ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო
უნივერსიტეტი

ქეთევან ქუთათელაძე

შეხვედრის ადგილის საიდუმლოდ ორგანიზების მობილური აპლიკაცია

კომპიუტერული მეცნიერება

ნაშრომი შესრულებულია კომპიუტერული მეცნიერების მაგისტრის აკადემიური ხარისხის მოსაპოვებლად

ხელმძღვანელი: მიხეილ თუთბერიძე, ფიზ.-მათ. მეცნ. კანდიდატი

თბილისი

2017

ანოტაცია

წინამდებარე ნაშრომის ფარგლებში შემუშავებულია მობილური აპლიკაცია, რომელიც ორ მომხმარებელს საშუალებას აძლევს, თვალყური ადევნონ ერთმანეთის გადაადგილებას რუკაზე, რათა გაუიოლდეთ ერთმანეთთან შეხვედრა. ამასთან, ინფორმაციის გადაგზავნა წარმოებს დაშიფრულად, მესამე პირებისგან დაცულად, რომელიც უზრუნველყოფილია კრიპტოგრაფიული მეთოდის გამოყენებით. პროგრამული უზრუნველყოფა შემუშავებულია, როგორც კლიენტ-სერვერული აპლიკაცია და ამასთან, კლიენტის აპლიკაცია ორიენტირებულია მობილურ მოწყობილობაზე. პროგრამული უზრუნველყოფა შეიძლება გამოყენებულ იქნას სოციალური ქსელის - Facebook მომხმარებლის მიერ მნიშვნელოვანი შეხვედრების დასაგეგმად.

Annotation

Ketevan Kutateladze

Mobile Application for Secretly Organizing Meeting Place

In the scope of the present work the mobile application, which enables two users to keep track of each other in order to meet easily, is developed. In addition, the information transmission is encrypted using cryptographic methods, to protect it from third parties. Software is developed as client-server application and client application is oriented to mobile device. The software can be used by social network – Facebook's user to plan important meetings.

სარჩევი

შესავალი

პირადი მონაცემების პრივატულობა 21-ე საუკუნის ერთ-ერთ ყველაზე პრიორიტეტულ ამოცანად იქცა. ადამიანები ცდილობენ თავიანთი ცხოვრების დეტალები არ გახადონ საჯარო და მხოლოდ ახლობლების ვიწრო წრეს გაუზიარონ ესა თუ ის ინფორმაცია. მეორე მხრივ, საინფორმაციო ტექნოლოგიების განვითარებამ გარკვეულწილად გამჭვირვალე გახადა ადამიანების ცხოვრება და მესამე პირებს (მათ შორის სახელმწიფოსაც) მისცა საშუალებ,ა უფრო მეტი იცოდეს ადამიანის ცხოვრებაზე, ვიდრე თავად ეს ადამიანი ისურვებდა ამას.

მობილური კავშირი, სოციალური ქსელები, ელექტრონული ფოსტა, ყველა ადამიანის პირად ინფორმაციას (მონაცემებს) ინახავს და მიმოცვლის. მიუხედავად იმისა, რომ კომუნიკაციის ყველა ამ სახეობაში უსაფრთხოების საკმაოდ დიდი ზომებია მიღებული, მაინც არსებობს საშიშროება, რომ ადამიანის პირადი ინფორმაცია ცნობილი გახდება თუნდაც ამ სისტემების მომსახურე ტექნიკური პერსონალისათვის და ასევე პირებისთვის, რომლებიც არამართლზომიერი გზით მოიპოვებენ წვდომას ამ საკომუნიკაციო სისტემებზე. ამის გამო დგება საკითხი, რომ მოხდეს ამ ინფორმაციის იმგვარად დაშიფრვა, რომ მხოლოდ იმ ორ ადამიანს შეეძლოს მისი ნახვა, რომლებისთვისაც იგი არის განკუთვნილი.

ორ ადამიანს შორის კომუნიკაცია შეიძლება იყოს სხვადასხვა სახის: შეიძლება იგზავნებოდეს უბრალოდ ტექსტები, შეიძლება იგზავნებოდეს ხმოვანი ინფორმაცია, სურათები, ვიდეო. ჩვენს ნაშრომში შევეხებით ინფორმაციის სპეციფიურ სახეს - გეოლოკაციას, რომელიც ასევე არის ადამიანის პრივატული ინფორმაცია და საჭიროებს მესამე პირებისგან დაცვას. ერთმა ადამიანმა მეორე ადამიანს შეიძლება გაუგზავნოს გეოლოკაცია, იმისათვის რომ უფრო ადვილად მოხერხდეს მათი შეხვედრა, მაგრამ ეს არ ნიშნავს იმას, რომ სხვებმაც უნდა იცოდნენ მათი შეხვედრის ადგილის და ან საერთოდ შეხვედრის შესახებ. სწორედ ყოველივე ამან გადაგვაწვეტინა ამ ნაშრომის შემუშავება. წინამდებარე ნაშრომის ფარგლებში შემუშავებულია მობილური აპლიკაცია, რომელიც ორ მომხმარებელს საშუალებას აძლევს, თვალყური ადევნონ ერთმანეთის გადაადგილებას რუკაზე, რათა გაუიოლდეთ ერთმანეთთან შეხვედრა. ამასთან,

ინფორმაციის გადაგზავნა წარმოებს დაშიფრულად და მისი ნახვა მესამე პირის მიერ გამორიცხულია იმ შემთხვევაშიც კი, როდესაც ეს მესამე პირი წარმოადგენს სერვერის ადმინისტრატორს.

