ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ
Π.Μ.Σ. ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΕΦΑΡΜΟΓΗ ΣΤΕΓΑΝΟΓΡΑΦΙΑΣ ΣΕ ANDROID ΚΙΝΗΤΗ ΣΥΣΚΕΥΗ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΛΕΞΑΚΗΣ ΙΩΑΝΝΗΣ (Α.Μ.: 1201)

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΚΩΝΣΤΑΝΤΙΝΟΣ ΛΑΜΠΡΙΝΟΥΔΑΚΗΣ

ΠΕΙΡΑΙΑΣ

ΔΕΚΕΜΒΡΙΟΣ, 2015

Πίνακας Περιεχομένων

Εικόνα Εξωφύλλου: Στεγανογραφία σε εικόνες , Πηγή:

<http://www.threatgeek.com/2011/10/the-da-vinci-node.html>, Downloaded 20/11/2015

1 Περίληψη

Στην εργασία αυτή επιχειρείται μια πρώτη επαφή με την τεχνική της Στεγανογραφίας ή αλλιώς την τέχνη της απόκρυψης δεδομένων από τρίτους, στα πλαίσια του ψηφιακού κόσμου που μας περιβάλει και μέσα από τη χρήση της σε ψηφιακά αρχεία. Αρχικά, γίνεται μια αναφορά στη σημασία και τη χρήση της Στεγανογραφίας ενώ γίνεται λόγος και για την ιστορική της εξέλιξη. Στη συνέχεια ακολουθεί η αναφορά εφαρμογών της Στεγανογραφίας, υπό το πρίσμα της ψηφιακής επικοινωνίας, σε ψηφιακά αρχεία διαφόρων τύπων (εικόνας, ήχου, βίντεο). Στο κύριο μέρος της εργασίας γίνεται ενδεικτικά η περιγραφή μιας εκ των τεχνικών Στεγανογραφίας σε ψηφιακό αρχείο εικόνας με τη μέθοδο της αντικατάστασης του λιγότερο σημαντικού bit (LSB) και της υλοποίησης αντίστοιχης εφαρμογής – εργαλείου σε περιβάλλον Android η οποία αποκρύπτει ψηφιακά δεδομένα, που παίζουν το ρόλο ενός ιδιωτικού κλειδιού, κωδικοποιώντας τα μέσα σε μια ψηφιακή εικόνα αλλά και αποκωδικοποιώντας τα εν τέλει μέσα από αυτήν. Στο τέλος της εργασίας μελετώνται προτάσεις για την περαιτέρω βελτίωση της ασφάλειας της εφαρμογής που αναπτύχθηκε.

2 Εισαγωγή

Σε ένα κόσμο που η ψηφιοποιημένη πληροφορία αποτελεί το Α και το Ω στη σύγχρονη επικοινωνία τα όρια των εφαρμογών της δείχνουν να περιορίζονται μόνο από φαντασία μας. Όσο πιο διευρυμένο γίνεται το πεδίο χρήσης της, τόσο πιο πολύ αυξάνει και η ανάγκη για την προστασία και την ασφάλεια της. Η έννοια της ασφάλειας μπορεί να περιγραφεί ολοκληρωμένα από την τήρηση και προστασία των βασικών αρχών της Εμπιστευτικότητας, της Ακεραιότητας και της Διαθεσιμότητας[1]. Στο πλαίσιο αυτό εκτεταμένες πολιτικές έχουν αναπτυχθεί κατά το παρελθόν και συνεχίζουν να εξελίσσονται με τη συμβολή και ανάπτυξη δοκιμασμένων αλλά και νέων μεθόδων και τεχνικών. Μια ευρέως δοκιμασμένη μέθοδος είναι και η μέθοδος της Κρυπτογραφίας(Cryptography). Η Κρυπτογραφία είναι η μέθοδος κατά την οποία η πληροφορία κωδικοποιείται χρησιμοποιώντας κάποιον αλγόριθμο (Κρυπτό - σύστημα) με σκοπό να μην μπορεί το περιεχόμενό της να γίνει αναγνώσιμο και να εκτεθεί σε τρίτους χωρίς τη χρήση του «κλειδιού» του Κρυπτό - συστήματος[2]. Είναι λοιπόν προφανές ότι η κρυπτογραφημένη πληροφορία είναι διαθέσιμη και ανιχνεύσιμη σε κάποιον τρίτο χωρίς όμως να μπορεί να εξάγει τα κωδικοποιημένα δεδομένα αν δεν είναι εξουσιοδοτημένος. Αντίστοιχα μια συγγενής μέθοδος που μπορεί να χρησιμοποιηθεί είναι η Στεγανογραφία(Steganography). Στην περίπτωση αυτή τα δεδομένα της πληροφορίας μεταμφιέζονται με τέτοιο τρόπο ώστε να μην επιτρέπεται σε οποιοσδήποτε τρίτο (επιτιθέμενο) να ανιχνεύσει – εντοπίσει την ύπαρξη των δεδομένων αυτών εξαρχής. Με άλλα λόγια η πληροφορία κρύβεται ώστε να μην μπορεί να γίνει αντιληπτή σε μη εξουσιοδοτημένα άτομα.

