Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра информационной безопасноти

Исраелян Армен Давидович

Реализация децентрализованного хранения защищенной базы данных

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:

д.ф.-м.н., профессор

М. А. Черепнев

Москва, 2023

Содержание

1 Введение

Децентрализованные технологии становятся все более популярными в нашем мире, и это не удивительно, ведь они позволяют создавать более безопасные и надежные системы хранения данных. В данной выпускной квалификационной работе рассматривается реализация децентрализованного хранения защищенной базы данных на языке kotlin для смартфонов с операционной системой android.

В работе рассмотрены существующие методы хранения данных и их недостатки, а также принципы децентрализованной технологии. Особое внимание уделено разработке механизма хранения данных в децентрализованной среде и обеспечению безопасности базы данных посредством разработки алгоритма для процесса аутентификации пользователей.

Для реализации данной задачи были использована технология wifi Direct которая позволяет создать независимую от интернета сеть, библиотека DSA шифрования, которые обеспечивают надежность данных.

Результатом данной работы является реализованный прототип децентрализованной базы данных с собсвтенным протоколом аутентификации, который может быть использован в различных сферах, где требуется высокий уровень безопасности и надежности хранения данных.

2 Обзор методов хранения базы данных

Существует множество методов хранения данных, от традиционных реляционных баз данных до новых технологий, таких как NoSQL и графовые базы данных. Каждый метод имеет свои преимущества и недостатки, и выбор определенного метода зависит от конкретных потребностей и задач. Однако, с появлением децентрализованных технологий, таких как блокчейн, появилась возможность создания более безопасных и надежных систем хранения данных.

Основными принципами децентрализованной технологии являются отсутствие централизованной власти и распределенность. В децентрализованных системах все участники сети имеют равные права и контролируют процессы в сети. Это делает систему более устойчивой к атакам и более надежной в целом.

На сегодняшний день существует множество решений в области децентрализованного хранения данных, таких как IPFS, Swarm, Storj, Sia и другие. Они предлагают различные подходы к хранению данных в децентрализованной среде и имеют свои преимущества и недостатки. Например, IPFS использует распределенную файловую систему, основанную на хэшах, чтобы хранить и распространять файлы, тогда как Swarm использует инфраструктуру, основанную на контрактах Ethereum, чтобы хранить и распространять контент. Sia и Storj используют блокчейн для хранения данных и предлагают вознаграждение за участие в сети. Каждое решение имеет свои преимущества и недостатки, и выбор зависит от конкретных потребностей и задач. В этой работе мы будем рассматривать вариант протокола p2p с модификацией в виде хранения абонентами полной базы данных. У такой схемы хранения множество приемуществ перед оригиналом, например, если какой-то узел выходит из сети, остальные узлы продолжают хранить копию базы данных, что обеспечивает высокую доступность данных. Для обеспечения безопасности и целостности данных будет использоваться протокол ЭЦП DSA и алгоритм аутентификации, который позволит определять можно ли доверять пользователю.

3 Постановка задачи

Результатом работы должно стать приложение для Android смартфонов в котором реализованны децентрализованная база данных, протокол аутентификации, протокол DSA. База данных будет формироваться из записей состоящих из временной метки, координат и цифовой подписи абонента.Так же должен быть реализован нативный интерфейс для просмотра базы данных и поиска по ней. Протокол аутентификации будет работать так. Каждую минуту все абоненты сети будут добавлять в базу данных запись и каждый абонент сети будет проводить процес верификации записи который будет состоять из проверки расстояния от себя, проверки насколько новой является временная метка записи и проверки ЭЦП. После верификации, каждый абонент проверивший запись пользователя должен будет опубликовать запись состоящую из результатов проверки, если хоть одна проверка не будет пройденна, результат будет считаться негативным, если все проверки пройденны успешно результат - позитивный. В случае негативного результата проверки все абоненты смогут увидеть, что один из них не прошел проверку. Это будет значить, что записям этого абонента нельзя доверять. По четырем возможным причинам.