ამოცანის დასმა

როგორც შესავალში იყო აღნიშნული, ჩვენი ამოცანაა, შევუწყოთ ხელი ორ ადამიანს, რათა ისინი ადვილად შეხვდნენ ერთმანეთს, რუკაზე ერთმანეთის ადგილმდებარეობისთვის თვალის მიდევნების გზით. ამასთან, ინფორმაცია მათი ადგილმდებარეობის შესახებ დაცული უნდა იყოს მესამე პირებისგან კრიპტოგრაფიული მეთოდების გამოყენებით. მობილურ აპლიკაციებს შორის კავშირი არ იქნება პირდაპირი, არამედ კავშირი განხორციელდება მობილური ინტერნეტის მეშვეობით. ინფორმაციის გადასაგზავნად გამოყენებული იქნება google-ის შესაბამისი სერვისი - Firebase. მომხმარებელთა მონაცემები სერვერზე არ უნდა ინახებოდეს გარდა მოწყობილობის იდენტიფიკატორებისა, რომელიც აუცილებელია Firebase-თვის მოწყობილობის შეტყობინების გასაგზავნად. ინფორმაციის დაშიფრვა მოხდება სიმეტრიული ალგორითმით, ოღონდ სიმეტრიული ალგორითმის პაროლის გაცვლა უნდა მოხდეს დიფი-ჰელმანის ალგორითმის გამოყენებით. მომხმარებლის მიერ ადგილმდებარეობის ყოველი ცვლილება უნდა ფიქსირდებოდეს მობილური აპლიკაციის მიერ და ეგზავნებოდეს Firebase-ის სერვერს, რომელიც, თავის მხრივ, გადაუგზავნის ამ ინფორმაციას მეორე მომხმარებლის მობილურ აპლიკაციას. მობილური აპლიკაცია ორივე მომხმარებლის ადგილმდებარეობას უნდა აჩვენებდეს რუკაზე. მობილურ აპლიკაციაში აუთენტიფიკაცია უნდა ხორციელდებოდეს სოციალური ქსელის Facebook-ის მეშვეობით. სისტემაში შესვლის შემდეგ მომხმარებელს მისდის ინფორმაცია მისი Facebook მეგობრების შესახებ. მომხმარებელი უგზავნის მოთხოვნას თავის რომელიმე მეგობარს და თუ ის მეგობარი დათანხმება, ამ მოთხოვნას, მაშინ ისინი ხედავენ ერთმანეთის ადგილდებარეობას რუკაზე და თვალს ადევნებენ ერთმანეთს თავიანთ მობილურ აპლიკაციაში.

ზოგიერთი ფაქტი Firebase Realtime Database-ის შესახებ

Firebase Realtime Database ინახავს და ასინქრონებს მონაცემს NoSql cloud database-თან ერთად. მონაცემი სინქრონიზებულია ყველა კლიენტთან რეალურ დროში და რჩება ხელმისაწვდომი, როცა აპლიკაცია გადის offline-ზე (როცა გამოვდივართ app-დან).

მონაცემები ინახება JSON სახით და სინქრონიზებულია რეალურ დროში ყველა დაკავშირებულ კლიენტთან. როცა ვქმნით cross-platform აპლიკაციას, ყველა კლიენტს ვუზიარებთ Realtime Database instance-ს და ავტომატურად ვიღებთ უახლოესი მონაცემების განახლებებს.

Firebase Realtime Database საშუალებას გვაძლევს, შევქმნათ მდიდარი, ერთობლივი აპლიკაციები, რომლებიც იძლევა მონაცემთა ბაზის უსაფრთხო ხელმისაწვდომობის საშუალებას პირდაპირ კლიენტის კოდის მხრიდან. მონაცემები ჩნდება ლოკალურად, და თუნდაც offline დროს, realtime ივენთები აგრძელებს კავშირს.

როცა მოწყობილობა ხელახლა უკავშირდება აპლიკაციას, Realtime Database ასინქრონებს ლოკალური მონაცემების ცვლილებას დისტანციურ განახლებებთან ერთად, რომელიც კლიენტის offline რეჟიმში დაფიქსირდა, ავტომატურად აერთიანებს ნებისმიერ კონფლიქტს.

Realtime Database არის NoSQL ბაზა და როგორც ასეთი, აქვს განსხვავებული ოპტიმიზაციები და ფუნქციები რელაციურ ბაზასთან შედარებით. The Realtime Database API განკუთვნილია მხოლოდ ოპერაციებისთვის, რომლებიც შეიძლება შესრულდეს სწრაფად. ეს საშუალებას გვაძლევს შევქმნათ realtime გამოცდილება, რომელსაც შეეძლება ემსახუროს მილიონობით მომხმარებელს.

Firebase-ს ხეები პროექტში

ამ სექციის ქვეშ ინახება მომხმარებლების (user-ების) ცხრილი ველებით: მეილი, პროვაიდერი (რითიც გავიარეთ ავტორიზაცია), დარეგისტრირების თარიღი და მომხმარებლის უნიკალური იდენტიფიკატორი (Firebase-ის იდენტიფიკატორი).

Search by email address, phone number, or user UID

Identifier	Providers	Created	Signed In	User UID
shotaioramashvili@gmail.com		May 27, 2017	Jul 1, 2017	8PeQbrHv68MNLg9tvFjfvswVjkS2
itevzo@gmail.com		May 29, 2017	Jun 1, 2017	92FLSUlGxyfGXlfqkenlpLsljhD2
melikishvilimari@yahoo.com		May 28, 2017	Jun 6, 2017	LC8D7XUUHBgbXHNUE65EEv3rU1...
tamuna.kapanadzee@gmail....		Jul 1, 2017	Jul 1, 2017	h2mkmc3mPAhvoHfXezuu9H4CR...

Rows per page: 50 1-4 of 4

Database

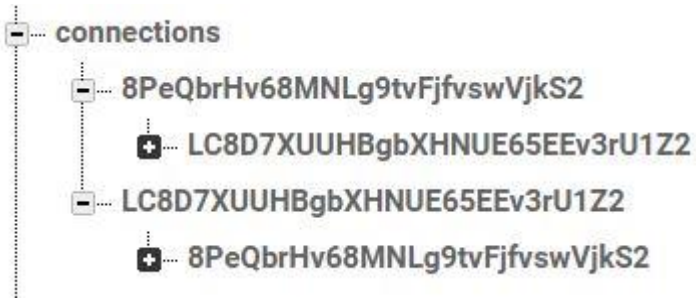
https://locationsharing-4d329.firebaseio.com/

locationsharing-4d329
- connections
- fbusers
- locations
- users

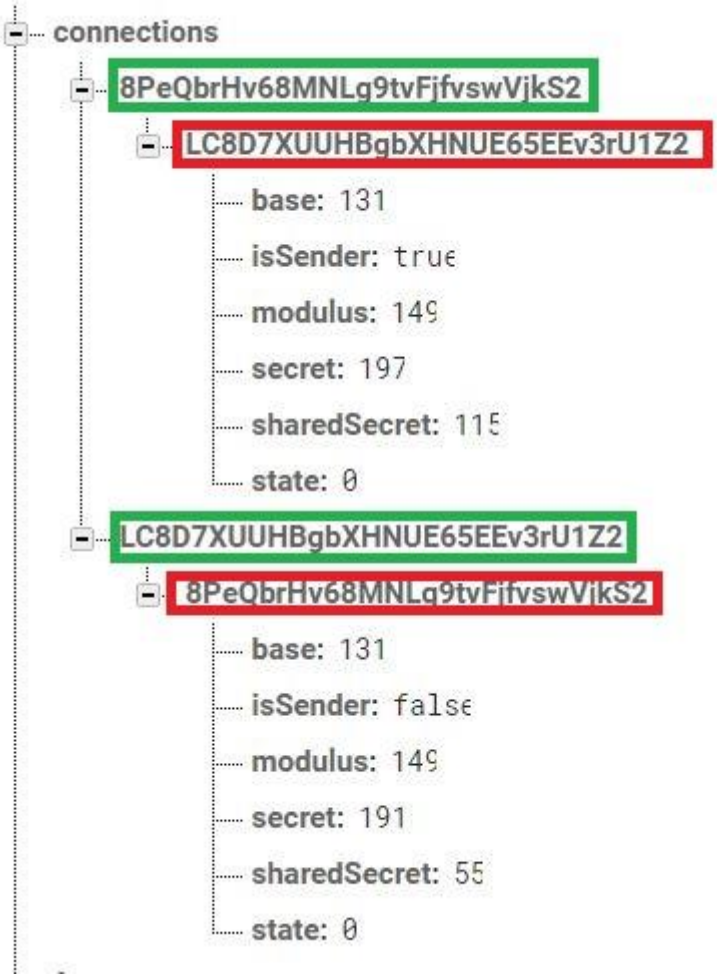
1. თუ ჩავშლით connections ხეს მივიღებთ ასეთ სურათს:

