Πανεπιστήμιο Πειραιά
« Τμήμα Ασφάλειας Ψηφιακών Συστημάτων»

Διπλωματική Εργασία

Ανάπτυξη εφαρμογής κρυπτογραφημένων τηλεφωνικών κλήσεων μέσω δικτύων IP για το σύστημα Android

Δημιουργός :
Αλέξανδρος -Ιάσονας Καμπάνης

Επιβλέπων:
Χριστόφορος Νταντογιάν

Ακαδημαϊκό έτος 2017-218

Πίνακας περιεχομένων

Ευρετήριο εικόνων

1. Περιγραφή του σκοπού της εφαρμογής

Στην σημερινή εποχή η πλατφόρμα του Android έχει επικρατήσει και η εξέλιξη των δικτύων και της επεξεργαστικής ισχύς δίνει δυνατότητα ανάπτυξης εφαρμογών που παλαιότερα δεν ήταν δυνατόν να αναπτυχθούν λόγω σχετικών περιορισμών. Ένα τέτοιο παράδειγμα είναι και η δυνατότητα τηλεφωνικών κλήσεων πάνω σε δίκτυα IP. Αυτή η δυνατότητα αποκτά περισσότερο ενδιαφέρον καθώς τα δίκτυα των παρόχων κινητής τηλεφωνίας θα εξελιχθούν ολοκληρωτικά σε δίκτυα IP στα επόμενα χρόνια, έτσι η ασφάλεια τέτοιων κλήσεων αποκτά μεγαλύτερο ενδιαφέρον και σημασία. Ο πρωταρχικός στόχος αυτής της εργασίας είναι η ανάπτυξη εφαρμογής η οποία θα παρέχει όσο το δυνατόν περισσότερη ασφάλεια πάνω στο δίκτυο προσφέροντας παράλληλα ικανοποιητική ποιότητα υπηρεσίας καθιστώντας το παραχθέν προϊόν λειτουργικό. Σκοπός είναι μία λύση όπου όσες λειτουργίες είναι δυνατόν θα πραγματοποιούνται peer to peer με χρήση ενός server μόνο για τα απολύτως απαραίτητα. Με βάση αυτές τις απαιτήσεις διαλέγουμε τα κατάλληλα εργαλεία και τεχνολογίες κατά την διάρκεια της υλοποίησης ώστε να ικανοποιηθούν αυτές οι απαιτήσεις. Δευτερεύον στόχος αυτής της εργασίας είναι να υποδείξει κατευθύνσεις προς τους χρήστες της εφαρμογής ώστε να δημιουργήσουν όσο πιο κατάλληλο και ασφαλές περιβάλλον στο οποίο θα λειτουργεί η εφαρμογή. Όταν αναφερόμαστε σε περιβάλλον αναφερόμαστε στην φυσική ασφάλεια της συσκευής του χρήστη αλλά κυρίως στο λογισμικό που υπάρχει στα smartphones εγκατεστημένο είτε από τους παρόχους τηλεπικοινωνιών είτε από τους κατασκευαστές. Η εμπειρία έχει δείξει ότι και οι δύο έχουν την δυνατότητα και πολλές φορές την εκμεταλλεύονται δημιουργώντας λογισμικό που προ εγκαθιστούν ή εγκαθιστούν μετά από αίτημα τρίτων το οποίο έχει πρόσβαση σε κάθε είδους προσωπικών δεδομένων και λειτουργιών της συσκευής. Η αναζήτηση και ανίχνευση τέτοιου λογισμικού μπορεί να εξελιχθεί σε διαδικασία μεγάλης πολυπλοκότητας καθιστώντας την έως και σχεδόν αδύνατη. Γενικά υπάρχουν δύο στρατηγικές που μπορεί κάποιος να ακολουθήσει. Η πρώτη είναι η απόπειρα ασφάλισης ενός υπάρχοντος συστήματος και η δεύτερη η δημιουργία ενός ασφαλούς περιβάλλοντος από τη βάση του. Πρέπει να σημειώσουμε σε αυτό το σημείο ότι η χρήση της εφαρμογής σε μη ασφαλές περιβάλλον την καθιστά έως και άχρηστη. Περισσότερα πάνω σε αυτό το θέμα στην συνέχεια της εργασίας. Η εργασία αυτή δεν θα ασχοληθεί με πρωτόκολλα υποδομής δικτύων VoIP όπως H.323 και SIP καθώς στόχος είναι μία αυτόνομη λύση καθώς και με τα βασικά ανάπτυξης εφαρμογών Android τα οποία θεωρούνται γνωστά.

2. Αρχιτεκτονική και ανάλυση των μερών της εφαρμογής

Ο στόχος της σχεδίασης της εφαρμογής είναι να προσφέρει όσο το δυνατό περισσότερο Peer to Peer λειτουργικότητα και να ελαχιστοποιεί την ανάγκη για έναν δυνατό Server. Με βάση αυτό δημιουργούμε έναν Server όπου στην απλή περίπτωση απλά χρησιμοποιείται για να δηλώνει παρουσία κάποιος κόμβος και να διαμοιράζει τις IP διευθύνσεις μεταξύ των κόμβων. Στην περίπτωση όπου ένας κόμβος δέν έχει IP διεύθυνση public ο ρόλος του server επεκτείνεται στο να μπορεί να καταγράφει αιτήματα σχετικά με τον τρόπο με τον οποίο μπορεί να συνδεθεί ένας κόμβος με κάποιον άλλο και να τα μεταφέρει καθώς και να διατηρεί ένα ανοιχτό κανάλι επικοινωνίας με κάθε κόμβο ώστε να προωθεί να μηνύματα. Έτσι ο server αποτελεί το κανάλι επικοινωνίας μεταξύ των κόμβων. Το συστατικό μέρος που θα συμπληρώνει την σύνδεση μεταξύ των κόμβων είναι το service που θα χρησιμοποιεί η εφαρμογή. Ο ρόλος του θα είναι στην απλή περίπτωση να συνδέει τα μέρη της εφαρμογής που εκκινούν μία διαδικασία με αυτά του αντίστοιχου κόμβου που την εξυπηρετούν. Στην περίπτωση που δεν έχουμε public IP ο ρόλος του service είναι να δέχεται αιτήματα που προέρχονται από άλλους κόμβους και να τα προωθεί στο κατάλληλο συστατικό μέρος της εφαρμογής ώστε να τα εξυπηρετήσει ορίζοντας ταυτόχρονα τον τρόπο επικοινωνίας μεταξύ των κόμβων. Τέλος τα Activities της εφαρμογής χωρίζονται σε αυτά που εκκινούν διαδικασίες και σε αυτά που τις εξυπηρετούν. Τα ζεύγη αυτά είναι Εισερχόμενη/εξερχόμενη κλήση, απομακρυσμένο αίτημα φιλίας/εξυπηρέτηση του και επιβεβαίωση ταυτότητας/εξυπηρέτηση του. Η εφαρμογή θα χωριστεί σε μέρη με σκοπό να υλοποιούνται σε κάθε μέρος μόνο οι απαραίτητες ενέργειες έτσι ώστε να μην προκύψει πολύπλοκο και κατά συνέπεια δυσνόητο και δύσχρηστο για τον χρήστη αποτέλεσμα.

2.1 Το Service

Το πρώτο μέρος της εφαρμογής θα είναι ένα Service σκοπός του οποίου θα είναι να ακούει για εισερχόμενες κλήσεις, για απομακρυσμένα αιτήματα φιλίας καθώς και αιτήματα επιβεβαίωσης ταυτότητας. Θα μπορεί να ενεργοποιηθεί//απενεργοποιηθεί από το Activity Initialize. Θα υπάρχουν δύο services, ένα για την περίπτωση που η εφαρμογή τρέχει σε περιβάλλον με public IP και ένα για την περίπτωση που η εφαρμογή τρέχει σε περιβάλλον με private IP . Το πιο θα εκκινηθεί θα αποφασίζεται από την εφαρμογή κατά τη φάση της εκκίνησης και της δήλωσης παρουσίας στον server διευθυνσιοδότησης. Το service για την περίπτωση που έχουμε public IP θα εκκινεί ένα server socket σε καινούργιο νήμα και θα περιμένει σύνδεση από κάποιο κόμβο. Αφού γίνει η σύνδεση θα δέχεται κατάλληλα δομημένο μήνυμα για 1)εισερχόμενη κλήση 2)απομακρυσμένο αίτημα φιλίας 3)επιβεβαίωση ταυτότητας. Αν το μήνυμα είναι σωστά δομημένο θα το προωθεί στο κατάλληλο Activity εκκινώντας το με intent και περνώντας τις κατάλληλες παραμέτρους, αν πρόκειται για αίτημα επιβεβαίωσης ταυτότητας θα το εξυπηρετεί χωρίς την βοήθεια κάποιου Activity. Το service για την περίπτωση που ο κόμβος έχει private IP θα εκκινεί σύνδεση με τον server η οποία θα μένει ενεργή και θα δέχεται κατάλληλα δομημένα μηνύματα που θα προέρχονται από άλλους χρήστες. Ανάλογα με το μήνυμα θα εκκινεί το κατάλληλο Activity ή θα στέλνει broadcast σε κάποιο ενεργό Activity.

2.2 Το Main Activity

Το οποίο θα δίνει την δυνατότητα στον χρήστη να μεταβεί σε όποια λειτουργία της εφαρμογής επιθυμεί. Θα συνοδεύεται από διεπαφή χρήστη με τα κατάλληλα πλήκτρα που θα εκκινούν τα κατάλληλα Activities χρησιμοποιώντας Intents. Τα Activities που θα μπορούν να εκκιννηθούν θα είναι τα 1)OutboundCall 2)AddContact 3)RemoveContact 4)WebOfTrust 5)Initialize 6)AddRemoteContact.

2.3 Το InboundCall Activity

Το οποίο θα χειρίζεται την εισερχόμενη κλήση και θα πραγματοποιεί τις απαραίτητες ενέργειες ώστε να εγκατασταθεί. Θα συνοδεύεται από διεπαφή χρήστη με πλήκτρα για την αποδοχή ή απόρριψη της κλήσης καθώς και πεδίο το οποίο θα ενημερώνει για το ποιος καλεί. Για την αποδοχή της κλήσης θα εκτελούνται δύο κύριες ενέργειες.Η πρώτη θα αφορά την διαδικασία παραγωγής του μυστικού κλειδιού. Το κλειδί κρυπτογράφησης θα είναι συμμετρικό(AES) σε mode CTR(stream cipher) και θα προκύπτει από την εκτέλεση του αλγορίθμου DiffieHellman Ephemeral(DHE). Επιλέγουμε mode CTR επειδή έχει δύο ιδιότητες που χρειαζόμαστε 1)η διαδικασία κρυπτογράφησης και αποκρυπτογράφησης είναι συμμετρικές, δηλαδή διαρκούν τον ίδιο χρόνο και 2)αν χάνονται πακέτα ο αλγόριθμος μπορεί να συγχρονίζεται. Δεδομένου ότι τα APIs του Android δεν υποστηρίζουν τον συγκεκριμένο αλγόριθμο θα πρέπει να υλοποιηθεί από την εφαρμογή. Η δεύτερη αφορά την διαδικασία καταγραφής, κρυπτογράφησης, αποστολής καθώς και τη διαδικασία αποδοχής, αποκρυπτογράφησης και αναπαραγωγής του ήχου. Για την αποστολή και αποδοχή του ήχου θα χρησιμοποιήσουμε το UDP πρωτόκολλο καθώς δεν μας ενδιαφέρει να αναπαράγουμε καθυστερημένα πακέτα δεδομένων. Για την δεύτερη διαδικασία θα τρέχουμε παράλληλα σε δύο νήματα τις διαδικασίες 1)αποδοχής αποκρυπτογράφησης και αναπαραγωγής και της 2)καταγραφής κρυπτογράφησης και αποστολής.Το Activity αυτό θα εκκινείται με Intent από ένα από τα δύο services με πληροφορίες για τον τρόπο σύνδεσης, παραμέτρους και άλλα δεδομένα.

2.4 Το OutboundCall Activity

Το οποίο θα στέλνει το κατάλληλο αίτημα στο Service και θα πραγματοποιεί τις κατάλληλες ενέργειες ώστε να εγκατασταθεί η κλήση. Θα συνοδεύεται από διεπαφή χρήστη με πεδίο όπου ο χρήστης θα διαλέγει πια επαφή θέλει να καλέσει, πλήκτρο εκκίνησης κλήσης, πλήκτρο τερματισμού κλήσης καθώς και πεδίο όπου θα εμφανίζεται η κατάσταση της κλήσης. Όπως και το InboundCall Activity θα εκτελεί δύο βασικές λειτουργίες 1)αλγόριθμος παραγωγής κλειδιού 2)διαδικασία καταγραφής, κρυπτογράφησης, αποστολής καθώς και τη διαδικασία αποδοχής, αποκρυπτογράφησης και αναπαραγωγής του ήχου. Οι Λειτουργίες θα είναι αντίστοιχες με του InboundCall Activity εκτός από κάποιες διαφορές στα βήματα της διαδικασίας παραγωγής κλειδιού δηλαδή στην εκτέλεση του αλγορίθμου DHE.Το Activity αυτό θα ανιχνεύει τις IP διευθύνσεις των δύο κόμβων και θα επιλέγει με ποιόν τρόπο θα πραγματοποιηθεί η κλήση. Ανάλογα με τις IP θα πραγματοποιούνται οι κατάλληλες ενέργειες και θα αποστέλλονται τα κατάλληλα μηνύματα είτε μέσω του ServicePublic είτε μέσω του ServicePrivate.

2.5 Το AddContact Activity

Το οποίο θα δίνει την δυνατότητα να εισαχθεί τοπικά ένα πιστοποιητικό(X.509 Certificate) και να συσχετιστεί με ένα όνομα επαφής ως καταχώρηση στη βάση δεδομένων των επαφών. Θα συνοδεύεται από διεπαφή χρήστη με πεδίο επιλογής ονόματος επαφής, πλήκτρο που θα εκτελεί την διαδικασία καταχώρησης καθώς και πεδίο που θα ενημερώνει για την επιτυχία της ενέργειας. Θα αναμένει την ύπαρξη ενός αρχείου κωδικοποιημένου πιστοποιητικού X.509 Certificate με συγκεκριμένο όνομα(π.χ myNewContact) σε συγκεκριμένο φάκελο (π.χ Pictures) της συσκευής. Για την αποθήκευση των πιστοποιητικών που θα συνοδεύονται από το όνομα της επαφής θα χρησιμοποιήσουμε την υποδομή KeyStore που παρέχει η Java και το Android και εξυπηρετεί αυτό ακριβώς το σκοπό. Το Activity θα ανοίγει ένα ήδη δημιουργημένο keystore, θα καταχωρεί τις αλλαγές και στη συνέχεια θα το αποθηκεύει. Προαιρετικά θα μπορούμε να προστατεύουμε την ακεραιότητα του KeyStore με κάποιο κωδικό συνεπώς προστίθεται και ένα πεδίο εισαγωγής του κωδικού στην διεπαφή του χρήστη.

2.6 Το RemoveContact Activity

Το οποίο θα δίνει δυνατότητα κατάργησης μιας αποθηκευμένης επαφής στη βάση δεδομένων. Θα συνοδεύεται από διεπαφή χρήστη με πεδίο επιλογής ονόματος χρήστη προς διαγραφή, πλήκτρο που θα εκτελεί την διαδικασία διαγραφής καθώς και πεδίο που θα ενημερώνει για το αποτέλεσμα της ενέργειας. Όπως και το AddContact Activity θα ανοίγει το ήδη υπάρχον keystore θα πραγματοποιεί τις αλλαγές και θα αποθηκεύει το αρχείο.

2.7 Το WebOfTrust Activity

Το οποίο θα χρησιμοποιείται για επαφές που δημιουργήθηκαν από απόσταση(δηλ. Όχι από το AddContact Activity) έτσι ώστε ο χρήστης να αποφασίσει αν θα εμπιστευτεί την ταυτότητα ενός χρήστη. Θα συνοδεύεται από διεπαφή χρήστη με πεδίο επιλογής του ονόματος της επαφής, πλήκτρο εκτέλεσης της ενέργειας και πεδίο που θα ενημερώνει για το πόσες επαφές επιβεβαιώνουν την ταυτότητα(X.509 Certificate) ενός χρήστη. Το Activity αυτό θα επικοινωνεί είτε με τοServicePublic άν ο κόμβος που επικοινωνεί έχει Public διεύθυνση είτε με το Validate Activity αν έχει private και εκτελείται το δεύτερο service. Η διαδικασία επιβεβαίωσης ταυτότητας θα περιλαμβάνει την αποστολή μηνύματος σε κάθε επαφή με παράμετρο το όνομα της επαφής που θέλουμε να επιβεβαιώσουμε και του κωδικοποιημένου κλειδιού. Στη συνέχεια θα επιστρέφουμε το αποτέλεσμα. Το Activity αυτό θα ανιχνεύει τις IP διευθύνσεις των δύο κόμβων και θα επιλέγει με ποιόν τρόπο θα πραγματοποιηθεί η ενέργεια. Ανάλογα με τις IP θα πραγματοποιούνται οι κατάλληλες ενέργειες και θα αποστέλλονται τα κατάλληλα μηνύματα είτε μέσω του ServicePublic είτε μέσω του ServicePrivate.

2.8 Το Validate Activity

Το Activity αυτό θα χρησιμοποιείται για να επιβεβαιώσει την ταυτότητα μίας επαφής στην περίπτωση που η εφαρμογή τρέχει σε περιβάλλον με private IP και τρέχει το δεύτερο service. Το Activity αυτό θα εκκινείται με intent για να επιβεβαιώσει την ταυτότητα ενός χρήστη όταν ο κόμβος αυτός λειτουργεί με private διεύθυνση. Αφού εγκατασταθεί η σύνδεση είτε με Connection Reversing είτε με Hole Punching δέχεται το πιστοποιητικό και το όνομα της επαφής και επιστρέφει μήνυμα επιτυχίας ή αποτυχίας

2.9 Το ContactRequest Activity

Θα απαντάει σε αιτήσεις που θα εκ κινούνται από το AddRemoteContact Activity. Θα δίνει την δυνατότητα στο χρήστη να αποφασίσει αν θα εισάγει μία επαφή που αιτείται να προστεθεί. Θα εκκινείται με intent από το Service με παράμετρο το όνομα της επαφής και το κωδικοποιημένο κλειδί, θα δίνει στον χρήστη τη δυνατότητα να προσθέσει το χρήστη που έκανε αίτημα ή να απορρίψει το αίτημα. Αν ο χρήστης προσθέσει την επαφή αυτή θα προστίθεται στο υπάρχον keystore αφού πρώτα ελεγχθεί αν υπάρχει επαφή με το ίδιο όνομα και αφού γίνουν οι απαραίτητες ενέργειες επεξεργασίας του κωδικοποιημένου πιστοποιητικού.

2.10 Το AddRemoteContact Activity

Θα στέλνει αίτημα φιλίας σε μία υπάρχουσα επαφή. Το αίτημα θα αποτελείται από το όνομα της επαφής και το certificate που αντιστοιχεί σε αυτήν. Θα αποτελείται από ένα πεδίο όπου ο χρήστης θα επιλέγει την επαφή στην οποία θα στείλει το αίτημα φιλίας, δηλαδή το όνομα του και την ταυτότητα του(πιστοποιητικό) και ένα πλήκτρο που θα πραγματοποιεί την ενέργεια. Το Activity αυτό για χρήση με το servicePublic θα δημιουργεί ένα κατάλληλα μορφοποιημένο μήνυμα,θα

ανακτά την διεύθυνση της επαφής από τη βάση δεδομένων, θα δημιουργεί δικτυακή σύνδεση με τη επαφή και θα στέλνει το αίτημα στο service της επαφής ακολουθούμενο από το κωδικοποιημένο πιστοποιητικό του. Για το servicePrivate θα αποφασίζει για τον τρόπο σύνδεσης με βάση τις IP και θα στέλνει τα κατάλληλα μηνύματα στο server για την εκτέλεση της ενέργειας.

2.11 To Initialize Activity

Θα δίνει τη δυνατότητα στον χρήστη να εισάγει χειροκίνητα τις IP διευθύνσεις των επαφών παρακάμπτοντας την ανάγκη για την ύπαρξη ενός Server καθώς και την δυνατότητα να ανακτήσει διευθύνσεις επαφών που έχουν δηλώσει την παρουσία τους στον server και να δηλώσει και ο ίδιος την παρουσία του. Επίσης κατά τη φάση δήλωσης παρουσίας στον server θα ανιχνεύει αν τρέχει σε περιβάλλον με public ή private IP και θα εκκινεί το κατάλληλο service .

3. Σημαντικότερες κλάσεις που θα χρησιμοποιηθούν

Σε αυτό το κεφάλαιο θα παρουσιάσουμε αναλυτικά τις σημαντικότερες κλάσεις του Android API που θα χρησιμοποιήσουμε και την λειτουργικότητα που αυτές προσφέρουν. Οι κλάσεις αυτές προσφέρουν έτοιμη λειτουργικότητα που διευκολύνει την υλοποίηση αλλά και περιορισμούς που δημιουργούν προβλήματα.

3.1 Η κλάση Keystore

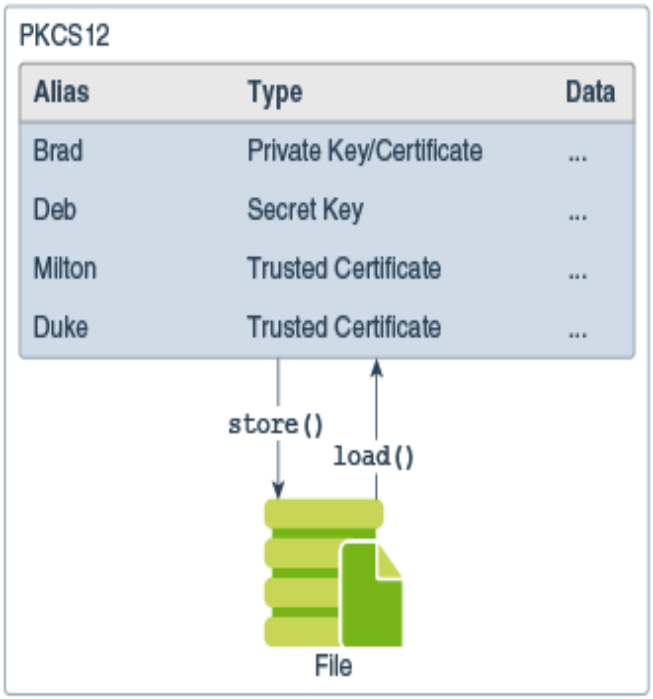
Για την ασφαλή αποθήκευση και ανάκτηση κλειδιών αντί μιας απλής βάσης δεδομένων τύπου SQL θα χρησιμοποιήσουμε την κλάση KeyStore η οποία αποτελεί μια υποδομή αποθήκευσης κρυπτογραφικών κλειδιών. Η κλάση παρέχει τρία ήδη καταχωρίσεων.
.https://developer.android.com/reference/java/security/KeyStore

KeyStore.PrivateKeyEntry
Αυτό το είδος καταχώρισης χειρίζεται ένα κρυπτογραφικό αντικείμενο τύπου **PrivateKey.** Αυτό αποτελεί ένα κρυπτογραφικό κλειδί συνοδευόμενο από μια αλυσίδα πιστοποιητικών (Certificate Entry) συνήθος selfsigned που αντιπροσωπεύει το ιδιωτικό και δημόσιο κλειδί. Προαιρετικά αυτή η καταχώριση μπορεί να προστατεύεται από κωδικό ώστε να αποτραπεί η μη εξουσιοδοτημένη πρόσβαση. Θα χρησιμοποιήσουμε αυτό το είδος καταχώρισης για να αποθηκεύσουμε το ζεύγος κλειδιών του χρήστη.
https://developer.android.com/reference/java/security/PrivateKey.html

KeyStore.SecretKeyEntry
Αυτό το είδος καταχώρισης αποθηκεύει ένα κρυπτογραφικό αντικείμενο τύπου SecretKey. Πρόκειται για συμμετρικό κρυπτογραφικό κλειδί. Στην εφαρμογή που θα αναπτύξουμε δεν θα χρησιμοποιήσουμε αυτού του είδους την καταχώριση καθώς τα συμμετρικά κλειδιά που θα παράγουμε θα είναι μίας χρήσης και δεν θα αποθηκεύονται πουθενά.
https://developer.android.com/reference/javax/crypto/SecretKey.html

KeyStore.TrustedCertificateEntry
Αυτό το είδος καταχώρισης αποθηκεύει ένα κρυπτογραφικό αντικείμενο τύπου Certificate. Πρόκειται για ένα δημόσιο κλειδί που αντιστοιχεί σε μία οντότητα, δηλαδή μία επαφή του χρήστη της εφαρμογής συνοδευόμενη απο ακόμα κάποιες πληροφορίες για την οντότητα. Όσο αφορά την εφαρμογή μας ενδιαφέρει μόνο το ιδιωτικό κλειδί. Στην ουσία όταν καταχωρούμε μία επαφή θα αποθηκεύουμε στο KeyStore μας ένα τέτοιο αντικείμενο συνοδευόμενο απο ένα ψευδώνυμο(alias).
https://developer.android.com/reference/java/security/cert/Certificate.html

Το αντικείμενο τύπου KeyStore φορτώνεται απο το αρχείο γίνεται η καταχώριση ή η ανάκτηση ενός κλειδιού και αποθηκεύεται πίσω στο αρχείο όπως δείχνει και η παρακάτω εικόνα.