- Абонент пытается подделать сообщение изменив временную метку
- Абонент пытается подключиться к базе данных при этом не находясь рядом
- Абонент пытается подделать цифровую подпись
- Абонент имеет плохое соединение

Таким образом сеть будет защищенна от несанкцианированного или небезопасного подключения, ведь для обмена данными абоненты должны находиться в непосредственной близости друг от друга.

Такая программа может быть применнена например для автоматической системы генерации рапортов отрядов в дозоре или рзаведке. Таким образом отряд будет иметь независимую ни от чего кроме абонентов, базу данных, на протяжении всего задания, а в конце можно будет узнать кто, когда и где находился.

3.1 Цели работы

Таким образом основные цели работы Это

- Реализация нативного пользовательского интерфейса для приложения
- Реализация p2p сети между смартфонами используя технологию Wifi Direct
- Реализация протокола создания и хранения базы данных
- Реализация протокола верификации записи
- Реализация протокола аутентификации пользователей на базе протокола верификации

4 Программная реализация

Для реализации были использованны следующие иснтрументы

- Язык xml : для нативного интерфейса
- Стандартная библиотека android.net.wifi.p2p и библиотеки из java java.net.InetSocketAdd java.net.ServerSocket, java.net.Socket : для реализации сети p2p коммуникация в которой работает через WifiDirect
- Библиотека androidx.room : для базы создания обработки их хранения базы данных
- Метод currentTimeMillis() из класса System : для временной метки
- Стандартные библиотеки android.location.Location android.location.LocationListener android.location.LocationManager : для получения координат
- Библиотеки из java java.security java.security.spec.PKCS8EncodedKeySpec : для реализцаии ЭЦП
- Библиотека гугла com.google.gson.Gson : для реализации передачи объектов по сокетам

4.1 Программная реализация

Для реализации подключения к устройствам через Wifi Direct была созданна функция

```
  val connectionInfoListener = object : WifiP2pManager.ConnectionInfoListener {
        var serverThread: Thread? = null
        var clientThread: Thread? = null
        override fun onConnectionInfoAvailable(wifiP2pinfo: WifiP2pInfo) {
            val groupOwnerAddress: InetAddress = wifiP2pinfo.groupOwnerAddress
            if (wifiP2pinfo.groupFormed && wifiP2pinfo.isGroupOwner) {
                connectionStatus.text = "Host"
                Log.v("I AM A HOST", "\n")
                if (serverThread == null) {
                    serverThread = Thread {

                        val serverConnectionHandler = ServerConnectionHandler(
                            serverSocket!!, dBase)
                        serverConnectionHandler.start()
                    }
                    serverThread!!.start()
                }
            } else if (wifiP2pinfo.groupFormed) {
                connectionStatus.text = "Client"
                if (clientThread == null) {
                    Log.v("I AM A CLIENT", "\n")
                    clientThread = Thread {
                        val clientHandler = ClientConnectionHandler(
                            groupOwnerAddress, 8888, dBase)
                        clientHandler.connect()
                    }
                    clientThread!!.start()
                }
            }
        }
    }
  }
}
```