- connections
  - 8PeQbrHv68MNLg9tvFjfvswVjkS2
  - LC8D7XUUHBgbXHNUE65EEv3rU1Z2

როცა რომელიმე მომხმარებლის მხრიდან ხდება მოთხოვნის გაგზავნა, თუ არ არსებობს, connections ხე იქმნება. ამ შემთხვევაში აპლიკაციაში დაკავშირებულია ორი მომხმარებელი. ჩავშალოთ ხეები (მომხმარებლები), რომლებიც ერთმანეთს ხვდებიან. მაგალითად 8PeQbrHv68MNLg9tvFjfvswVjkS2 და LC8D7XUUHBgbXHNUE65EEv3rU1Z2.



ამ შემთხვევაში 8PeQbrHv68MNLg9tvFjfvswVjkS2 მომხმარებელი დაკავშირებულია მხოლოდ ერთ ადამიანთან (იმ მომხმარებელთან, რომელიც უზიარებს მდებარეობას). იგივე მდგომარეობაა h2mkmc3mPAhvoHfXezuu9H4CR1F2 შემთხვევაში. თუ კიდევ ჩავშლით ხეს, ასეთ სურათს მივიღებთ:



ალგორითმისთვის წინასწარ შერჩეული 2 პარამეტრია : base და modulus, რომლებიც ცხადია ორივე მომხმარებლისთვის ერთნაირია. isSender პარამეტრი არის bool ტიპის ცვლადი, რომელიც ხდება true, თუ მომხმარებელი არის გამგზავნი, ანუ

ინიციატორი, წინააღმდეგ შემთხვევაში isSender არის false. State არის რიცხვი, რომელიც აღნიშნავს, მომხმარებელი მოთხოვნას ეთანხმება, აგზავნის თუ უარს ამბობს.

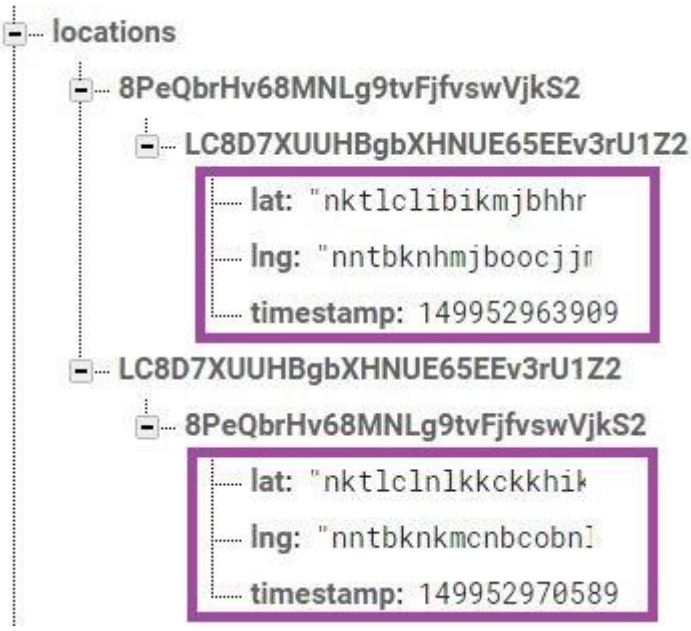
მნიშვნელოვანი კომპონენტებია modulus, sharedSecret და secret. ამ სამ პარამეტრზე დაყრდნობით ხდება მთავარი secret key-ს დაგენერირება, რომელიც ორივესთვის ერთი და იგივეა და რომელიც ცხადად არსად არ ჩანს.

2. ჩავშალოთ fbusers ხე:

fbusers არის user-ების ფესვი (root), რომელიც ინახავს ასეთ ასახვას fb id: Firebase id. ანუ თითოეული მომხმარებლის Facebook-ის id-ს შეუსაბამებს Firebase-ის id-ს. ეს იმისთვის გვჭირდება, რომ როცა აპლიკაციაში მეორე tab-ზე Friend-ზე გადავდივართ, პირველ რიგში ის მეგობრები მოდის, რომელთაც ამ აპლიკაციაში რაიმე აქვთ გაკეთებული. მეგობრების წამოღება ცალკე სერვისია, ის გვიბრუნებს ამ id-ბით რაღაც სიას, რომ შევძლოთ და გარდავქმნათ Facebook-ის id Firebase-ის id-ით. ეს იმიტომ გვჭირდება, რომ ყველა დანარჩენ ადგილებში, სადაც ხდება location გაზიარება, ყველგან გამოიყენება Firebase-ის id.

3. ჩავშალოთ locations ხე:

მთავარი secret key-ს დაგენერირების შემდეგ, მასზე დაყრდნობით location-ის გაცვლის დროს ვაკეთებთ დაშიფრვას. მშობელში ჩალაგებულია შვილები, ანუ საზიარო ადამიანები, რომლებიც ამ მომხმარებელთან რაღაცებს აზიარებს.



მდებარეობისთვის საჭირო პარამეტრები: lat, lng და timestamp, სადაც გაუგებარი ტექსტები ზიარდება, რომელიც ზუსტად იმ key-ს გამოყენებით არის დაშიფრული, რომელიც გავცვალეთ ალგორითმის მიხედვით. timestamp არის განახლების დრო, რომელიც ყოველ 5 წამში იცვლება. locations-ის ხეში მდებარეობის ცვლილების დროს ყვითლდება იდენტიფიკატორები. აქედან ინფორმაცია იგზავნება push-ის პრინციპით. ეს კარგი საშუალებაა იმისთვის, რომ ფიზიკურად დავაკვირდეთ ნებისმიერ ცვლილებას.

