**CHAPTER FIVE**

**CONCLUSION AND RECOMMENDATIONS**

**5.1 Summary**

The aim of this final year project work was to develop a concept based on best in practice methods that could be implemented on android platform to allow usage of IM via intranet connection in a secure way using encryption techniques.

This was achieved by first identifying the challenges and problems faced by Instant Messaging applications and the current security technologies in them and accordingly find the best-in-practice solutions. Using the research done, a concept was worked out which matched the requirements of authenticity, confidentiality and trustworthiness all achieved using the local server intranet connectivity, without depending on the internet. It was designed to be simple for use. The feasibility of this concept was then demonstrated by implementing it and thus presenting a working prototype.

Although there is nothing like 100% security, the concept was successfully developed based on a proven best-practice and successfully implemented on to of a real IM shows that it is possible to use the developed concept to put security and privacy on the next level.

**5.2 Conclusion**

This final year project presents an idea to develop a IM service for the intranet users, this service will be deployed on the intranet server of any organization that allows smartphone and tablet users to send and receive messages within an organization at free of cost. This communication does not need to interact with mobile service provider or no need to take any data plan. Internet connectivity is also not required. So this way it reduces the cost of communication, increases the security of the communication between various devices which gives compatibility with the Personal Computers with the help of local host based server which provides an interface between the user and the personal computers or tablets to provide an efficient communication by increasing its performance. It can be downloaded free of cost, so it is economical also.

**5.3 Recommendation and Future Work**

In order to design very effective secure system different factors should be considered. Although the goal of this project work was merely to develop an intranet based secure messages between two clients, there are some other factors that might be considered for future research such as usability of the application or scalability of the servers or how the design effects the cost of implementation. In order to complete this research the following topics are suggested for future researchers:

1. **Performance**: the proposed design has not been analyzed to investigate how it effects the performance of the mobile phone. How much it will consume CPU power and how it affects battery drain. Different security algorithms or encryption ciphers with back and forth communication with local host server to check if there are new messages might need more computational power and support from the underlying operating system.
2. To avoid **reverse engineering** attacks, it is always a great practice to automatically clearing all cache data with any change in the IM application activity. For example, when the application goes to the background. This means to avoid many mistakes in the implementation and coding of the application.
3. Implementation of the “**group chat**” feature might be considered as well. In group chat, more than two parties can join and chat with each other. Implementation of a secure group chat on intranet server is more complicated, mostly when the chat application has capability of offline messages and needs more considerations in the design phase of the architecture which could be a topic of a new research.

The app developed though rich in in its features can be enhanced in a number of ways. Some of them include extending the functionality to include multimedia messages and location messages. In addition to this functionality of encryption can be extended to group chats as well.

Features can also include the improvement of the way user contacts are shared to make them more "private" and also add a stronger way of verifying users, maybe using fingerprints. Also a method in which communication with the key server can be made more secure than just using SSL may be a possible extension to this application. In addition to this a further extension can be made to port the encryption features to the web app local host server, making the mobile app for other platforms like iOS and desktop versions.

**APPENDICES**

**Mobile App Code**

class ChatActivity : AppCompatActivity() {

val TAG = "ChatActivity"

var adapter : ChatListAdapter? = null

var appPreference : AppPreferences? = null

var friends : ArrayList<User>? = null

var user : User? = null

var db : DbHelper? = null

var appPreferences : AppPreferences? = null

private val mMessageReceiver = object : BroadcastReceiver() {

override fun onReceive(context: Context, intent: Intent) {

// Get extra data included in the Intent

val message = intent.getIntExtra("status",2)

if(message == 1 && adapter != null){

if(intent.hasExtra("friends")){

friends = ModelConverter.GsonToClass<ArrayList<User>>(intent.getStringExtra("friends"))

}

adapterWork()

}

//Log.e(TAG, message.toString())

//Log.e(TAG, friends.toString())

}

}

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_chat)

setSupportActionBar(toolbar)

db = DbHelper(this).open()

appPreference = AppPreferences(this)

toolbar.title = "Chat List"

val lv = findViewById<ListView>(R.id.chat\_list\_\_view)

appPreference = AppPreferences(this)

user = ModelConverter.GsonToClass<User>(appPreference!!.getUser()!!)

