

# Cryptography - project

Marta Gawron, Krzysztof Strzała, Marta Hendler

## Aim of the project

The aim of the project is to create an application that allows two of its users to tell if they are in close proximity to each other. The implemented functionality should return 1 (or other positive message) when users are next to each other, and 0 otherwise.

## Model

A protocol based on the socialist millionaires problem (also known as the Yao millionaires problem) will be used to achieve the project objective. The original problem concerns the ability of two millionaires to determine whether their wealth is the same, but without disclosing its size<sup>1</sup>. In general, this problem can be reduced to the issue of verifying that both parties involved in the communication process possess the same secret information.

The secret remains undisclosed throughout the communication - only the equality of information held by both parties is verified. This makes it impossible for a potential *man in the middle* to influence the implementation of the protocol in any way other than leading to a failure (verification with a negative result). This protocol has been adapted in the Off The Record (OTR) protocol and is used, for example, in online chat rooms to verify the identity of users.

## Protocol

Assumptions:

- Alice's and Bob's secrets are respectively  $x, y$ ,
- protocol parameters: cyclic group of order  $p$ , where  $p$  - a big prime number, group generator  $g$ .

1. Alice chooses two random exponents  $a_2, a_3$  and sends to Bob:

$$g_{2a} = g^{a_2}$$

$$g_{3a} = g^{a_3}$$

2. Bob chooses two random exponents  $b_2, b_3$  and computes:

$$g_{2b} = g^{b_2}, g_{3b} = g^{b_3}$$

$$g_2 = g_{2a}^{b_2}, g_3 = g_{3a}^{b_3}$$

He then chooses a random exponent  $r$  and computes:

$$P_b = g_3^r, Q_b = g^r \cdot g_2^y$$

Bob sends to Alice:  $g_{2b}, g_{3b}, P_b, Q_b$

3. Alice receives these values and computes:

$$g_2 = g_{2b}^{a_2}, g_3 = g_{3b}^{a_3}$$

She then chooses a random exponent  $s$  and computes:

$$P_a = g_3^s, Q_a = g^s \cdot g_2^x$$

$$R_a = \left( \frac{Q_a}{Q_b} \right)^{a_3}$$

Alice sends to Bob:  $P_a, Q_a, R_a$

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Socialist\\_millionaire\\_problem](https://en.wikipedia.org/wiki/Socialist_millionaire_problem)

4. Bob receives the above values and computes:

$$R_b = \left( \frac{Q_a}{Q_b} \right)^{b_3}$$

$$R_{ab} = R_a^{b_3}$$

Then he checks if the equality:  $R_{ab} == \frac{P_a}{P_b}$  occurs and sends back to Alice the value of  $R_b$

5. Alice receives this value and computes:

$$R_{ab} = R_b^{a_3}$$

Then she also checks whether equality holds  $R_{ab} == \frac{P_a}{P_b}$

In a valid run of the protocol, the value of  $R_{ab}$  is  $\frac{P_a}{P_b} \cdot \left( g_2^{a_3 b_3} \right)^{(x-y)}$ . This means that the equality will only occur on both sides in the case  $x == y$ . On the other hand, if  $x \neq y$  happens, no information is revealed because  $g_2^{a_3 b_3}$  is a random number unknown to either party<sup>2</sup>.

## Algorithm steps

---

**Algorithm 1:** Neighborhood check

---

**Result:** 1 - users are nearby; 0 - otherwise

```
1 Set group parameters  $p$  and  $g$ ;  
2 Select random exponents:  $a_2, a_3, s$  for user 1 and  $b_2, b_3, r$  for user 2;  
3 Get both parties secrets:  $x$  for user 1 and  $y$  for user 2;  
4 Perform socialist millionaire protocol on  $x$  and  $y$ ;  
5 if  $x == y$  then  
6   | return 1  
7 else  
8   | return 0  
9 end
```

---

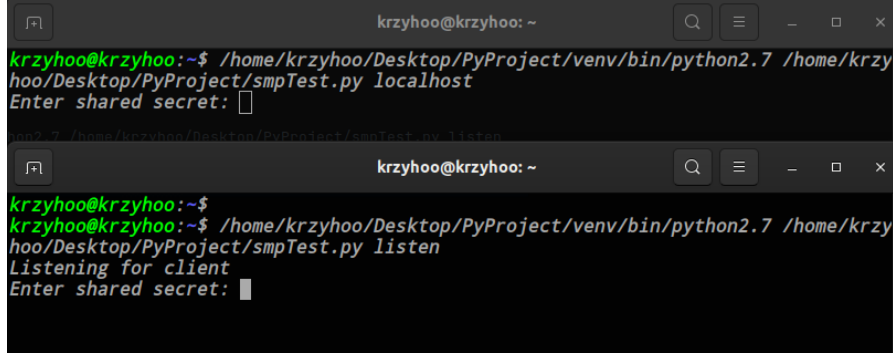
---

<sup>2</sup><https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html>

## Implementation

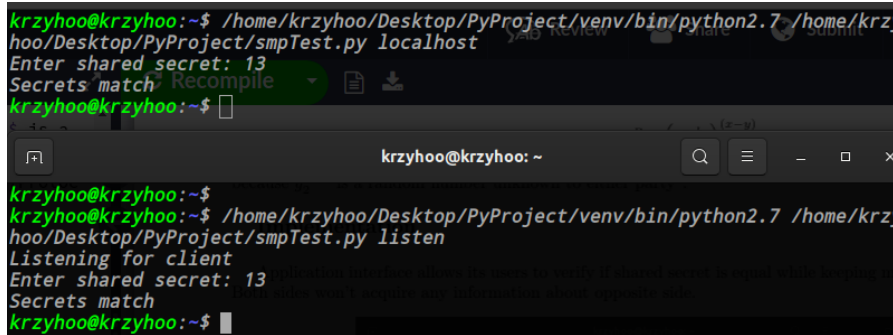
Application interface allows its users to verify if their secrets are equal while keeping maximum privacy level. Both sides won't acquire any information about opposite side. Usage is very simple. A secret of generic type is put, and if another user will put equal secret, application will return true.