4. ჩავშალოთ users ხე:

ეს არის მომხმარებლების ხე, აქ ინახება თითოეული მომხმარებლის უნიკალური იდენტიფიკატორი, რომელსაც Firebase ანიჭებს და თუ ჩავშლით მას, ვნახავთ ამ id-ზე მიბმული მომხმარებლის დამატებით ინფორმაციას, რომელიც მოდის Facebook-დან: firstname (სახელი Facebook-ზე), id (Facebook-ის უნიკალური იდენტიფიკატორი), lastname (გვარი Facebook-ზე) და picture (Facebook-დან წამოღებული სურათი).


users
8PeQbrHv68MNLg9tvFjfvswVjkS2
firstname: "Shota'
id: "1325710797466252
lastname: "Ioramashvili
picture: "https://fb-s-b-a.akamaihd.net/h-ak-fbx/v/t1.0-1
92FLSUlGxyfGXlfqkenIpLsljhD2
LC8D7XUUHBgbXHNUE65EEv3rU1Z2
firstname: "Mari'
id: "150025431002560
lastname: "Melikishvili
picture: "https://fb-s-d-a.akamaihd.net/h-ak-fbx/v/t1.0-1
ZkbXPkVh0JeiOUIiDmSzIPKUAux1
h2mkmc3mPAhvoHfXezuu9H4CR1F2

დიფი-ჰელმანის ალგორითმი

საჭიროდ მიგვაჩნია, რომ დეტალურად აღვწეროთ დიფი-ჰელმანის ალგორითმი, რისთვისაც ვისარგებლებთ [4]-ით.

<u>**გასაღებების შექმნა უშუალოდ ღია არხში.**</u> კერკჰოფსის პრინციპი გვასწავლის, რომ ნებისმიერი კრიპტოალგორითმის გამძლეობა უნდა ეფუძნებოდეს გასაღებს, რადგანაც ესაა ერთადერთი პარამეტრი, რომელიც არ იცის მოწინააღმდეგემ, ალგორითმი მისთვის ცნობილია. სიმეტრიულ კრიპტოგრაფიაში გასაღებების გასაცვლელად აუცილებელია დახურული არხი. კლასიკური კრიპტოგრაფიის გამოყენების პერიოდში გასაღებების გაცვლა ხდებოდა გარკვეული დროის შემდეგ აუცილებლად დახურული არხის საშუალებით. მას შემდეგ, რაც საინფორმაციო არხი გარდა ქაღალდისა გახდა აგრეთვე ტელეგრაფი და განსაკუთრებით რადიო, დაიწყეს გასაღებების გაცვლა ღია არხის საშუალებით დაშიფრული სახით. მიუხედავად ამისა, სიმეტრიულ კრიპტოგრაფიაში გასაღებების შექმნის, განაწილების და შემდეგ გამოყენებული გასაღებების განადგურების პრობლემა ყოველთვის წარმოადგენდა ყველაზე რთულ პრობლემას, რომელზეც მოდიოდა სისტემაზე დახარჯული რესურების ყველაზე დიდი ნაწილი.

სწორედ ეს პრობლემა გახდა ღიაგასაღებიანი კრიპტოგრაფიის შექმნის ძირითადი მიზეზი. გასული საუკუნის სამოცდაათიან წლებში ამერიკელმა კრიპტოლოგებმა დიფიმ და ჰელმანმა გამოაქვეყნეს სტატია, რომელშიც დამტკიცებული იყო ღია გასაღებიანი კრიპტოგრაფიული სისტემების შესაძლებლობა, ხოლო ორი წლის შემდეგ გამოაქვეყნეს სტატია, რომელშიც აღწერილია საერთო საიდუმლო გასაღების შექმნის პროცედურა ღია, მოწინააღმდეგის მიერ სრულად კონტროლირებად არხში. ამ ალგორითმმა შემდეგში მიიღო დიფი - ჰელმანის ალგორითმის სახელი.

დიფი-ჰელმანის ალგორითმის იმით განსხვავდება ყველა დანარჩენისგან, რომ ამ შემთხვევაში ხდება არა წინასწარ განზადებული გასაღების გადაცემა, არამედ საერთო საიდუმლო გასაღების გამომუშავება მოწინააღდეგის მიერ სრულად კონტროლირებად არხში. ამ ალგორითმით, რომელიც შეიქმნა გასაღებების განაწილების პრობლემის გადასაჭრელად, დაიწყო ღიაგასაღებიანი კრიპტოგრაფია. დიფიმ და ჰელმანმა მოძებნეს

ისეთი ცალმხრივმიმართული ფუნქცია, რომელმაც მათ საშუალება მისცათ, შესრულებულიყო ტოლობა:

$$E_A(E_B(M)) = E_B(E_A(M)).$$

ეს არის სასრული ველის F_p^* ჯგუფში რიცხვის ახარისხების ფუნქცია, რომელსაც გააჩნია გასაღების გამომუშავების განაწესის შესასრულებლად ყველა საჭირო თვისება. ფუნქცია ცალმხრივმიმართულია, რადგანაც შებრუნებული ამოცანა, დისკრეტული ლოგარითმის მოძებნა F_p^* ჯგუფში ზოგადად წარმოადგენს არაპოლინომურ ამოცანას, რაც უზრუნველყოფს ალგორითმის კრიპტომედეგობას, ამასთან F_p^* ჯგუფი წარმოადგენს აბელის ჯგუფს გამრავლების მიმართ, რაც ნიშნავს, რომ სრულდება ზემოთ აღნიშნული კომუტატურობის თვისება. ალგორითმი შედგება ორი ძირითადი ეტაპისგან:

მოსამზადებელი ეტაპი. ალგორითმის შესასრულებლად ანი და ბექა წინასწარ უნდა შეთანხმდნენ ორ პარამეტრზე, p მარტივ და $g \in F_p^*$ რიცხვებზე. ამასთან, g უნდა იყოს ამ ჯგუფის წარმომადგენელი ელემენტი (ეს უკანასკნელი პირობა არაა აუცილებელი). რაც ყველაზე მთავარია, არაა საჭირო ამ რიცხვების დამალვა ეკასგან. ასე, რომ არაა საჭირო დახურული არხი.

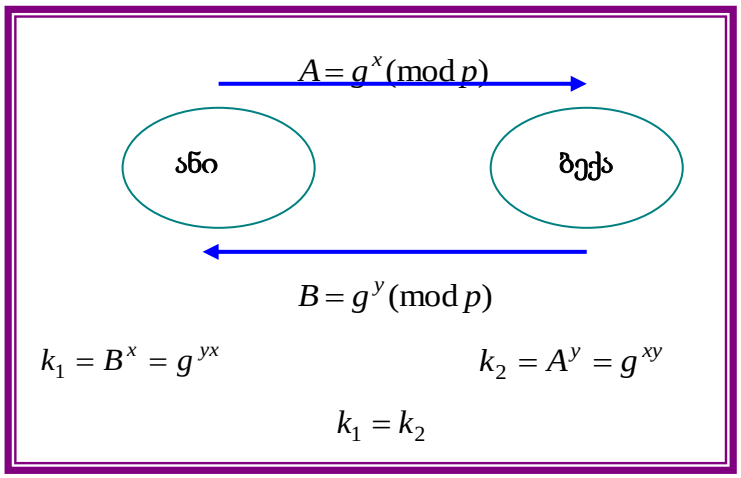
საერთო საიდუმლო გასაღების გამომუშავები ეტაპი. ამ ეტაპზე ღია არხში ანი და ბექა ცვლიან მხოლოდ ორ შეტყობინებას. ერთს აგზავნის ანი და ერთსაც ბექა. ეკა, რომელიც აკონტროლებს ღია არხს, მიიტაცებს ორივე შეტყობინებას, მაგრამ არ შეუძლია გამოთვალოს ანის და ბექას საიდუმლო გასაღები.

