PROJECT REPORT

⇒ Description of System: The VoipSec system is a secured VoIP application. The system consists of a third party SFL phone which we have used as a part of our project. We included our own SIP proxy which is used to securely transmit the packets over the network. SIP proxy consists of packet transfer among two clients which is encrypted. When the user enters the secure mode, our application runs the Python Voipsec Application components viz, UDP proxy, SSL receiver and RTP listener.

⇒ Python VoipSec Application:
  - UDP proxy receives the SIP packets from the local client and passes them over a secure tunnel.
  - SSL receiver receives SSL packets from UDP proxy and passes them over to local SIP client.
  - RTP listener sends and receives RTP media stream packets using AES encryption over UDP.

⇒ Working: Python Voipsec runs on client as Client Proxy and on Server as Server-Proxy.
  - Instead of connecting to the actual SIP server, the SIP client connects to the Client - Proxy which forms a secure tunnel using TLS between the Client Proxy and the Server Proxy.
  - On the server proxy side, the server listens for connections from client proxy to establish the TLS tunnel.
  - Once the tunnel is established, the client-proxy and server-proxy rewrite the information contained in the SIP packets so that communication occurs between the client and proxy-server instead of directly between the client and SIP server themselves.
  - Essentially, the Proxy on the client and server are acting like a man-in-the-middle attack except they are not attacking the security of the system; they are enhancing it.
  - At this point, the SIP conversation occurs as normal with the client and server proxy translating port information, encrypting, and decrypting the messages.
  - Since the SIP server relies on the information in the SIP packet headers, the secure tunnels are transparent to the Bob's client, Alice's client, and the SIP server.
The only change in the standard SIP conversation is that Bob's client proxy will add a SIP header containing a random secret key to the 200 OK message when Bob answers Alice's call.

This random secret key will be extracted and remembered by Alice's proxy and used to encrypt the direct SIP messages (RTP) between Alice and Bob's SIP proxy going forward (i.e. the ACK).
⇒ **Modified SFL client:**
- We modified the SFL phone application in order to incorporate the secure proxy through the application. Through this modification, a check box was provided to the client in their account settings and checking the box would automatically trigger the python application and appropriate SIP client proxy configuration.

⇒ **Challenges:**

⇒ **Technical Complications**
- Figuring out how SIP clients (user agents) made decisions regarding where to route packets was a challenge since SIP packet headers had to be parsed and edited.
- Figuring out how to call Python code from the C++ VoipClient was also a challenge; we embedded python within a existing Linux SIP client, sfphone.
- Network Address Translation (NAT) / Firewall Issues blocked a lot of our work with both linux and android

⇒ **Indirection:**
- Originally, we started out under the perception that we were mandated to use Android; we faced an uphill battle as we went about building from scratch a minimal android client that would do sip calling. The problems we encountered here were numerous, as we learned android development from the bottom up.
- At one point, found an open and complete SIP client that worked with android; this required a complete rework of the existing android client we had started from scratch.
- Slightly into the rework, it was discovered we could use any platform atop which to build a SIP client.
- We decided to author an encryption as a proxy based solution that way it would be agnostic of the SIP client.

⇒ **Final direction:**
- Finally rested on doing a Python-based application with the above components using SSL tunnels and AES symmetric key cipher.
- Decided to use a Linux desktop SIP client (SFLPhone) coupled with reSIProcate SIP server. We would redirect the clients typical channel to receive data on through a local python application which would perform encryption. So long as each client had this proxy, the media stream could remain encrypted.

⇒ **What did I learn from the project?** I learnt python from this project. I also familiarized myself with the concepts of packet routing.
Security Services and Mechanisms used:
- Security Services: data confidentiality and authentication.
- Security Mechanisms: encryption on RTP for media stream. TLS encryption for SIP signaling.

Contribution of each team member:
- Major work of the project was done by Mark, he was the one who has helped in getting this project done. Mark developed the upd_proxy, ssl_receiver and the rtp_listener, which are the basic pillars of the project. Initially while working for Android app, Mark helped with server connection.
- Adam has helped with setting the secure proxy channel for transmission. He modified the SFL client b incorporating the check box which enabled the python application to run. He also worked on the Android application initially and developed contact list.
- Raghav served with Mark while developing the udp_proxy and ssl_receiver. He also worked for the major part in GUI of the android app initially.