Whole implementation of protocol can be found in file `smp.py`. `smpTest.py` is a file for testing purpose.



```
krzyhoo@krzyhoo: ~  
krzyhoo@krzyhoo:~$ /home/krzyhoo/Desktop/PyProject/venv/bin/python2.7 /home/krzyhoo/Desktop/PyProject/smpTest.py localhost  
Enter shared secret:   
krzyhoo@krzyhoo:~$  
krzyhoo@krzyhoo:~$ /home/krzyhoo/Desktop/PyProject/venv/bin/python2.7 /home/krzyhoo/Desktop/PyProject/smpTest.py listen  
Listening for client  
Enter shared secret: 
```

Figure 1: Usage demo: split terminal for entering secrets from both parties.



```
krzyhoo@krzyhoo:~$ /home/krzyhoo/Desktop/PyProject/venv/bin/python2.7 /home/krzyhoo/Desktop/PyProject/smpTest.py localhost  
Enter shared secret: 13  
Secrets match  
krzyhoo@krzyhoo:~$  
krzyhoo@krzyhoo:~$ /home/krzyhoo/Desktop/PyProject/venv/bin/python2.7 /home/krzyhoo/Desktop/PyProject/smpTest.py listen  
Listening for client  
Enter shared secret: 13  
Secrets match  
krzyhoo@krzyhoo:~$
```

Figure 2: Usage demo: entering equal values results in success message.

Api can be linked as library with other application, and might be used in many different cases, such as verifying distance between two users. The protocol used proves to be useful in this situation as it allows check the location of a user by its relation with the location of another user, without revealing potentially sensitive geolocation data. We suggest app that fetch users longitude and latitude from GPS, and map it to values in range of 0 to  $\frac{7200}{(\frac{k}{3})^2}$  by following equation:

$$f(lat, long) = \frac{360}{k} \cdot \frac{lat + 90}{k} + \frac{long + 180}{k} \quad (1)$$

Therefore, the secret input for each user is the not the exact geolocation data but the number of square, in which the user is. This way, the equality of the secret means that both users are located within the same space limited by the squares.

For example, for  $k = 3$  we divide map into 7200 squares, each with size  $k \times k$ .  $k$  parameter should be chosen based on required distance accuracy level. Square  $3 \times 3$  has approximately size of  $70800 km^2$ .

Just for test purpose, we've created android app that allows us to send mapped by equation 1 longitude and latitude to server in python and verify, if protocol is working properly, with mocked data on server. Android app shows users if they are close to each other. In the pictures below are presented messages which are displayed.



Figure 3: Positive verification: equal secrets mean that the users are nearby.

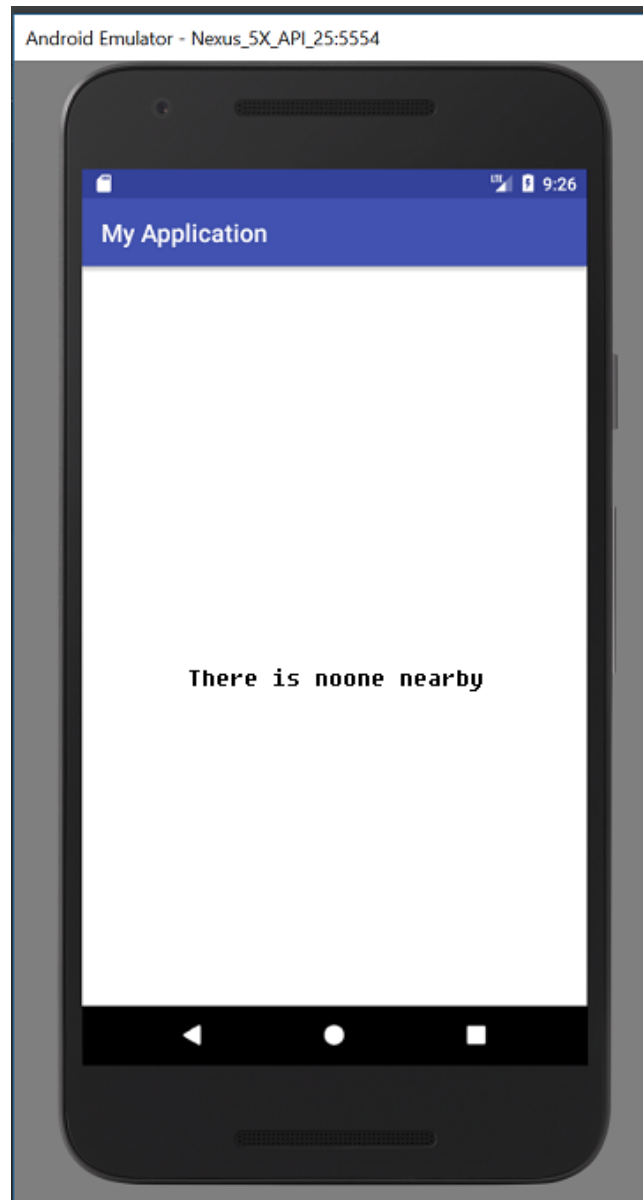


Figure 4: Negative verification: different secrets mean that the users are not close to each other.

## Project structure

- smp.py contains implementation of the protocol,
- smpTest.py is a file for testing if shared secret matches,
- server.py returns mocked verification,
- android app written in kotlin/java, for visualisation purposes, it shows the user if there is a person nearby.