1. ანი აირჩევს ნებისმიერ რიცხვს $x \in \{2,..., p-2\}$, გამოთვლის $A = g^x (\bmod p)$ და მიღებულ შედეგს გაუგზავნის ბექას, ამასთან, მის მიერ არჩეულ x რიცხვს კი ინახავს საიდუმლოდ.

2. ბექაც აირჩევს ნებისმიერ რიცხვს $y \in \{2,..., p-2\}$, გამოთვლის $B = g^y (\bmod p)$ და მიღებულ შედეგს გაუგზავნის ანის. ბექაც არჩეულ y რიცხვს ინახავს საიდუმლოდ.

3. ანი გამოთვლის გასაღებს $k = B^x (\bmod p)$. ახლა ანის შეუძლია გაანადგუროს თავისი საიდუმლო რიცხვი, რათა არ აღმოაჩინოს იგი ეკამ.

4. ბექაც გამოთვლის გასაღებს $k = A^y \pmod p$. ბექასაც შეუძლია თავისი საიდუმლო გასაღების განადგურება.



სურ. 14.2. დიფი - ჰელმანის ალგორითმის სქემა.

ადვილი დასანახია, რომ ანიმ და ბექამ გამოთვალეს ერთი და იგივე გასაღები. მართლაც, ანიმ გამოთვალა $k = g^{yx} \pmod p$ და ბექამ კი - $k = g^{xy} \pmod p$. რადგანაც F_p^* ჯგუფი აბელის ჯგუფია, ეს სიდიდეები ერთმანეთის ტოლია.

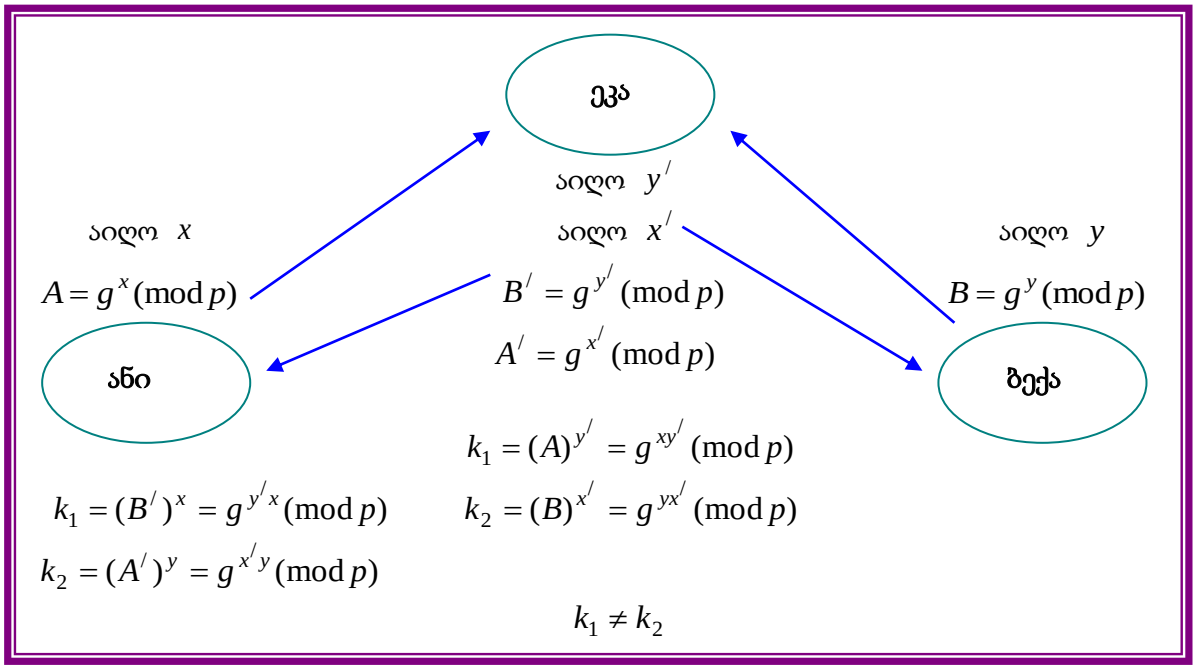
საიდუმლო გასაღებების მნიშვნელობა. როგორც ალგორითმის აღწერიდან ჩანს, ანიც და ბექაც იყენებენ საიდუმლო x და y პარამეტრებს, რომლებსაც ჩვენ ვუწოდეთ საიდუმლო გასაღებები, მაგრამ ისინი არ წარმოადგენენ საიდუმლო გასაღებებს იმ თვალსაზრისით, რომ ადრესატს არ სჭირდება მათი ცოდნა, მაგრამ თუ ეკა აღმოაჩენს ამ სიდიდეებიდან ერთს მაინც, ის თავისუფლად შეძლებს საერთო საიდუმლო გასაღების პოვნას და ამ გასაღების საშუალებით დაშიფრული ტექსტების გაშიფვრას. ამიტომ ასეთ გასაღებებს უწოდებენ **ეფემერულ გასაღებებს,** რეკომენდირებულია, რომ ეს გასაღები გამოყენებული იქნეს მხოლოდ ერთხელ და საერთო საიდუმლო გასაღების შექმნის შემდეგ მაშინვე იქნას განადგურებული.

აქტიური შეტევა სისტემაზე. დიფი-ჰელმანის ალგორითმის უსაფრთხოება ეფუძნება დისკრეტული ლოგარითმის პრობლემას და პარამეტრების სწორად შერჩევის შემთხვევაში, თუ ეკა განახორციელებს სისტემაზე მხოლოდ პასიურ შეტევას, ალგორითმის გატეხვა დღეს შეუძლებელია. მაგრამ ეკას ყოველთვის აქვს საშუალება

შეუტიოს სისტემას აქტიურად. აქტიური შეტევა ნიშნავს, რომ რადგანაც ეკა სრულად აკონტროლებს ღია არხს, ამიტომ მას შეუძლია ანის სახელით დაუკავშირდეს ბექას და შექმნას საერთო საიდუმლო გასაღები ბექასთან, შემდეგ ბექას სახელით დაუკავშირდეს ანის და შექმნას საერთო საიდუმლო გასაღები ანისთან.