LocalBroadcastManager.getInstance(this).registerReceiver(

mMessageReceiver, IntentFilter("MessageUpdate"))

friends = ModelConverter.GsonToClass<ArrayList<User>>(appPreference?.getFriends())

if(friends == null) {

friends = ArrayList()

}

getLastMessage()

adapter = ChatListAdapter(this, friends!!, user!!)

updateView()

lv.adapter = adapter

lv.setOnItemClickListener { \_, \_, position, \_ ->

var selected = lv.getItemAtPosition(position) as User

startActivity(Intent(this, ViewActivity::class.java).apply {

putExtra("friend", Gson().toJson(selected))

})

}

}

fun adapterWork(){

runOnUiThread {

getLastMessage()

updateView()

adapter!!.swapItem(friends!!)

}

}

override fun onResume() {

adapterWork()

super.onResume()

}

private fun getLastMessage() {

friends?.let {

friends!!.forEach { fr ->

fr.last\_message = Message.messageSingle(db!!.getLastMessage(fr.id!!, user!!.id!!))

}

sortByTime(it)

Log.e(TAG, it.toString())

}

}

private fun sortByTime(list: ArrayList<User>) {

if(list.size > 1){

Collections.sort(list, object : Comparator<User> {

internal var df = SimpleDateFormat("yyyy-MM-dd HH:mm", Locale.ENGLISH)

override fun compare(lhs: User, rhs: User): Int {

try {

df.timeZone = TimeZone.getDefault()

if (lhs.last\_message!!.created\_at == null && rhs.last\_message!!.created\_at == null) {

return 1

}

if (lhs.last\_message!!.created\_at == null && rhs.last\_message!!.created\_at != null) {

return 1

}

if (rhs.last\_message!!.created\_at == null && lhs.last\_message!!.created\_at != null) {

return -1

}

val a = df.parse(lhs.last\_message!!.created\_at)

val b = df.parse(rhs.last\_message!!.created\_at)

Log.e(TAG, "Result Of Comparison.....sortChatList, Line 328")

out.println(-a.compareTo(b))

return -a.compareTo(b)

} catch (e: Exception) {

e.printStackTrace()

return 0

}

}

})

friends = list

Log.e(TAG, list.toString())

}

}

private fun updateView(){

if(friends!!.size == 0){

chat\_list\_\_view.visibility = View.GONE

chat\_frag\_tv.visibility = View.VISIBLE

}else{

chat\_list\_\_view.visibility = View.VISIBLE

chat\_frag\_tv.visibility = View.GONE

}

}

override fun onCreateOptionsMenu(menu: Menu?): Boolean {

menuInflater.inflate(R.menu.chat\_menu,menu)

return super.onCreateOptionsMenu(menu)

}

fun convertTime(time: Long) :String{

val date = Date(time)

val sdf = SimpleDateFormat("yyyy-MM-dd HH:mm")

sdf.timeZone = TimeZone.getTimeZone("GMT+1")

return sdf.format(date)

}

override fun onOptionsItemSelected(item: MenuItem?): Boolean {

when(item!!.itemId){

R.id.search -> {

showSearchDialog()

}

R.id.sign\_out -> {

db!!.truncateMessage()

appPreference!!.clear()

stopService(Intent(this, BackgroundService::class.java))

startActivity(Intent(this,SplashActivity::class.java))

finish()

}

}

return super.onOptionsItemSelected(item)

}

private fun showSearchDialog() {

val dialog = AlertDialog.Builder(this).create()

val v = layoutInflater.inflate(R.layout.layout\_search, null)

val search\_text = v.findViewById<EditText>(R.id.search\_text)

val search\_btn = v.findViewById<Button>(R.id.search\_btn)

val search\_no = v.findViewById<TextView>(R.id.search\_no)

val search\_list = v.findViewById<ListView>(R.id.search\_lv)

search\_list.setOnItemClickListener { \_, \_, position, \_ ->

val temp = search\_list.getItemAtPosition(position) as User

friends!!.add(temp)

friends?.let ll@{

friends?.forEach {fd->

if(fd.id == temp.id){

return@ll

}

}

appPreference!!.setFriends(friends!!)

}

dialog.dismiss()

adapterWork()

startActivity(Intent(this, ViewActivity::class.java).apply {

putExtra("friend", Gson().toJson(temp))

})

}

search\_btn.setOnClickListener {

if(search\_text.text.isNotBlank()){

var pd = ProgressDialog(this)

pd.setMessage("Searching...")

pd.setCanceledOnTouchOutside(false)

pd.show()

val retrofit = RetrofitClient(this, RetrofitClient.Defaulthost)

retrofit.apiService!!.searchResult(search\_text.text.toString())

.subscribeOn(Schedulers.newThread())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(object : Subscriber<Response<User>>() {

override fun onCompleted() {

Log.e(TAG, "completed")

}

override fun onError(throwable: Throwable) {

pd.dismiss()

search\_text.error ="An Error Occurred."

