Real World Treasure Hunt Game with Secured Location Info

(prototype of homomorphic encrypted geofence)

B12902131 陳柏宇 B12902037 張聲元

Motivation

A real-world treasure hunt game on electronic devices typically needs user’s GPS location to check whether the user is inside the geofence or not.

And this compromises user’s privacy.

Is there a secure way to **check whether the user is in the geofence**, while not revealing **user’s location** and the **geofence location**?

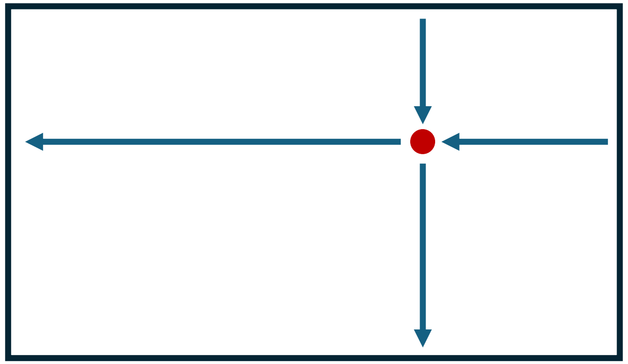
Goal

1. The server can tell whether the client is inside the geofence.
2. If the client don’t know the location of the geofence and isn’t inside the geofence, **there is no way for the client to trick the server to believe that he or she is inside the geofence**.
3. No one besides the client will know about his or her location.
4. No one besides the server will know about the geofence location.
5. No third party.
6. The accuracy should be <1m.

Case Study 1

crypto-geofence

```
north_offset = (north - latitude) * random()
south_offset = (latitude - south) * random()
east_offset = (east - longitude) * random()
west_offset = (longitude - west) * random()
```



DEF-CON Crypto & Privacy Village 2018
https://github.com/Georeactor/encrypted-geofence

Case Study 1

crypto-geofence

```
client: homomorphic encrypts location
              ↓
server: calculate offsets under homomorphic encryption
              ↓
client: decrypt offsets, check positive negative
```

Client can lie to the server!

DEF-CON Crypto & Privacy Village 2018
https://github.com/Georeactor/encrypted-geofence

Case Study 2

NEXUS: Using Geo-fencing Services without revealing your Location

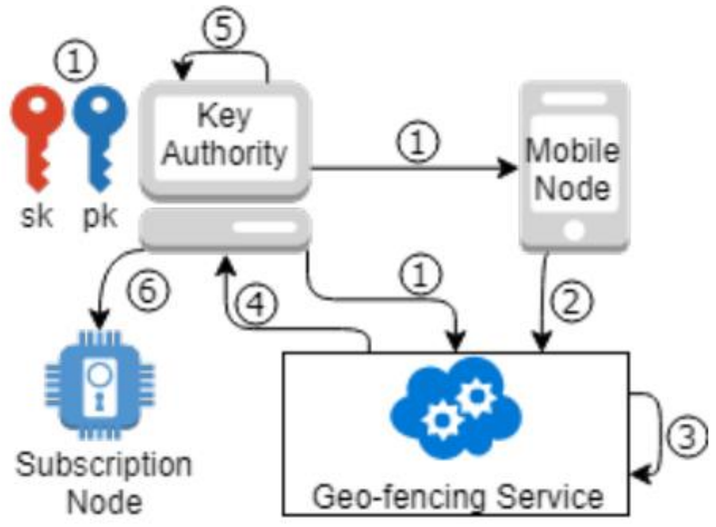


Figure 1. The actors and high level interactions in NEXUS.

requires third party

IEEE Global Internet of Things Summit (GIoTS) 2018
https://ieeexplore.ieee.org/abstract/document/8534577