Εικόνα 1: Λειτουργικότητα KeyStore

3.2 Η κλάση AudioRecord

Για την καταγραφή του ήχου απο το μικρόφωνο θα χρησιμοποιήσουμε την κλάση AudioRecord. Η κλάση αυτή διαχειρίζεται τις πηγές ήχου για εφαρμογές Java που θέλουν να χρησιμοποιήσουν το hardware της συσκευής. Αυτό επιτυγχάνεται διαβάζοντας τα δεδομένα από ένα αντικείμενο τύπου AudioRecord.
https://developer.android.com/reference/android/media/AudioRecord.html
Οι παρακάτω μέθοδοι είναι διαθέσιμες:

read(byte[] audioData, int offsetInBytes, int sizeInBytes)

read(short[] audioData, int offsetInShorts, int sizeInShorts)

read(ByteBuffer audioBuffer, int sizeInBytes)

Διάφορες παραλλαγές των τριών αυτών συναρτήσεων είναι διαθέσιμες.
Τέλος η συνάρτηση κατασκευαστής που δημιουργεί ένα αντικείμενο τύπου AudioRecord είναι η παρακάτω:
AudioRecord(int audioSource, int sampleRateInHz, int channelConfig, int audioFormat, int bufferSizeInBytes).

sampleRateInHz
Η συχνότητα δειγματοληψίας εκφρασμένη σε HZ. Η τιμή 44100 είναι η μόνη τιμή που δουλεύει εγγυημένα σε όλες τις συσκευές.

ChannelConfig
Μπορεί να πάρει τις τιμές CHANNEL_IN_MONO και CHANNEL_IN_STEREO

bufferSizeInBytes
Το μέγεθος του buffer στον οποίο αποθηκεύεται ο ήχος πριν διαβαστεί από τις συναρτήσεις read(). Το μέγεθος αυτού του buffer πρέπει να είναι τουλάχιστον ίσο με αυτό που επιστρέφει η συνάρτηση getMinBufferSize(int sampleRateInHz, int channelConfig, int audioFormat.

3.3 Η κλάση AudioTrack

Για την αναπαραγωγή του ήχου θα χρησιμοποιήσουμε την κλάση AudioTrack(). Η κλάση αυτή διαχειρίζεται και αναπαράγει μία πηγή ηχητικών δεδομένων για εφαρμογές. Αυτό επιτυγχάνεται εισάγοντας δεδομένα σε ένα αντικείμενο τύπου AudioTrack.
https://developer.android.com/reference/android/media/AudioTrack.html
Για αυτό το σκοπό οι παρακάτω μέθοδοι είναι διαθέσιμες.

int write(byte[] audioData, int offsetInBytes, int sizeInBytes)

int write(short[] audioData, int offsetInShorts, int sizeInShorts)

int write(float[] audioData, int offsetInFloats, int sizeInFloats, int writeMode)

audioData δομή τύπου float που κρατάει τα δεδομένα προς αναπαραγωγή.
OffsetInFloats ο δείκτης από όπου θα αρχίσει η αναπαραγωγή στην δομή audioData.
sizeInFloats ο αριθμός των shorts που θα εισαχθούν προς αναπαραγωγή στο αντικείμενο AudioTrack.
WriteMode μπορεί να πάρει μία απο τις δύο τιμές WRITE_BLOCKING όπου η write() θα μπλοκάρει έως ότου όλα τα δεδομένα γραφούν στο αντικείμενο καταναλωτή, WRITE_NON_BLOCKING όπου η write() θα επιστρέψει αμέσως αφού βάλει στην ουρά προς αναπαραγωγή όσα περισσότερα δεδομένα ήχου είναι δυνατό χωρίς να μπλοκάρει. Αυτή η παράμετρος χρησιμοποιείται μόνο σε stream mode.

Διάφορες παραλλαγές των συναρτήσεων write() είναι διαθέσιμες.

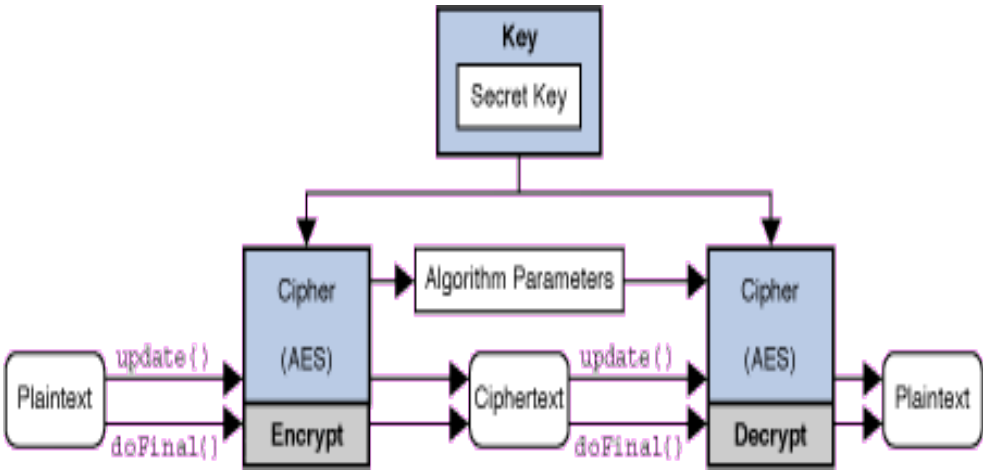
Ένα στιγμιότυπο ενός αντικειμένου AudioTrack μπορεί να λειτουργεί σε δύο καταστάσεις 1)Static 2)streaming. Στο streaming mode η εφαρμογή γράφει ένα συνεχές stream χρησιμοποιώντας μία από τις συναρτήσεις write(). Αυτές μπλοκάρουν και επιστρέφουν όταν τα δεδομένα μεταφερθούν απο το επίπεδο Java στο native επίπεδο και μπουν στην ουρά για αναπαραγωγή. Η κατάσταση streaming είναι χρήσιμη για καταστάσεις όπου τα ηχητικά δεδομένα είναι πολλά ώστε να χωρέσουν στη μνήμη λόγω της διάρκειας του ήχου ή λόγω της ποιότητας των δεδομένων προς αναπαραγωγή. Το static mode επιλέγεται όταν έχουμε να κάνουμε με ήχους μικρής διάρκειας που χωράνε στη μνήμη και υπάρχει η απαίτηση να αναπαραχθούν με τη μικρότερη δυνατή καθυστέρηση. Για την εφαρμογή μας θα χρησιμοποιήσουμε την κλάση AudioTrack σε stream mode.

3.4 Η κλάση Cipher

Για την κρυπτογράφηση και αποκρυπτογράφηση των δεδομένων θα χρησιμοποιήσουμε την κλάση Cipher. Ένα αντικείμενο Cipher μπορεί να αρχικοποιηθεί ώστε να υλοποιεί έναν από τους αλγόριθμους που υποστηρίζει το Android. Η εφαρμογή μας θα κρυπτογραφεί/αποκρυπτογραφεί τα δεδομένα χρησιμοποιώντας τον αλγόριθμο AES.Ο AES μπορεί να λειτουργεί είτε ως block cipher

όπου τροφοδοτούμε τον αλγόριθμο με δεδομένα σταθερού μεγέθους(blocks) ή ως stream cipher όπου όπου κρυπτογραφούμε τα δεδομένα σε μικρότερα μέρη ή ακόμα και bit με bit άν και συνήθως παράγεται ακολουθία ενός byte.
https://developer.android.com/reference/javax/crypto/Cipher

Η διαδικασία περιγράφεται σχηματικά ως εξής.



Εικόνα 2: Διαδικασία Κρυπτογράφησης/Αποκρυπτογράφησης

Ένα αντικείμενο Cipher δημιουργείται με κλήση στην μέθοδο getInstance(String transformation) όπου το πεδίο transformation είναι της μορφής "algorithm/mode/padding" ή "algorithm", ή την getInstance(String transformation, Provider). Στήν συνέχεια αρχικοποιείται με κλήση σε μία απο τις μεθόδους init(int opmode , …) η οποία επιλέγεται ανάλογα με τον αλγόριθμο που επιλέξαμε με την κλήση στην getInstance(). Η διαδικασία κρυπτογράφησης/αποκρυπτογράφησης τελειώνει με κλήσεις σε update() και doFinal().

3.5 Η κλάση Signature

Για την υλοποίηση του αλγορίθμου DHE θα χρειαστούμε την κλάση Signature.Η κλάση αυτή χρησιμοποιείται για να παρέχει λειτουργικότητα ψηφιακών υπογραφών σε εφαρμογές οι οποίες χρειάζονται αυθεντικοποίηση και ακεραιότητα δεδομένων. Ένα αντικείμενο τύπου Signature μπορεί να χρησιμοποιηθεί είτε για παραγωγή είτε για επιβεβαίωση ψηφιακών υπογραφών.

Ένα αντικείμενο τύπου Signature περνάει τρεις φάσεις κατά την διάρκεια της χρήσης του:

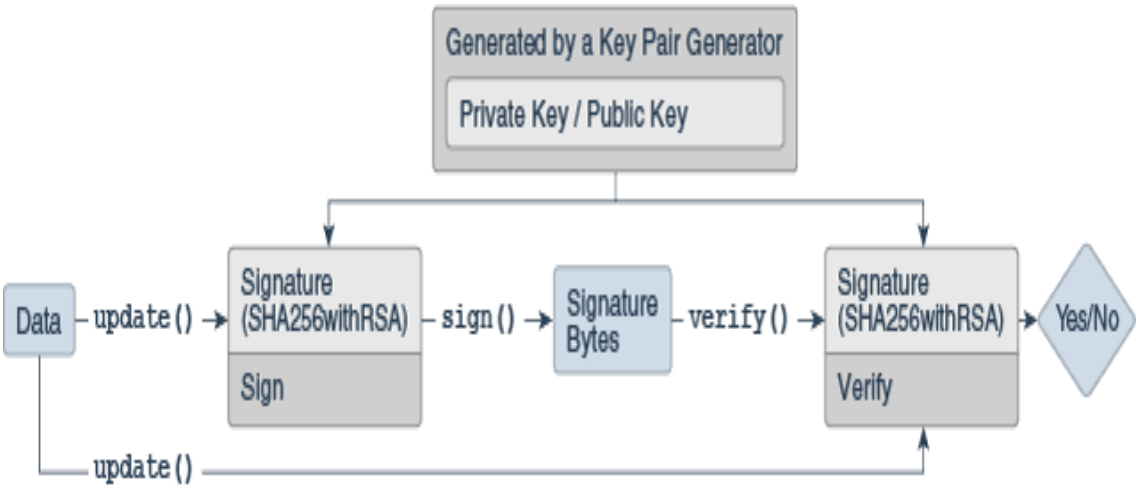
Αρχικοποίηση
Σε αυτή τη φάση αρχικοποιούμε το αντικείμενο είτε με κλήση στην void initVerify(PublicKey public key) εάν πρόκειται να χρησιμοποιήσουμε το αντικείμενο για επιβεβαίωση υπογραφής είτε με την void initSign(PrivateKey privatekey) εάν πρόκειται να χρησιμοποιήσουμε το αντικείμενο για παραγωγή ψηφιακής υπογραφής.

Ενημέρωση(Updating)
Σε αυτή τη φάση ανάλογα με τον τύπο της αρχικοποίησης και χρησιμοποιώντας μία από τις μεθόδους update(). Θα τροφοδοτούμε το αντικείμενο με δεδομένα προς υπογραφή ή επιβεβαίωση μέχρι αυτά να τελειώσουν.

Υπογραφή/Επιβεβαίωση
Σε αυτό το στάδιο πραγματοποιείτε η διαδικασία υπογραφής μέσω κλήσης σε κάποια από τις μεθόδους sign(). Η δεύτερη μέθοδος είναι η verify().

Η διαδικασία ψηφιακών υπογραφών περιγράφεται σχηματικά ως εξής

Για την εφαρμογή μας επιλέγουμε τον αλγόριθμο SHA256withRSA.



Εικόνα 3: Διαδικασία Ψηφιακών Υπογραφών

3.6 Η κλάση KeyPairGenerator

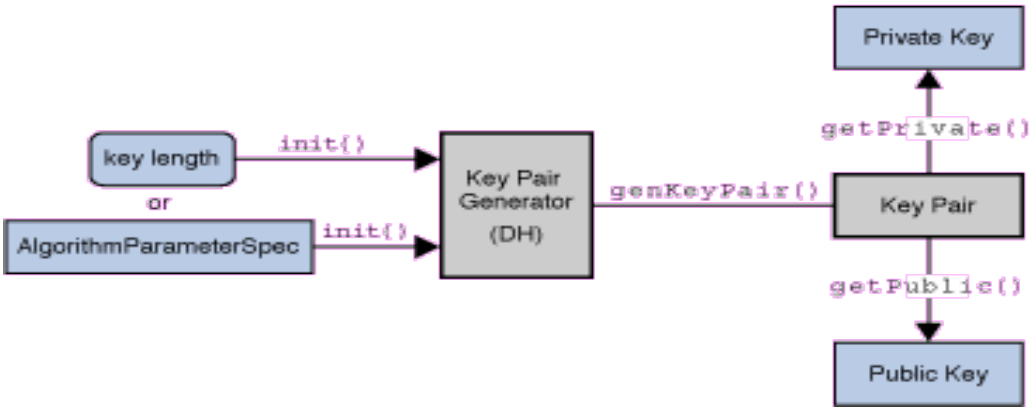
Η κλάση KeyPairGenerator χρησιμοποιείται για να παράγει ζεύγη Public και Private Keys. Τα αντικείμενα αυτού του είδους δημιουργούνται με κλήση στην μέθοδο getInstance(). Ένα KeyPairGenerator για ένα συγκεκριμένο αλγόριθμο δημιουργεί κλειδιά που μπορούν να χρησιμοποιηθούν με αυτόν τον αλγόριθμο μόνο. Επίσης συσχετίζει με τα κλειδιά παραμέτρους σχετικούς με τον αλγόριθμο.
https://developer.android.com/reference/java/security/KeyPairGenerator.html

Υπάρχουν δύο τρόποι για να παραχθούν ζεύγη κλειδιών 1)με ανεξάρτητο σχετικά με τον αλγόριθμο τρόπο(algorithm independent) και με συγκεκριμένο ως προς τον αλγόριθμο τρόπο(algorithm specific manner). Η μόνη διαφορά μεταξύ τους είναι η αρχικοποίηση του αντικειμένου.

Algorithm Independent
Σε αυτόν τον τρόπο αρχικοποίησης χρησιμοποιούμε μία από τις μεθόδους initialize Εφόσον δέν παρέχουμε συγκεκριμένες παραμέτρους για την αρχικοποίηση ο default Provider που θα χρησιμοποιηθεί επιλέγει τιμές για τις παραμέτρους εάν αυτές υπάρχουν(π.χ πρώτοι αριθμοί p, q).

Algorithm Specific
Σε αυτόν τον τρόπο αρχικοποίησης χρησιμοποιούμε μία άπω τις μεθόδους initialize, Εφόσον παρέχουμε συγκεκριμένες παραμέτρους ο Provider θα τις χρησιμοποιήσει. Τέλος η διαδικασία ολοκληρώνεται με κλήση στη μέθοδο KeyPair generateKeyPair().

Σχηματικά η διαδικασία περιγράφεται σχηματικά ως εξής



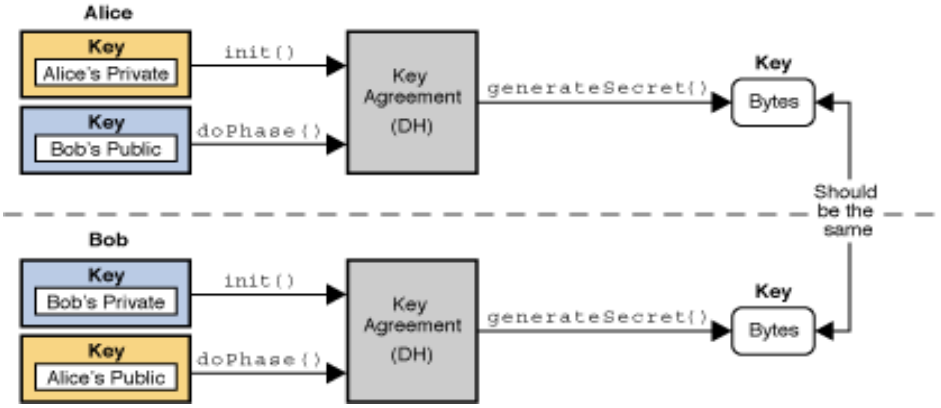
Εικόνα 4: Διαδικασία παραγωγής κλειδιού

3.7 Η κλάση KeyAgreement

Η κλάση KeyAgreement χρησιμοποιείται για να έρθουν σε συμφωνία δύο ή περισσότερα μέρη σχετικά με ένα μυστικό που θα παράγουν αυτόνομα(δηλαδή δεν θα σταλεί ποτέ) από το οποίο θα προκύψει ένα συμμετρικό κρυπτογραφικό κλειδί. Η εφαρμογή μας θα χρησιμοποιήσει αυτή την κλάση ως συστατικό μέρος για να δημιουργήσει τον αλγόριθμο DHE ο οποίος δεν προσφέρεται έτοιμος στα APIs του Android. Τα αντικείμενα αυτού του είδους δημιουργούνται με κλήση στην μέθοδο getInstance().
https://developer.android.com/reference/javax/crypto/KeyAgreement.html

Η διαδικασία συνεχίζεται με διαδοχικές κλήσεις στην μέθοδο dophase(). Για συμφωνία κλειδιού μεταξύ δύο μερών η dophase() εκτελείται σε κάθε μέρος μία φορά με παράμετρο το δημόσιο κλειδί του άλλου μέρους και lastphase=true. Η διαδικασία ολοκληρώνεται με κλήση στη μέθοδο byte[] generateSecret() η οποία επιστρέφει το παραχθέν μυστικό.

Η διαδικασία περιγράφεται σχηματικά ως εξής.



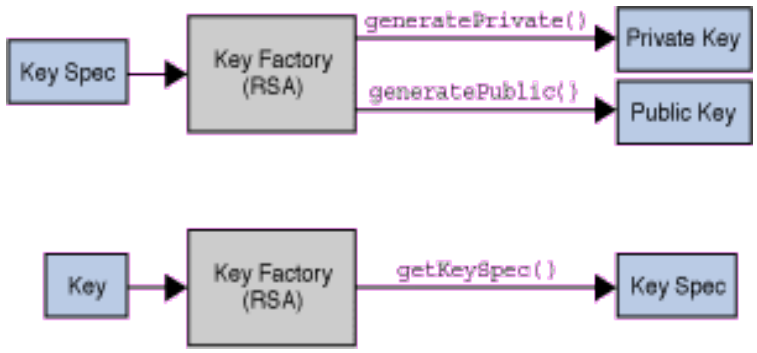
Εικόνα 5: Διαδικασία συμφωνίας κλειδιού

3.8 Η κλάση KeyFactory

Η κλάση KeyFactory χρησιμοποιείται για να μετασχηματίσει αντικείμενα τύπου Key σε προδιαγραφές(Specifications) δηλαδή διαφανείς αναπαραστάσεις των συστατικών ενός κλειδιού(π.χ οι πρώτοι αριθμοί p, q) και αντίστροφα. Θα χρησιμοποιήσουμε αυτή την κλάση κατά την ανάπτυξη του αλγορίθμου DHE.
(https://developer.android.com/reference/java/security/KeyFactory.html

Ένα αντικείμενο KeyFactory δημιουργείτε με κλήση στην μέθοδο getinstance και ολοκληρώνεται με κλήσεις στις μεθόδους PrivateKey generatePrivate(KeySpec keySpec) και PublicKey generatePublic(KeySpec keySpec) ή την <T extends KeySpec> T getKeySpec(Key key, Class<T> KeySpec) για την αντίστροφη διαδικασία.

Σχηματικά η διαδικασία περιγράφεται ως εξής.



Εικόνα 6: Διαδικασία KeyFactory

4. Προβλήματα που προέκυψαν κατά την υλοποίηση

Κατά τη διαδικασία της υλοποίησης βρεθήκαμε μπροστά σε προβληματικές καταστάσεις που επηρεάζουν σημαντικά και περιορίζουν τη λειτουργικότητα της εφαρμογής έτσι όπως αρχικά σχεδιάστηκε. Δύο από αυτά είναι τα σημαντικότερα τα οποία κρίνουμε πως χρήζουν επεξήγησης, για τους λόγους δηλαδή που προέκυψαν, το βαθμό και τον τρόπο που επηρεάζουν τη λειτουργικότητα καθώς και τις πιθανές λύσεις που υπάρχουν για την αντιμετώπιση τους.

4.1 Πρόβλημα 1:Απώλεια συγχρονισμού του stream cipher

Το πρώτο απο τα δύο σημαντικότερα προβλήματα είναι αυτό της απώλειας συγχρονισμού του stream cipher. Το πρόβλημα αυτό προκύπτει διότι επειδή χρησιμοποιούμε το πρωτόκολλο UDP κάποια πακέτα αναπόφευκτα χάνονται. Αυτή η κατάσταση παρόλο που δέν επηρεάζει σημαντικά την ποιότητα της υπηρεσίας ώστε να δημιουργεί πρόβλημα διότι άν για παράδειγμα ο αποστολέας έχει στείλει πέντε πακέτα δεδομένων και χαθεί το τέταρτο και από όταν φτάσει το 5 και τα επόμενα πακέτα ο παραλήπτης θα προσπαθεί να τα αποκρυπτογραφήσει με λάθος counter. Πιο συγκεκριμένα αντί να το αποκρυπτογραφεί με counter = IV+5 θα το αποκρυπτογραφεί με counter = IV+4 παράγοντας λάθος αποτέλεσμα. Παράλληλα το API του Android και συγκεκριμένα η κλάση Cipher δέν παρέχει κάποια μέθοδο για να επηρεάσουμε τον counter με τον οποίο αποκωδικοποιούνται τα δεδομένα. Οι μονάδικοί τρόποι να επηρεάσουμε τον counter είναι εμμέσως μέσω κλήσης της update() ή της doFinal(), ή άμεσα μηδενίζοντας τον counter και θέτοντας έναν καινούργιο IV μέσω κλήσης στην init().