ანი და ბექა ფიქრობენ, რომ მათ შექმნეს საერთო საიდუმლო გასაღები, სინამდვილეში კი თითოეულმა შექმნა საიდუმლო გასაღები ეკასთან. ამიტომ ეკა მიიღებს ანის გაგზავნილ შეტყობინებას, მოახდენს მის დეშიფრაციას წაიკითხავს შეტყობინებას. ამის შემდეგ ის დაშიფრავს ამ შეტყობინებას იმ გასაღებით, რომელიც შექმნეს მან და ბექამ და გაუგზავნის ბექას.

ასეთი შეტევის არსებობა მიუთითებს იმაზე, რომ დამოუკიდებლად, დამატებითი პროცედურების გარეშე, ამ ალგორითმის გამოყენება არ შეიძლება. თუ შევადარებთ ამ პროცედურას ინფორმაციის აუთენტიფიკაციის პროცედურას, დავინახავთ, რომ საჭიროა შეტყობინების ავტორის აუთენტიფიკაცია, რომელიც საერთო საიდუმლო გასაღების გამოყენების დროს ავტომატურად სრულდებოდა.



სურ. 14.3. აქტიური შეტევა დიფი - ჰელმანის ალგორითმზე.

პროგრამული უზრუნველყოფის კოდი

პროგრამული უზრუნველყოფა შესრულდა iOS პლატფორმის გამოყენებით. პროგრამირების ენად არჩეულ იქნა Swift. სერვერული მხარე რეალიზებულია google-ის სერვისით Firebase-ით, რომელსაც უკავშირდება კლიენტი აპლიკაცია. კლიენტი აპლიკაცია რეალიზებულია, როგორც iOS აპლიკაცია და თავსებადია Apple-ის მობილურ მოწყობილობებთან.

პირველი გვერდის Login-ის ზოგადი აღწერა

LoginViewController-ში ხდება Facebook ავტორიზაცია, ღია პროფილის წამოღება. Facebook token-ის Firebase-თვის გადაწოდება. (Facebook-ის token-ს ვაწვდით Firebase-ს, ხოლო Firebase ამ token-ზე დაყრდნობით წამოიღებს მონაცემებს და რეგისტრაციას გააკეთებს). აქ არის მომხმარებლის რეგისტრაცია/ლოგინი და მთავარი გვერდის ჩატვირთვა, სადაც რუკა და მეგობრების სიაა.

რუკა

MapViewController-ში მნიშვნელოვანი ობიექტია CLLocationManager(), რომელსაც არაერთი თვისება აქვს. ერთ-ერთი მნიშვნელოვანი თვისება არის მდებარეობის სიზუსტე, რომელსაც kCLLocationAccuracyBest გვაწვდის. ასევე მნიშვნელოვანია allowsBackgroundLocationUpdates, რომელიც საშუალებას გვაძლებს, მივიღოთ მდებარეობის ავტომატური განახლებები background რეჟიმში.

1) მნიშვნელოვანი მეთოდია createTimer(), რომელიც მომხმარებლის მიმდინარე ადგილს (მდებარეობას) Push-ით გაუგზავნის ყველა მეგობარს, რომლებთანაც ის აზიარებს ადგილს. როგორც კი მომხმარებლის მეგობრების სია ჩაიტვირთება, იმ მეგობრებთან, რომლებთანაც მომხმარებელი აზიარებს location-ს და ამასთან State არის accept, მოხდება latitude და logtitude-ის გაგზავნა. რაც შეეხება Generator-ს, იგი ახდენს ამ location-ის დაშიფრვას secret key-ს მიხედვით და აგზავნის ამ დაშიფრულ მონაცემს სერვერზე. შესაძლებელია რამდენიმე მომხმარებელთან კავშირი, რადგან ყველასთვის

სხვადასხვა გასაღებები დაგენერირდება. ამ დაშიფრული location-ის გაგზავნა ყველა მეგობრისთვის ხდება ყოველ 5 წამში ერთხელ.

```
func createTimer() {
    timer = Timer.scheduledTimer(withTimeInterval: 5, repeats: true) { [weak self] (timer) in
        guard let location = self?.lastLocation else { return }
        guard let uid = Auth.auth().currentUser?.uid else { return }
        guard let users = self?.delegate?.model.connectedUsers else { return }

        for user in users.filter({ $0.value.state == .accept }) {
            let locationRef =
Database.database().reference().child("locations").child(user.key).child(uid)
            let lat = Generator.send(message: "\(location.coordinate.latitude)", exchange:
user.value.exchange, secret: user.value.secret)
            let lng = Generator.send(message: "\(location.coordinate.longitude)", exchange:
user.value.exchange, secret: user.value.secret)
            locationRef.updateChildValues([
                "lat": lat,
                "lng": lng,
                "timestamp": [".sv": "timestamp"],
            ])
        }
    }
}
```

2) listenToConnections()-ში ხდება მომხმარებლის connections-ზე მოსმენა ახლის დამატებაზე, ცვლილებაზე ან წაშლაზე. ამის მიხედვით ხდება მიმდინარე მეგობრების წამოღება, რომლებთანაც აზიარებს locations-ს. ამ სიაზე დაყრდნობით ხდება მომხმარებლის მეგობრების გამოჩენა რუკაზე. თითოეულ connection-ს მოყვება modulus, base, uid(Firebase-ის უნიკალური id) და connection state.

```
func listenToConnections() {
    guard let uid = Auth.auth().currentUser?.uid else { return }
    let myConnectionsRef = Database.database().reference().child("connections").child(uid)

    myConnectionsRef.observe(.childAdded, with: { [weak self] snapshot in
        print("connections .childAdded")

        self?.delegate?.model.createOrEditConnectedUser(for: snapshot.key, with:
snapshot.value, isRemoved: false)
```

```
    }, withCancel: nil)

    myConnectionsRef.observe(.childChanged, with: { [weak self] snapshot in
      print("connections .childChanged")

      self?.delegate?.model.createOrEditConnectedUser(for: snapshot.key, with:
snapshot.value, isRemoved: false)

    }, withCancel: nil)

    myConnectionsRef.observe(.childRemoved, with: { [weak self] snapshot in
      print("connections .childRemoved")

      self?.delegate?.model.createOrEditConnectedUser(for: snapshot.key, with:
snapshot.value, isRemoved: true)

    }, withCancel: nil)
  }
```

3) listenToSharedLocations()-ში ხდება მომხმარებელთან გაზიარებული location-ების მოსმენა. როგორც კი რომელიმე შეიცვლება, მისი ასახვა მოხდება რუკაზე.