Η μέθοδος της Στεγανογραφίας, στα πλαίσια της ασφάλειας της ψηφιακής πληροφορίας, περιγράφεται στη εργασία αυτή. Στα παρακάτω κεφάλαια θα περιγραφεί η έννοια και η σημασία της, οι εφαρμογές της και οι τεχνικές που χρησιμοποιούνται καθώς επίσης και η ανάλυση και περιγραφή ανάπτυξης εφαρμογής Στεγανογραφίας σε κινητή συσκευή λειτουργικού συστήματος Android χρησιμοποιώντας ενδεικτικά μια από τις τεχνικές και συγκεκριμένα τη χρήση του αλγορίθμου Least Significant Bit (LSB) που εξετάζεται αργότερα.

3 Η μέθοδος της Στεγανογραφίας και η σημασία της

Σύμφωνα με ότι ειπώθηκε στην εισαγωγή, η Στεγανογραφία είναι η μέθοδος του να επικοινωνεί κανείς με τέτοιο τρόπο ούτος ώστε να μη γίνεται αντιληπτή εξαρχής η ύπαρξη μιας τέτοιας επικοινωνίας.

Ποιοι μπορεί να είναι όμως οι λόγοι για μια τέτοια μορφή επικοινωνίας; Η Στεγανογραφία όπως θα περιγραφεί αργότερα είναι απλώς ένα εργαλείο, και όπως κάθε εργαλείο στην υπηρεσία του ανθρώπου μπορεί να χρησιμοποιηθεί είτε κακόβουλα ή εγκληματικά, είτε κατά του εγκλήματος αλλά και για την προστασία των ανθρώπινων δικαιωμάτων. Έτσι θα μπορούσε να χρησιμοποιηθεί για να εξυπηρετήσει τους σκοπούς κατασκοπείας, εμπόρων ναρκωτικών, τρομοκρατών, πορνογράφων αλλά και κάθε άλλη παράνομη ή εγκληματική δραστηριότητα στην οποία συνεπάγεται επικοινωνία και εν τέλει ανταλλαγή πληροφορίας. Σε αντιπαραβολή θα μπορούσε να χρησιμοποιηθεί για καλούς σκοπούς όπως η ανταλλαγή ευαίσθητων πληροφοριών μεταξύ σωμάτων ασφαλείας , ή και εταιριών-βιομηχανιών μιας χώρας για την προστασία εναντίων της κατασκοπείας, τη διατήρηση της ανωνυμίας και της εμπιστευτικότητας σε περιπτώσεις καταγγελιών και παροχή πληροφοριών στις αρχές μιας χώρας αλλά και την προστασία ευαίσθητων και προσωπικών δεδομένων από κακόβουλα άτομα όπως τρομοκράτες, πορνογράφους κ.τ.λ. Πρέπει να αναφέρουμε στο σημείο αυτό και όσον αφορά την περίπτωση της ιδιωτικότητας. πως σε πολλές χώρες έχουν ψηφιστεί νόμοι που είτε περιορίζουν, είτε απαγορεύουν την χρήση της κρυπτογραφίας λόγω του ότι η χρήση της περιορίζει την πρόσβαση των αρχών σε πληροφορίες που μπορούν να χρησιμοποιηθούν από ένδικα μέσα εναντίων εγκληματιών αφήνοντας έτσι ακάλυπτους τους μη κακόβουλους χρήστες που απλά επιθυμούν την ιδιωτικότητα των προσωπικών τους δεδομένων. Εδώ ή Στεγανογραφία μπορεί να δώσει λύση και να κρύψει ευαίσθητη πληροφορία μέσα σε κάποιο αρχείο έτσι ώστε μόνο εξουσιοδοτημένα άτομα που προορίζονται να λάβουν την πληροφορία αυτή να γνωρίζουν ότι υπάρχει.