Πρόβλημα 1:Προτεινόμενη λύση

Η λύση που προτείνεται για την αντιμετώπιση αυτού του προβλήματος είναι η εξής. Στη πλευρά του αποστολέα θα έχουμε έναν μετρητή με εύρος τιμών για παράδειγμα από 0-65000 ο οποίος θα αυξάνεται κάθε φορά που θα στέλνουμε ένα πακέτο και θα τον κάνουμε concatenate στο πακέτο των δεδομένων που θα στέλνουμε. Στην πλευρά του παραλήπτη των δεδομένων θα έχουμε έναν αντίστοιχο μετρητή ο οποίος θα αυξάνεται κάθε φορά που λαμβάνει ένα πακέτο ταυτόχρονα ο παραλήπτης θα ξέρει πόσα πακέτα έχει κρυπτογραφήσει ο αποστολέας. Με αυτόν τον τρόπο αν υπάρχει απώλεια έως 10000 πακέτα θα είναι ανιχνεύσιμη. Έχοντας ανιχνεύσει τον αριθμό των πακέτων που έχουν χαθεί μπορούμε να διορθώσουμε τον counter με διαδοχικές κλήσεις στις update() ή doFinal(), χωρίς να χρησιμοποιούμε το αποτέλεσμα αν ο αριθμός των πακέτων που έχουν χαθεί είναι μικρός ή αν ο αριθμός των πακέτων που έχουν χαθεί είναι μεγάλος μπορούμε να επαναυπολογίσουμε τον σωστό IV και να τον θέσουμε με κλήση στην init().

4.2 Πρόβλημα 2:Ιδιωτικές διευθύνσεις και NATs

Το δεύτερο από τα σημαντικότερα προβλήματα είναι αυτό της διευθυνσιοδότησης στο Internet. Όταν η εφαρμογή τρέχει σε συσκευές όπου τους έχει δοθεί global IP όλα δουλεύουν κανονικά αλλά όταν μία από τις δύο συσκευές ή και οι δύο βρίσκονται πίσω από ένα ή πολλαπλά NATs τότε οι εφαρμογή δέν μπορεί να εγκαταστήσει την σύνδεση αφού η private IP address δέν είναι ορατή στο Internet.

Στην περίπτωση που βρισκόμαστε σε ένα NAT οικιακού δικτύου και ο router έχει μία global IP διεύθυνση είναι εύκολο να ανοίξουμε την κατάλληλη πόρτα στο router και να δρομολογούμε την κίνηση στην κατάλληλη private IP. Τί γίνεται όμως άν ο router βρίσκεται και αυτός επίσης πίσω από NAT ή η εφαρμογή τρέχει σε δίκτυο 3G/4G και δέν έχουμε δυνατότητα ρύθμισης του router.

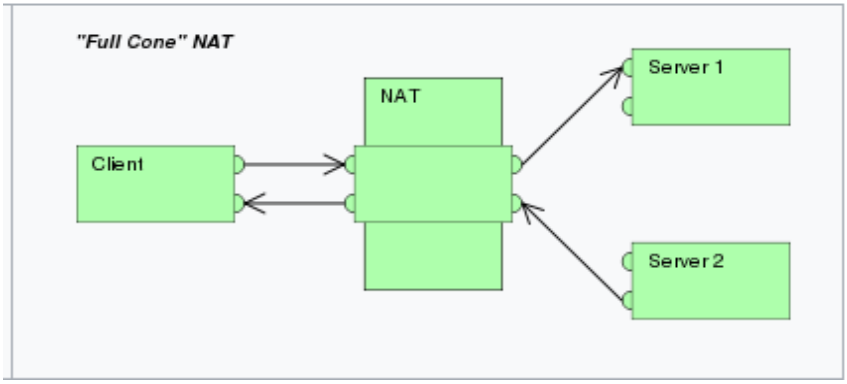
Τα σενάρια είναι τρία:
1)Και οι δύο συσκευές έχουν global IP διευθύνσεις
2)Και οι δύο συσκευές έχουν private IP διευθύνσεις
3)Η μία συσκευή έχει private διεύθυνση και η άλλη global

Λίγα λόγια για τα NATs

Network Address Translation(NAT) είναι μία μέθοδος η οποία αντιστοιχεί ένα χώρο διευθύνσεων σε έναν άλλο αλλάζοντας τα IP Headers για δεδομένα που περνάνε από τη συσκευή. Σημαντικό ρόλο παίζει η έννοια της συνόδου(session). Μία σύνοδος αναγνωρίζεται από δύο ζεύγη διευθύνσεων/πόρτας. Η τεχνική αυτή αρχικά χρησιμοποιήθηκε για να αντιμετωπιστεί η επαναδιευθυνσιοδότιση των κόμβων όταν το δίκτυο μεταφέρεται.

Η κυριότερη κατηγορία NAT είναι αυτή των Outbound NATs. Αυτά τα NATs απορρίπτουν κάθε πακέτο που φτάνει αν προηγουμένως δεν έχει προηγηθεί η δημιουργία ενός session μέσω της εκκίνησης μίας σύνδεσης από μία private address του εσωτερικού δικτύου. Υπάρχουν τέσσερις βασικές μέθοδοι NAT και διάφορες παραλλαγές τους. Στην συνέχεια παραθέτουμε τον τρόπο λειτουργίας τους:

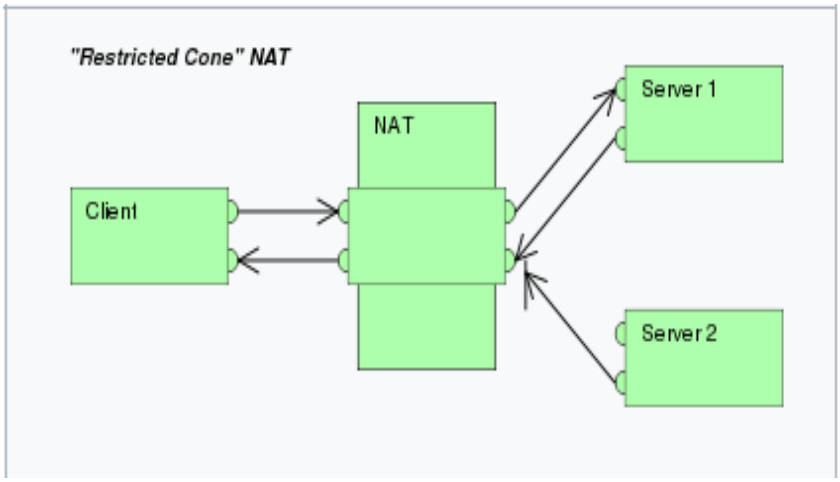
Full Cone NAT
Αυτή η μέθοδος αφού δημιουργηθεί ένα session μέσω της εκκίνησης μίας σύνδεσης από το εσωτερικό δίκτυο δέχεται κάθε πακέτο που φτάνει στην εξωτερική διεύθυνση/πόρτα που έχει ανοίξει και το προωθεί στην εσωτερική διεύθυνση/πόρτα. Αν φτάσει πακέτο από διαφορετικό από τον αρχικό παραλήπτη αυτό γίνεται δεκτό και προωθείται στο εσωτερικό endpoint.
Σχηματικά περιγράφεται ως εξής:



Εικόνα 7: Full Cone NAT Configuration

Address Restricted Cone NAT
Αυτή η μέθοδος είναι ίδια με την προηγούμενη με την διαφορά ότι αφού δημιουργηθεί το session, για να μην απορριφθεί ένα πακέτο πρέπει να προέρχεται από την ίδια διεύθυνση IP. Για να γίνει δεκτό ένα πακέτο από έναν νέο παραλήπτη πρέπει να έχει προηγηθεί αποστολή πακέτου προς αυτόν τον παραλήπτη από την private διεύθυνση/πόρτα.

Σχηματικά περιγράφεται ως εξής

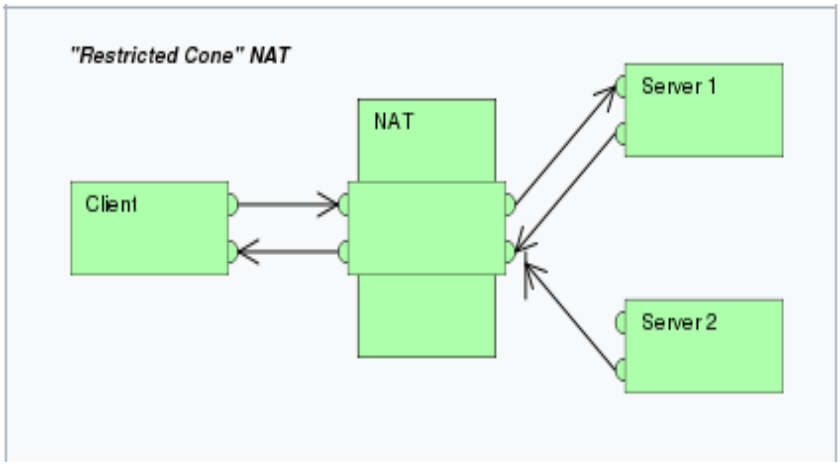


Εικόνα 8: Address Restricted NAT Configuration

Port Restricted Cone NAT

Αυτή η μέθοδος είναι ίδια με την παραπάνω με διαφορά ότι εκτός από τον περιορισμό για την διεύθυνση έχουμε περιορισμό και για την πόρτα. Αν ληφθεί πακέτο από τη ίδια IP αλλά από άλλο port απορρίπτεται.

Σχηματικά περιγράφεται ως εξής



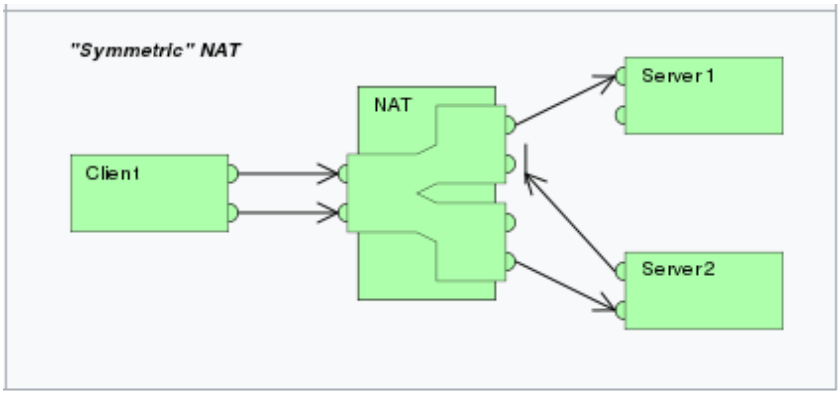
Εικόνα 9: Port Restricted NAT Configuration

Symmetric NAT

Αυτή η μέθοδος διαφέρει πολύ απο τις παραπάνω καθώς συμπεριφέρεται ως εξής. Αφού δημιουργηθεί ένα session και δημιουργηθούν τα endpoints δεκτά γίνονται πακέτα που προέρχονται απο τον αρχικό παραλήπτη(διεύθυνση/πόρτα). Αν ο αποστολέας στείλει απο το εσωτερικό endpoint του session πακέτο σε έναν δεύτερο παραλήπτη τότε η μέθοδος δημιουργεί ένα καινούργιο

endpoint συνεπώς ένα καινούργιο session.

Σχηματικά περιγράφεται ως εξής



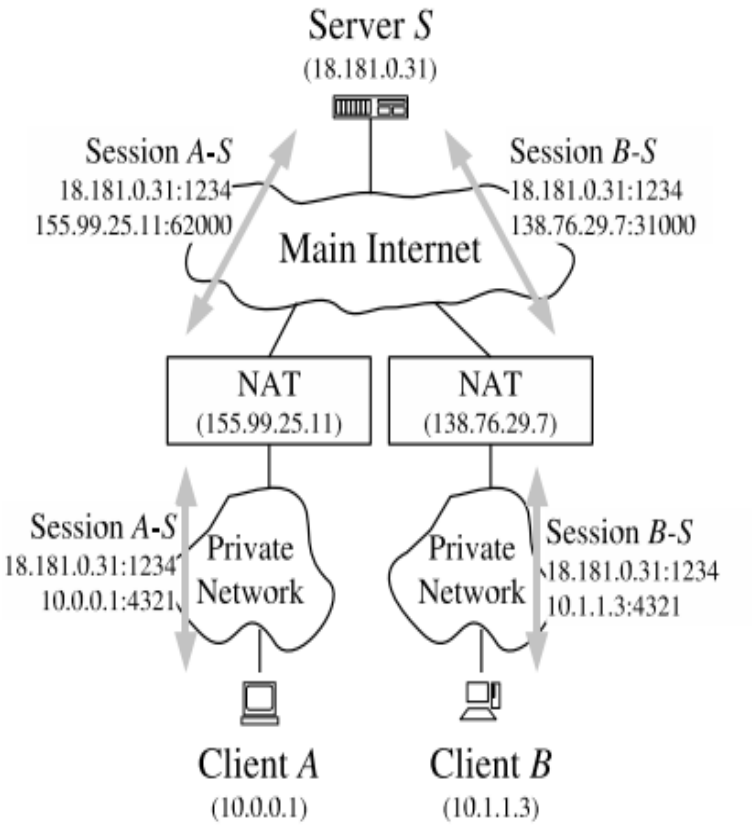
Εικόνα 10: Symmetric NAT Configuration

Πρόβλημα 2 :Διαθέσιμες λύσεις

Υπάρχουν τρεις λύσεις στη βιβλιογραφία που μπορούν να εφαρμοστούν στα τρία σενάρια του προβλήματος όπως ορίστηκαν παραπάνω.

1)Relaying
Η πιο αποτελεσματική αλλά λιγότερο λειτουργική λύση όσο αφορά την P2P αρχιτεκτονική που θέλουμε για την εφαρμογή, εφαρμόζεται σε όλα τα σενάρια. Εδώ όταν δύο εφαρμογές θέλουν να επικοινωνήσουν χρησιμοποιούν έναν Server με public IP όπου αφού δηλώσουν την παρουσία τους στέλνουν αίτημα επικοινωνίας στον Server, ο Server προωθεί το αίτημα από το ένα peer στο άλλο και ξεκινάει η επικοινωνία. Όλα τα δεδομένα της επικοινωνίας περνούν μέσα από τον Server ο οποίος τα προωθεί στα peers. Δεδομένου ότι τα peers ξεκίνησαν την επικοινωνία με τον Server δεν αντιμετωπίζουμε κανένα πρόβλημα με τα Outbound NATs.

Σχηματικά περιγράφεται ως εξής



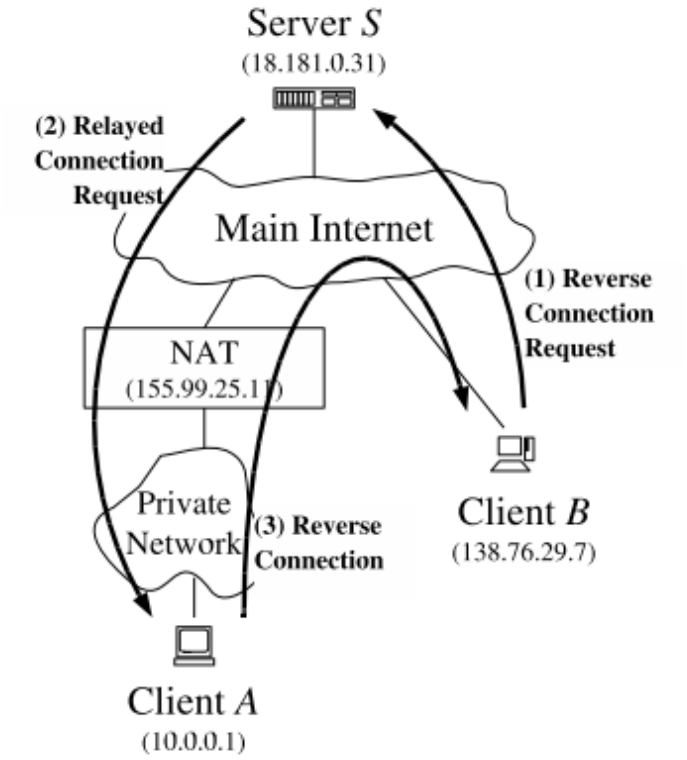
Εικόνα 11: Διαδικασία Relaying

2)Connection Reversal

Αυτή η λύση εφαρμόζεται στο σενάριο στο οποίο ένα από τα δύο peers έχει global IP διεύθυνση. Τα σενάρια εδώ είναι δύο. Ο κόμβος που έχει private διεύθυνση επιθυμεί να καλέσει τον κόμβο που έχει global IP διεύθυνση. Αφού χρησιμοποιήσει τον Server για να μάθει την διεύθυνση του εκκινεί σύνδεση με αυτόν η οποία επιτυγχάνεται εφόσον έχει ορατή διεύθυνση IP.

Στο δεύτερο σενάριο ο κόμβος που έχει global IP διεύθυνση επιθυμεί να καλέσει αυτόν με την private IP. Αρχικά και επειδή δεν μπορεί να εκκινήσει σύνδεση με μία private διεύθυνση ενημερώνει για την πρόθεση του να καλέσει τον Server ο οποίος με τη σειρά του ενημερώνει τον δεύτερο κόμβο, ο δεύτερος κόμβος εκκινεί τότε σύνδεση με τον πρώτο όπως περιγράφηκε παραπάνω και η επικοινωνία επιτυγχάνεται.

Σχηματικά περιγράφεται παρακάτω

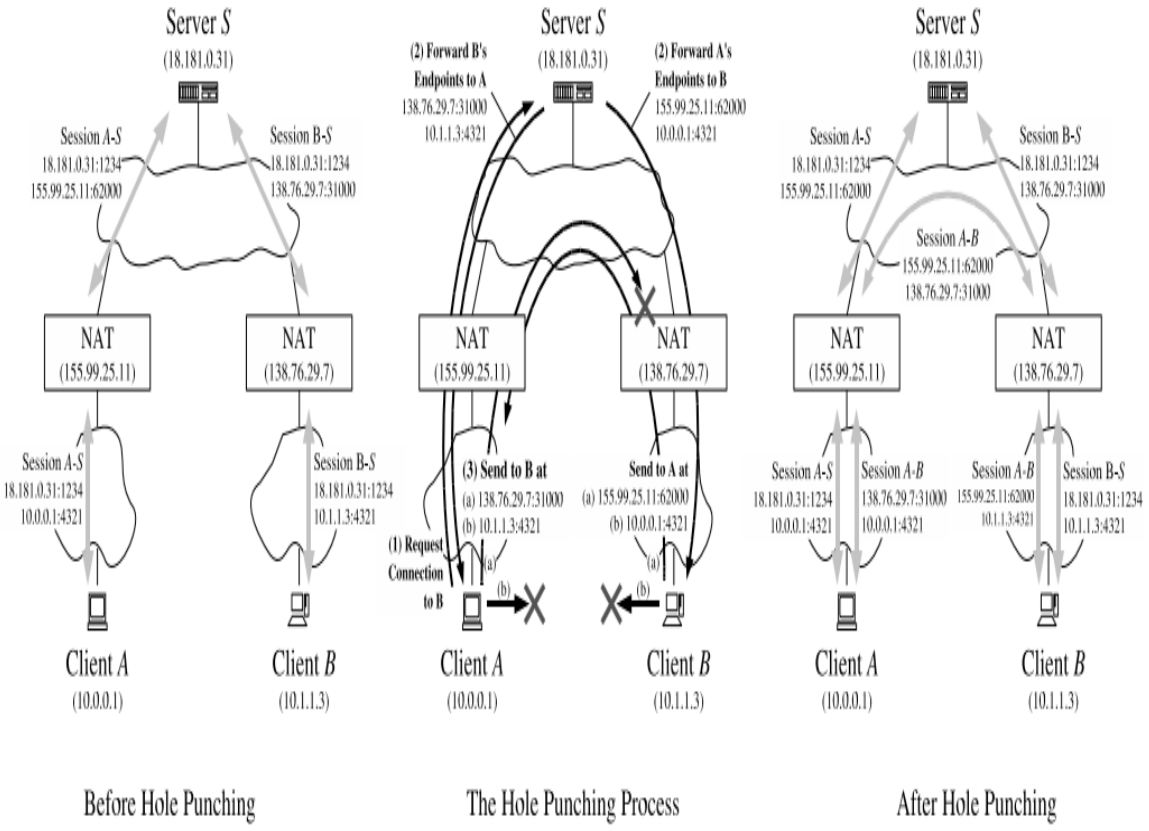


Εικόνα 12: Διαδικασία Connection Reversal

3)Hole Punching

Αυτή η τεχνική εφαρμόζεται στην περίπτωση όπου και οι δύο κόμβοι που θέλουν να επικοινωνήσουν βρίσκονται πίσω από NAT. Η λύση αυτή εφαρμόζεται για όλες τις παραλλαγές NATs εκτός από τα συμμετρικά για λόγους που θα εξηγήσουμε παρακάτω. Εδώ οι δύο κόμβοι διατηρούν μία σύνδεση(UDP) ανοιχτή στέλνοντας πακέτα ανά τακτά χρονικά διαστήματα ώστε το NAT να μην τερματίσει το session. Στην συνέχεια ο κόμβος Α στέλνει αίτημα κλήσης προς τον Β όπου φτάνει στον Server ο οποίος το προωθεί στον Β. Τέλος ο Server στέλνει στον Α το public endpoint του Β και στον Β το public endpoint του Α. Αφού ολοκληρωθεί η συναλλαγή με τον Server ο Α και ο Β στέλνουν πακέτα στα endpoints που πλέον γνωρίζουν. Πλέον αφού σταλεί το πακέτο μέσα από τα private endpoints στα δύο NATs επαναχρησιμοποιούμε τα sessions που δημιουργήθηκαν για επικοινωνία με το Server επίσης πλέον έχει αλλάξει ο κανόνας στο NAT το οποίο πλέον περιμένει δεδομένα απο το public endpoint του άλλου κόμβου τέλος η σύνδεση φαίνεται να έχει εκκινηθεί από το private δίκτυο οπότε δεν έχουμε πρόβλημα ούτε με το Outbound NAT. Η μέθοδος αυτή δεν λειτουργεί με συμμετρικά NATs επειδή όταν οι δύο κόμβοι αρχίσουν να στέλνουν πακέτα ο ένας στο public endpoint του άλλου το NAT στην ουσία θα δημιουργήσει καινούργιο Session

Σχηματικά περιγράφεται ως εξής



Εικόνα 13: Διαδικασία Hole Punching

Η ύπαρξη του Server στην εφαρμογή και δεδομένων όλων των σεναρίων επικοινωνίας είναι αναπόφευκτη. Σχεδιαστικός στόχος της εφαρμογής ήταν να περιορίσει όσο γίνεται την ανάγκη για έναν "δυνατό" Server. Ιδανικά θα θέλαμε έναν Server ο οποίος απλά θα κατέγραφε την παρουσία κόμβων και θα διαμοίραζε τις IP και θα μπορούσε να τρέχει ακόμα και σε ένα οικιακό δίκτυο. Δεδομένου του προβλήματος με τα NATs και απόρριψης της λύσης 1)Relaying οδηγούμαστε στην λύση ενός Rendezvous Server ο οποίος θα έχει τη δυνατότητα να αντιλαμβάνεται πιο από τα τρία σενάρια επικοινωνίας έχει να αντιμετωπίσει και να καθοδηγεί τους κόμβους που θέλουν να επικοινωνήσουν κατάλληλα.