ზოგადი კონტეინერი

ViewController წარმოადგენს კონტეინერს, სადაც Map და Friends დევს. აქედან ხდება "Log out" ღილაკის გამოყენება და გადასვლა LogIn გვერდზე. ასევე ხდება tab-ებზე ანიმაციით აქეთ-იქით გადასვლა.

მეგობრები

FriendsViewController-ში გვაქვს Facebook მეგობრების სია, რომლებიც იყენებენ აპლიკაციას. changeConnectionState-ში ხდება მეგობრისთვის სხვადასხვა მოთხოვნის გაგზავნა (LocationSharingState). მომხმარებელს შეუძლია მოთხოვნა გაგზავნოს, შემოთავაზებულ მოთხოვნას დაეთანხმოს, ან უარყოს. ამავე ნაწილში ხდება შემთხვევითად მიმდინარე მომხმარებლის და მისი მეგობრისsecret-ის გენერირება.

მუშაობს რამდენიმე switch. თუ მე ვაგზავნი მოთხოვნას ჩემს მეგობართან, იმუშავებს .request, თუ უარვყოფ ან ვეთანხმები, მაშინ შესაბამისად .deny, .accept. MultiplicativeGroupOfIntegersModulo არის მნიშვნელოვანი კლასი, რომელიც ინახავს modulus და base-ს, რომელთა დაგენერირება ხდება შემთხვევითად 10-დან 300-მდე.

ვთქვათ generator0 არის A მეგობარი, მისი უნიკალური იდენტიფიკატორი და secret. generator1 არის B მეგობარი. sharedSecrets გენერირდება A და B მეგობრების secret-ების, ასევე modulus და base-ს მიხედვით. აქ ხდება ალგორითმის მნიშვნელოვანი პარამეტრების გენერირება და საბოლოოდ ამ ყველაფრის ჩაწერა ბაზის connections ხეში.

```
func changeConnectionState(uid: String, friendUID: String, for state: LocationSharingState) {

    let myConnections =
Database.database().reference().child("connections").child(uid).child(friendUID)
    let friendConnections =
Database.database().reference().child("connections").child(friendUID).child(uid)

    switch state {
    case .request:
        let exchange = MultiplicativeGroupOfIntegersModulo(modulus: random(), base:
random())
        let generator0 = Generator(id: uid, secret: random())
        let generator1 = Generator(id: friendUID, secret: random())
        let sharedSecrets = Generator.createSharedSecrets(generator0, generator1, exchange:
exchange)

        myConnections.updateChildValues([
            "state" : state.rawValue,
            "isSender" : false,
            "secret" : generator0.secret,
            "sharedSecret" : sharedSecrets.s1,
            "modulus": exchange.modulus,
            "base": exchange.base
        ])

        friendConnections.updateChildValues([
            "state" : state.rawValue,
            "isSender" : true,
            "secret" : generator1.secret,
            "sharedSecret" : sharedSecrets.s2,
            "modulus": exchange.modulus,
            "base": exchange.base
        ])
    case .deny, .accept:
        myConnections.updateChildValues(["state" : state.rawValue ])
        friendConnections.updateChildValues(["state" : state.rawValue ])
    case .unknown:
        myConnections.removeValue()
```

```
        friendConnections.removeValue()
      }
    }
}
```

მოდელები

1)ConnectedUserSetting მოდელში არის Firebase-ის connections ხეში არსებული ობიექტი.

2) Friend მოდული გვეხმარება მეგობრის Firebase id, მისი უნიკალური აიდი, რომელიც Facebook-დან მოდის, firstname, lastname და picture. მოდული აპლიკაციაში Friends-ის გვერდზე არსებულ პარამეტრებს ემსახურება.

3) EncryptedLocation მოდული ემსახურება Firebase-ის locations ხის latitude და lotitude-ის გამოთვლას. ასევე ხდება დაშიფრული ინფორმაციის აღდგენა. Facebook სურათის ჩახატვა აპლიკაციაში.

4) Model არის მთავარი მოდელი, რომელიც მოიცავს როგორც მეგობრების სიას, ასევე კონკრეტულად მომხმარებელთან დაკავშირებულ მეგობრებს და ადგილებს, რომლებიც მასთანაა გაზიარებული.

5) LocationSharingState მოდელში ხდება state-ების მიხედვით მომხმარებლებისთვის სხვადასხვა შეტყობინებების ამოგდება.

ალგორითმის კოდის მხარე

კლიენტის აპლიკაციის კოდის იმ ნაწილს, რომელიც პასუხისმგებელია კრიპტოგრაფიული მეთოდებით მდებარეობის ინფორმაციის დაცვაზე, აქვს შემდეგი სახე:

ორი მნიშვნელოვანი რიცხვი, რომელიც თავიდანვე ორივესთვის გაცხადებულია, კოდში გვხვდება, როგორც base და modulus. თითოეული მომხმარებლის მიერ შერჩეულ საიდუმლო გასაღებს წარმოადგენს secret. გასაღებს, რომელსაც ისინი ცხადად ცვლიან ღია არხით არის sharedSecret, ხოლო ის secret key, რომელიც მათთვის საერთოა, ცხადი სახით არსად არ ჩანს.

ალგორითმის კოდის სეგმენტში გვაქვს Exchange პროტოკოლი (ინტერფეისი), რომელიც შედგება 2 მეთოდისგან და რომელთა იმპლემენტაციასაც ახდენს MultiplicativeGroupOfIntegersModulo სტრუქტურა.

Exchange.swift

```
import Foundation

protocol Exchange {
   func sharedBaseForInsecureTransmission(secret: Int) -> Int
   func sharedSecretByMixing(receivedSharedBase: Int, with secret: Int) -> Int
}
```

ეს სტრუქტურა მოიცავს modulus-ს და base-ს. პირველი მეთოდია sharedBaseForInsecureTransmission, რომელსაც გადაეცემა პარამეტრად secret და აბრუნებს sharedSecret-ს. ხოლო მეორე მეთოდი sharedSecretByMixing იღებს პარამეტრად მიღებულ sharedSecret-ს და secret-ს, და აბრუნებს secret key-ს. ორივე ეს ფუნქცია იყენებს powMod მეთოდს, რომელიც ალგორითმის ფორმულის მიხედვით ახდენს ახარისხებას და მოდულის აღებას.