Log.e(TAG, "new friends onError: +" + throwable.toString())

}

override fun onNext(response: Response<User>) {

if(response.status != 1){

search\_text.error = response.message

search\_no.text = response.message

Log.e(TAG, "could not get friends")

}else{

val temp = response.data

if(temp!!.size == 0){

search\_text.error = "User not found"

}else{

temp.remove(user)

for(i in 0 until temp.size-1){

if(temp[i].id == user!!.id){

temp.removeAt(i)

}

}

val adapter = SearchAdapter(this@ChatActivity, temp)

search\_list.adapter = adapter

}

}

pd.dismiss()

}

})

}else{

search\_text.error = "Field Can not be empty"

}

}

dialog.setView(v)

dialog.setTitle("User Search")

dialog.setCancelable(true)

dialog.show()

}

override fun onBackPressed() {

stopService(Intent(this, BackgroundService::class.java))

super.onBackPressed()

}

class ViewActivity : AppCompatActivity() {

val TAG = "ViewActivity"

var adapter : ViewChatAdapter? = null

var appPreference : AppPreferences? = null

var friends : ArrayList<User>? = null

var user : User? = null

var db : DbHelper? = null

var retrofitClient : RetrofitClient? = null

private val mMessageReceiver = object : BroadcastReceiver() {

override fun onReceive(context: Context, intent: Intent) {

// Get extra data included in the Intent

val message = intent.getIntExtra("status",2)

if(message == 1 && adapter != null){

if(intent.hasExtra("friends")){

friends = ModelConverter.GsonToClass<ArrayList<User>>(intent.getStringExtra("friends"))

friends?.forEach {

if(it.id == friend!!.id){

messages!!.add(it.last\_message!!)

}

}

}

refresh()

//adapter!!.swapItem(friends!!)

}

Log.e(TAG, message.toString())

Log.e(TAG, friends.toString())

}

}

var friend : User? = null

var messages : ArrayList<Message>? = null

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_view)

setSupportActionBar(toolbar)

supportActionBar?.setDisplayHomeAsUpEnabled(true)

appPreference = AppPreferences(this)

db = DbHelper(this).open()

user = ModelConverter.GsonToClass<User>(appPreference!!.getUser())

friend = ModelConverter.GsonToClass<User>(intent.getStringExtra("friend"))

toolbar.title = friend!!.username

LocalBroadcastManager.getInstance(this).registerReceiver(

mMessageReceiver, IntentFilter("MessageUpdate"))

messages = Message.messageList(db!!.getChatHistory(friend!!.id!!, user!!.id!!))

if(messages == null)

messages = ArrayList()

adapter = ViewChatAdapter(this,messages!!,user!!, friend!!)

val lv = findViewById<ListView>(R.id.rv\_chat)

lv.transcriptMode = AbsListView.TRANSCRIPT\_MODE\_ALWAYS\_SCROLL

lv.isStackFromBottom = true

lv.adapter = adapter

adapter!!.registerDataSetObserver(object : DataSetObserver() {

override fun onChanged() {

super.onChanged()

lv.setSelection(adapter!!.count - 1)

}

})

Log.e(TAG, messages.toString())

btn\_send.setOnClickListener {

if(et\_chat.text.isNotEmpty()){

var msg = Message()

msg.from = user!!.id

msg.to = friend!!.id

msg.message = et\_chat.text.toString()

msg.mid = Random().nextInt(99999999)

msg.status = 0

msg.created\_at = getDate()

db!!.addMessage(msg)

msg = Message.messageSingle( db!!.getMessageByMid(msg.mid!!))!!

messages!!.add(msg)

sendMessage(msg)

adapter!!.notifyDataSetChanged()

et\_chat.text.clear()

Log.e(TAG, msg.toString())

}

}

Thread(Runnable {

updateStatus()

}).start()

}

fun refresh(){

messages = Message.messageList(db!!.getChatHistory(friend!!.id!!, user!!.id!!))

runOnUiThread {

adapter!!.swapItem(messages!!)