5.Άλλα θέματα που πρέπει να αντιμετωπιστούν ώστε η εφαρμογή να πετύχει το σκοπό της

Η εφαρμογή που αναπτύξαμε προστατεύει τα δεδομένα όσο αυτά μεταφέρονται μέσα στα δίκτυα IP αλλά αυτό δεν είναι αρκετό για να ισχυριστούμε ότι η εφαρμογή επιτυγχάνει το σκοπό της ο οποίος είναι η ασφάλεια της επικοινωνίας. Τα smartphones είναι πολύπλοκες συσκευές με πολύπλοκο λογισμικό και οι περισσότεροι χρήστες έχουν εγκατεστημένες δεκάδες εφαρμογές αμφιβόλου προέλευσης ακόμα και στο store. Επίσης έχουν βρεθεί πολλές ευπάθειες οι οποίες έχουν αποτέλεσμα το rooting μίας συσκευής και άρα τον πλήρη έλεγχο από τον επιτιθέμενο. Ακόμα χειρότερα πολλές εταιρείες τηλεπικοινωνιών που εμπορεύονται συσκευές παρέχουν support για πολύ περιορισμένο διάστημα αφήνοντας τις συσκευές τους ευάλωτες σε όλες τις ευπάθειες που θα ανακαλυφθούν στο μέλλον. Όλα αυτά οδηγούν το επιτιθέμενο στο να έχει πρόσβαση στις λειτουργίες της συσκευής απλά εγκαθιστώντας μία εφαρμογή ή εκμεταλλευόμενος μία ευπάθεια. Στην συνέχεια έχουν τη δυνατότητα να ενεργοποιούν την κάμερα, το μικρόφωνο και να στέλνουν τα δεδομένα μέσω δικτύου. Όλα αυτά αφορούν επιτιθέμενους χωρίς πολλούς πόρους όπως πρόσβαση σε κάποιο σημείο του δικτύου παρόχου. Επιτιθέμενοι με πόρους όπως εταιρείες τηλεπικοινωνιών ή κατασκευαστές συσκευών έχουν ήδη εγκατεστημένες εφαρμογές που έχουν πρόσβαση σε μικρόφωνο κάμερα και δίκτυο. Επίσης έχουν δυνατότητα να εγκαταστήσουν εφαρμογές όποτε θελήσουν και να συλλέγουν δεδομένα για δικό τους όφελος όπως έχουν κάνει σε πολλές περιπτώσεις στο παρελθόν. Τέλος το προφανές είναι ότι οι νόμιμες υποκλοπές είναι γρήγορες και εύκολες με την υπόθεση ότι οι εταιρείες, πάροχοι και κατασκευαστές, συνεργάζονται όπως κάνουν σχεδόν πάντα.

Η αντιμετώπιση του προβλήματος όσον αφορά τους επιτιθέμενους χωρίς πόρους είναι αρκετά απλή και αποτελεσματική. Το μόνο που χρειάζεται είναι να κάνεις είναι να χρησιμοποιείται μία συσκευή μόνο για να τρέχει την εφαρμογή μας. Αυτό σημαίνει πως ο χρήστης πρέπει να κάνει και να μην κάνει συγκεκριμένες ενέργειες.

Ενέργειες που πρέπει να γίνονται και να μην γίνονται:

Πάντα εγκαθιστούμε ενημερώσεις ασφαλείας που παρέχει ο κατασκευαστής το συντομότερο. Αν κάποια εφαρμογή που υπάρχει στην συσκευή έχει γνωστό πρόβλημα ασφάλειας ένας επιτιθέμενος μπορεί απλά να υποθέσει ότι είμαστε ευάλωτοι και να δοκιμάσει μία επίθεση(π.χ να στείλει ένα απλό SMS).

Ποτέ δεν εγκαθιστούμε καινούργιες εφαρμογές. Αν το κάνουμε δεν εγκαθιστούμε εφαρμογές που έχουν δικαίωμα να χρησιμοποιούν κάμερα μικρόφωνο και δυνατότητα να εγκαθιστούν καινούργιες εφαρμογές

Μειώνουμε το attack surface. Το ότι δεν έχει ανιχνευθεί μία ευπάθεια δεν σημαίνει ότι δεν υπάρχει ,όσο λιγότερες εφαρμογές χρησιμοποιούμε τόσο πιο ασφαλείς είμαστε. Υποψήφιες ευάλωτες εφαρμογές είναι ο browser, το πρόγραμμα ανάγνωσης PDFs ακόμα και το MP3 player.

Πάντα προσέχουμε την φυσική ασφάλεια της συσκευής. Απενεργοποιούμε το debug mode και κλειδώνουμε την οθόνη

Η αντιμετώπιση του προβλήματος όσον αφορά τους επιτιθέμενους με πόρους είναι δύσκολη επίπονη και πολλές φορές αδύνατη.

Αν ο επιτιθέμενος έχει τη συνεργασία του παρόχου τηλεπικοινωνιών τότε η εφαρμογή μας παρέχει ασφάλεια διότι η μόνη δυνατή επίθεση θα ήταν μία τύπου Man In the Middle. Επίθεση τέτοιου τύπου θα αποτύχει και μάλιστα θα ανιχνευτεί από την εφαρμογή μας. Αν τώρα ο επιτιθέμενος έχει την συνεργασία του κατασκευαστή της συσκευής τότε τα πράγματα γίνονται πολύ δύσκολα. Μπορούμε να κάνουμε δύο υποθέσεις με βάση τις οποίες θα
αξιολογήσουμε τις επιλογές μας.

Υπόθεση 1: Η συσκευή δεν έχει spyware και θέλουμε να αποτρέψουμε την εγκατάσταση του.
Σε αυτή τη περίπτωση εξετάζουμε όλες τις εφαρμογές που βρίσκονται ήδη εγκατεστημένες από κατασκευαστή και Google για το αν έχουν πρόσβαση σε κάμερα μικρόφωνο δίκτυο και εγκατάσταση νέων εφαρμογών. Αφού τις εντοπίσουμε τις απ εγκαθιστούμε . Ταυτόχρονα εφαρμόζουμε όλους τους κανόνες που αφορούν τους επιτιθέμενους χωρίς πόρους όπως περιγράφηκαν παραπάνω.

Υπόθεση 2: Η συσκευή έχει ήδη ενεργό spyware και θέλουμε να το εντοπίσουμε
Με βάση αυτήν την υπόθεση η επίτευξη του στόχου γίνεται δύσκολη. Ένας απλός και αποτελεσματικός τρόπος να ανιχνεύσουμε ενεργό spyware είναι να περνάει η δικτυακή κίνηση της συσκευής μας μέσω ενός proxy όπου μπορούμε να αναλύουμε τα πακέτα και αφού ανιχνεύσουμε τη διεργασία που τα παράγει και να την σταματήσουμε ελπίζοντας ότι η συσκευή θα συνεχίσει να δουλεύει.

Υπόθεση 3: Η συσκευή έχει ανενεργό spyware
Σε αυτή την περίπτωση η επίτευξη του στόχου είναι σχεδόν αδύνατη αφού ο μόνος τρόπος να εντοπίσουμε το spyware είναι να αναλύσουμε όλο το kernel το framework του εγκατεστημένου Android καθώς και drivers συσκευών και bootloader. Σε αυτή την περίπτωση αυτό που προτείνουμε είναι η εύρεση ενός κατασκευαστή που παρέχει όλο το λογισμικό της συσκευής opensource και η διεξοδική ανάλυση του ή η κατασκευή μίας συσκευής με χρήση Arduino..

Προτείνουμε στον χρήστη της εφαρμογής να μην κάνει υποθέσεις ως προς τον επιτιθέμενο και την αποτελεσματικότητα της εφαρμογής σε σχέση με τα σενάρια που περιγράφηκαν καθώς και να προσπαθήσει να καλύψει όλες τις υποθέσεις πριν την χρησιμοποιήσει.

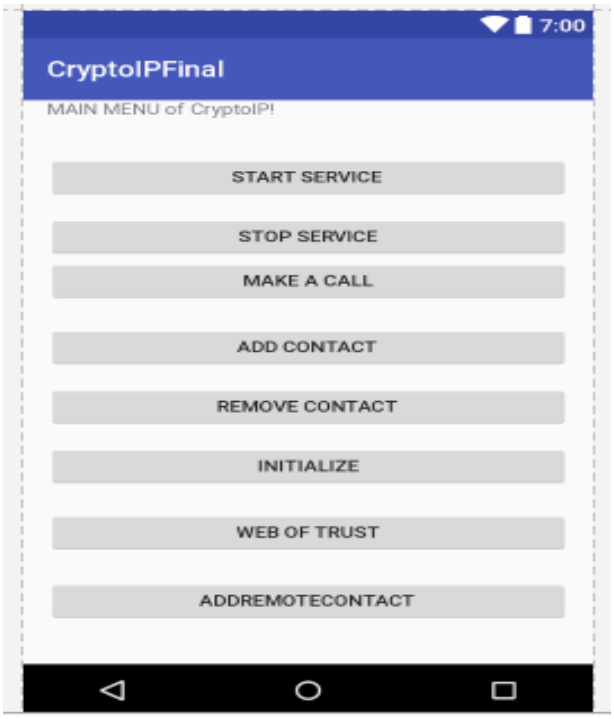
6.Υλοποίηση των μερών της εφαρμογής

Σε αυτό το κεφάλαιο θα παραθέσουμε αναλυτικά τον κώδικα της εφαρμογής ανά μέρος. Εξαιτίας της σχεδίασης του Android δεν μπορούμε να καλέσουμε μεθόδους που μπλοκάρουν για μεγάλο χρονικό διάστημα το κεντρικό νήμα του Activity, όπως δικτυακές κλήσεις και πρόσβαση σε βάσεις δεδομένων διότι το Android θα θεωρήσει ότι η εφαρμογή δεν ανταποκρίνεται και θα προτρέψει τον χρήστη να τερματίσει την εφαρμογή. Για το λόγο αυτό υλοποιούμε όλες αυτές τις χρονοβόρες διαδικασίες με την κλάση IntentService δημιουργώντας ένα καινούργιο νήμα χωρίς περιορισμούς όπου εκτελούμε όλες τις διεργασίες και επιστρέφουμε πληροφορίες και δεδομένα όπου αυτό χρειάζεται μέσω της κλάσης ResultReceiver η οποία υλοποιεί callbacks στο κεντρικό νήμα. Έτσι κάθε Activity συνοδεύεται από ένα αντίστοιχο IntentService.

6.1 Το MainActivity Activity

Το UserInterface του MainActivity θα αποτελείται απο ένα text view που θα εμφανίζει το κείμενο “MainMenu” και οκτώ πλήκτρα. 1)Start Service 2)Stop Service 3)WebOfTrust 4)Make Call 5)Add Contact 6)Remove Contact 7)AddRemoteContact 8)Initialize

και η οθόνη που εμφανίζει είναι η εξής



Εικόνα 14: Οθόνη Main Activity

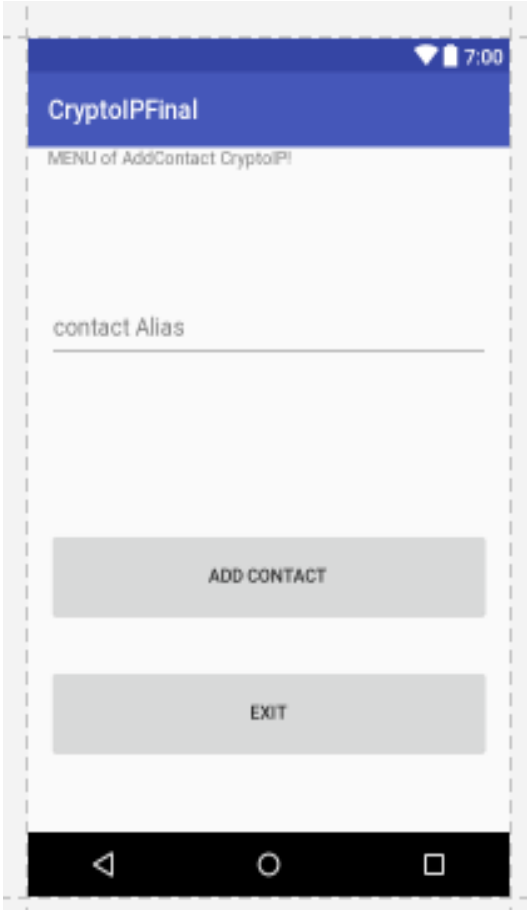
Στην συνέχεια ο κώδικας θα περιλαμβάνει τη δημιουργία ενός Activity με όνομα αρχείου MainActivity. Αρχικά ορίζουμε την κλάση και δημιουργούμε τα επτά Intents και στη συνέχεια τα οκτώ πλήκτρα που θα εκκινούν τα intents. Στη συνέχεια κάνουμε override την μέθοδο onCreate 1)δημιουργούμε τα Intents 2)συσχετίζουμε τα πλήκτρα που δηλώσαμε παραπάνω με αυτά του

αρχείου xml. Στην συνέχεια μέσα στη onCreate() προσθέτουμε τους OnClickListeners για κάθε πλήκτρο. Αντίστοιχα δημιουργούμε και τους υπόλοιπους. Για να εκκινήσουμε το service χρησιμοποιήσαμε την μέθοδο boolean startService(Intent intent) για να τερματίσουμε το service την boolean stopService(Intent intent) και για να εκκινήσουμε τα υπόλοιπα Activities την boolean startActivity(Intent intent).Τέλος κλείνουμε μετά την εκκίνηση των intents το MainActivity με τη μέθοδο finish().

6.2 Το AddContact Activity

Το UserInterface του AddContact Activity θα αποτελείται από ένα textview που θα προτρέπει τον χρήστη να διαλέξει το όνομα της επαφής που θέλει να δημιουργήσει, ένα πεδίο εισαγωγής κειμένου EditText όπου θα εισάγεται το alias με το οποίο θα αποθηκευτεί η επαφή και ένα πλήκτρο το οποίο θα εκτελεί την λειτουργία.

Και εμφανίζει την παρακάτω οθόνη



Εικόνα 15: Οθόνη AddContact Activity

Στη συνέχεια κάνουμε override την μέθοδο onCreate(). Δηλώνουμε τα δύο πλήκτρα 1)buttonExitAddContact 2) buttonAddContact τα συσχετίζουμε με αυτά απο το αρχείο xml και δημιουργούμε τους Listeners για κάθε ένα από αυτά. Μέσα στον listener διαβάζουμε το String που

εισήγαγε ο χρήστης μέσω του EditText(το όνομα της επαφής. Στη συνέχεια καλούμε την μέθοδο addContact() και τέλος δημιουργούμε ένα Toast που ενημερώνει το χρήστη για το αποτέλεσμα της διαδικασίας. Στην συνέχεια θα δημιουργήσουμε την συνάρτηση addContact την οποία καλούμε όταν πατηθεί το κουμπί AddContact. Ο ορισμός της συνάρτησης είναι ο εξής: **public String addContact(String Cname).** Η τιμή που θα επιστρέψει τη addContact() θα χρησιμοποιηθεί για να ενημερωθεί ο χρήστης για το αποτέλεσμα. Η συνάρτηση αναμένει να βρεί ένα αρχείο KeyStore τύπου PKCS12 στον φάκελο PICTURES και ένα κωδικοποιημένο αρχείο με το όνομα mycert, στον ίδιο φάκελο, το οποίο περιέχει το πιστοποιητικό της επαφής που θα δημιουργήσουμε. Αν όλα πάνε καλά η συνάρτηση επιστρέφει success.

Ο κώδικας της addContact() είναι ο εξής.

```
public String addContact(String Cname) {
    try {
        File file = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURE
S), "keystore.p12");
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        FileInputStream instreamKS = new FileInputStream(file);
        keyStore.load(instreamKS, "password".toCharArray());
        instreamKS.close();
        if(keyStore.containsAlias(Cname)) {  return "alias already exists";   }
```

Δημιουργούμε ένα block try/catch για τον κώδικα. Στην συνέχεια δημιουργούμε ένα αντικείμενο τύπου File που αντιπροσωπεύει το αρχείο KeyStore. Δημιουργούμε ένα αντικείμενο KeyStore τύπου PKCS12, στην συνέχεια φορτώνουμε το KeyStore και κλείνουμε το αρχείο. Τέλος ελέγχουμε αν υπάρχει ήδη επαφή με το όνομα που διάλεξε ο χρήστης. Αν όλα πάνε καλά εκτελείται ο υπόλοιπος κώδικας της μεθόδου.

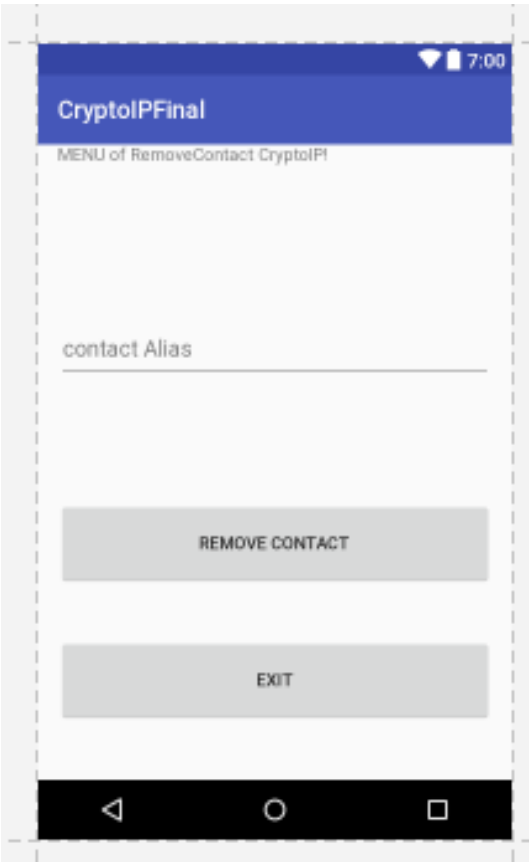
```
File certfile = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURE
S), "mycert");
if(!certfile.exists()) { return "please place the contact cert"; }
InputStream inStream = new FileInputStream(certfile);
CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate cert = (X509Certificate)cf.generateCertificate(inStream);
keyStore.setCertificateEntry(Cname, cert);
inStream.close();
OutputStream outStream = new FileOutputStream(file);
keyStore.store(outStream, "password".toCharArray());
```

Στη συνέχεια δημιουργούμε ένα αντικείμενο τύπου File που αντιπροσωπεύει το αρχείο με το κωδικοποιημένο πιστοποιητικό. Ελέγχουμε αν υπάρχει. Άν δεν υπάρχει η συνάρτηση επιστρέφει το κατάλληλο μήνυμα. Αν υπάρχει ανοίγει το αρχείο, δημιουργεί ένα αντικείμενο CertificateFactory και παράγει ένα πιστοποιητικό X.509 από το κωδικοποιημένο αρχείο. Το παραχθέν πιστοποιητικό αποθηκεύεται στο keystore με τη μέθοδο setCertificateEntry(). Στη συνέχεια κλείνουμε το αρχείο και αποθηκεύουμε το ανανεωμένο keystore πίσω στο αρχείο. Τέλος βρίσκεται ο κώδικας catch για κάθε exception που μπορεί να προκύψει στο κώδικα.

6.3 Το RemoveContact Activity

Το UserInterface του RemoveContact Activity θα αποτελείται από ένα textview που θα προτρέπει τον χρήστη να διαλέξει το όνομα της επαφής που θέλει να διαγράψει, ένα πεδίο εισαγωγής κειμένου EditText όπου θα εισάγεται το alias το οποίο θα διαγραφεί και ένα πλήκτρο το οποίο θα εκτελεί την λειτουργία.

Και εμφανίζει την παρακάτω οθόνη



Εικόνα 16: Οθόνη RemoveContact Activity

Στη συνέχεια κάνουμε override την μέθοδο onCreate(). Δηλώνουμε τα δύο πλήκτρα 1)buttonExitRemoveContact 2) buttonRemoveContact τα συσχετίζουμε με αυτά από το αρχείο xml και δημιουργούμε τους Listeners για κάθε ένα από αυτά. Μέσα στον listener διαβάζουμε το String που εισήγαγε ο χρήστης μέσω του EditText το οποίο αντιπροσωπεύει το όνομα της επαφής. Στη συνέχεια καλούμε την μέθοδο removeContact() και τέλος δημιουργούμε ένα Toast που ενημερώνει το χρήστη για το αποτέλεσμα της διαδικασίας.
Ο κώδικας της removeContact() είναι ο εξής.

```
public String removeContact(String Cname) {
    try {
        File file = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURE
S), "keystore.p12");
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        FileInputStream instreamKS = new FileInputStream(file);
        keyStore.load(instreamKS, "password".toCharArray());
```

```
        instreamKS.close();
        if(!keyStore.containsAlias(Cname)) {   return "alias doesnt exist";    }
```

Εδώ όπως και στο AddContact Activity δημιουργούμε ένα try/catch block κώδικα. Δημιουργούμε ένα αντικείμενο File που αντιπροσωπεύει το αρχείο KeyStore και ένα αντικείμενο keystore τύπου PKCS12. Στη συνέχεια φορτώνουμε το keystore ελέγχουμε εάν υπάρχει το alias που εισήγαγε ο χρήστης. Αν δεν υπάρχει η συνάρτηση επιστρέφει το κατάλληλο μήνυμα, αν ναι συνεχίζει και πραγματοποιεί την διαδικασίας διαγραφής.

```
keyStore.deleteEntry(Cname);
OutputStream outStream = new FileOutputStream(file);
keyStore.store(outStream, "password".toCharArray());
```

Τέλος βρίσκεται ο κώδικας catch για κάθε exception που μπορεί να προκύψει στο κώδικα.
Τέλος αποθηκεύει το ανανεωμένο keystore και επιστρέφει το κατάλληλο μήνυμα.

6.4 To Service

6.4.1 To Service1

Το service για την περίπτωση που ο κόμβος έχει public IP είναι το παρακάτω. Το Service δεν έχει κάποιο User Interface. Εκκινείται και σταματάει από το Main Activity και το Initialize Activity. Θα αποδέχεται τρία διαφορετικά μηνύματα με τις κατάλληλες παραμέτρους και θα εκκινεί τα αντίστοιχα Activities περνώντας αυτές τις παραμέτρους. Τα μηνύματα είναι τα ακόλουθα:

Friend Request
Το μήνυμα θα έχει την μορφή FRIEND:<contactName>:<encodedCert>
Όπου η πρώτη παράμετρος το όνομα της επαφής και η δεύτερη η ταυτότητα της.
Incoming Call
Το μήνυμα θα έχει τη μορφή CALL:<contactName>:<IP>
Όπου η πρώτη παράμετρος το όνομα της επαφής και η δεύτερη η IP της.
IDValidation
Το μήνυμα θα έχει την μορφή WEBOFTRUST:<contactname>:<encodedCert>:<certContactName>
Όπου η πρώτη παράμετρος το όνομα της επαφής που ζητά την επιβαιβέωση ταυτότητας η δεύτερη το κωδικοποιημένο X.509, τρίτη το όνομα της επαφής που αντιστοιχεί στο πιστοποιητικό και τέταρτη η IP του αιτούντος.
Το Service θα υλοποιεί νήμα για να πραγματοποιεί την λειτουργία του έτσι αφού ορίσουμε την κλάση κάνουμε override τη μέθοδο onCreate() όπου θα δημιουργήσουμε το αντικείμενο τύπου Server και θα το εκκινήσουμε με κλήση στη μέθοδο start(). Κάνουμε override την onDestroy() όπου σταματάμε το αντικείμενο τύπου Server με κλήση στην stop().

Στη συνέχεια θα δημιουργήσουμε ένα νέο αρχείο JAVA όπου θα ορίσουμε το αντικείμενο Server που χρησιμοποιήσαμε παραπάνω. Το αντικείμενο Server θα είναι νήμα οπότε θα κάνουμε override τη μέθοδο run(). Πέρα από την run() θα υπάρχουν άλλες τέσσερις μέθοδοι. Η start() η οποία θα εκκινεί το νήμα η stop() η οποία θα το σταματάει η μέθοδος Server() η οποία είναι η μέθοδος κατασκευαστής και η handle(Socket socket) η οποία θα υλοποιεί όλη τη λειτουργικότητα. Συνεχίζοντας κάνουμε override τη μέθοδο run(). Μέσα στη run() δημιουργείται ένα ServerSocket. Στη συνέχεια μέσω ενός while βρόχου ο κώδικας μπλοκάρει την λειτουργία μέχρι να συνδεθεί κάποιος πελάτης μέσω της accept(). Όταν γίνει σύνδεση καλείται η handle(). Τέλος μόλις επιστρέψει η handle() κλείνουμε το socket που δημιουργήθηκε και η διαδικασία επαναλαμβάνεται

έως ότου γίνει κλήση στη stop().

