Praneeth Manubolu
MSCS 630

Android Jabber Application
Project Milestone

Abstract –

This paper explains the implementation of Android based chat application, methodology and technologies. To secure the information, AES has been implemented where the AES keys are initially encrypted using RSA. The server used for connecting users only shares the encrypted information there by making it impossible to decrypt even by the provider. XML is used for exchanging information where only sensitive data like actual message is encrypted. For verifying or validation SHA-2(256 bits) has been used.

Although this project is aimed to implement a secure chat application, the entire system (client & server) is designed to look and work like commercial chat application like WhatsApp, messenger etc.

Introduction –

In the present world, people communicate with others through messages more, than by voice or video. The challenge lies in not only securely transmitting them but to implement an efficient encryption system as millions of messages fly around the globe at any given second. It comes to no surprise that we already live in the most secured era where an attack on a well encrypted data takes more time than the formation of universe. The project takes inspiration from this, implementing latest encryption standards and techniques.

Background –

One of the highly used crypto system for instant messaging is OTR (off-the-record messaging) which is a combination of AES symmetric key algorithm of 128 bits key length, the Diffie–Hellman key exchange with 1536 bits group size, and the SHA-1 hash function. This project uses a similar approach to OTR. E2EE (End-to-end encryption) is another kind of crypto system where the even the provider cannot decrypt the data. This is a true implementation of security where even the provider cannot decrypt messages which makes government meddling or cyber threats impossible. E2EE has been implemented in WhatsApp and has been gaining popularity extensively.

Methodology –

To create a user-friendly application, where the user can also see statistics of the encryption and decryption. Using OTR as reference, the application is programmed to generate RSA and AES keys on initializing, on the client app. The RSA public key and AES encrypted keys are shared during the handshake process between users. It was required to understand how AES works as encrypting messages was causing issues due to varied lengths. Padding was required to overcome this problem.

After the handshake is successful and the AES key is achieved at both sides the real magic happens. Every time a message is sent it is encrypted using the AES key. Along with this the message is hashed and then encrypted using the same key. The results are shown in '**Project demo**' section with pics.

At the receiver end the message is decrypted then hashed. It is then compared to the decrypted hash message received. If this matches then and only the message is displayed. The hash acts as a signature in this case.

Due to the dynamic nature of the system, generating AES key each time the application tries to connect to new user makes the system highly secured.

Experiments –

Initially two applications were created one for just sending/receiving message and other to check encryption and decryption. It could encrypt and decrypt text file successfully using AES keys of size 128, 196 and 256. Also, used only RSA for encrypting/decrypting messages to understand why RSA alone cannot be used. From experimenting RSA showed sever performance issues where it took 17 seconds to encrypt a piece of data, the same data took 294 milliseconds if AES was used.

Discussion –

Experiments proved that AES is much faster than RSA. An important observation is that, while comparing encryption/decryption with AES and RSA it was seen that AES takes more time to encrypt than to decrypt with a ratio of 6:1(E:D - where E is time taken for encryption and D is the time taken for decryption). In case of RSA, Decrypting took more time than encrypting the same data with a ratio of 1:8(E:D).

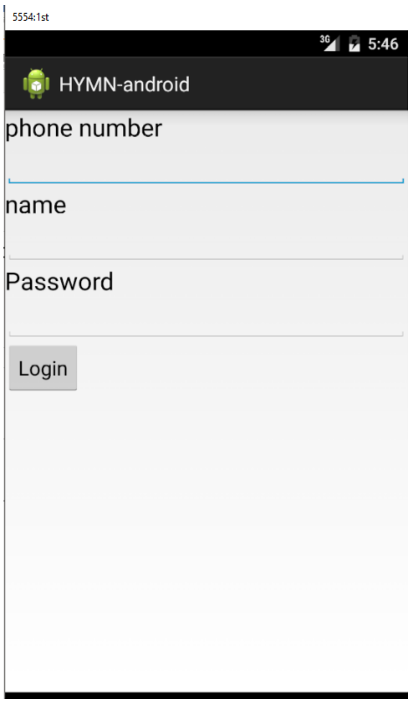
Exchanging RSA Public keys over the internet might be tricky. In case of this project, the modulus and exponent are extracted from the generated public key. These are then shared to the desired devices. Initially the RSA key was encoded with base 64 and then shared but this didn't work as reconstructing the key resulted a different key. The reason this happened is explained later in challenges section. The drawback of the existing way of sharing keys is that the modulus and exponent which is basically a 'int' type is converted to string to transmit over internet. The size of this string is substantial and has reduced the efficiency.

I have written XML parser to read the incoming messages. The values like encrypted message, type of message is found from keys in the xml. Although this works, the amount of time taken to put the entire thing into xml and parse it at the receiver at didn't feel efficient. Also, the entire XML can be encrypted and then placed in one XML tag, then at the receiver end it can be decrypted to the original XML which has keys like – message, hash value, to, from can be generated this could make the transmission line more secure making the sender/destination information hidden. This approach is time crunching but could be used where security is crucial.