}

}

private fun updateStatus() {

var mids = ""

if(messages != null){

messages?.forEach {

if((it.from != user!!.id) && (it.status!! < 3))

if(it.status!! == 0){

sendMessage(it)

}else{

if(mids.isNotEmpty())

mids += ",${it.mid}"

else

mids = "${it.mid}"

}

}

if(mids != ""){

Log.e(TAG, "mids=$mids")

updateStatusApi(mids)

}

}

}

fun getDate() : String{

val date = Date()

val sdf = SimpleDateFormat("yyyy-MM-dd HH:mm")

sdf.timeZone = TimeZone.getTimeZone("GMT+1")

return sdf.format(date)

}

private fun sendMessage(msg : Message) {

Log.e(TAG, "Send Message")

retrofitClient = RetrofitClient(this,RetrofitClient.Defaulthost)

retrofitClient!!.apiService!!.sendMessage(msg.mid!!,msg.from!!, msg.to!!, msg.message!!)

.subscribeOn(Schedulers.newThread())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(object : Subscriber<Response<Message>>() {

override fun onCompleted() {

Log.e(TAG, "completed")

}

override fun onError(throwable: Throwable) {

Log.e(TAG, "new friends onError: +" + throwable.toString())

}

override fun onNext(response: Response<Message>) {

if(response.status != 1){

Log.e(TAG, "Message Could Not Be Sent")

}else{

db!!.updateStatus(msg.mid!!, 1)

for (i in messages!!.size-1..0){

if(messages!![i].mid == msg.mid){

messages!![i] = msg

//db!!.addMessage(msg)

Log.e(TAG, "replaced message after send")

break

}

}

adapter!!.notifyDataSetChanged()

Log.e(TAG,response.toString())

}

}

})

}

private fun updateStatusApi(mid : String){

Log.e(TAG, "Get New Friends")

retrofitClient = RetrofitClient(this, RetrofitClient.Defaulthost)

retrofitClient!!.apiService!!.setMessageStatus(mid, 3)

.subscribeOn(Schedulers.newThread())

.observeOn(AndroidSchedulers.mainThread())

.subscribe(object : Subscriber<Response<Message>>() {

override fun onCompleted() {

Log.e(TAG, "completed")

}

override fun onError(throwable: Throwable) {

Log.e(TAG, "Status onError: +" + throwable.toString())

}

override fun onNext(response: Response<Message>) {

if(response.status != 1){

Log.e(TAG, "Message Not Found")

}else{

response.data!!.forEach {

db!!.updateStatus(it.mid!!,3)

}

Log.e(TAG,response.toString())

}

}

})

}

**Encryption Code**

public static byte[] encrypt(byte[] data, byte[] publicKey) {

try { boolean failure = true;

IESEngine engine = new IESEngine(new ECDHBasicAgreement(), new KDF2BytesGenerator(new SHA1Digest()), new HMac(new SHA1Digest()));

ECPoint publicPoint = ecparms.getCurve().decodePoint(publicKey);

ECPublicKeyParameters pubKey = new ECPublicKeyParameters(publicPoint, ecparms);

ECPrivateKeyParameters k = null;

ECPublicKeyParameters R = null; ECPoint Z = null; initPRNG(); while (failure) {

BigInteger privNum = new BigInteger(PRNG.generateSeed(NONCE\_SIZE)); // select k k = new ECPrivateKeyParameters(privNum, ecparms); // this is just a container

R = generatePublicKey(k); // R = kP

// calculate Z=hkQ

BigInteger z = new BigInteger(ecparms.getH().toByteArray()); // z=h z = z.multiply(privNum); // z=hk Z = pubKey.getQ(); // Z=Q if (Z instanceof ECPoint.F2m) {

ECPoint.F2m Z2 = new ECPoint.F2m(Z.getCurve(), Z.getX(), Z.getY()); // clone Z = Z2.multiply(z); // Z=zQ <=> Z=hkQ

}

if (Z instanceof ECPoint.Fp) {

ECPoint.Fp Z2 = new ECPoint.Fp(Z.getCurve(), Z.getX(), Z.getY()); // clone

Z = Z2.multiply(z); // Z=zQ <=> Z=hkQ

}

if (!Z.isInfinity()) { failure = false; // see step 2

} }

// step 3 and 4:

// setup KDF(xz,R) - see new IESParameters(...)