В этой функции при подключении к устройству оно переходит в одно из двух состояний : состояние сервера, состояние клиента. Состояние сервера имплементировано в классе ServerConnectionHandler

```
class ServerConnectionHandler(
        private val serverSocket: ServerSocket,
        private val DB: mroomDatabase,
    ) {
        private val clients = mutableSetOf<OutputStream>()
        fun start() {
            val threadPool = Executors.newCachedThreadPool()
            val timer = Timer()
            timer.scheduleAtFixedRate(object : TimerTask() {
                override fun run() {
                    broadcastMessage()
                }
            }, 2000, 60 * 1000)
            Thread {
                while (true) {
                    val clientSocket = serverSocket.accept()
                    threadPool.submit { handleClientConnection(clientSocket) }
                }
            }.start()
        }

        private fun handleClientConnection(clientSocket: Socket) {
            val inputStream: InputStream = clientSocket.getInputStream()
            val outputStream: OutputStream = clientSocket.getOutputStream()
            clients.add(outputStream)
            var lasttime :Long =0
            var Abo : String? = null
            while (true) {
                val buffer = ByteArray(1024)
                val size = inputStream.read(buffer)
                val gson = Gson()
                val jsonString = buffer.sliceArray(0 until size).toString(
                    Charsets.UTF_8)
                val receivedMessage = gson.fromJson(jsonString, Record::class.
                    java)
                lasttime = receivedMessage.timestamp
                if (Abo != null) Abo = receivedMessage.sender
                if (System.currentTimeMillis() - lasttime > 120000){
                    val apr = Record(
                        approval = 2,
                        sender = "${Build.MODEL}",
                        message ="user ${Abo} gone for longer than 2 minutes",
                        sign = generateSignature("user ${Abo} gone for longer
                            than 2 minutes").toString(),
                        timestamp = System.currentTimeMillis()
                    )
                    handleClientMessage(apr)
```

```
                    lasttime = System.currentTimeMillis()
                }
                handleClientMessage(receivedMessage)
                if (receivedMessage.approval == 0) approve(
                    receivedMessage)
            }
        }

        private fun handleClientMessage(message: Record) {
            // insert message into database
            clients.forEach {
                val gson = Gson()
                val jsonString = gson.toJson(message)
                val data = jsonString.toByteArray(Charsets.UTF_8)
                it.write(data) // send message to each client
            }
            DB.dbDao().insertMessage(message)
        }

        private fun broadcastMessage() {
            val time = System.currentTimeMillis()
            val location = mLocation
            val message = Record(approval = 0,
                sender = Build.MODEL,
                message = "${location?.latitude} ${location?.longitude}",
                sign = generateSignature(Build.MODEL.toString() + location.
                    toString() + time.toString()).toString(),
                timestamp = time)
            DB.dbDao().insertMessage(message)
            clients.forEach() {
                val gson = Gson()
                val jsonString = gson.toJson(message)
                val data = jsonString.toByteArray(Charsets.UTF_8)
                Log.d("SERVER", "BROADCASTING ${message.timestamp} to $it")
                it.write(data)
            }
        }

        private fun approve(message: Record) {
            var approval: Int = 1
            var pod = "Sign confirmed"
            var tim = "Time confirmed"
            val time = System.currentTimeMillis()
            var loc = "${getDateTime(time)} Location confirmed"
            if ((time - message.timestamp) > 10000) {
                Log.d("Time check", "FAILED")
                approval = 2
                tim = "Time confirmation failed"
            }
            val location = getLocationFromMessageText(message.message)
            val lat = location?.first
```

```
95              val lon = location?.second
96              val Dist = 0.01
97              if (getDistance(mLocation!!.latitude, mLocation!!.longitude, lat!!,
                    lon!!) > Dist) {
98                  Log.d("Location Check", "FAILED")
99                  approval = 2
100                 loc = "${getDateTime(time)} User to far"
101             }
102             if (verifySignature(message.message, message.sign, publicKey) > Dist
                    ) {
103                 Log.d("Location Check", "FAILED")
104                 approval = 2
105                 loc = "${getDateTime(time)} User to far"
106             }
107             val apr = Record(
108                 approval = approval,
109                 sender = "${Build.MODEL} checks ${message.sender}",
110                 message = loc+"\n"+tim+"\n"+pod,
111                 sign = generateSignature(message.message).toString(),
112                 timestamp = message.timestamp + 1)
113             handleClientMessage(apr)
114         }
115         private fun getDateTime(s: Long): String? {
116             val timeFormat = SimpleDateFormat("HH:mm:ss", Locale.getDefault())
117             return timeFormat.format(Date(s))
118         }
119
120         private fun getDistance(lat1: Double, lon1: Double, lat2: Double, lon2:
                Double): Double {
121             val earthRadius = 6371 // in km
122             val dLat = Math.toRadians(lat2 - lat1)
123             val dLon = Math.toRadians(lon2 - lon1)
124             val a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
125                     Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(
                            lat2)) *
126                     Math.sin(dLon / 2) * Math.sin(dLon / 2)
127             val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
128             return earthRadius * c
129         }
130
131         fun getLocationFromMessageText(messageText: String): Pair<Double, Double
                >? {
132             val parts = messageText.split(" ")
133             if (parts.size != 2) {
134                 return null
135             }
136             val latitude = parts[0].toDoubleOrNull() ?: return null
137             val longitude = parts[1].toDoubleOrNull() ?: return null
138             return Pair(latitude, longitude)
139         }
140
```

```
141         private fun generateSignature(message: String): BigInteger {
142             val file = File("/data/data/com.example.diploma/files/SecretKey.txt"
                    )
143             val privateKeyHex = file.readText(Charset.defaultCharset())
144             val privateKeyBytes =
145                 privateKeyHex.chunked(2).map { it.toInt(16).toByte() }.
                        toByteArray()
146             val keySpec = PKCS8EncodedKeySpec(privateKeyBytes)
147             val keyFactory = KeyFactory.getInstance("DSA")
148             val x = keyFactory.generatePrivate(keySpec)
149             val signature: Signature = Signature.getInstance("SHA1withDSA")
150             signature.initSign(x)
151             signature.update(message.toByteArray())
152
153
154             return BigInteger(signature.sign())
155         }
156         private fun verifySignature(message: String, signatureBytes: BigInteger)
                : Boolean {
157             val signature = Signature.getInstance("SHA1withDSA")
158             signature.initVerify(publicKey)
159             signature.update(message.toByteArray())
160             return !signature.verify(signatureBytes.toByteArray())
161         }
162     }
```