Case Study 3

Privacy-preserving proof-of-location using homomorphic encryption

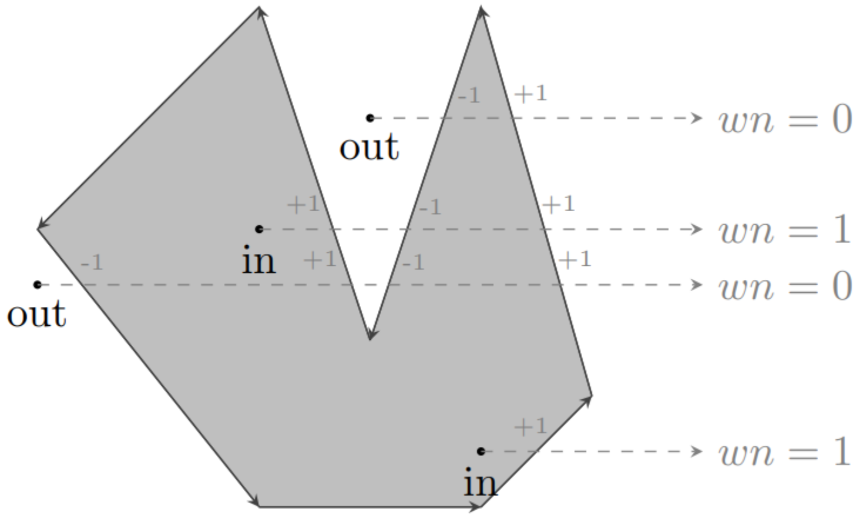


Figure 2.6.: Points outside of polygon have $wn = 0$ and points inside $wn = 1$.

Winding Number Point-in-Polygon Algorithm

is_left, Δy for every edge

↓

point is in the polygon or not

2020

https://www.diva-portal.org/smash/record.jsf?dswid=-8729&pid=diva2%3A1445878

send homomorphic encrypted location to server

Algorithm 4 Vectorised computation with masking (yellow: client; blue: server)

$wn = 0$, point P
$P = \texttt{Enc}(\texttt{pk}, P)$
Send P

Vertices $\{\mathbf{V}_i\}$, flags $flag_1$, $flag_2$.
Compute $\Delta\mathbf{x}$, $\Delta\mathbf{y}$:
$$\Delta\mathbf{x} = \mathbf{V}.\mathbf{x} - P.x, \quad \Delta\mathbf{y} = \mathbf{V}.\mathbf{y} - P.y$$
Cyclically shift the arrays to the left by one element to obtain $\Delta\mathbf{x}_{next}$, $\Delta\mathbf{y}_{next}$
Compute the `is_left` indicator array
$$\texttt{is\_left} = \Delta\mathbf{x} \cdot \Delta\mathbf{y}_{next} - \Delta\mathbf{x}_{next} \cdot \Delta\mathbf{y}$$
Masking with flags
$$\Delta\mathbf{y} = \Delta\mathbf{y} \cdot flag_1, \quad \texttt{is\_left} = \texttt{is\_left} \cdot flag_2$$
Send $\Delta\mathbf{y}$ and `is_left`

Decrypt intermediate results:
$$\Delta\mathbf{y} = \texttt{Dec}(\texttt{sk}, \Delta\mathbf{y}), \quad \texttt{is\_left} = \texttt{Dec}(\texttt{sk}, \texttt{is\_left})$$
Masking with the sgn function
$$\Delta\mathbf{y} = \text{sgn}(\Delta\mathbf{y}), \quad \texttt{is\_left} = \text{sgn}(\texttt{is\_left})$$
Send $\Delta\mathbf{y}$ and `is_left`

De-masking with flags
$$\Delta\mathbf{y} = \Delta\mathbf{y} \cdot flag_1, \quad \texttt{is\_left} = \texttt{is\_left} \cdot flag_2$$
Add the times when edges cross upward and p is left of edges
$$wn \mathrel{+}= \Sigma\Big((\Delta\mathbf{y} <= 0) \cdot (\Delta\mathbf{y}_{next} > 0) \cdot \texttt{is\_left}\Big)$$
Subtract the times when edges cross downward and p is right of edges
$$wn \mathrel{+}= \Sigma\Big((\Delta\mathbf{y} > 0) \cdot (\Delta\mathbf{y}_{next} <= 0) \cdot \texttt{is\_left}\Big)$$
return $wn \neq 0$

Algorithm 4 Vectorised computation with masking (yellow: client; blue: server)

$wn = 0$, point P

$P = \mathtt{Enc}(\mathtt{pk}, P)$

Send P

Vertices $\{\mathbf{V}_i\}$, flags $flag_1$, $flag_2$.