Project Demo –

- RSA Sharing XML –

Below XML is sent to server to save 'cornado' public key in server session on login. Same way all android clients save their public key in session.



```
    {"RSA":"29912073352436273164856694569476174287163747531625381819564779
91518774055099784726007040520825172359960353251089912159498055026261337512
966454578649418837868520608570861734108823889058709148315800177294853782551
0672442534447019309237829480134254001031399855923746041780636504627409794856
24983462151492453115436589449697204094027368085290049381385928873240158109
87792262910174351517275437403482308914545239356773250203467091273383406852
841558954801432290419924007975078680084440012936683442833821419337043945902
52773451118332948248343747314907191764784664888936568235384721375979692541604
6014624954115965497901300693l7","user":"praneeth","type":"validate","expon
ent":"65537","Pass":"praneeth","PH":"8454535043"}
```

- Received message (by all devices)-

A list of all the users and their public key is sent to all the users in that network. This information is used to reconstruct rsa public keys for all users

```
    {"type":"UserList","list":[{"mod":"29912073352436273164856694569476l7
428716374753162538181956477991518774055099784726007040520825172359960353251
0899121594980550262613375129664545786494188378685206085708617341088238890588
7091483158001772948537825510672442534447019309237829480134254001031399855592
3746041780636504627409794856249834621514924531154365894496972040940273680885
2900493813859288732401581098779226291017435151727543740348230891454523935667
7325020346709127338340685284155895480143229041992400797507868008444001293666
834428338214193370439459025277345111833294824834374731490719176478466488893
65682353847213759796925416046014624954115965497901300693l7","name":"praneet
h","exponent":"65537"},{"mod":"23526251606944093228350705916579054174903559
8603696702165751769209480717381165399521208198941322479547458020069632062l0
46198835540224530385863669582876814922386984986063317040795965986989129986 3
63973426135555283538283762069969244990843333278454673987863648102203575590 7
41966474703046053885033262161056571329832603571761499123902976740471681543 8
57702811062788219401065299432411334102693294429315720047839709911568652402 1
09436884541342578387739389401071026767825126513264015625551609410485908424 9
46872609347415430256261576261585090266444935595390671510161673025541925520 3
8890497213992560746014063889736988795999989l029l","name":"cornado","exponen
t":"65537"}]}
```

- Handshake –

The first user to initiate handshake will generate AES key and send a message to the other user the encrypted AES key. The sent message is as below

```
    {"AES":"0bb243eb6433bafe3c8821f735e205587bb3ae8ebcd31c825c1c8bdd4e134
6d941e7d78d0a6a5fa627fa9ed0ec45848644e4504df714a593891fc403c43d362b79aa1ab2
5c026ede7c75a4d6d959b697596465e169aeaa5f16c4dde0fa94ace5074a137ed7cbd1cf379
29980af86a05d81b21efc44444c4033fabd032d2277698eeba139c869d83b64bc9e99775b59
abe694de798a455b87b2194ce9088e898e2a9c34644fab44a519aaeab081235ef1459ce27d2
42c193eed4340b49c2dac5ae857f0d53e75bd2137c0420ff38440ac901ab6eea46b4852bccf
```

6ba4f0e03ac31b6cd01f5a4d7c41933388864f549a609eb8fb1da31a2d55e87ea0d2d62454e2","to":"praneeth","from":"cornado","type":"Secure"}

- Message sharing –

From then on the message are transmitted as shown below

{"text":"YnIpZWGmZXs4QoQf1D8RFg==","to":"cornado","from":"praneeth","hash":"Jhao6k71sA3o8fvZsQBHYP\/zHr6f+tNi45YsbrBoEZICPzRnUa5tjn2mADaoHfsTLW6kgQVVV\/jguJ\/TcRLC4w==","type":"message"}

Conclusion –

It is proven that AES is much faster than RSA and can be used for securing any type of data. A major problem still existing is the sharing of the encryption keys. RSA still plays a dominant role in this field due its unique implementation of using public and private keys. Where private key is kept secret, and is used for decryption. Using these two encryption systems together increases security to the highest level. SHA-2 is used in aid of the above system to verify the data which acts as a digital signature.

It has taken more time to debug than to code, but this ambitious feat has been accomplished. This project has not only made me understand end to end encryption but also other things like - ways of sharing keys, which point to implement security, security leaks etc. which wouldn’t be possible if I haven’t worked on a project.

References –

1. Jiang Bian, Remzi Seker and Umit Topaloglu “Off-the-Record Instant Messaging for Group Conversation”. *Information Reuse and Integration*. IEEE International Conference on Information Reuse and Integration 2007.

2. Alessandro Barenghi, Michele Beretta, Alessandro Di Federico, Gerardo Pelosi “An End-to-End Encrypted Online Social Network”. 2014 IEEE Intl Conf on High *Performance Computing and Communications*, 2014 IEEE 6th Intl Symp on *Cyberspace Safety and Security*, 2014 IEEE 11th Intl Conf on *Embedded Software and Syst* (HPCC, CSS, ICESS).