Этот класс реализует сервер, который постоянно "слушает"подключения пользователей, и постоянно рассылает и обрабатывает их сообщения в handleClientMessage(). Так же сервер раз в минуту рассылает клинетам запись со своими данными. В этом классе описаны ключевые функции имплементированные в программной реализации. Для подписывания и проверки подписи используются функции generateSignature и verifySignature. Для проверки локации реализова на функция getDistance которая возвращает расстояние и класс locationManager

```
1         locationManager = getSystemService(LOCATION_SERVICE) as LocationManager
2         val lastKnownLocation = locationManager.getLastKnownLocation(
              LocationManager.GPS_PROVIDER)
3         if (lastKnownLocation != null) {
4             mLocation = lastKnownLocation
5         }
```

Состояние клиента имеет в себе те же функции, что и состояние сервера, но оно подключается только к одному пользователю, и не рассылает сообщения при получении.

```
1
2     class ClientConnectionHandler(
3         private val serverAddress: InetAddress,
4         private val serverPort: Int,
5         private val DB: mroomDatabase,
6     ) {
7         private lateinit var clientSocket: Socket
8         fun connect() {
9             Log.v("Connect", "Creating connection")
10            try {
11                clientSocket = Socket(serverAddress, serverPort)
12                Log.v("connect", "Connection successful")
13                handleServerConnection(clientSocket)
14                sendPeriodicMessage()
15            } catch (e: Exception) {
16
17                e.printStackTrace()
18            }
19
20        }
21
22        fun sendPeriodicMessage() {
23            val timer = Timer()
24            timer.schedule(object : TimerTask() {
25                override fun run() {
26                    sendMessage()
27                }
28            }, 0, 60 * 1000) // send message every 1 minute (60 seconds * 1000
                  milliseconds)
29        }
30
31        private fun handleServerConnection(socket: Socket) {
32            Thread {
33                try {
34                    val inputStream = socket.getInputStream()
35                    val buffer = ByteArray(1024)
36                    while (true) {
37                        val size = inputStream.read(buffer)
38                        if (size <= 0) {
39                            // End of stream reached, break out of the loop
40                            break
41                        }
42                        val gson = Gson()
43                        val jsonString = buffer.sliceArray(0 until size).
                              toString(Charsets.UTF_8)
44                        val receivedMessage = gson.fromJson(jsonString, Record::
                              class.java)
```

```
                    handleMessage(receivedMessage)
                }
            } catch (e: Exception) {
                e.printStackTrace()
            }
        }.start()
    }

    private fun handleMessage(message: Record) {
        Log.d("TAG", "SAY from :${message.message} ${message.timestamp} to:
            ${Build.MODEL}")
        Log.d("Client handling message", "\n")
        DB.dbDao().insertMessage(message)
        if ((message.approval == 0) && (message.sender != Build.MODEL)) {
            Log.d("Client Approval", "STARTS")
            approve(message)
        }
    }

     private fun approve(message: Record) {
        var approval: Int = 1
        var pod = "Sign confirmed"
        var tim = "Time confirmed"
        val time = System.currentTimeMillis()
        var loc = "${getDateTime(time)} Location confirmed"
        if ((time - message.timestamp) > 10000) {
            Log.d("Time check", "FAILED")
            approval = 2
            tim = "Time confirmation failed"
        }
        val location = getLocationFromMessageText(message.message)
        val lat = location?.first
        val lon = location?.second
        val Dist = 0.01
        if (getDistance(mLocation!!.latitude, mLocation!!.longitude, lat!!,
            lon!!) > Dist) {
            Log.d("Location Check", "FAILED")
            approval = 2
            loc = "${getDateTime(time)} User to far"
        }
        if (verifySignature(message.message, message.sign, publicKey) > Dist