Compute $\Delta\mathbf{x}$, $\Delta\mathbf{y}$:

$$\Delta\mathbf{x} = \mathbf{V}.\mathbf{x} - P.x, \quad \Delta\mathbf{y} = \mathbf{V}.\mathbf{y} - P.y$$

Cyclically shift the arrays to the left by one element to obtain $\Delta\mathbf{x}_{next}$, $\Delta\mathbf{y}_{next}$

Compute the `is_left` indicator array

$$\mathtt{is\_left} = \Delta\mathbf{x} \cdot \Delta\mathbf{y}_{next} - \Delta\mathbf{x}_{next} \cdot \Delta\mathbf{y}$$

Masking with flags

$$\Delta\mathbf{y} = \Delta\mathbf{y} \cdot flag_1, \quad \mathtt{is\_left} = \mathtt{is\_left} \cdot flag_2$$

Send $\Delta\mathbf{y}$ and `is_left`

Dec

Mas

Sen

De-

Add

Sub

return $wn \neq 0$

```python
class Server(Participant):

    def masking(self, cipher_arr):
        """
        Mask the cipher_arr with a sign change
        :param cipher_arr: the cipher_arr to mask
        :return: masked cipher_arr
        """
        num = len(cipher_arr)
        flag = np.random.choice([-1, 1], num)

        for i in range(num):
            self._evaluator.multiply_plain_inplace(cipher_arr[i], self._encoder.encode(flag[i]))
        return flag, cipher_arr
```

only 2^(2edges) possible combinations, client can easily brute force out the geofence location

Algorithm 4 Vectorised computation with masking (yellow: client; blue: server)

$wn = 0$, point P
$P = \texttt{Enc}(\texttt{pk}, P)$
Send P

Vertices $\{\mathbf{V}_i\}$, flags $flag_1$, $flag_2$.
Compute $\Delta\mathbf{x}$, $\Delta\mathbf{y}$:
$$\Delta\mathbf{x} = \mathbf{V}.\mathbf{x} - P.x, \quad \Delta\mathbf{y} = \mathbf{V}.\mathbf{y} - P.y$$
Cyclically shift the arrays to the left by one element to obtain $\Delta\mathbf{x}_{next}$, $\Delta\mathbf{y}_{next}$
Compute the `is_left` indicator array
$$\texttt{is\_left} = \Delta\mathbf{x} \cdot \Delta\mathbf{y}_{next} - \Delta\mathbf{x}_{next} \cdot \Delta\mathbf{y}$$
Masking with flags
$$\Delta\mathbf{y} = \Delta\mathbf{y} \cdot flag_1, \quad \texttt{is\_left} = \texttt{is\_left} \cdot flag_2$$
Send $\Delta\mathbf{y}$ and `is_left`

Client can randomly modify is_left and Δy. And wn will become ≠0. I have tested and proved this.

Decrypt intermediate results:
$$\Delta\mathbf{y} = \texttt{Dec}(\texttt{sk}, \Delta\mathbf{y}), \quad \texttt{is\_left} = \texttt{Dec}(\texttt{sk}, \texttt{is\_left})$$
Masking with the sgn function
$$\Delta\mathbf{y} = \mathrm{sgn}(\Delta\mathbf{y}), \quad \texttt{is\_left} = \mathrm{sgn}(\texttt{is\_left})$$
Send $\Delta\mathbf{y}$ and `is_left`

De-masking with flags
$$\Delta\mathbf{y} = \Delta\mathbf{y} \cdot flag_1, \quad \texttt{is\_left} = \texttt{is\_left} \cdot flag_2$$
Add the times when edges cross upward and p is left of edges
$$wn \mathrel{+}= \Sigma\Big((\Delta\mathbf{y} <= 0) \cdot (\Delta\mathbf{y}_{next} > 0) \cdot \texttt{is\_left}\Big)$$
Subtract the times when edges cross downward and p is right of edges
$$wn \mathrel{+}= \Sigma\Big((\Delta\mathbf{y} > 0) \cdot (\Delta\mathbf{y}_{next} <= 0) \cdot \texttt{is\_left}\Big)$$
return $wn \neq 0$

If wn≠0, server will believe client is in the geofence.