// pass the parameters for step4 engine.init(true, (CipherParameters) k, (CipherParameters) pubKey, new IESParameters(Z.getX().toBigInteger().toByteArray(), R.getQ().getEncoded(), 64));

// step 4 generate C and t byte[] cAndT = engine.processBlock(data, 0, data.length);

// collect the result in format

// byte[0] = length of R (encoded)

// R in encoded form

// C and t as coming from the engine byte[] publicBytes = R.getQ().getEncoded(); byte[] out = new byte[1 + publicBytes.length + cAndT.length]; out[0] = (byte) publicBytes.length; // WARN this will crash if

// key.length > 255

System.arraycopy(publicBytes, 0, out, 1, publicBytes.length);

System.arraycopy(cAndT, 0, out, 1 + publicBytes.length, cAndT.length); out1 = new byte[1 + publicBytes.length + cAndT.length];

out11 = out; pr = publicKey; return out;

} catch (Exception excep) { return null;

} catch (Throwable t) { return null; } }

**Decryption Code**

public static byte[] decrypt(byte[] data, AsymmetricCipherKeyPair pair) throws InvalidCipherTextException {

IESEngine engine = new IESEngine(new ECDHBasicAgreement(), new KDF2BytesGenerator(new SHA1Digest()), new HMac( new SHA1Digest()));

ECPrivateKeyParameters p = (ECPrivateKeyParameters) pair.getPrivate();

// deserialize data into (R,C,t) int lengthOfR = data[0]; if (lengthOfR > data.length) { throw new InvalidCipherTextException("Lengthfield invalid");

}

byte[] RasBytes = new byte[lengthOfR];

System.arraycopy(data, 1, RasBytes, 0, lengthOfR);

ECPoint RPoint = ecparms.getCurve().decodePoint(RasBytes); // decode into R

ECPublicKeyParameters R = new ECPublicKeyParameters(RPoint, ecparms); // this is just a container byte[] cAndT = new byte[data.length - lengthOfR - 1];

System.arraycopy(data, 1 + lengthOfR, cAndT, 0, data.length - lengthOfR- 1);

// compute Z=hdR (step 2)

BigInteger z = new BigInteger(ecparms.getH().toByteArray()); // z=h z = z.multiply(p.getD()); // z=hd ECPoint Z = R.getQ(); // Z=Q if (Z instanceof ECPoint.F2m) {

ECPoint.F2m Z2 = new ECPoint.F2m(Z.getCurve(), Z.getX(), Z.getY()); // clone Z = Z2.multiply(z); // Z=zQ <=> Z=hdQ

}

if (Z instanceof ECPoint.Fp) {

ECPoint.Fp Z2 = new ECPoint.Fp(Z.getCurve(), Z.getX(), Z.getY()); // clone Z = Z2.multiply(z); // Z=zQ <=> Z=hdQ

}

if (Z.isInfinity()) { throw new InvalidCipherTextException("Z is infinite"); // reject if Z is infinite (see step 2)

}

// step 3

// setup KDF(xz,R) - see new IESParameters(...)

// pass the parameters for step4 engine.init(false, (CipherParameters) p, (CipherParameters) R, new IESParameters(Z.getX().toBigInteger().toByteArray(), R.getQ().getEncoded(), 64));

// step 4 and 5 byte[] clear = engine.processBlock(cAndT, 0, cAndT.length);

return clear;

}

**Blowfish Code**

public Encryptor( byte[] key ){ cipher = new PaddedBlockCipher( new CBCBlockCipher( new BlowfishEngine() ) ); this.key = new KeyParameter( key );

}

// Initialize the cryptographic engine. // The string should be at least 8 chars long. public Encryptor( String key ){ this( key.getBytes() );

}

// Private routine that does the gritty work. private byte[] callCipher( byte[] data ) throws CryptoException { int size = cipher.getOutputSize( data.length ); byte[] result = new byte[ size ]; int olen = cipher.processBytes( data, 0,data.length, result, 0 ); olen += cipher.doFinal( result, olen ); if( olen < size ){ byte[] tmp = new byte[ olen ]; System.arraycopy( result, 0, tmp, 0, olen ); result = tmp;

}

return result;

}

// Encrypt arbitrary byte array, returning the // encrypted data in a different byte array.

public synchronized byte[] encrypt( byte[] data ) throws CryptoException { if( data == null || data.length == 0 ){ return new byte[0];

}

cipher.init( true, key );

System.out.print("encrypted byte:"+callCipher( data ));

System.out.print("encrypted String:"+new String(callCipher( data ))); return callCipher( data );

}

// Encrypts a string.

public byte[] encryptString( String data ) throws CryptoException { if( data == null || data.length() == 0 ){ return new byte[0];

}

return encrypt( data.getBytes() );

}

// Decrypts arbitrary data.

public synchronized byte[] decrypt( byte[] data ) throws CryptoException { if( data == null || data.length == 0 ){ return new byte[0];

}

cipher.init( false, key ); return callCipher( data );

}

// Decrypts a string that was previously encoded

// using encryptString. public String decryptString( byte[] data ) throws CryptoException { if( data == null || data.length == 0 ){ return "";

}

return new String( decrypt( data ) );

}