4 Ιστορική Αναδρομή

Η πρώτη χρήση της Στεγανογραφίας καταγράφεται από το Ηρόδοτο, τον «πατέρα» της ιστορίας. Ο Ηρόδοτος στα κείμενά του αναφέρει δύο ιστορίες σχετικά με κάποιες στεγανογραφικές τεχνικές που χρησιμοποιούνταν στην Αρχαία Ελλάδα κατά την περίοδο του 440 Π.Χ. Η πρώτη αναφέρει πως ο βασιλιάς Δαρείος ξύρισε το κεφάλι ενός αιχμαλώτου και έγραψε ένα μήνυμα στο κεφάλι του. Όταν τα μαλλιά του αιχμαλώτου μεγάλωσαν και πάλι τότε τον έστειλε στον γαμπρό του βασιλιά της Μιλήτου Αρισταγόρα χωρίς να το ανιχνεύσει κανείς. Μια άλλη αναφέρει πως ένας στρατιώτης, ο Δημήρατος θέλησε να προειδοποιήσει τη Σπάρτη για την πρόθεση του βασιλιά Ξέρξη να εισβάλει στην Ελλάδα. Έτσι, χάραξε ένα μήνυμα σε ξύλινη πλάκα η οποία καλύφθηκε με κερί. Οι πλάκες αυτές χρησίμευαν, την εποχή εκείνη, στον να χαράσσεται ένα κείμενο στο κερί ως μέσω γραφής, Έτσι λοιπόν το κρυφό μήνυμα παρέμεινε μη ανιχνεύσιμο κάτω από το κερί. Αργότερα οι Ρωμαίοι επίσης χρησιμοποίησαν αόρατο μελάνι το οποίο εξαγόταν από οργανικά υποκατάστατα όπως χυμούς φρούτων αλλά και γάλα. Το μήνυμα μπορούσε να εξαχθεί θερμαίνοντας το στεγανογραφημένο κείμενο[2].

Στην αρχαία Κίνα μια μέθοδος απόκρυψης μηνύματος ήταν με τη χρήση μιας διάτρητης μάσκας κειμένου που αποτελούνταν από έναν αριθμό από τρύπες σε τυχαία σημεία. Ο αποστολέας τοποθετούσε τη μάσκα πάνω από ένα φύλλο χαρτί γράφοντας το κείμενο που ήθελε να αποκρύψει μέσα στις τρύπες και έπειτα γέμιζε τα κενά του χαρτιού με άλλους χαρακτήρες με σκοπό να σχηματίσει ένα άλλο κείμενο. Ο παραλήπτης χρησιμοποιώντας την ίδια μάσκα μπορούσε να διαβάσει το μυστικό κείμενο τοποθετώντας τη μάσκα πάνω από το χαρτί στο οποίο ήταν γραμμένο το ολοκληρωμένο κείμενο. Μάλιστα, πολύ αργότερα ένας Ιταλός μαθηματικός, ο Καρντάν, επαναχρησιμοποίησε τη μέθοδο αυτή το 16^{ο} αιώνα και η οποία υιοθετήθηκε από μια αγγλική τράπεζα το 1992 για την απόκρυψη του προσωπικού αριθμού ταυτοποίησης (PIN: Personal Identification Number) των πελατών τις[3].