Problem

1. Client can easily trick server into believing client is inside the geofence.
2. Rely on trust in a third party.

Methodology

Develop a scheme to check whether client fakes or not.

I’m not familiar with researching, homomorphic encryption, security level and accuracy.

So I’m afraid that this may be a little bit informal.

What I’m trying to do is improving the methodology and make it usable in a treasure hunting game.

Methodology

Client homomorphic encrypted latitude, longitude and send to server

// ABCD are all homomorphic encrypted
A = (north - latitude)
B = (latitude - south)
C = (east - longitude)
D = (longitude - west)

Ask client to decrypt obfuscated A, B, C, D and random obfuscated combination of [A, B, C, D]
// r(x) = x * random_floats_between(1, -1), r(x) is still homomorphic encrypted
message = [r(A), r(B), r(C), r(D)] and [r(A*C), r(1), r(A*B*D), r(C), r(B*D), r(A*B*C*D), r(B*D) …….]

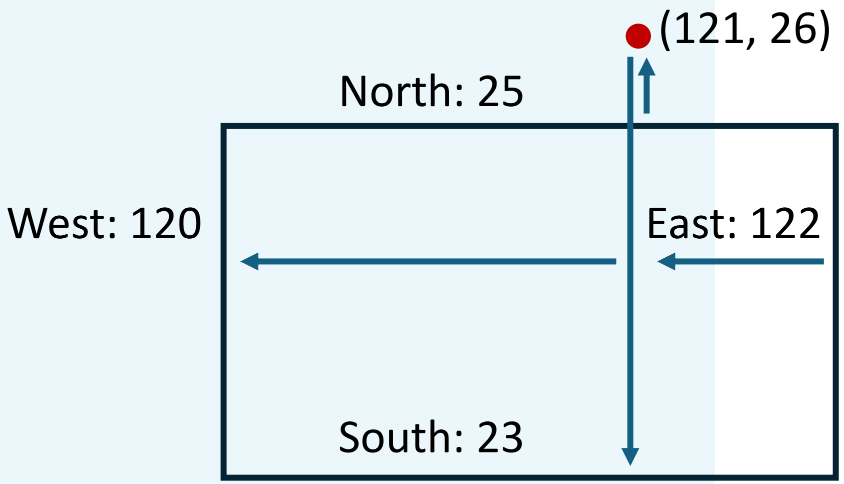
Client decrypt and sign all of the message and send to server
decrypted_and_signed_message = [1, -1, -1, 1] and [1, -1, -1, 1, -1, -1, 1, 1, -1, -1, -1……]

Server check if all of the sign matches, if not, client fakes the decrypted_and_signed_message
And if A>0, B>0, C>0, D>0, the client is inside the geofence

Methodology(example)

Client send encrypted(121, 26) to server

// are all homomorphic encrypted, server don’t know the values and signs
A = 25 – 26 = -1 < 0
B = 26 – 23 = 1 > 0
C = 122 – 121 = 1 > 0
D = 121 – 120 = 1 > 0
// server will save the random number and random combination
r(A) = 0.8 * -1 = -0.8
r(B) = 0.2 * 1 = 0.2
r(C) = -0.9 * 1 = -0.9
r(D) = -0.6 * 1 = -0.6
r(A*C) = -0.5 * -1 * 1 = 0.5
send [-0.8, 0.2, -0.9, -0.6] and [0.5] to client



decrypted_and_signed_message = [-1, 1, -1, -1] and [1]

A: -1*0.8<0 B: 1*0.2>0 C: -1*-0.9>0 D: -1*-0.6>0
r(A*C): negative(A) * positive(C) * negative(-0.5) > 0 matched, assume client didn’t fake the data

Methodology(possible attack1)