            ) {
            Log.d("Location Check", "FAILED")
            approval = 2
            loc = "${getDateTime(time)} User to far"
        }
        val apr = Record(
            approval = approval,
            sender = "${Build.MODEL} checks ${message.sender}",
            message = loc+"\n"+tim+"\n"+pod,
            sign = generateSignature(message.message).toString(),
```

```
                timestamp = message.timestamp + 1)
            sendMessage(apr)
        }
        private fun getDateTime(s: Long): String? {
            val timeFormat = SimpleDateFormat("HH:mm:ss", Locale.getDefault())
            return timeFormat.format(Date(s))
        }

        private fun getDistance(lat1: Double, lon1: Double, lat2: Double, lon2:
            Double): Double {
            val earthRadius = 6371 // in km
            val dLat = Math.toRadians(lat2 - lat1)
            val dLon = Math.toRadians(lon2 - lon1)
            val a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                    Math.cos(Math.toRadians(lat1)) * Math.cos(Math.toRadians(
                        lat2)) *
                    Math.sin(dLon / 2) * Math.sin(dLon / 2)
            val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
            return earthRadius * c
        }

        fun getLocationFromMessageText(messageText: String): Pair<Double, Double
            >? {
            val parts = messageText.split(" ")
            if (parts.size != 2) {
                return null
            }
            val latitude = parts[0].toDoubleOrNull() ?: return null
            val longitude = parts[1].toDoubleOrNull() ?: return null
            return Pair(latitude, longitude)
        }

        fun sendMessage(record: Record? = null) {
            try {
                if (::clientSocket.isInitialized && clientSocket.isConnected) {
                    val outputStream = clientSocket.getOutputStream()
                    val time = System.currentTimeMillis()
                    val location = mLocation
                    val sender = Build.MODEL
                    val messageText = "${location?.latitude} ${location?.
                        longitude}"
                    val sign = generateSignature(messageText).toString()
                    val approval = if (record == null) 0 else record.approval
                    val message = record ?: Record(approval = approval,
                        sender = sender,
                        message = messageText,
                        sign = sign,
                        timestamp = time)
                    val gson = Gson()
                    val jsonString = gson.toJson(message)
                    val data = jsonString.toByteArray(Charsets.UTF_8)
```

```
                    outputStream.write(data)
                } else {
                    Log.v("Failed to send message",
                        "clientSocket is uninitialized or not connected")
                }
            } catch (e: Exception) {
                Log.v("Failed to send message", "$clientSocket is unavailable")
                e.printStackTrace()
            }
        }

        private fun generateSignature(message: String): BigInteger {
            val file = File("/data/data/com.example.diploma/files/SecretKey.txt"
                )
            val privateKeyHex = file.readText(Charset.defaultCharset())
            val privateKeyBytes =
                privateKeyHex.chunked(2).map { it.toInt(16).toByte() }.
                    toByteArray()
            val keySpec = PKCS8EncodedKeySpec(privateKeyBytes)
            val keyFactory = KeyFactory.getInstance("DSA")
            val x = keyFactory.generatePrivate(keySpec)
            val signature: Signature = Signature.getInstance("SHA1withDSA")
            signature.initSign(x)
            signature.update(message.toByteArray())
            return BigInteger(signature.sign())
        }
        private fun verifySignature(message: String, signatureBytes: BigInteger)
            : Boolean {
            val signature = Signature.getInstance("SHA1withDSA")
            signature.initVerify(publicKey)
            signature.update(message.toByteArray())
            return signature.verify(signatureBytes.toByteArray())
        }

    }
```