Τέλος ορίζουμε την μέθοδο private void handle(Socket socket). Η μέθοδος αυτή θα πραγματοποιεί όλες τις ενέργειες από τη στιγμή που κάποιος client συνδεθεί. Αφού ορίσουμε τις μεταβλητές μας δημιουργούμε ένα try/catch block όπου διαβάζουμε το εισερχόμενο μήνυμα διαμορφωμένο όπως ορίσαμε στην αρχή.

```
private void handle(Socket socket) throws IOException {
    BufferedReader reader = null;
    PrintStream output = null;
    try {
        String certContactName;
        String name;
        String IP;
        String cert;
        String mode;
        boolean flag = false;
        reader = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        String line;
        byte[] certificate=null;
        line = reader.readLine();
```

Αφού διαβάσουμε το μήνυμα ακολουθεί η εξαγωγή των παραμέτρων ανάλογα με το τι αίτημα λάβαμε. Ενδεικτικά για το αίτημα φιλίας ο ακόλουθος κώδικας εξάγει τις παραμέτρους όνομα και ταυτότητα όπως φαίνεται παρακάτω:

```
if (line.startsWith("FRIEND")) {

    flag = true;
    String info[] = line.split(":");
    name = info[1];
    mode = info[2];

    reader.close();
    InputStream in = socket.getInputStream();
    in.read(certificate);
    in.close();

    Intent ContactIntent = new Intent(mContext, ContactRequest.class);
    ContactIntent.putExtra("mode", mode);
    ContactIntent.putExtra("Name", name);
    ContactIntent.putExtra("ContactCert", certificate);
    startActivity(ContactIntent);
}
```

Στη συνέχεια δημιουργούμε το Intent που θα εκκινήσει το Activity ContactRequest, εισάγουμε τις παραμέτρους με τη μέθοδο putExtra() και εκκινούμε το Intent με τη μέθοδο startActivity(). Αντίστοιχα χειριζόμαστε όλα τα υπόλοιπα σενάρια που αφορούν αιτήματα. Εδώ να αναφέρουμε ότι το αίτημα επιβεβαίωσης ταυτότητας εξυπηρετείται κατευθείαν από το Service1 και δεν εκκινεί κάποιο Activity όπως γίνεται με το Service2. Τέλος αν το αίτημα που λάβαμε είναι λάθος στέλνουμε κατάλληλο μήνυμα

6.4.2 Το Service2

Το service αυτό θα χρησιμοποιείται για την περίπτωση που η εφαρμογή τρέχει με private IP. Η λειτουργία του θα αφορά σύνδεση στον Server και αποδοχή μηνυμάτων που στέλνουν άλλοι χρήστες προς τον κόμβο. Αφού δεχθεί το μήνυμα το αναλύει και είτε εκκινεί το κατάλληλο Activity είτε στέλνει ένα Broadcast σε κάποιο Activity. Στην συνέχεια κάνουμε override την μέθοδο onCreate. Εδώ μέσα σε ένα try/catch block κώδικα αρχικά ανακτούμε την διεύθυνση του server, δημιουργούμε ένα socket που συνδέεται στον server. Τέλος δημιουργούμε τα In/Out Streams και στέλνουμε το αίτημα στον server. Τέλος δημιουργώντας ένα βρόχο ακούμε για αιτήματα που αφορούν αυτό το κόμβο και όταν φτάσει κάποιο καλούμε την handle για να το χειριστεί. Τέλος κάνουμε catch πιθανά exceptions.
Η handle θα δέχεται ως είσοδο το αίτημα που λάβαμε από τον server και θα ξεχωρίζει περιπτώσεις ώστε να προωθήσει τα δεδομένα στα κατάλληλα μέρη της εφαρμογής Το αίτημα θα είναι της μορφής MODE:TYPE:ACTIVITY:FROM:TO:ENDPOINT1:ENDPOINT2. Δεν είναι απαραίτητες όλες οι παράμετροι για όλα τα μηνύματα. Περισσότερα για τις παραμέτρους και την μορφοποίηση του μηνύματος στο κεφάλαιο με την υλοποίηση του server. Αφού ορίσουμε την συνάρτηση και επεξεργαστούμε το αίτημα ακολουθεί η ανάλυση των περιπτώσεων. Αρχικά οι περίπτωση όπου το MODE είναι CR(Connection Reversing).
Η πρώτη υποπερίπτωση είναι το πεδίο activity του μηνύματος να είναι CALL. Σε αυτή την περίπτωση δημιουργούμε ένα intent με ενέργεια να εκκινήσει το InboundCall Activity, εισάγει τις παραμέτρους και εκτελεί την startActivity όπως φαίνεται παρακάτω.

```
public String handle(String inline) {
    String[] result;
    result = inline.split(":");
    if(result[0].equals("CR")) {
        if(result[2].equals("CALL")) {
            Intent CallIntent = new Intent(this, InboundCallActivity.class);
            CallIntent.putExtra("mode", result[0]);
            CallIntent.putExtra("fromAlias", result[3]);
            CallIntent.putExtra("IP", result[4]);
            startActivity(CallIntent);
        }
```

Η δεύτερη υποπερίπτωση είναι το πεδίο activity να είναι ADDREMOTECONTACT. Εδώ και πάλι δημιουργούμε ένα intent εισάγουμε τις παραμέτρους και εκκινούμε το Activity ContactRequest όπως κάναμε και στην πρώτη περίπτωση
Η τρίτη υποπερίπτωση είναι το πεδίο activity να είναι WEBOFTRUST. Εδώ εξάγουμε παραμέτρους και εκκινούμε το κατάλληλο Activity όπως κάναμε και στην πρώτη περίπτωση.

Στην συνέχεια διακρίνουμε την περίπτωση το πεδίο mode να είναι HP και το πεδίο type RESPONSE. Όλα τα αιτήματα αυτού του είδους απευθύνονται σε ενεργά Activities οπότε στέλνουμε ένα broadcast με κατάλληλο action και όλες τις παραμέτρους. Το broadcast θα αφορά μέρη της δικής μας εφαρμογής και δεν θα μπορεί να ληφθεί από άλλες εφαρμογές.
Ο κώδικας φαίνεται παρακάτω.

```
else if(inline.startsWith("HP:RESPONSE")) {
    Intent intent = new Intent();
    intent.setAction("com.example.iasonas.cryptoipfinal.HP"+result[2]);
    intent.putExtra("mode", result[0]);
```

```
    intent.putExtra("type", result[1]);
    intent.putExtra("fromAlias", result[3]);
    intent.putExtra("toAlias", result[4]);
    intent.putExtra("endpoint1", result[5]);
    intent.putExtra("endpoint2", result[6]);
    LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
}
```

Η τελευταία περίπτωση είναι το mode να είναι HP και το type REQUEST. Η περίπτωση αυτή έχει τρείς υποπεριπτώσεις. Η πρώτη από αυτές είναι το πεδίο activity να είναι CALL. Εξάγουμε τις παραμέτρους δημιουργούμε το intent, περνάμε τις παραμέτρους και εκκινούμε το InboundCall Activity όπως φαίνεται παρακάτω

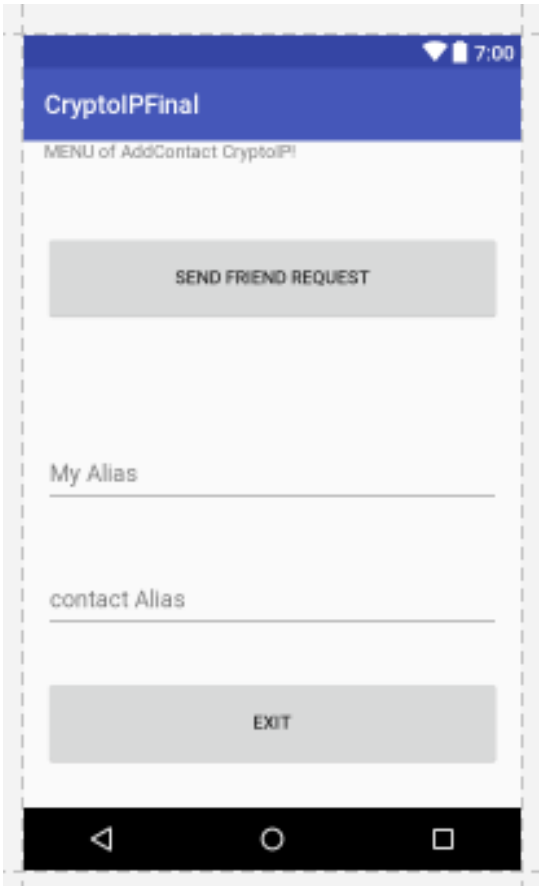
```
else if(inline.startsWith("HP:REQUEST")) {
    if(result[2].equals("CALL")) {
        Intent CallIntent = new Intent(this, InboundCallActivity.class);
        CallIntent.putExtra("mode", result[0]);
        CallIntent.putExtra("type", result[1]);
        CallIntent.putExtra("fromAlias", result[3]);
        CallIntent.putExtra("toAlias", result[4]);
        CallIntent.putExtra("endpoint1", result[5]);
        CallIntent.putExtra("endpoint2", result[6]);
        startActivity(CallIntent);
    }
```

Η δεύτερη υποπερίπτωση είναι το πεδίο activity να είναι ADDREMOTECONTACT. Και σε αυτήν τη περίπτωση κάνουμε ότι και στην προηγούμενη. Η τελευταία υποπερίπτωση είναι το πεδίο activity να είναι WEBOFTRUST. Και σε αυτήν την περίπτωση κάνουμε ότι κάναμε και στις προηγούμενες υποπεριπτώσεις.

6.5 Το AddRemoteContactActivity

Το Activity αυτό θα στέλνει κατάλληλο μήνυμα στο Service ή τον server αφού ανιχνεύσει με πιο τρόπο θα πραγματοποιηθεί η επικοινωνία, όπου θα ζητάει από το χρήστη της εφαρμογής να τον προσθέσει σαν επαφή. Το UI θα αποτελείται από ένα πλήκτρο που θα τερματίζει το Activity, ένα που θα στέλνει το κατάλληλο μήνυμα στην επαφή, ένα edittext όπου θα επιλέγει το alias του και το alias της επαφής στην οποία θα στείλει το αίτημα.

Και η οθόνη που βλέπει ο χρήστης είναι η παρακάτω:



Εικόνα 17: Οθόνη AddRemoteContact

Στη συνέχεια θα γράψουμε το κώδικα για το Activity. Αρχικά δημιουργούμε την κλάση και τις κατάλληλες μεταβλητές που χρειαζόμαστε κάνουμε override τη μέθοδο onCreate και συσχετίζουμε τα συστατικά του UI Στη συνέχεια δημιουργούμε τους listeners για τα δύο πλήκτρα. Ο listener για το πλήκτρο exit απλά καλεί τη finish ενώ ο listener για το πλήκτρο sendrequest ανακτά τις IP των κόμβων που θα επικοινωνήσουν και αποφασίζει με πιο τρόπο θα γίνει η επικοινωνία. Ανάλογα με πιο τρόπο θα γίνει η επικοινωνία εκτελούνται και οι κατάλληλες ενέργειες.

Mode Normal/Normal1

Στη συνέχεια αν το mode λειτουργίας είναι Normal/Normal1 εκτελείται η μέθοδος sendrequest όπως φαίνεται παρακάτω. Η sendrequeest συνδέεται στον κόμβο και στέλνει το σωστά μορφοποιημένο αίτημα.

```
if(mode.equals("Normal") || mode.equals("Normal1")) {
    result = sendrequest(ContactName, myName);
}
```

Η μέθοδος αυτή θα στέλνει στην κατάλληλη επαφή το αίτημα φιλίας. Θα ανακτά το πιστοποιητικό του χρήστη της εφαρμογής και θα το στέλνει κωδικοποιημένο στην επαφή. Η μέθοδος θα παίρνει δύο παραμέτρους , το alias του χρήστη και το alias της επαφής και θα επιστρέφει μία μεταβλητή String. Θα ανοίγει το keystore θα ανακτά το πιστοποιητικό που αντιστοιχεί στο keypair του χρήστη

της εφαρμογής, στη συνέχεια θα ανακτά την διεύθυνση του χρήστη από τη βάση δεδομένων και θα στέλνει το αίτημα. Όλη η λειτουργικότητα θα πραγματοποιείται με υλοποίηση ενός AsyncTask.

Mode CR

Αν mode είναι CR τότε εκτελούνται οι παρακάτω ενέργειες.

```
if(mode.equals("CR")) {
    serverIP = getIP("server");
    request = "CR:" + myName + ":" + ContactName + ":" + "ADDREMOTECONTACT" +
myIP;
    getMyCert();
    sendrequestCR();
    sendmessageCR();
}
```

Αρχικά ανακτούμε την διεύθυνση του server με κλήση στην getIP η οποία εκτελεί το κατάλληλο query στην βάση δεδομένων , στην συνέχεια δημιουργούμε το request που θα στείλουμε στον server, ανακτούμε το πιστοποιητικό με κλήση στην getMyCert, στέλνουμε το request στον server διευθυνσιοδότησης με κλήση στην sendrequestCR και τέλος λαμβάνουμε το μήνυμα με κλήση στην sendmessageCR.

Mode HP

Εδώ αν το mode λειτουργίας είναι HP εκτελούνται οι παρακάτω ενέργειες.

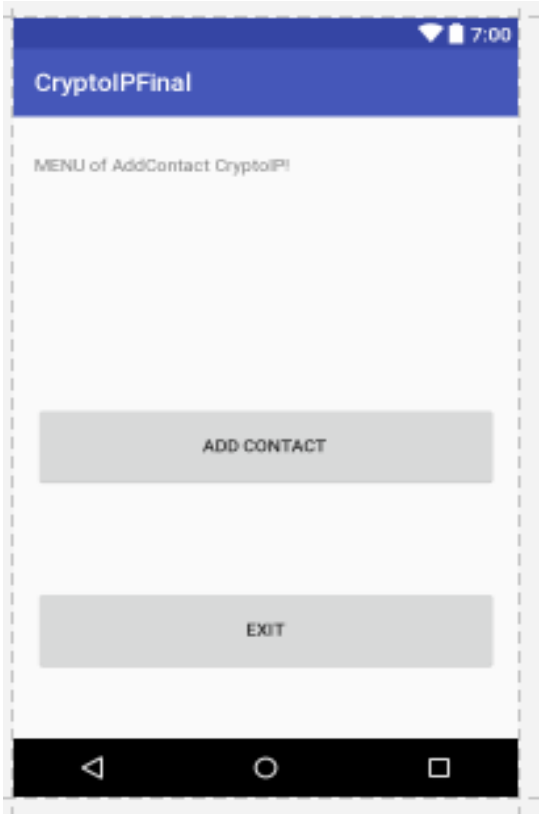
```
if(mode.equals("HP")) {
    serverIP = getIP("server");
    request = "HP:REQUEST:" + myName + ":" + ContactName + ":" +
"ADDREMOTECONTACT";
    sendrequestHP();

    while(!responseflag) {}

    initHP();
    getMyCert();
    sendmessageHP();
}
```

Εδώ αρχικά ανακτούμε την IP του server, στη συνέχεια δημιουργούμε το request και το στέλνουμε στον server με την sendrequestHP, ταυτόχρονα η μέθοδος αυτή δημιουργεί και τα δύο UDP Sockets που θα χρησιμοποιήσουμε για την επικοινωνία. Στην συνέχεια μπλοκάρουμε την εκτέλεση μέχρι που η μεταβλητή responseflag να γίνει true. Αυτό θα συμβεί μόλις φτάσει το HP:RESPONSE μήνυμα μέσω του service το οποίο θα στείλει broadcast στο Activity. Στην συνέχεια θα εκτελεστεί η initHP η οποία θα στείλει από τα δύο UDP Sockets που δημιουργήσαμε πακέτα προς τα δύο endpoints που λάβαμε μέσω του HP:RESPONSE μηνύματος ώστε να αλλάξει ο κανόνας στο NAT. Στην συνέχεια εκτελείται η getMyCert η οποία είναι η ίδια με αυτήν που εκτελέστηκε στο Mode CR. Τέλος εκτελείται η sendmessageHP. Η μέθοδος αυτή θα στείλει το πακέτο στον κόμβο χρησιμοποιώντας το ένα socket που δημιουργήθηκε κατά την αποστολή του HP:REQUEST μηνύματος και το ένα endpoint που έλαβε κατά την παραλαβή του HP:RESPONSE μηνύματος.

6.6 Το ContactRequest Activity

Το activity αυτό θα εκκινείται με intent από το service όταν ληφθεί το κατάλληλο μήνυμα και θα δίνει στον χρήστη την δυνατότητα να προσθέσει απομακρυσμένα μια επαφή. Θα αποτελείται από δύο πλήκτρα και ένα TextView.

Και η οθόνη που βλέπει ο χρήστης η παρακάτω



Εικόνα 18: Οθόνη ContactRequest Activity

Στην συνέχεια γράφουμε το κώδικα του Activity. Αρχικά ορίζουμε την κλάση και δηλώνουμε τις μεταβλητές που θα χρησιμοποιήσουμε. Στην συνέχεια κάνουμε override τη μέθοδο onCreate όπου αρχικά θέτουμε το UI που χρησιμοποιούμε, παίρνουμε τις παραμέτρους που συνοδεύουν το Intent που εκκίνησε το Activity ανάλογα σε ποίο mode βρισκόμαστε. Στη συνέχεια δημιουργούμε τους listeners για τα δύο πλήκτρα. Το πλήκτρο buttonDeny απλά καλεί τη μέθοδο finish ενώ το πλήκτρο buttonAccept αφού ελέγξει σε τί mode λειτουργεί η εφαρμογή πραγματοποιεί τις απαραίτητες ενέργειες. Αρχικά αν το mode είναι Normal/Normal1 καλεί την addToKeyStore με τις παραμέτρους που εξάγαμε προηγουμένως.

Mode Normal / Normal1

```
if(mode.equals("Normal") || mode.equals("Normal1")) {
    result = addToKeyStore(name, key);
}
```

Εδώ καλούμε την addToKeyStore με τις παραμέτρους που εξάγαμε απο το intent.

Mode CR

Αν το mode είναι CR τότε καλεί την getMessageCR για να λάβει το κωδικοποιημένο πιστοποιητικό του χρήστη και στην συνέχεια καλεί την addToKeyStore για να εισάγει στο keystore. Ο κώδικας φαίνεται παρακάτω.

```
else if(mode.equals("CR")) {
    getMessageCR();
   result = addToKeyStore(from, key);
}
```

Η μέθοδος getMessageCR δημιουργεί τη σύνδεση με τον άλλο κόμβο και δέχεται το πιστοποιητικό του άλλου κόμβου.

Mode HP

Η τρίτη περίπτωση είναι αν το mode είναι HP όπως φαίνεται παρακάτω

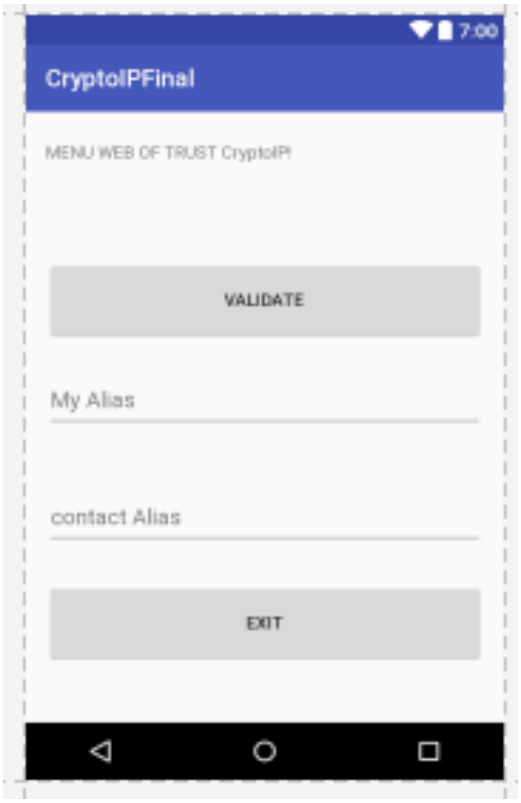
```
else if(mode.equals("HP")) {
    serverIP = getIP("server");
    request = "HP:RESPONSE:" + myName + ":" + name + ":" + "ADDREMOTECONTACT";
    sendresponseHP();
    initHP();
    receivemessageHP();
    result = addToKeyStore(name, key);
}
```

Εδώ αρχικά ανακτούμε την διεύθυνση του server δημιουργούμε το request που θα στείλουμε πίσω στο Activity που έστειλε το αντίστοιχο request και το στέλνει με κλήση στην sendresponseHP. Στην συνέχεια η μέθοδος initHP χρησιμοποιείται για να στείλει ένα πακέτο στο endpoint του άλλου κόμβου ώστε να αλλάξει ο κανόνας στο NAT. Στην συνέχεια εκτελείται η receivemessageHP. Η μέθοδος αυτή θα λάβει το πιστοποιητικό όταν βρισκόμαστε σε mode λειτουργίας HP μέσω των δύο sockets και των δύο endpoints που έχουμε διαθέσιμα. Εδώ χρησιμοποιούμε μόνο το ένα socket/endpoint.
Τέλος καλούμε την addToKeystore για να αποθηκεύσουμε το κλειδί. Ημέθοδος αυτή αρχικά δημιουργεί ένα αρχείο με όνομα remotecert σε συγκεκριμένο φάκελο της συσκευής όπου αποθηκεύει το πιστοποιητικό που έλαβε απο τον κόμβο. Στη συνέχεια ανοίγει το keystore και ελέγχει αν υπάρχει επαφή με αυτό το alias. Αν δεν υπάρχει καταχωρεί την επαφή και κλείνει το keystore.

6.7 To WebOfTrust Activity

Το Activity αυτό θα στέλνει αίτημα επιβεβαίωσης ταυτότητας σε όλες τις επαφές του χρήστη και θα αναφέρει πόσες απο αυτές επιβεβαιώνουν την ταυτότητα. Το UI του Activity θα αποτελείται από ένα πεδίο EditText όπου θα εισάγεται το όνομα της επαφής της οποίας θέλουμε να επιβεβαιώσουμε την ταυτότητα, ένα ακόμη EditText όπου θα εισάγεται το alias του χρήστη, ένα textView όπου θα εμφανίζεται το αποτέλεσμα και δύο Button, το πρώτο θα εκκινεί την διαδικασία και το δεύτερο θα τερματίζει το Activity. Το Activity θα ανιχνεύει σε πιο mode θα λειτουργήσει και θα εκτελεί τις απαραίτητες ενέργειες για κάθε περίπτωση.

Και η οθόνη που θα βλέπει ο χρήστης η παρακάτω.