Client send encrypted(121, 26) to server

// are all homomorphic encrypted, server don't know the values and signs

10^3 A = 25 – 26 = -1 < 0

10^3 B = 26 – 23 = 1 > 0

C = 122 – 121 = 1 > 0

D = 121 – 120 = 1 > 0

// server will save the random number and random combination

10^3 r(A) = 0.8 * -1 = -0.8

10^3 r(B) = 0.2 * 1 = 0.2

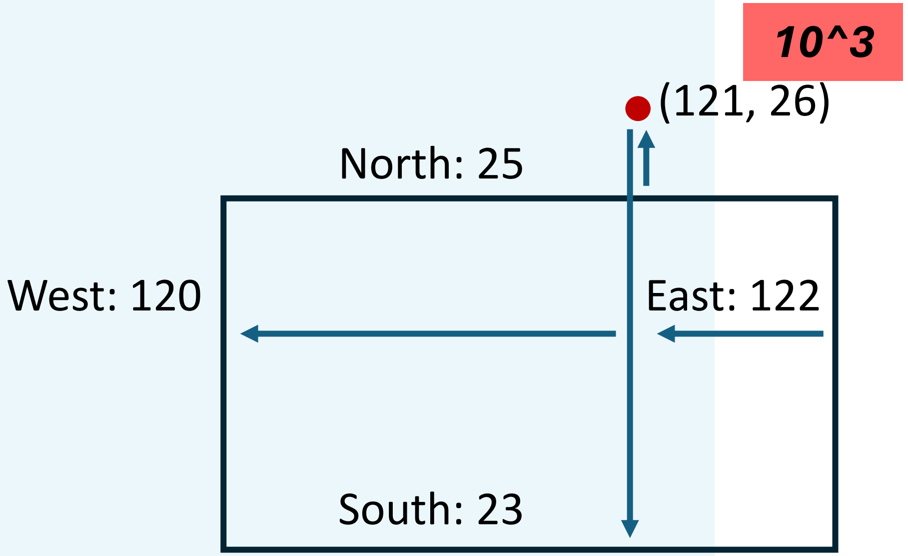
r(C) = -0.9 * 1 = -0.9

r(D) = -0.6 * 1 = -0.6

r(A*C) = -0.5 * -1 * 1 = 0.5

***10^3**, very possibly contains either A or B*

send [-0.8, 0.2, -0.9, -0.6] and [0.5] to client



decrypted_and_signed_message = [-1, 1, -1, -1] and [1]

A: -1*0.8<0 B: 1*0.2>0 C: -1*-0.9>0 D: -1*-0.6>0

r(A*C): negative(A) * positive(C) * negative(-0.5) > 0 matched, assume client didn't fake the data

Methodology(security level)

With the previous mentioned attack method.

Mallory can know each additional data is any of these:

1. either a or b
2. a and b
3. either c or d
4. c and d
5. (either a or b) and (either c or d)
6. …

When “either” occurs (0.75), Mallory has to guess(0.5).

If we have 128 additional data, the chance of getting right is: $\frac{1}{2^{128*0.75}}$

Methodology(accuracy)

```
ckks_params = {
    'scheme': 'CKKS', // avoid client factor and calculate the geofence
    'n': 2**14,
    'scale': 2**50, // x_fix = round(x_float * scale)
    'qi_sizes': [60, 50, 50, 50, 60] // needs to multiply 3 times, so 3 intermediates
}
// all the latitudes and longitudes will times 10^2 to increase accuracy
```

$2^{50} \cong 10^{15}$

The random number is in range [1, 0.1] and [-0.1, -1]

$10^{15}/10 = (10^{3.5})^4$

$10^{-(3.5+2)}$ *in latitude and longitude around* $(121.5, 25)$ *is about* $\mathbf{0.47m}$.

Testing on the code can achieve about $\mathbf{0.015m}$ accuracy.

Methodology(possible attack2)

The random number is in range [1, 0.1] and [-0.1, -1]

So Mallory can learn about the geofence location after requesting enough times. (With the distribution)

Methodology

https://colab.research.google.com/drive/14gjKktiU0w5zj0S-CNtRlhTb6vwqOjRi?usp=sharing

Applications

1. Bombing Area
2. Covid 19
3. …