the geofence location. When in the implementation phase, the server should restrict the number of requests in a period of time to make the client unable to guess the geofence region out by observing the distribution of the offsets from significantly amount of requests.

Comparisons with existing approaches

Crypto-Geofence

As discussed in the adopted solutions, only the client will know whether the client is in the geofence or not. By such the server could not be able to determine if the client is truthful about the result, as the server can't validate.

For our modification, we entails the client to decrypt more information, the checksum, such that the server can determine whether the client fakes the data or not.

Privacy-Preserving Proof-of-Location using Homomorphic Encryption

Also discussed in the adopted solutions, clients can modify the y offset and the is left values that requires to be returned to the server to make the final result of the wn to be nonzero. Furthermore, the mask applied by the server has a computational feasible amount to brutal force out the correct mask, thus, leads to the result of client can crack out the correct mask and find out the correct geofence location with its size.

For our modification, our checksum solution prevents the client from modifying the offset since the checksum will be incorrect if the offset is modified. Also, our method uses floats to mask the data, hence, the client will not be able to crack out the correct geofence by brute forcing out the correct mask.

NEXUS: Using Geo-fencing Service without Revealing your Location

NEXUS [3] is a method that relies on the trust of a third party to validate if the client is returning the cor-

rect location and protect the server from revealing the geofence. First it generates a public-private key pair to encrypt and decrypt the homomorphic encryption. Secondarily, the mobile node will constantly send position information in homomorphic space to the geo-fencing service. Third, the geo-fenceing service will compute the result if the client is inside the geofence or not without knowing the actual information before decrypting the data. Finally, the geo-fencing server passes the computed information to the Certificate Authority who holds the private key to decrypt the data and send the final result to the server.

NEXUS is highly relying on the third party which in implementation, it is hard to provide a third party of trust and protect the data from an additional transmission. Whereas, our solution only entails both client and server to participate the geofence calculation.

Conclusions and Comments

Conclusion

The paper performs a simple prototype of the geofence with homomorphic encryption. The client and server are both unable to know either the location of the client or the grid of the geofence. The method applied as a checksum protects the server from being faked by the clients data if the client is able to modify the return data. The client encrypts their location in homomporphic encoding allows the server to calculate if the client is inside the geofence or not without the need to decrypt the data and knowing the client's location.

The comparisons showed that our method improves the security from the possible attack from the client to the server to obtain the geofence data, and that it enhances the performance and reduces the risk by eliminating the trust third party to participate the calculation.

Comment

Our project about real world treasure hunting is because of our passion towards games like Pokemon GO. We discover that those games often have to continuously request client for their location information, and may be able to track each client and monitor their life, which is terrible. Hence, we determine to develop a method that the server and the client can both keep their data safe from abuse of each other.

Even masking with random floats, we find it difficult to protect the server from geofence leak. Because the client may be able to approximate the geofence location from the distribution after significant amount of requests. By only constraining the client's request time in a period may only slow down the geofence leak. We think we have to develop a better scheme in the future, that the client cannot approximate the geofence location from the given masked offset and checksum, even with infinite amount of request.

However, the improvement of the method keeps both side safe which can cause positive impact to both the privacy of the client and the valuable information security of the server. We holds positive looking into our project and the potential for developing the prototype of our system into more general uses, such as home quarantine, bombard area warning and so on.

References

[1] `https://github.com/Georeactor/encrypted-geofence`
[2] `https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1445878&dswid=9450`
[3] `https://ieeexplore.ieee.org/abstract/document/8534577`

Footnotes

The complete implementation and additional documentation are available on `https://colab.research.google.com/drive/14gjKktiU0w5zj0S-CNtRlhTb6vwq0jRi?usp=sharing`.