Εικόνα 19: Οθόνη WebOfTrust Activity

Στη συνέχεια ακολουθεί ο κώδικας του WebOfTrust Activity. Αρχικά δημιουργούμε την κλάση και δηλώνουμε τις παραμέτρους. Στη συνέχεια κάνουμε override την μέθοδο onCreate συσχετίζουμε τα συστατικά του UI με τις μεταβλητές μας. Στη συνέχεια δημιουργούμε τους listeners για τα δύο πλήκτρα. Αρχικά παίρνουμε το όνομα του χρήστη και της επαφής της οποίας θέλουμε να επιβεβαιώσουμε την ταυτότητα από το UI. Στη συνέχεια ανακτούμε την IP του χρήστη της εφαρμογής με κλήση στην μέθοδο getIP και τέλος εκτελούμε την διαδικασία validate με τις κατάλληλες παραμέτρους. Τέλος εμφανίζουμε το αποτέλεσμα στο textview. Η μέθοδος getIP θα διαβάζει μία καταχώρηση από μία βάση δεδομένων η οποία δημιουργείται και ενημερώνεται από άλλο συστατικό της εφαρμογής είναι ίδια με αυτήν που χρησιμοποιήθηκε σε συστατικά μέρη της εφαρμογής που περιγράφηκαν προηγουμένως. Στη συνέχεια εκτελείται η μέθοδος validate η οποία θα στείλει αίτημα επιβεβαίωσης σε κάθε κόμβο που έχουμε αποθηκεύσει στην βάση δεδομένων.
Αρχικά η μέθοδος ανοίγει το keystore και ανακτά όλες τις διαθέσιμες επαφές καθώς και το πιστοποιητικό του χρήστη που θέλουμε να επιβεβαιώσουμε. Τέλος δημιουργεί το μήνυμα που θα

σταλεί όπως φαίνεται παρακάτω.

```
public int validate(String cName, String MyName, String myIP){
    try {
        File file = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURE
S), "keystore.p12");
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        FileInputStream instreamKS = new FileInputStream(file);
        keyStore.load(instreamKS, "password".toCharArray());
        instreamKS.close();
        X509Certificate peerCert = (X509Certificate)
keyStore.getCertificate(cName);
        peer = peerCert.getPublicKey();
        cert = peer.toString();
        message = "WEBOFTRUST:" + MyName + ":" + cert + ":" + cName + ":" + myIP
;
```

Στη συνέχεια για κάθε alias που υπάρχει στο keystore υπολογίζει με πιο τρόπο θα γίνει η σύνδεση.

```
for (Enumeration<String> aliases = keyStore.aliases();
aliases.hasMoreElements();) {
    String alias = aliases.nextElement();
    cIP = getIP(alias);
    if(myIP.equals("0.0.0.0")&cIP.equals("0.0.0.0")) {
        mode = "HP";
    } else if(cIP.equals("0.0.0.0")) {
        mode = "CR";
    }else if(myIP.equals("0.0.0.0")) {
        mode = "Normal1";
    }
    else { mode = "Normal"; }
```

Στη συνέχεια ανάλογα με το mode που βρισκόμαστε εκτελούνται οι κατάλληλες ενέργειες.

Mode Normal/Normal1

Σε αυτό το mode το αίτημα θα εξυπηρετηθεί από το service του κόμβου.

```
if(mode.equals("Normal") || mode.equals("Normal1") ) {
    sendmessage();
}
```

Η μέθοδος αυτή είναι παρόμοια με αυτήν που χρησιμοποιήθηκε σε άλλα Activities (AddRemoteContact). Η μέθοδος αυτή συνδέεται με το service του κόμβου, στέλνει το μήνυμα που δημιουργήθηκε προηγουμένως και δέχεται το response.

Mode CR

```
else if(mode.equals("CR")) {
    request = "CR:" + me + ":" + alias + ":" + "WEBOFTRUST:" + myIP;
    sendrequestCR();
```

```
    sendmessageCR();
}
```

Σε αυτό το mode το αίτημα θα εξυπηρετηθεί από το Activity Validate. Αρχικά κατασκευάζουμε το request που θα στείλουμε στον server. Με κλήση στην sendrequestCR συνδεόμαστε με τον server και στέλνουμε το μήνυμα. Η μέθοδος αυτή είναι ίδια με αυτή που χρησιμοποιήθηκε σε άλλα Activities(AddRemoteContact). Στην συνέχεια στέλνουμε το μήνυμα με κλήση στην sendmessageCR. Η μέθοδος αυτή είναι ίδια με αυτή που χρησιμοποιήθηκε σε άλλα Activities(AddRemoteContact).Δημιουργεί ένα serverSocket όπου θα συνδεθεί ο κόμβος για να στείλουμε το μήνυμα. Αφού σταλεί το μήνυμα λαμβάνουμε το response.

Mode HP

```
else if (mode.equals("HP")) {
    serverIP = getIP("server");
    request = "HP:REQUEST:" + me + ":" + alias + ":" + "WEBOFTRUST";
    sendrequestHP();
    while(!responseflag) {}
    initHP();
    sendmessageHP();
    receivemessageHP();
}
```

Σε αυτό το mode το αίτημα θα εξυπηρετηθεί από το Activity Validate. Αρχικά κατασκευάζουμε το request που θα στείλουμε στον server. Με κλήση στην sendrequestHP συνδεόμαστε με τον server και στέλνουμε το μήνυμα. Η μέθοδος αυτή είναι ίδια με αυτή που χρησιμοποιήθηκε σε άλλα Activities(AddRemoteContact). Στην συνέχεια περιμένουμε μέχρι να ενεργοποιηθεί ο Broadcast Receiver και να εκτελεστεί η onReceive η οποία θα ξεμπλοκάρει την εκτέλεση και θα μας δώσει τα δύο UDP endpoints. Ακολουθεί η κλήση στην μέθοδο initHP η οποία είναι ίδια με αυτή που χρησιμοποιήθηκε σε άλλα Activities(AddRemoteContact) και χρησιμοποιείται ώστε να αλλάξει ο κανόνας στο NAT ώστε να γίνονται δεκτά πακέτα απο το συγκεκριμένο endpoint. Ακολουθεί η sendmessageHP η οποία είναι ίδια με αυτή του AddRemoteContact Activity και χρησιμοποιείται για να σταλεί το request. Τέλος ακολουθεί κλήση στην receivemessageHP η οποία είναι ίδια με αυτή που χρησιμοποιήθηκε στο ContactRequest Activity και χρησιμοποιείται για να λάβουμε το response.
Τέλος για κάθε κόμβο που συνδεθήκαμε αν το response είναι θετικό αυξάνουμε έναν μετρητή και ενημερώνουμε τον χρήστη για το αποτέλεσμα η μέθοδος validate επιστρέφει τον αριθμό των επαφών που επιβεβαίωσαν την ταυτότητα. Το αποτέλεσμα προβάλλεται μέσω ενός textview.

6.8 To Validate Activity

Το Activity αυτό θα χρησιμοποιείται για κόμβους που τρέχουν το δεύτερο Service και θα απαντά σε αιτήματα που εκ κινούνται από το WebOfTrust Activity. Το activity αυτό θα λειτουργεί σε mode CR ή HP, δηλαδή όχι σε Normal/Normal1. Το UI αυτού του Activity θα αποτελείται από ένα textview που θα ενημερώνει το χρήστη ότι ένα αίτημα επιβεβαίωσης ταυτότητας εξυπηρετήθηκε με επιτυχία.
Συνεχίζοντας με την υλοποίηση αρχικά ορίζουμε την κλάση και δηλώνουμε τις απαραίτητες μεταβλητές. Στην συνέχεια κάνουμε override τη μέθοδο onCreate όπου συσχετίζουμε με το UI και

μέσω του intent με βάση το οποίο εκ κινήθηκε το Activity ενημερωνόμαστε σε τί mode βρισκόμαστε.

Mode CR

```
mode = ContactRequestIntent.getStringExtra("mode");
if (mode.equals("CR")) {
    from = ContactRequestIntent.getStringExtra("fromAlias");
    fromIP = ContactRequestIntent.getStringExtra("IP");
    displayContact.setText(from);
    getMessageCR();
}
```

Αν το mode είναι CR τότε εξάγουμε απο το intent τις απαραίτητες παραμέτρους και καλούμε την μέθοδο getmessageCR. Η μέθοδος αυτή συνδέεται στο socket που έχει ανοίξει το webOfTrust Activity και δέχεται το μήνυμα. Στη συνέχεια καλεί την μέθοδο validate και ενημερώνει τον χρήστη για το αποτέλεσμα.

Mode HP

Αν το mode είναι HP τότε εκτελείται ο παρακάτω κώδικας

```
else if(mode.equals("HP")) {
    from = ContactRequestIntent.getStringExtra("fromAlias");
    myalias = ContactRequestIntent.getStringExtra("toAlias");
    UDPendpoint1 = ContactRequestIntent.getStringExtra("endpoint1");
    UDPendpoint2 = ContactRequestIntent.getStringExtra("endpoint2");
    displayContact.setText(from);

    serverIP = getIP("server");
    response = "HP:RESPONSE:" + myalias + ":" + from + ":" + "WEBOFTRUST";
    sendresponseHP();
    initHP();
    receivemessageHP();
    String requestsplit[] = Rresponse.split(":");
    result = validate(requestsplit[3], requestsplit[2]);

    sendmessageHP();
}
```

Εδώ αρχικά εξάγουμε τις παραμέτρους απο το intent που εκκίνησε το Activity. Στη συνέχεια ανακτούμε τη διεύθυνση του server με χρήση της getIP την οποία έχουμε ξαναδεί σε άλλα μέρη της εφαρμογής, μετά κατασκευάζουμε το request που θα στείλουμε στον server με κλήση στη μέθοδο sendresposneHP η οποία είναι ίδια με αυτήν που χρησιμοποιήθηκε σε άλλα Activities(ContactRequest). Ακολουθεί κλήση στην initHP η οποία χρησιμοποιείται για να αλλάξει τον κανόνα αντιστοίχησης στο NAT στέλνοντας ένα πακέτο σε κάθε endpoint από το κατάλληλο socket. Η μέθοδος αυτή είναι ίδια με αυτές που χρησιμοποιήθηκαν σε άλλα μέρη της εφαρμογής. Ακολουθεί κλήση στην receivemessageHP η οποία χρησιμοποιείται για να λάβει το request που στέλνει το WebOfTrust Activity το οποίο και εκκίνησε την διαδικασία. Ο κώδικας είναι

ίδιος με αυτόν που χρησιμοποιήθηκε σε άλλα Activities(ContactRequest) . Από το μορφοποιημένο μήνυμα που λάβαμε εξάγουμε το alias και το κλειδί και τα περνάμε στην validate. Τέλος στέλνουμε το response στο Activity που εκκίνησε τη διαδικασία με κλήση στην sendmessageHP η οποία είναι ίδια με άλλες που χρησιμοποιήσαμε σε άλλα Activities(AddRemoteContact).

Η μέθοδος validate αρχικά ανοίγει το keystore στην συνέχεια ανακτά το κλειδί του alias που πέρασε ως παράμετρος στη μέθοδο και το μετατρέπει σε String. Τέλος συγκρίνει το String αυτό με αυτό που λάβαμε για επιβεβαίωση και επιστρέφει το αποτέλεσμα.

6.9 Το OutboundCall Activity

Το Activity αυτό θα εκκινείται απο το MainActivity. Οι δύο βασικές ενέργειες που θα πραγματοποιεί είναι η διαδικασία handshake, ώστε οι δύο κόμβοι να καταλήξουν σε ένα συμμετρικό κλειδί για την επικοινωνία και η διαδικασία makeCall η οποία θα πραγματοποιεί την κλήση. Η makeCall μπορεί να χωριστεί και αυτή σε διαδικασίες βάση λειτουργικότητας σε διαδικασία παραγωγής δεδομένων, κρυπτογράφησης δεδομένων, αποστολής δεδομένων και αντίστοιχα αποδοχής δεδομένων, αποκρυπτογράφησης, κατανάλωσης δεδομένων. Το Activity αυτό επικοινωνεί με το InboundCall Activity και χρησιμοποιεί δύο μεθόδους για να στέλνει και να λαμβάνει δεδομένα την sendpacket και τη receivepascket. Για να λειτουργούν αυτές οι συναρτήσεις σε όλα τα mode λειτουργίας δηλαδή,
1)Normal όπου και οι δύο κόμβοι έχουν public IPs.
2) Normal1 όπου ο κόμβος που εκκινεί τη διαδικασία έχει private IP και ο άλλος κόμβος public.
3)CR Mode όπου ο κόμβος που εκκινεί τη διαδικασία έχει public IP και ο δεύτερος private.
Θα τροποποιήσουμε τον κώδικα τους ώστε να βρίσκονται σε κατάλληλο συνδυασμό server/client με τις αντίστοιχες μεθόδους του InboundCall.

- Για mode Normal

Όταν η receivepacket του OutboundCall βρίσκεται σε server mode η sendpacket του Inbound θα βρίσκεται σε client.

- Για Normal1

Όταν η receivepacket του OutboundCall βρίσκεται σε client mode η sendpacket του Inbound θα βρίσκεται σε server.

- Για mode CR

Όταν η receivepacket του OutboundCall βρίσκεται σε server mode η sendpacket του Inbound θα βρίσκεται σε client.

- Για mode Normal

Όταν η sendepacket του OutboundCall βρίσκεται σε client mode η receivepacket του Inbound θα βρίσκεται σε server.

- Για Normal1

Όταν η sendpacket του OutboundCall βρίσκεται σε client mode η receivepacket του Inbound θα βρίσκεται σε server.

- Για mode CR

Όταν η sendpacket του OutboundCall βρίσκεται σε server mode η receivepacket του Inbound θα βρίσκεται σε client

Με αυτό το τρόπο η μόνη τροποποίηση για να λειτουργεί κανονικά η εφαρμογή στα παραπάνω modes είναι η απόφαση για το πιο mode λειτουργίας έχουμε να αντιμετωπίσουμε και στη συνέχεια η ρύθμιση των μεθόδων που χρησιμοποιούμε για να υλοποιήσουμε τις μεθόδους sendpacket/receivepacket.

Με τη ίδια λογική αντιμετωπίζουμε και τα DatagramSockets που χρησιμοποιούμε για την αποστολή/λήψη δεδομένων ήχου.

- Για mode Normal

Όταν το datagramsocket sender OutboundCall βρίσκεται σε server mode το datagramsocket receiver θα βρίσκεται σε client mode.

- Για mode Normal1

Όταν το datagramsocket sender OutboundCall βρίσκεται σε client mode το datagramsocket receiver θα βρίσκεται σε server mode.

- Για mode CR

Όταν το datagramsocket sender OutboundCall βρίσκεται σε server mode το datagramsocket receiver θα βρίσκεται σε client mode.

- Για mode Normal

Όταν το datagramsocket receiver OutboundCall βρίσκεται σε server mode το datagramsocket sender θα βρίσκεται σε client mode.

- Για mode Normal1

Όταν το datagramsocket receiver OutboundCall βρίσκεται σε client mode το datagramsocket sender θα βρίσκεται σε server mode.

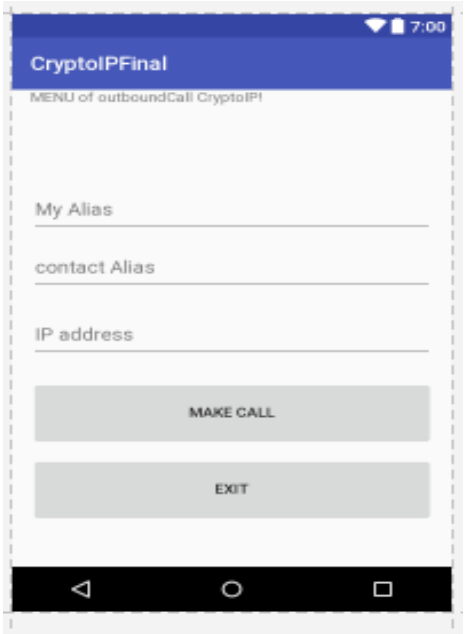
- Για mode CR

Όταν το datagramsocket receiver OutboundCall βρίσκεται σε server mode το datagramsocket sender θα βρίσκεται σε client mode.

Για το mode λειτουργίας HP αφού δημιουργήσουμε τα endpoints με τον ίδιο τρόπο που το κάναμε σε προηγούμενα μέρη της εφαρμογής θα δημιουργήσουμε δύο καινούργιες μεθόδους sendpacket/receivepacket και αφού πραγματοποιηθεί η εκτέλεση του αλγορίθμου DHE χρησιμοποιώντας ένα από τα δύο UDP sockets για μεταφορά/παραλαβή δεδομένων θα χρησιμοποιήσουμε και τα δύο UDPsockets για μεταφορά/παραλαβή δεδομένων ήχου.

Το user interface αυτού του activity θα αποτελείται απο ένα textfield όπου ο χρήστης θα εισάγει το όνομα της επαφής που θέλει να επικοινωνήσει και ένα ακόμα textfield στο οποίο θα εισάγει την IP.

Και η οθόνη που βλέπει ο χρήστης είναι η παρακάτω:



Εικόνα 20: Οθόνη OutboundCall Activity

Στη συνέχεια θα δημιουργήσουμε την κλάση OutboundCallActivity, θα ορίσουμε τις απαραίτητες μεταβλητές. Θα κάνουμε override τη μέθοδο onCreate() όπου θα συσχετίσουμε τα αντικείμενα του UI. Στη συνέχεια θα δημιουργήσουμε τους Listeners για τα πλήκτρα. Αρχικά ανακτούμε το alias του χρήστη της εφαρμογής και το alias της επαφής που θέλουμε να καλέσουμε και ανακτούμε τις IP τους από τη βάση δεδομένων με κλήση στην getIP, μέθοδο που χρησιμοποιήσαμε και σε άλλα μέρη της εφαρμογής. Στην συνέχεια ανιχνεύουμε το mode στο οποίο θα λειτουργήσει η εφαρμογή.

Mode Normal/Normal1

Εδώ αρχικά στέλνουμε το request στον κόμβο που θέλουμε να επικοινωνήσουμε. Στην συνέχεια καλούμε την OutHandshake η οπία θα εκτελέσει τον αλγόριθμο DHE. Στην συνέχεια εκτελούμε την makeCall η οποία και πραγματοποιεί την κλήση.

```
if(mode.equals("Normal") || mode.equals("Normal1")) {
    sendcallrequest();
    OutHandshake(contactName);
    makeCall(key, IV);
}
```

Στην μέθοδο sendcallrequest αρχικά δημιουργούμε το κατάλληλο request και στη συνέχεια συνδεόμαστε στον κόμβο, το στέλνουμε και τερματίζουμε τα streams. Τέλος εκτελείται η OutHandshake και η makeCall στις οποίες θα αναφερθούμε στη συνέχεια.

Mode CR

```
if(mode.equals("CR")) {
    sendrequestCR();
    OutHandshake(contactName);
    makeCall(key, IV);
}
```

Εδώ εκτελείται η μέθοδος sendrequestCR η οποία χρησιμοποιείται για να στείλει το request που δημιουργήσαμε κατα τη φάση ανίχνευσης του mode λειτουργίας. Η μέθοδος αυτή είναι ίδια με άλλες που χρησιμοποιήσαμε σε άλλα μέρη της εφαρμογής(AddRemoteContact). Τέλος όπως και στο mode Normal/Normal1 εκτελείται η OutHandshake και η makeCall.

Mode HP

```
else if(mode.equals("HP")) {
    request = "HP:REQUEST:" + myName + ":" + contactName + ":" + "CALL";
    sendrequestHP();
    while(!responseflag) {}
    initHP();
    OutHandshake(contactName);
    makeCall(key, IV);
}
```

Εδώ αρχικά δημιουργούμε το request και στη συνέχεια το στέλνουμε στον server με κλήση στη μέθοδο sendrequestHP η οποία στέλνει το request στον server. Η μέθοδος αυτή είναι ίδια με άλλες που χρησιμοποιήσαμε σε άλλα μέρη της εφαρμογής(AddRemoteContact). Στη συνέχεια μπλοκάρεται η εκτέλεση μέχρι να λάβει το response ο Broadcast Receiver του Activity. Ο Broadcast Receiver είναι ίδιος με αυτόν από τα AddRemoteContact, WebOfTrust. Ακολουθεί κλήση στην initHP η οποία επίσης είναι ίδια με προηγούμενες. Στη συνέχεια όπως και στις προηγούμενες περιπτώσεις εκτελείται η OutHandshake και η makeCall.

Η receivepacket

Αν το mode είναι HP τότε χρησιμοποιούμε τα UDP sockets και τα αντίστοιχα endpoints που δημιουργήθηκαν κατά τη διαδικασία αποστολής request για Hole Punching για να λάβει το μήνυμα από τον κόμβο. Η μέθοδος αυτή χρησιμοποιεί ACKs/ timeouts για να ληφθεί εγγυημένα το μήνυμα χρησιμοποιώντας το πρωτόκολλο UDP. Αν το mode δεν είναι HP τότε υλοποιούμε τις αντιστοιχίες client/server μεταξύ receivepacket/sendpacket που ορίσαμε στην αρχή της ενότητας. Αν το mode είναι Normal/CR δημιουργείται serversocket. Αν είναι το mode Normal1 δημιουργείται client socket.

Η sendpacket

Εδώ κάνουμε ότι κάναμε με την receivepacket με τη διαφορά ότι αυτή η μέθοδος χρησιμοποιείται για να στέλνουμε μηνύματα. Αν το mode είναι HP χρησιμοποιούμε τα UDP sockets και endpoints, αλλιώς εφαρμόζουμε τις αντιστοιχίσεις μεταξύ client/server sockets που αναφέραμε στην αρχή της ενότητας. Αν το mode είναι Normal/Normal1 έχουμε client socket με το οποίο στέλνουμε το πακέτο αφού γίνει σύνδεση. Αν το mode είναι CR τότε έχουμε ServerSocket.

Η διαδικασία OutHandshake

Στην συνέχεια ακολουθεί η OutHandshake. Αυτή η συνάρτηση θα παίρνει μία παράμετρο, το όνομα της επαφής που θέλουμε να επικοινωνήσουμε και θα αλληλεπιδρά με την InHandshake την οποία θα περιγράψουμε στην επόμενη ενότητα με στόχο να καταλήξουνε σε συμφωνία μυστικού κλειδιού για την επικοινωνία, δηλαδή να εκτελεστεί ο αλγόριθμος DHE. Τα βήματα που εκτελεί η OutHandshake είναι τα παρακάτω:

- Φόρτωση του keypair μου και του public της επαφής απο keystore
- Παραγωγή του ephemeral key pair μου
- Αποστολή του ephemeral public μου
- Υπογραφή του ephemeral public μου με private
- Αποδοχή του ephemeral public της επαφής
- Αποστολή υπογραφής
- Αποδοχή υπογραφής
- Έλεγχος υπογραφής
- Παραγωγή μυστικού
- Παραγωγή κλειδιού
- Παραγωγή IV String
- Αποστολή IV String

Στην συνέχεια ακολουθεί ο κώδικας της OutHandshake.

```
public void OutHandshake(String Cname) {

        loadMyKeyPair(contactName);

        loadmyEphemeralKeyPair();

        sendEphemeralPublic(myEphemeralPublicKey);

        signature = signEphemeralPublic(myPrivateKey, myEphemeralPublicKey);

        EphemeralPubliceReceived = receivePublic();

        sendSignature(signature);

        signatureReceived = receiveSignature();

        checkSignature(signatureReceived, peerPublicKey);

        secret = generateSecret(myEphemeralPrivateKey,
EphemeralPubliceReceived);

        secretKeyGenerated = generateSecretKey(secret);

        IV = generateIV();

        sendIV(IV);

    }
```

Στην συνέχεια θα αναλύσουμε τις μεθόδους που υλοποιούν τη λειτουργικότητα. Αυτές είναι οι

δώδεκα που φαίνονται παραπάνω.

Η loadMyKeyPair

Παίρνει μία παράμετρο, το όνομα της επαφής που επικοινωνούμε. Ανοίγει το keystore και ανακτά το δημόσιο κλειδί της επαφής και το ιδιωτικό κλειδί του χρήστη.

Η loadMyEphemeralKeyPair

Η μέθοδος αυτή παράγει ένα ζεύγος ιδιωτικού/δημόσιου κλειδιού για μελλοντική χρήση από τον αλγόριθμο DHE. Αρχικά δημιουργούμε ένα αντικείμενο KeyPairGenerator τύπου "DH", το αρχικοποιούμε με μέγεθος κλειδιού 2048 και στη συνέχεια με κλήση στην generateKeyPair παράγουμε το ζεύγος τα οποία και αποθηκεύουμε για μελλοντική χρήση.

Η sendEphemeralPublic

Παίρνει ως παράμετρο το ephemeral δημόσιο κλειδί μας και το στέλνει στην επαφή που θέλουμε να επικοινωνήσουμε, η μέθοδος στέλνει σε κωδικοποιημένη μορφή το κλειδί με χρήση της sendpacket.

Η signEphemeralPublic

Η μέθοδος αυτή παίρνει δύο παραμέτρους. Η πρώτη είναι το ιδιωτικό κλειδί και η δεύτερη είναι το ephemeral δημόσιο κλειδί το οποίο θα υπογράψουμε με το ιδιωτικό. Η μέθοδος επιστρέφει την υπογραφή σε ένα αντικείμενο τύπου byte[]. Αρχικά δημιουργούμε ένα αντικείμενο Signature το οποίο θα υλοποιήσει τον αλγόριθμο SHA256withRSA. Στη συνέχεια αρχικοποιούμε το αντικείμενο για χρήση υπογραφής με το ιδιωτικό μας κλειδί, περνάμε ως παράμετρο την κωδικοποιημένη μορφή του ephemeral δημοσίου κλειδιού, παράγουμε την υπογραφή και επιστρέφουμε την υπογραφή.