4.2 База данных

Для реализации базы данных был написан файл Database.kt

```
package com.example.diploma

import android.content.Context
import androidx.lifecycle.LiveData
import androidx.room.*
import java.math.BigInteger


@Database(entities = [Record::class], version = 1)
abstract class mroomDatabase : RoomDatabase() {
    abstract fun dbDao(): DbDao

    companion object {
        @Volatile
        private var INSTANCE: mroomDatabase? = null
        fun getDatabase(context: Context): mroomDatabase {
            return INSTANCE ?: synchronized(this) {
                val instance = Room.databaseBuilder(
                    context.applicationContext,
                    mroomDatabase::class.java,
                    "Records"
                ).build()
                INSTANCE = instance
                instance
            }
        }
    }
}

@Entity(tableName = "Records")
data class Record(
    @PrimaryKey(autoGenerate = true)
    val id: Int = 0,
    val type: Int?,
    val approval: Int = 0,
    val sender: String,
    val message: String,
    val sign: String,
    val timestamp: Long
) {
    @Ignore
    constructor(approval: Int, sender: String?, message: String, sign: String,
        timestamp: Long) : this(0, 0, approval, sender ?: "", message, sign,
        timestamp)
}


@Dao
```

```
47 interface DbDao {
48     //@Query("SELECT * FROM Records")
49     @Query("SELECT * FROM Records ORDER BY timestamp ASC")
50     fun getAllMessages(): List<Record>
51
52     @Query("SELECT * FROM Records WHERE sender LIKE :sender")
53     fun getRecordsBySender(sender: String): List<Record>
54
55     @Query("SELECT COUNT(*) FROM Records WHERE sign = :sign")
56     fun hasRecord(sign : String): LiveData<Boolean>
57
58     @Insert(onConflict = OnConflictStrategy.REPLACE)
59     fun insertMessage(message: Record)
60
61     @Delete
62     fun deleteMessage(message: Record)
63
64
65 }
```

В данном файле описана основная структура сообщения, которой является класс Record. Так же описанны конструкторы для этого класса и иструмент взаимодействия с базой данных DbDao.

4.3 Инетрфейс

Интерфейс приложения состоит из двух экранов главного меню(Рис 1) и меню просмотра базы данных (Рис 2).