MultiplicativeGroupOfIntegersModulo.swift

```
import Foundation

struct MultiplicativeGroupOfIntegersModulo: Exchange {

   let modulus: Int
   let base: Int

   init(modulus: Int, base: Int) {
      self.modulus = modulus
      self.base = base
   }

   func sharedBaseForInsecureTransmission(secret: Int) -> Int {
      return powMod(radix: base, power: secret)
   }

   func sharedSecretByMixing(receivedSharedBase: Int, with secret: Int) -> Int {
      return powMod(radix: receivedSharedBase, power: secret)
   }
```

```
    private func powMod(radix: Int, power: Int) -> Int {
        let value = (BInt(integerLiteral: radix) ^ power) % BInt(integerLiteral: modulus)
        return value.toInt()!
    }
}
```

Generator-ის აღწერა. Generator-ს ვაწვდით საკუთარ უნიკალურ იდენტიფიკატორს და საკუთარ secret-ს. თუ მაგალითად 2 user-სთვის გვინდა sharedSecret-ების დაგენერირება, ამას უზრუნველყოფს createSharedSecrets-ს მეთოდი, რომელიც ორივესთვის შექმნის ობიექტს და sharedSecret-ს sharedBaseForInsecureTransmission მეთოდის გამოყენებით, რომელიც ზევითაა აღწერილი. გენერატორი საბოლოოდ აბრუნებს ორივესთვის sharedSecret-ებს, რომელთა ჩაწერაც ხდება ბაზაში. გენერატორს აქვს 2 სტატიკური მეთოდი: send და receive. რადგან ბაზაში ვინახავთ ინფორმაციას connections-ის ხეში, ამ ინფორმაციის გადაწოდება ხდება send-თვის, send იღებს location-ს, Exchange-ს, რომელიც არის base და modulus-ის ერთობლიობა და Secret-ს, რომელიც არის secret და sharedSecret. ამ მეთოდის შიგნით ცვლადი sharedSecret არის ის secret key, რომლითაც ხდება ტექსტის დაშიფრვა და ამ დაშიფრული ტექსტის (კოორდინატების) ჩაწერა ბაზაში. ამას უზრუნველყოფს encrypt მეთოდი.

გვაქვს Cypher პროტოკოლი (ინტერფეისი), რომელიც შედგება ორი მეთოდისგან:

Cypher.swift

```
import Foundation

protocol Cipher {
    func encrypt(message: String, secret: Int) -> String
    func decrypt(message: String, secret: Int) -> String
}
```

ამ მეთოდების იმპლემენტაციას ახდენს XOR სტრუქტურა. ამ სტრუქტურაში გვაქვს 3 მეთოდი: encrypt, decrypt და impl.

```
import Foundation

struct XOR: Cipher {
    private func impl(message: String, secret: Int) -> String? {
        return message.unicodeScalars.map { UnicodeScalar(UInt32($0.value) ^ UInt32(secret))
}.flatMap{ $0 }.reduce("") { $0 + String($1) }
    }
```

```
    func encrypt(message: String, secret: Int) -> String { return impl(message: message, secret:
secret)! }
    func decrypt(message: String, secret: Int) -> String { return impl(message: message, secret:
secret)! }
}
```

encrypt მეთოდს გადაეცემა დასაშიფრი ტექსტი და secret key, და აბრუნებს დაშიფრულ ტექსტს. იგი იყენებს დაკრიპტვის ალგორითმს, რომელიც შეიძლება ბევრნაირად შევასრულოთ, მთავარია, რომ secret key-ს მიხედვით ხდება თითოეული სიმბოლოს ცვლილება. ეს impl იღებს ტექსტს და secret key-ს და ახდენს ალგორითმის მიერ დაგენერირებული დიდი რიცხვის თითოეული სტრიქონის რაღაც უნიკოდის სხვა სტრიქონით ჩანაცვლებას.

Generator.swift

```
import Foundation

class Generator {
    let id: String
    let secret: Int

    init(id: String, secret: Int) {
        self.id = id
        self.secret = secret
    }

    static func createSharedSecrets(_ g0: Generator, _ g1: Generator, exchange: Exchange) -> (s1:
Int, s2: Int) {
        let s2 = exchange.sharedBaseForInsecureTransmission(secret: g0.secret)
        let s1 = exchange.sharedBaseForInsecureTransmission(secret: g1.secret)

        return (s1, s2)
    }

    static func send(message text: String, exchange: Exchange, secret: Secret) -> String {
        let cipher: Cipher = XOR()
        let sharedSecret = exchange.sharedSecretByMixing(receivedSharedBase: secret.shared,
with: secret.private)
        return cipher.encrypt(message: text, secret: sharedSecret)
    }

    static func recieve(message text: String, exchange: Exchange, secret: Secret) -> String {
        let cipher: Cipher = XOR()
```

```
        let sharedSecret = exchange.sharedSecretByMixing(receivedSharedBase: secret.shared,
with: secret.private)
        return cipher.encrypt(message: text, secret: sharedSecret)
    }
}

func random() -> Int {
    let fileURL = Bundle.main.url(forResource: "prime", withExtension: "txt")
    let content = try! String(contentsOf: fileURL!, encoding: String.Encoding.utf8)
    let primes = content.components(separatedBy: "\n")
    let count = primes.count
    let r = Int(arc4random_uniform(UInt32(count)))

    return Int(primes[r])!
}
```

დასკვნა

ამრიგად, წინამდებარე ნაშრომის ფარგლებში შექმნილია შეხვედრის ადგილის საიდუმლოდ ორგანიზების სისტემა, რომელსაც აქვს კლიენტ-სერვერული აპლიკაციის სახე. სისტემაში გარანტირებულია ინფორმაციის გავრცელების ფარულობა, რაც უზრუნველყოფილია კრიპტოგრაფიული მეთოდებით. სისტემის კლიენტური ნაწილი არ მოითხოვს დიდ რესურსებს და შესაძლებელია მისი ინსტალაცია მობილურ მოწყობილობებზე, რაც საშუალებას იძლევა, რომ მომხმარებლებმა უსაფრთხოდ და დაცულად გაცვალონ თავიანთი ადგილმდებარეობები. სისტემა შექმნილია თანამედროვე ტექნოლოგიების გამოყენებით, მისი მოქმედების პრინციპი დაფუძნებულია სამეცნიერო კვლევებზე და მას მიცემული აქვს საბოლოო სახე, რაც შესაძლებელს ხდის მის გამოყენებას კომერციული მიზნებისათვის.

გამოყენებული ლიტერატურა

1. Firebase Realtime Database- https://Firebase.google.com/docs/database/
2. კრიპტოგრაფია- https://ka.wikipedia.org/wiki/%E1%83%99%E1%83%A0%E1%83%98%E1%83%9E%E1%83%A2%E1%83%9D%E1%83%92%E1%83%A0%E1%83%90%E1%83%A4%E1%83%98%E1%83%90
3. პროგრამული ენა swift-ს დოკუმენტაცია- https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/TheBasics.html#//apple_ref/doc/uid/TP40014097-CH5-ID309
4. ქოჩლაძე ზ. თანამედროვე კრიპტოგრაფიის საფუძვლები. თბილისის უნივერსიტეტის გამომცემლობა, 2013