Η receivepublic

Η μέθοδος αυτή δέν παίρνει παραμέτρους και επιστρέφει ένα αντικείμενο τύπου PublicKey όπου θα αποθηκευτεί το ephemral public key της επαφής. Η συνάρτηση αυτή κάνει χρήση της συνάρτησης receivepacket την οποία αναλύσαμε προηγουμένως . Στη συνέχεια δημιουργούμε ένα αντικείμενο KeyFactory τύπου DH και ένα αντικείμενο X509EncodedKeySpec όπου περνάμε ως παράμετρο το κωδικοποιημένο πιστοποιητικό που λάβαμε. Τέλος παράγουμε το δημόσιο κλειδί και το επιστρέφουμε.

Η sendSignature

Η μέθοδος αυτή παίρνει μία παράμετρο την υπογραφή που προέκυψε από τη διαδικασία υπογραφής του ephemeral δημόσιου κλειδιού με το ιδιωτικό. Με χρήση της sendpacket στέλνει την υπογραφή.

Η receiveSignature

Η οποία παραλαμβάνει μέσω δικτύου την υπογραφή που προέκυψε από το ephemeral δημόσιο κλειδί της επαφής υπογεγραμμένο από το ιδιωτικό κλειδί της επαφής.

Η checkSignature

Η μέθοδος αυτή παίρνει δύο παραμέτρους. Η πρώτη είναι η υπογραφή που λάβαμε από την επαφή

με την οποία επικοινωνούμε και η δεύτερη είναι το δημόσιο κλειδί της επαφής. Η μέθοδος επαληθεύει την υπογραφή. Στη μέθοδο αυτή αρχικά δημιουργούμε ένα αντικείμενο τύπου Signature με παράμετρο τον αλγόριθμο SHA256withRSA. Στη συνέχεια αρχικοποιούμε το αντικείμενο για χρήση επαλήθευσης υπογραφής με παράμετρο το δημόσιο κλειδί της επαφής που επικοινωνούμε. Τέλος εκτελούμε την διαδικασία επαλήθευσης και επιστρέφουμε το αποτέλεσμα.

Η generateSecret

Η μέθοδος αυτή παίρνει δύο παραμέτρους η πρώτη είναι το ephemeral ιδιωτικό κλειδί του χρήστη της εφαρμογής και η δεύτερη είναι το ephemeral δημόσιο κλειδί της επαφής που επικοινωνούμε. Η μέθοδος επιστρέφει το παραχθέν μυστικό από την εκτέλεση του αλγορίθμου Key Agreement DH. Αρχικά δημιουργούμε ένα αντικείμενο KeyAggrement τύπου DiffieHellman. Στη συνέχεια το αρχικοποιούμε με το ephemeral ιδιωτικό κλειδί μας, εκτελούμε την μέθοδο dophase με παράμετρο το ephemeral δημόσιο κλειδί της επαφής που επικοινωνούμε και τέλος παράγουμε το μυστικό το οποίο επιστρέφει η συνάρτηση.

Η generateSecretKey

Η μέθοδος αυτή παίρνει μία παράμετρο τύπου byte[], το μυστικό που παρήγαγε η προηγούμενη μέθοδο. Με βάση αυτό δημιουργεί ένα αντικείμενο SecretKeySpec το οποίο είναι και το τελικό κλειδί που θα χρησιμοποιηθεί για την κρυπτογράφηση της επικοινωνίας. Η μέθοδος δημιουργεί ένα αντικείμενο SecretKeySpec με παράμετρο το μυστικό και τον αλγόριθμο AES. Επιστρέφει το παραχθέν κλειδί.

Η generateIV

Η συνάρτηση αυτή δέν παίρνει παραμέτρους και επιστρέφει ένα string που θα χρησιμοποιηθεί για την παραγωγή του IV. Εδώ δημιουργούμε ένα αντικείμενο SecureRandom το οποίο αρχικοποιούμε με την τιμή SHA512 και με κλήση στην nextBytes παράγουμε ένα τυχαίο αριθμό που αποθηκεύουμε στην μεταβλητή την οποία και επιστρέφουμε.

Η sendIV

Η συνάρτηση αυτή στέλνει τον IV στην επαφή που επικοινωνούμε χρησιμοποιώντας την sendpacket.

Η διαδικασία makeCall

Τέλος εκτελείται η makeCall. Η συνάρτηση αυτή παίρνει δύο παραμέτρους, το κλειδί και το String με βάση το οποίο θα δημιουργηθεί ο IV. Η συνάρτηση αυτή είναι σχεδόν ίδια με την answerCall της διαδικασίας InHandshake καθώς δημιουργεί δύο νήματα, ένα για να δέχεται/αποκρυπτογραφεί/αναπαράγει δεδομένα και ένα για να παράγει/κρυπτογραφεί/στέλνει δεδομένα. Εδώ φροντίζουμε να υλοποιήσουμε τις αντιστοιχίσεις μεταξύ client/server sockets με βάση το mode λειτουργίας που βρισκόμαστε. Για το mode HP θα χρησιμοποιήσουμε τα δύο sockets που δημιουργούμε κατά την αποστολή αίτήματος HP στο server και τα αντίστοιχα endpoints που λάβαμε στη συνέχεια.
Ακολουθεί ο κώδικας για το πρώτο νήμα το οποίο θα παράγει δεδομένα και θα τα στέλνει στον κόμβο που επικοινωνούμε.

```
public void makeCall(final SecretKeySpec generatedSecretKey, final byte[]
generatedIV) {

    Thread streamThread = new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                AudioRecord recorder;
                recorder = new AudioRecord(MediaRecorder.AudioSource.MIC,
sampleRate, channelConfig, audioFormat, minBufSize * 10);
                Cipher cipher = Cipher.getInstance("AES/CTR/NoPadding");
                cipher.init(Cipher.ENCRYPT_MODE, generatedSecretKey, new
IvParameterSpec(generatedIV));
```

Εδώ αρχικά ορίζουμε την μέθοδο makeCall και δημιουργούμε το πρώτο νήμα. Στη συνέχεια δημιουργούμε και αρχικοποιούμε το αντικείμενο recorder καθώς και το αντικείμενο cipher με το κλειδί και τον IV που παρήγαγε η διαδικασία OutHandshake. Έτσι το νήμα είναι έτοιμο να παράγει και να κρυπτογραφεί δεδομένα.
Στη συνέχεια εφαρμόζουμε τις αντιστοιχίσεις για τα socket όπως ορίστηκαν στην αρχή της ενότητας με βάση το mode λειτουργίας και εκκινούμε το αντικείμενο recorder.

```
if(mode.equals("Normal") || mode.equals("CR")) {
    socket = new DatagramSocket(6666);
    while (!socket.isConnected()) {
    }
}
if(mode.equals("Normal1")) {
    socket = new DatagramSocket(6666);
    while(!socket.isConnected()) { socket.connect(destination, 3333);}
} else if(mode.equals("HP")) {
    String[] params;
    params = UDPendpoint2.split(":");
    destination = InetAddress.getByName(params[0]);
    portHP = Integer.getInteger(params[1]);
}
recorder.startRecording();
```

Στη συνέχεια δημιουργούμε ένα βρόχο while ο οποίος θα τρέχει μέχρι ο χρήστης να πατήσει το πλήκτρο endCALL. Εδώ διατηρούμε έναν μετρητή ο οποίος θα μηδενίζεται όταν φτάνει στην τιμή 65000. Ο μετρητής αυτός θα ενσωματώνεται στο πακέτο δεδομένων ώστε ο παραλήπτης να μπορεί να συγχρονίζει τον stream cipher όταν χάνονται πακέτα. Τέλος δημιουργούμε το τελικό πακέτο. Ο κώδικας φαίνεται παρακάτω.

```
while(status) {
    if(packetcounter==65000) { packetcounter = 0; }
    packetcounter++;
    dbuf.putInt(packetcounter);
    byte[] sync = dbuf.array();
    sizeC = sync.length;
    recorder.read(buffer, 0, buffer.length);
    byte[] ciphertext = cipher.doFinal(buffer)
    ByteArrayOutputStream outputStream = new ByteArrayOutputStream( );
    outputStream.write(sync);
    outputStream.write(ciphertext);
    byte[] finalpacket = outputStream.toByteArray();
```

Στην συνέχεια ανάλογα με το mode που βρισκόμαστε στέλνουμε το πακέτο

```
if(mode.equals("Normal") || mode.equals("Normal1") || mode.equals("CR")) {
    packet = new DatagramPacket(finalpacket, finalpacket.length);
    socket.send(packet);
} else if(mode.equals("HP")) {
    packet = new DatagramPacket(finalpacket, finalpacket.length, destination,
portHP);
    Dsock1.send(packet);
}
```

Τέλος κλείνουμε το αντικείμενο recorder και το socket και κάνουμε catch τα exceptions και ξεκινάμε το νήμα.

Και ο κώδικας για το δεύτερο νήμα είναι ο παρακάτω:

```
Thread streamThread2 = new Thread(new Runnable() {
    @Override
    public void run() {
        try {
            AudioTrack track;
            track = new AudioTrack(AudioManager.MODE_NORMAL, 16000,
AudioFormat.CHANNEL_CONFIGURATION_MONO,
                    AudioFormat.ENCODING_PCM_16BIT, minBufSize,
AudioTrack.MODE_STREAM);
            Cipher cipher2 = Cipher.getInstance("AES/CTR/NoPadding");
            cipher2.init(Cipher.DECRYPT_MODE, generatedSecretKey, new
IvParameterSpec(generatedIV));
```

Εδώ αρχικά δημιουργούμε το αντικείμενο AudioTrack το οποίο θα τροφοδοτούμε με δεδομένα που θα λαμβάνουμε από τον κόμβο που επικοινωνούμε. Το αντικείμενο αυτό αρχικοποιείται με τις ίδιες παραμέτρους που αρχικοποιήθηκε το αντικείμενο AudioRecord. Στη συνέχεια αρχικοποιούμε το αντικείμενο Cipher το οποίο θα χρησιμοποιήσουμε για την αποκρυπτογράφηση των δεδομένων.

Στη συνέχεια ανιχνεύουμε σε ποιο mode βρισκόμαστε και εκτελούμε τις απαραίτητες ενέργειες όπως και στο πρώτο νήμα.

```
if(mode.equals("Normal") || mode.equals("CR")) {
    socket2 = new DatagramSocket(4444);
    while (!socket2.isConnected()) {}
}
if(mode.equals("Normal1")) {
    socket2 = new DatagramSocket(4444);
    while(!socket2.isConnected()) { socket2.connect(destination, 5555); }
}
track.play();
```

Στη συνέχεια διαβάζουμε το εισερχόμενο πακέτο ανάλογα με το mode.

```
while(status) {
    if(mode.equals("Normal") || mode.equals("CR") || mode.equals("Normal1")) {
        socket2.receive(packet);
    } else {
```

```
        Dsock2.receive(packet);
    }
    packetCounter++;
```

Στη συνέχεια ελέγχουμε αν έχουν χαθεί πακέτα εξάγοντας τον counter από το πακέτο που λάβαμε. Η εφαρμογή μπορεί να συγχρονιστεί έχοντας χάσει έως και 2000 συνεχόμενα πακέτα. Αν ο μετρητής του πακέτου είναι μεγαλύτερος από τον counter του νήματος και δεν έχουν χαθεί πάνω από 2000 πακέτα ο stream cipher συγχρονίζεται εκτελώντας αντίστοιχες φορές τη doFinal αλλιώς αγνοεί το πακέτο. Εναλλακτικά μπορούμε να συγχρονίσουμε τον Stream cipher υπολογίζοντας την τιμή που πρέπει να έχει ο counter, δημιουργώντας έναν καινούργιο IV και θέτοντας τον με κλήση στην init.

```
packetcount = Arrays.copyOfRange(originalpacket, 0, 3);
ByteBuffer wrapped = ByteBuffer.wrap(packetcount);
int num = wrapped.getInt();
if(num > packetCounter) {
    if((num - packetCounter) < 2000) {
        for (int i = 0; i < (num - packetCounter); i++) {
            byte[] plaintext = cipher2.doFinal(buffer2);
        }
        packetCounter = num;
    } else {
        packetCounter--;
        break;
    }
}
```

Αν πρόκειται για καθυστερημένο πακέτο ή για πακέτο που παρήχθη αφού μηδενίστηκε ο counter εκτελείται ο παρακάτω κώδικας

```
if(num < packetCounter)  {
    if((65000-packetCounter+num) < 2000 ) {
        for (int j = 0; j < (num + 65000 - packetCounter); j++) {
            byte[] plaintext = cipher2.doFinal(buffer2);
        }
        packetCounter = num;
    }
} else {
    packetCounter--;
    break;
}
```

Τέλος αποσπούμε τα δεδομένα απο το πακέτο τα αποκρυπτογραφούμε και τα τροφοδοτούμε στο αντικείμενο AudioTrack για αναπαραγωγή, τέλος σταματάμε το AudioTrack και κλείνουμε το socket.

```
 buffer2 = Arrays.copyOfRange(originalpacket, 4, originalpacket.length);

    byte[] plaintext = cipher2.doFinal(buffer2);
    track.write(plaintext, 0, plaintext.length);
    track.flush();
    if(packetCounter==65000) { packetCounter = 0; }
}
track.stop();
```

```
track.release();
socket2.disconnect();
socket2.close();
```

6.10 Το InboundCall Activity

Το Activity αυτό θα εκκινείται απο το Service με Intent. Οι δύο βασικές ενέργειες που θα πραγματοποιεί είναι η διαδικασία handshake, ώστε οι δύο κόμβοι να καταλήξουν σε ένα συμμετρικό κλειδί για την επικοινωνία και η διαδικασία answerCall η οποία θα πραγματοποιεί την κλήση. Η answerCall μπορεί να χωριστεί και αυτή σε διαδικασίες βάση λειτουργικότητας σε διαδικασία παραγωγής δεδομένων, κρυπτογράφησης δεδομένων, αποστολής δεδομένων και αντίστοιχα αποδοχής δεδομένων, αποκρυπτογράφησης, κατανάλωσης δεδομένων όπως η αντίστοιχη του OutboundCall Activity. Το Activity αυτό επικοινωνεί με το InboundCall Activity και χρησιμοποιεί δύο μεθόδους για να στέλνει και να λαμβάνει δεδομένα την sendpacket και τη receivepascket οι οποίες είναι ίδιες με αυτές OutboundCall Activity. Για να λειτουργούν αυτές οι συναρτήσεις σε όλα τα mode λειτουργίας δηλαδή,
1)Normal όπου και οι δύο κόμβοι έχουν public IPs.
2) Normal1 όπου ο κόμβος που εκκινεί τη διαδικασία έχει private IP και ο άλλος κόμβος public.
3)CR Mode όπου ο κόμβος που εκκινεί τη διαδικασία έχει public IP και ο δεύτερος private. Θα τροποποιήσουμε τον κώδικα τους ώστε να βρίσκονται σε κατάλληλο συνδυασμό server/client με τις αντίστοιχες μεθόδους του OutboundCall όπως ορίστηκε στην προηγούμενη ενότητα. Τέλος το UI της εφαρμογής θα έχει ένα textview όπου θα εμφανίζεται το όνομα της επαφής που καλεί, ένα πλήκτρο για αποδοχή της κλήσης και ένα για απόρριψη της.

Και η οθόνη που βλέπει ο χρήστης είναι η παρακάτω:

Εικόνα 21: Οθόνη InboundCall Activity

Στη συνέχεια θα δημιουργήσουμε την κλάση InboundCallActivity, θα ορίσουμε τις απαραίτητες μεταβλητές. Θα κάνουμε override τη μέθοδο onCreate() όπου θα συσχετίσουμε τα αντικείμενα του UI. Στη συνέχεια εξάγουμε τις παραμέτρους απο το Intent ανάλογα με το mode που βρισκόμαστε.Εδώ αφού ανιχνεύσουμε το mode λειτουργίας εξάγουμε τις παραμέτρους για τα

modes Normal, CR, Normal1 και HP. Τέλος θα δημιουργήσουμε τους Listeners για τα δύο κουμπιά όπου ανάλογα το mode εκτελούνται οι κατάλληλες ενέργειες. Ο listener για το πλήκτρο Ignore απλά καλεί τη finish αφού στείλει το κατάλληλο broadcast στο IntentService στο οποίο εκτελείται η λειτουργικότητα ώστε να τερματιστεί.
Ο listener για το πλήκτρο Answer ανάλογα με το mode εκτελεί τις κατάλληλες λειτουργίες. Για mode Normal, Normal1, CR ο κώδικας είναι ο παρακάτω:

```
if(mode.equals("Normal") || mode.equals("Normal1") || mode.equals("CR")) {
    InHandshake(contactName, IP);
    answerCall(key, IV, IP);
}
```

Εδώ χρησιμοποιούμε τις παραμέτρους που εξάγαμε απο το intent και εκτελούμε τις διαδικασίες InHandshake και answerCall στις οποίες θα αναφερθούμε στη συνέχεια.

Για mode HP ο κώδικας είναι ο παρακάτω

```
else if(mode.equals("HP")) {
    serverIP = getIP("server");
    response = "HP:RESPONSE:" + myalias + ":" + contactName + ":" + "CALL";
    sendresponseHP();
    initHP();
    InHandshake(contactName, IP);
    answerCall(key, IV, IP);
}
```

Αν το mode είναι HP εκτελείται η sendresponseHP η οποία είναι ίδια με αυτή που έχουμε χρησιμοποιήσει σε άλλα μέρη της εφαρμογής(Validate). Η initHP επίσης είναι ίδια με άλλα μέρη της εφαρμογής. Τέλος εκτελείται η InHandshake και η answerCall.

Η διαδικασία InHandshake

Η διαδικασία αυτή είναι παρόμοια με την OutHandshake και αποτελεί την υλοποίηση του αλγορίθμου DiffiHellman Ephemeral(DHE). Η διαδικασία InHandshake της κλάσης InboundCal lActivity θα αλληλεπιδρά με την διαδικασία OutHandshake της κλάσης OutboundCallActivity. Τα βήματα απο την πλευρά της InHandshake είναι τα παρακάτω:

- 1) Φόρτωση του keyPair μου και του public της επαφής από το KeStore.
- 2) Αποδοχή του Ephemeral public της επαφής
- 3) Παραγωγή του Ephemeral KeyPair μου
- 4) Υπογραφή του Ephemeral Public μου με το Private μου
- 5) Αποστολή του Ephemeral Public μου
- 6) Αποδοχή υπογραφής
- 7) Αποστολή υπογραφής
- 8) Έλεγχος υπογραφής
- 9) Παραγωγή μυστικού
- 10) Παραγωγή συμμετρικού κλειδιού
- 11) Αποδοχή IV

Ξεκινώντας την υλοποίηση της InHandshake παραθέτουμε τον ορισμό της συνάρτησης και τα βήματα

```
public void InHandshake(String Contact, String mIP) {

    loadMyKeyPair(Contact);

    EphemeralPubliceReceived = receivePublic();


    loadmyEphemeralKeyPair(EphemeralPubliceReceived);


    signature = signEphemeralPublic(myPrivateKey, myEphemeralPublicKey);


    sendEphemeralPublic(myEphemeralPublicKey);


    signatureReceived = receiveSignature();


    sendSignature(signature);


    checkSignature(signatureReceived, peerPublicKey);


    secret = generateSecret(myEphemeralPrivateKey, EphemeralPubliceReceived);


    secretKeyGenerated = generateSecretKey(secret);


    IV = receiveIV();

}
```

Στην συνέχεια θα αναλύσουμε τις μεθόδους που υλοποιούν τη λειτουργικότητα.

Η loadMyKeyPair

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η receivepublic

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.
Η loadMyEphemeralKeyPair

Η μέθοδος αυτή παίρνει μία παράμετρο. Η παράμετρος είναι το ephemeral δημόσιο κλειδί της επαφής που θα επικοινωνήσουμε και το χρειαζόμαστε γιατί τα ephemeral ζεύγη κλειδιών των δύο μερών που επικοινωνούν πρέπει να παραχθούν με τις ίδιες παραμέτρους. Ο κώδικας φαίνεται παρακάτω:

```
public void loadmyEphemeralKeyPair(PublicKey EphPublicPeer){
    try {
        DHParameterSpec dhParamFromPeer = ((DHPublicKey)
EphPublicPeer).getParams();
        KeyPairGenerator keygen = KeyPairGenerator.getInstance("DH");
        keygen.initialize(dhParamFromPeer);
        KeyPair myEphKeyPair = keygen.generateKeyPair();
        myEphemeralPrivateKey = myEphKeyPair.getPrivate();
        myEphemeralPublicKey = myEphKeyPair.getPublic();
    }catch(java.security.NoSuchAlgorithmException e) { result =
"NoSuchAlgorithmException";  }
    catch(java.security.InvalidAlgorithmParameterException e) { result =
"InvalidAlgorithmException"; }
}
```

Ο κώδικας αποτελείται από ένα try/catch block αρχικά δημιουργούμε ένα αντικείμενο DHParameterSpec στο οποίο εξάγουμε τις παραμέτρους από το ephemeral δημόσιο κλειδί της επαφής που λάβαμε παραπάνω. Στη συνέχεια δημιουργούμε ένα αντικείμενο τύπου KeyPairGenerator το οποίο αρχικοποιούμε με τις παραμέτρους που εξάγαμε παραπάνω. Τέλος δημιουργούμε το keypair, αποθηκεύουμε το δημόσιο και ιδιωτικό κλειδί για μελλοντική χρήση και κάνουμε catch τα exceptions.

Η signEphemeralPublic

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η sendEphemeralPublic

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η receiveSignature

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η sendSignature

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η checkSignature

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η generateSecret

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η generateSecretKey

Αυτή είναι ίδια με την αντίστοιχη της OutHandshake του OutboundCall Activity.

Η receiveIV

Η μέθοδος αυτή διαβάζει ένα αντικείμενο byte[] μέσω της receivepacket και το επιστρέφει.

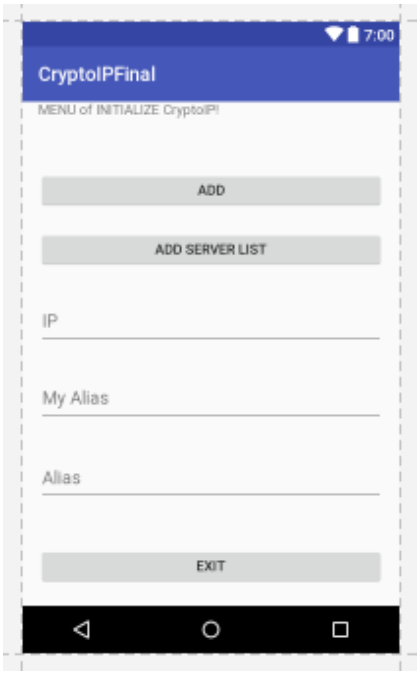
Η διαδικασία answerCall

Η διαδικασία αυτή είναι ίδια με την makeCall του OutboundCall Activity με τη μόνη διαφορά να βρίσκεται στα UDP sockets που χρησιμοποιεί για να στείλει καI να λάβει τα δεδομένα. Έτσι για το πρώτο νήμα με το οποίο στέλνει τα δεδομένα το UDP Socket ακούει στην θύρα 5555 και προσπαθεί να συνδεθεί στην 4444 του κόμβου ανάλογα το mode. Και στο δεύτερο νήμα το οποίο λαμβάνει τα δεδομένα το UDP Socket ακούει στην θύρα 3333 και προσπαθεί να συνδεθεί στην 6666 του κόμβου ανάλογα το mode. Όλη η υπόλοιπη λειτουργικότητα είναι ίδια.