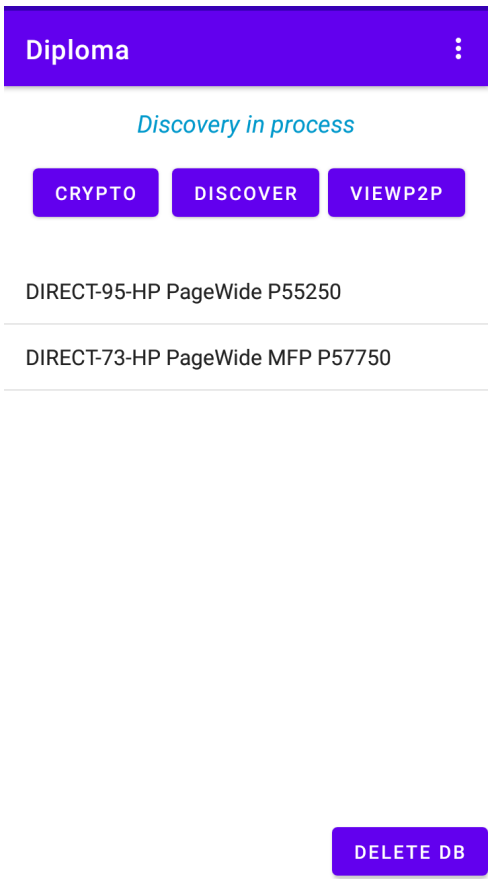

Рис. 1: Главное меню

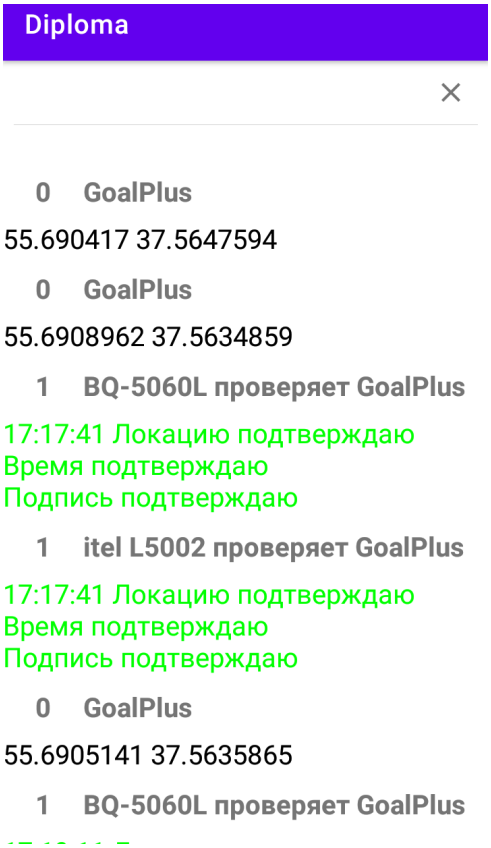

Рис. 2: Меню просмотра БД

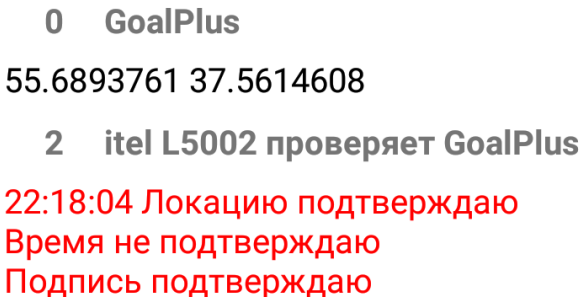

Рис. 3: Негативный результат аутентификации абонента

На экране главного меню мы можем наблюдать 4 кнопки и список доступных устройств в Wifi Direct. Кнопка "DISCORVER"отвечает за обновление или включение поиска устройств в сети. Кнопка "Crypto"отвечает за генерацию пары ключей DSA. Кнопка "VIEWP2P"отвечает за переход к экрану просмотра базы. Кнопка "DELETE DB"за очищение базы данных.

В базе данных (Рис. 2) можно наблюдать два типа записей: записи синхронизации и записи аутентификации. Запись синхронизации состоит из имени устройтсва абонентка и его координат. Запись аутентификации состоит из имён устройств проверяющего и проверяемого, записи с временной меткой, в которой говорится о поддтверждennии или опроверженнии тех или инных факторов при аутентификации. Запись с негативным результатом аутентификации представленна на рис. 3.

В экране просмотра базы данных сверху реализованна поисковая строка. Если вбить в неё имя устройтсва абонента, будет выведенна база данных всех его записей синхронизации и всех негативных аутентификационных записей о нем.

Таким образом, абоненты смогут составить мнение о надежности любого абонента и решить, доверять дальше конкретному абоненту или нет. В случае утраты доверия абонента можно отключить от сети WifiDirect, и абоненту придется восстанавливать доверие, если ему будет нужно войти в нее снова.

5 Заключение

В данной работе было программно реализованно приложение для создания децентрализованной базы данных с нативным интерфейсом. Так же был имплементирован протокол аутентификации пользователей через проверку сообщений. Не смотря на то что для подключенния пользователей используется Клиент/Сервер-ная архитекутра, пользователи могут в любой момент сменить центрального пользователя. Что делает базу децентрализованной, так как никто не может контролировать кто станет сервером при подключении, и пользователя который будет выступать в роли сервера, можно в любой момент отключить или заменить кем-то другим.

Список литературы

[1] *Черепнев М.А.*Криптографические протоколы М.: ООО "МАКС Пресс 2018. – 125 с.

[2] *Черепнев М.А.*Децентрализованная схема защищенного создания и хранения баз данных. International Journal of Open Information Technologies, Лаборатория Открытых Информационных Технологий факультета ВМК МГУ им. М.В. Ломоносова, 2020