6.11 To Initialize Activity

Το Activity αυτό θα δίνει την δυνατότητα στο χρήστη να εισάγει στην βάση δεδομένων την διεύθυνση IP που αντιστοιχεί σε κάποια επαφή. Εκτός από αυτή την λειτουργία θα δίνεται η δυνατότητα να γίνει σύνδεση με τον server ώστε να ανακτηθούν οι διευθύνεις των επαφών που έχουν δηλώσει την παρουσία τους και να δηλώσει και ο ίδιος την παρουσία του. Εκτός από αυτό το Activity αυτό θα αποφασίζει πιο service θα εκκινηθεί ανάλογα με το response του server κατά τη διαδικασία δήλωσης παρουσίας. Το UI θα αποτελείται από τρία πλήκτρα ένα θα εκτελεί την εισαγωγή στην βάση, το δεύτερο θα ανακτά τις διευθύνσεις από τον server και το τρίτο θα τερματίζει το Activity. Επίσης θα υπάρχουν δύο EditTexts όπου ο χρήστης θα εισάγει το alias της επαφής και την διεύθυνση IP ή την διεύθυνση του server.

Και η οθόνη που θα βλέπει ο χρήστης η παρακάτω



Εικόνα 22: Οθόνη Initialize Activity

Στη συνέχεια θα δημιουργήσουμε το Activity όπου θα δηλώσουμε την κλάση και τις μεταβλητές που χρειάζονται. Στη συνέχεια κάνουμε override τη μέθοδο onCreate όπου συσχετίζουμε με το κατάλληλο UI. Στη συνέχεια δημιουργούμε τους listeners για τα τρία πλήκτρα. Ο listener για το πλήκτρο buttonExit στέλνει το κατάλληλο broadcast στο Intent Service ώστε να τερματιστεί και

καλεί την μέθοδο finish. Ο listener για το πλήκτρο buttonAdd αρχικά παίρνει τις τιμές από τα edittexts και στη συνέχεια περνάει τις τιμές σε ένα IntentService το οποίο και εκκινεί. Το IntentService αυτό καλεί τη μέθοδο addToDB και το πλήκτρο buttonAddServer παίρνει την παράμετρο περνάει τις παραμέτρους σε ένα IntentService το οποίο εκκινεί και καλεί την register και την retriveFromServer.

```
buttonAdd.setOnClickListener(new View.OnClickListener() {
    public void onClick(View _view) {
        aliasName = alias.getText().toString();
        aliasNameIP = aliasIP.getText().toString();
        InitializeService.putExtra("mode", "ADD");
        InitializeService.putExtra("aliasName", aliasName);
        InitializeService.putExtra("aliasNameIP", aliasNameIP);
        startService(InitializeService);
    }
});
buttonAddServer.setOnClickListener(new View.OnClickListener() {
    public void onClick(View _view) {
        myalias = myAlias.getText().toString();
        serverIP = aliasIP.getText().toString();
        InitializeService.putExtra("mode", "SERVER");
        InitializeService.putExtra("myalias", myalias);
        InitializeService.putExtra("serverIP", serverIP);
        startService(InitializeService);
    }
});
```

Η μέθοδος addToDB παίρνει τις τιμές που εισήγαγε ο χρήστης ως παραμέτρους και τις εισάγει στην βάση. Αρχικά ανοίγει τη βάση ή την δημιουργεί αν δεν υπάρχει, δημιουργεί τον πίνακα αν δεν υπάρχει και στην συνέχεια δημιουργεί το query και το εκτελεί. Τέλος κλείνει την βάση.

```
public void addToDB(String name, String address) {

    SQLiteDatabase mydb;
    String query;
    query = "INSERT INTO IPTable VALUES('"+name+"', '"+address+"');";
    mydb = openOrCreateDatabase("mydata.db", Context.MODE_PRIVATE, null);
    mydb.execSQL("CREATE TABLE IF NOT EXISTS IPTable(name TEXT,IP TEXT);");
    mydb.execSQL(query);
    mydb.close();

}
```

Η μέθοδος register θα επικοινωνήσει με τον server και θα δηλώσει την παρουσία του. Στην συνέχεια θα ελέγξει άν έχει public ή private IP και αναλόγως θα εκκινήσει το κατάλληλο service Ο κώδικας φαίνεται παρακάτω.

```
public void register() {
    String myIP = retrieveMyIp();
    message = "POST:" + myalias + ":" + myIP;
    sendmessage();
    if(response.equals("public")) {
        addToDB(myalias, myIP);
        startService(Server);
    } else if(response.equals("private")) {
        addToDB(myalias, "0.0.0.0");
         startService(Serverv2);
    }
```

```
}
```

Η μέθοδος retrieveMyIp ανακτά την IP της συσκευής που τρέχει η εφαρμογή.

Η μέθοδος retriveFromServer στέλνει το κατάλληλα διαμορφωμένο μήνυμα στον Server και χειρίζεται την απάντηση. Αρχικά ανοίγει το keystore και ανακτά όλες τις επαφές. Στην συνέχεια δημιουργούμε ένα βρόχο for και για κάθε επαφή που υπάρχει στο keystore επιχειρούμε να ανακτήσουμε την διεύθυνση από τον server. Αν αυτή υπάρχει την καταχωρούμε στην βάση δεδομένων.
Ο κώδικας φαίνεται παρακάτω

```
public void retriveFromServer() {
    try {
        File file = new
File(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURE
S), "keystore.p12");
        KeyStore keyStore = KeyStore.getInstance("PKCS12");
        FileInputStream instreamKS = new FileInputStream(file);
        keyStore.load(instreamKS, "password".toCharArray());
        instreamKS.close();
        for (Enumeration<String> aliases = keyStore.aliases();
aliases.hasMoreElements();) {
            String alias = aliases.nextElement();
            message = "GET:" + alias ;
            sendmessage();
            if(!response.equals("Not Found")) { addToDB(alias, response); }

        }
```

6.12 To Manifest

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.iasonas.cryptoip_v1">
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <receiver android:name=".AddRemoteContactService$MyBroadcastReceiverARC"
android:exported="false">
            <intent-filter>
                <action
android:name="com.example.iasonas.cryptoipfinal.HPADDREMOTECONTACT"/>
            </intent-filter>
        </receiver>
        <receiver android:name=".WebOfTrustService$MyBroadcastReceiverWOT"
android:exported="false">
            <intent-filter>
                <action
```

```
android:name="com.example.iasonas.cryptoipfinal.HPWEBOFTRUST"/>
            </intent-filter>
        </receiver>
        <receiver
android:name=".OutboundCallActivityService$MyBroadcastReceiverOUTC"
android:exported="false">
            <intent-filter>
                <action
android:name="com.example.iasonas.cryptoipfinal.HPCALL"/>
            </intent-filter>
        </receiver>
        <receiver
android:name=".OutboundCallActivityService$MyBroadcastReceiverOUTCEndCall"
android:exported="false">
            <intent-filter>
                <action
android:name="com.example.iasonas.cryptoipfinal.ENDOUTCALL"/>
            </intent-filter>
        </receiver>
        <receiver
android:name=".InboundCallActivityService$MyBroadcastReceiverINCEndCall"
android:exported="false">
            <intent-filter>
                <action
android:name="com.example.iasonas.cryptoipfinal.ENDINCALL"/>
            </intent-filter>
        </receiver>
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".AddContact"/>
        <activity android:name=".RemoveContact"/>
        <activity android:name=".AddRemoteContact"/>
        <activity android:name=".ContactRequest"/>
        <activity android:name=".WebOfTrust"/>
        <activity android:name=".Validate"/>
        <activity android:name=".OutboundCallActivity"/>
        <activity android:name=".InboundCallActivity"/>
        <activity android:name=".Initialize"/>
        <service android:name=".Service2"/>
        <service android:name=".ServerService"/>
        <service android:name=".AddContactService"/>
        <service android:name=".RemoveContactService"/>
        <service android:name=".AddRemoteContactService"/>
        <service android:name=".ContactRequestService"/>
        <service android:name=".WebOfTrustService"/>
        <service android:name=".ValidateService"/>
        <service android:name=".OutboundCallActivityService"/>
        <service android:name=".InboundCallActivityService"/>
        <service android:name=".InitializeService"/>

    </application>
</manifest>
```

8. Ο Server διευθυνσιοδότησης

Ο Server αυτός θα τρέχει σε έναν οποιονδήποτε υπολογιστή με public IP διεύθυνση και θα δέχεται αιτήματα καταχώρισης/ανάκτησης διεύθυνσης επαφών από τους κόμβους που έχουν public IP. Για τους κόμβους που έχουν private IP θα υπάρχουν άλλα τρία είδη μηνυμάτων 1)αίτημα Conection Reversing για την περίπτωση που ο ένας χρήστης έχει public και ο άλλος private IP, 2)αίτημα για Hole punching σε περίπτωση που και οι δύο κόμβοι έχουν private IP, 3)απάντηση σε αίτημα για σύνδεση Hole Punching. Τέλος θα υπάρχει αίτημα για δημιουργία μόνιμης σύνδεσης με τον server ώστε οι κόμβοι να μπορούν να στέλνουν τα προηγούμενα μηνύματα στους άλλους κόμβους. Ο server θα διατηρεί ένα αρχείο όπου θα αποθηκεύει ζεύγη ονομάτων/διευθύνσεων. Κατά το αίτημα καταχώρισης διεύθυνσης ο κόμβος θα στέλνει την IP που βλέπει ότι έχει στον server ο οποίος θα την συγκρίνει με την διεύθυνση που βλέπει από το socket. Σε περίπτωση που είναι ίδιες θα την καταχωρεί μαζί με το Alias και θα ενημερώνει τον κόμβο ότι έχει public IP, αν είναι διαφορετικές θα καταχωρεί το alias με IP 0.0.0.0 και θα ενημερώνει τον κόμβο οτι έχει private IP.
Τα αιτήματα θα είναι κατάλληλα μορφοποιημένα όπως ορίζονται παρακάτω

- POST:<myAlias>:<myIP>
- GET:<alias>
- HP:REQUEST:<fromAlias>:<toAlias>:<Activity>
- HP:RESPONSE:<fromAlias>:<toAlias>:<Activity>
- CR:<fromAlias>:<toAlias>:<Activity>:<fromIP>
- KEEPALIVE:<myAlias>

Για τα αιτήματα τύπου KEEPALIVE θα δημιουργείτε ένα καινούργιο νήμα τύπου ClientHandler. Οι παράμετροι θα είναι το αρχικό socket και streams που δημιουργήθηκαν κατά την αρχική σύνδεση καθώς και μία BLOCKINGQUEUE η οποία θα είναι κοινή μεταξύ όλων των ενεργών νημάτων ώστε να διαμοιράζονται τα αιτήματα καθώς και το alias του χρήστη. Αρχικά ορίζουμε την κλάση, τις μεταβλητές της και την μέθοδο κατασκευαστή. Στην συνέχεια κάνουμε override τη μέθοδο run και δημιουργούμε ένα βρόχο while όπου θα ελέγχουμε την ουρά αν έχει μηνύματα για τον συγκεκριμένο χρήστη. Στην συνέχεια ελέγχουμε εάν υπάρχουν μηνύματα στη ουρά, αν δέν υπάρχουν χρησιμοποιούμε την break επιστρέφοντας στην αρχή του βρόχου, αν υπάρχουν τα περνάμε όλα σε μία λίστα. Ο κώδικας φαίνεται παρακάτω.

```
while(true)
{
	try {

	int itemsNum = theQueue.remainingCapacity();

	if(itemsNum==100) { break; }

	List<Message> myList = new ArrayList<Message>();
	theQueue.drainTo(myList);
```

Στην συνέχεια ελέγχουμε εάν κάποιο από τα μηνύματα προορίζεται για τον συγκεκριμένο χρήστη που του ανήκει το νήμα. Εάν βρεθεί κάποιο μήνυμα αφαιρείται από την λίστα και με βάση αυτό δημιουργείται το incomingmessage. Τέλος εισάγουμε πίσω στην ουρά όλα τα μηνύματα που δεν προορίζονταν για εμάς. Ο κώδικας φαίνεται παρακάτω.

```
for(int i=0; i<myList.size(); i++) {
```

```
        Message current = myList.get(i);

        if (current.getTo()==me){

incomingMessage = current.getMode()+":"+current.getType()
+":"+current.getActivity()+":"+current.getFrom()+":"+current.getTo()
+":"+current.getIpPort1()+":"+current.getIpPort2()+"/";

myList.remove(i);
break;
}
}

for(int j=0; j < myList.size(); j++){

        Message current = myList.get(j);
        theQueue.add(current);
}
```

Τέλος χρησιμοποιούμε το outstream για να στείλουμε το μήνυμα στον client αν αυτό υπάρχει. Το είδος του αντικειμένου που εισάγεται στην ουρά είναι τύπου Message όπως ορίζεται παρακάτω με get/set μεθόδους για όλες τις μεταβλητές.

```
public class Message {

        String mode;
        String activity;
        String type;
        String from;
        String to;
        String IPport1;
        String IPport2;
```

Προχωρώντας στην υλοποίηση αρχικά δηλώνουμε την κλάση, την μέθοδο main, τις απαραίτητες μεταβλητές και δημιουργούμε ένα ServerSocket.Στην συνέχεια δημιουργούμε ένα βρόχο while όπου αρχικά περιμένουμε να συνδεθεί κάποιος client και αφού συνδεθεί διαβάζουμε το request του.

Αν το request ξεκινάει με GET τότε εκτελείται ο παρακάτω κώδικας

```
if(line.startsWith("GET")) {

String info[] = line.split(":");

String myname = info[1];

IP = getIP(myname);

   output.println(IP);
}
```

Εδώ αρχικά εξάγουμε τις παραμέτρους από το μήνυμα και καλούμε την getIP ώστε να ανακτήσουμε την διεύθυνση του κόμβου. Αφού ανακτηθεί την στέλνουμε πίσω στον κόμβο που έκανε το αίτημα. Ο κώδικας για την getIP είναι ο παρακάτω

```
public static String getIP (String cName) {

      try {
      FileReader inFile = new FileReader("myfile");
      BufferedReader instream = new BufferedReader(inFile);
      String input;

      while((input = instream.readLine()) != null) {

      String info[] = input.split(" ");
      if(info[0] == cName) { instream.close(); return info[1];
    }

      }
      instream.close();

}catch(IOException e){}

return "Not Found";
}
```

Εδώ αφού ανοίξουμε το αρχείο με τις καταχωρίσεις διαβάζουμε όλες τις καταχωρίσεις και αν βρεθεί αντιστοιχία επιστρέφουμε την καταχώριση. Αν δεν βρεθεί επιστρέφουμε NotFound.

Αν το request ξεκινάει με POST τότε εκτελείται ο παρακάτω κώδικας

```
else if(line.startsWith("POST")){

      String info[] = line.split(":");
      String myname = info[1];
      String myIP = info[2];

      IP = socket.getRemoteSocketAddress().toString();

      response = saveIP(myname, IP, myIP);

      output.println(response);

}
```

Εδώ αφού εξάγουμε τις παραμέτρους από το request ανακτούμε την IP με την οποία συνδέεται ο κόμβος και καλούμε την saveIP αποθηκεύοντας το αποτέλεσμα στη μεταβλητή response και το στέλνουμε πίσω στον κόμβο. Το αποτέλεσμα θα είναι της μορφής public/private. Η μέθοδος saveIP θα αναγνωρίσει αν ο κόμβος έχει private ή public IP και θα καταχωρίσει το κατάλληλο αποτέλεσμα στο αρχείο.
Ο κώδικας για την saveIP είναι ο παρακάτω.

```
public static String saveIP(String Cname, String CIP, String IP2) {

      String result = null;
      String toAppend;

      try {
      FileReader inFile = new FileReader("myfile");
      BufferedReader instream = new BufferedReader(inFile);
      String input;
```

```
      while((input = instream.readLine()) != null) {

            String info[] = input.split(" ");
            if(info[0] == Cname) { instream.close(); return "already exists"; }

                  }
```

Αρχικά ανοίγουμε το αρχείο με τις καταχωρίσεις και ελέγχουμε αν υπάρχει καταχώριση με το ίδιο alias. Αν δεν υπάρχει η εκτέλεση συνεχίζεται αλλιώς επιστρέφει κατάλληλο μήνυμα.

```
if(CIP!=IP2){

 toAppend = Cname + " " + "0.0.0.0" + "\n";
   result = "private";
      }else{

      toAppend = Cname + " " + CIP + "\n";
      result = "public";

      }
```

Στη συνέχεια ελέγχουμε άν η IP του κόμβου είναι public ή private και δημιουργούμε την κατάλληλη καταχώριση. Τέλος αποθηκεύουμε την καταχώριση στο αρχείο και επιστρέφουμε το αποτέλεσμα της διαδικασίας όπως φαίνεται παρακάτω.

```
BufferedWriter writer = new BufferedWriter(new FileWriter("myfile", true));
      writer.write(toAppend);
      writer.close();

            return result;
      }
```

Αν το request ξεκινάει με KEEPALIVE τότε εκτελείται ο παρακάτω κώδικας

```
else if(line.startsWith("KEEPALIVE")) {

            String info[] = line.split(":");
            String myAlias = info[1];

      Thread t = new ClientHandler(socket, reader, output, myAlias, bq );
            t.start();
      flag=false;
}
```

Εδώ αρχικά εξάγουμε τις παραμέτρους από το requestκαι δημιουργούμε ένα καινούριο νήμα τύπου ClientHandler όπως περιγράφηκε προηγουμένος περνώντας τις κατάλληλε παραμέτρους και το εκκινούμε.

Αν το request ξεκινάει με CR τότε εκτελείται ο παρακάτω κώδικας.

```
else if(line.startsWith("CR")) {

            String info[] = line.split(":");
            String fromAlias = info[1];
            String toAlias = info[2];
            String Activity = info[3];
```

```
            String IPport = info[4];

      Message msg = new Message();

      msg.setMode("CR");
      msg.setType("empty");
      msg.setActivity(Activity);
      msg.setFrom(fromAlias);
      msg.setTo(toAlias);
      msg.setIpPort1(IPport);
      msg.setIpPort2("empty");

      bq.add(msg);

}
```

Αρχικά εξάγουμε τις παραμέτρους από το request και στη συνέχεια δημιουργούμε ένα αντικείμενο Message περνώντας τις κατάλληλες παραμέτρους. Τέλος εισάγουμε το αντικείμενο στην ουρά.

Αν το request ξεκινάει με HP:REQUEST τότε εκτελείται ο παρακάτω κώδικας. Αρχικά εξάγουμε τις παραμέτρους

```
else if(line.startsWith("HP:REQUEST")){

      String info[] = line.split(":");
      String fromAlias = info[2];
      String toAlias = info[3];
      String Activity = info[4];
```

Στην συνέχεια ο server ανοίγει ένα UDP Socket και ένα πακέτο, εξάγει την IP/port του socket και την στέλνει με μήνυμα στο κόμβο.

```
DatagramSocket Dsock = new DatagramSocket();
byte[] array = new byte[24];
DatagramPacket packet = new DatagramPacket(array, array.length);
int port = Dsock.getLocalPort();
String sAdd = Dsock.getLocalAddress().toString() + ":" + Integer.toString(port);

output.println(sAdd);
```

Στην συνέχεια ο κόμβος στέλνει ένα πακέτο στον server από το οποίο ο server ανακτά το πρώτο UDP endpoint το οποίο πλέον γνωρίζει.

```
Dsock.receive(packet);
int port2 = packet.getPort();
String address = packet.getAddress().toString() + ":" + Integer.toString(port2);
```

Στην συνέχεια θα ακολουθηθεί η ίδια διαδικασία για τη δημιουργία του δεύτερου UDP endopoint. Ο κώδικας φαίνεται παρακάτω.

```
DatagramSocket Dsock2 = new DatagramSocket();
byte[] array2 = new byte[24];
DatagramPacket packet2 = new DatagramPacket(array2, array2.length);
```

```
int port3 = Dsock2.getLocalPort();
String sAdd2 = Dsock2.getLocalAddress().toString() + ":" +
Integer.toString(port3);

output.println(sAdd2);

Dsock.receive(packet2);
int port4 = packet.getPort();
String address2 = packet2.getAddress().toString() + ":" +
Integer.toString(port4);
```

Τέλος δημιουργείται ένα καινούριο αντικείμενο τύπου Message με τις κατάλληλες παραμέτρους και εισάγεται στην ουρά όπως φαίνεται παρακάτω.

```
Message msg = new Message();

      msg.setMode("HP");
      msg.setType("REQUEST");
      msg.setActivity(Activity);
      msg.setFrom(fromAlias);
      msg.setTo(toAlias);
      msg.setIpPort1(address);
      msg.setIpPort2(address2);

      bq.add(msg);

      }
```

Αν το request ξεκινάει με HP:RESPONSE τότε ακολουθείται η ίδια διαδικασία που ακολουθήθηκε για το HP:REQUEST με την διαφορά ότι το πεδίο type θα ισούται με RESPONSE. Τέλος κλείνουμε τα streams και sockets που δεν χρειάζονται, δηλαδή σε όλες τις περιπτώσεις εκτός από την περίπτωση KEEPALIVE.

9.Συμπεράσματα

Στην εφαρμογή που αναπτύξαμε σε αυτή την εργασία χρησιμοποιήσαμε στο μέγιστο τα Android APIs με όλα τα πλεονεκτήματα και τα μειονεκτήματα τους. Ο αλγόριθμος DHE θα μπορούσε να χρησιμοποιηθεί έτοιμος κάνοντας χρήση της κλάσης SSLSocket η οποία υποστηρίζει AES/GCM. Ο λόγος που δεν χρησιμοποιήσαμε την κλάση αυτή είναι επειδή δεν υποστηρίζει το πρωτόκολλο UDP. Στην περίπτωση που χρησιμοποιούσαμε το πρωτόκολλο TCP θα είχαμε καθυστερημένα πακέτα να αναπαράγονται καθώς επίσης και την καθυστερημένη αναπαραγωγή των πακέτων που θα έφταναν μετά από αυτά. Η συμπεριφορά αυτή θα προκαλούσε προβλήματα στην ποιότητα της υπηρεσίας. Τέλος, όπως αναφέραμε και στο κεφάλαιο 6 ο χρήστης της εφαρμογής πρέπει να κατανοήσει ότι παρόλο που η εφαρμογή προστατεύει τα δεδομένα του όταν αυτά κινούνται στο δίκτυο δεν σημαίνει πως του παρέχει απόλυτη ασφάλεια στις τηλεπικοινωνίες του καθώς ο επιτιθέμενος μπορεί να έχει πρόσβαση στα δεδομένα από κάποιο άλλο σημείο.
Σαν μελλοντικές προσθήκες που θα μπορούσαν να βελτιώσουν την ασφάλεια που παρέχει η εφαρμογή μπορούμε να προτείνουμε την υλοποίηση ψηφιακών υπογραφών για την σηματοδοσία μεταξύ του server διευθυνσιοδότησης και των κόμβων ώστε να πετύχουμε ακεραιότητα και αυθεντικότητα των αιτημάτων που στέλνονται και λαμβάνονται από τον server μειώνοντας τον κίνδυνο για επιθέσεις άρνησης εξυπηρέτησης.

Αναφορές

[1] Tariq Latif, Kranthi Kumar Malkajgiri, "Adoption of VoIP", Luleå University of Technology 2007

[2] Hakeem Adewale Idowu, "Security Awareness and Challenges in VoIP Technology", Luleå University of Technology 2014

[3] Piero Fontanini, "VoIP Security", Gjøvik University College, 2008

[4] Chintan Vaishnav, "Voice over Internet Protocol (VoIP): The Dynamics of Technology and Regulation", Massachusetts Institute of Technology 2006

[5] Prajwol Kumar Nakarmi, "Evaluation of VoIP Security for Mobile Devices", KTH Royal Institute of Technology 2011

[6] Richa Marwaha, "Security in Peer-to-Peer SIP VoIP", San Jose State University 2010

[7] Selvakumar Vadivelu, "Evaluating the Quality of Service in VOIP and comparing various encoding techniques", University of Bedfordshire 2011