Μεταξύ 15^{ου} και 16^{ου} αιώνα πολλοί συγγραφείς όπως ο Γιοχάνες Τρίθιμους, συγγραφέας του συγγράμματος Steganographia, και ο Γκασπάρι Σκότι, συγγραφέας του συγγράμματος Steganographica, έγραψαν για Στεγανογραφικές μεθόδους για

κωδικοποιημένο κείμενο, αόρατα μελάνια, και μηνύματα κρυμμένα μέσα σε μουσική[2].

Στα τέλη του 19^{ου} αιώνα μέχρι και την πρώτη δεκαετία του 20^{ου} οι Αύγουστος Κέρκοφ(Cryptographic Militaire) και Κάρολος Μπρικέτ(Les Filigranes) στα συγγράμματα τους εισήγαγαν συστήματα σχετικά με τεχνικές υδατογραφήματος. Επίσης κατά τη διάρκεια του μεσοπολέμου συντελείται σημαντική πρόοδος στον τομέα της Στεγανογραφίας αποτελώντας πεδίο για την ανάπτυξη νέων τεχνικών. Τεχνικές όπως microdots(σμίκρυνση δεδομένων ή εικόνων σε πάρα πολύ μικρό μέγεθος, μη ανιχνεύσιμα από το γυμνό μάτι, καταγεγραμμένα σε χαρτί) αλλά και null ciphers(π.χ. λαμβάνοντας κάθε δεύτερο ή τρίτο γράμμα ενός απλού κειμένου τα οποία μαζί σχηματίζουν το μυστικό κείμενο)[2].

Σήμερα χρησιμοποιώντας ως επί το πλείστον ψηφιακά δεδομένα, χρησιμοποιούνται διάφορες τεχνικές από εργαλεία και συστήματα στην κωδικοποίηση αρχείων, εικόνας, ήχου, βίντεο αλλά ακόμα και microdots και null ciphers.

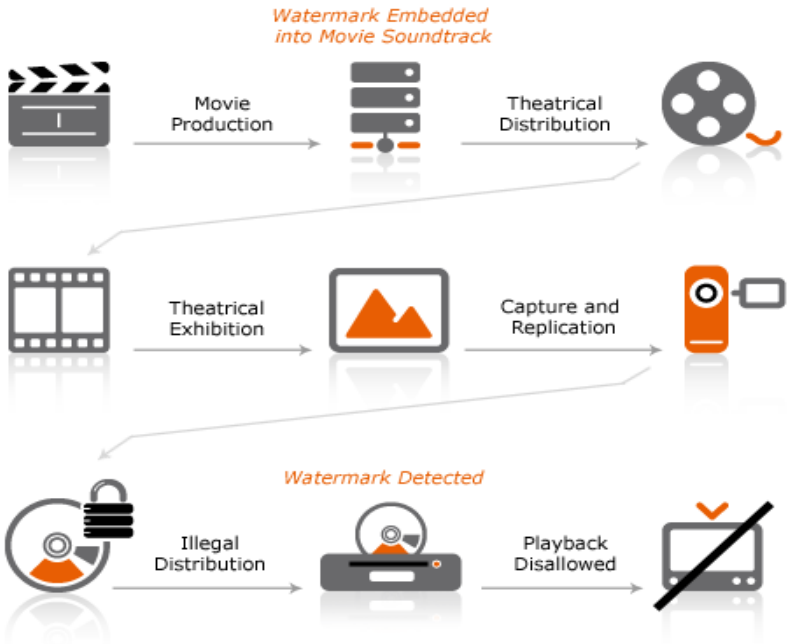
5 Εφαρμογές Στεγανογραφίας

- **Παραμετροποίηση και εμπλουτισμός Δομών Δεδομένων**

Καθώς οι δομές δεδομένων μπορούν να γίνουν παρωχημένες, υπάρχει ανάγκη για εμπλουτισμό τους με νέους τύπους δεδομένων και μάλιστα χωρίς αυτό να μπορεί να προβλεφτεί. Στην περίπτωση αυτή η Στεγανογραφία μπορεί να αποτελέσει λύση. Για παράδειγμα μπορεί να προστεθεί επιπλέον πληροφορία σε μια εικόνα ως μέρος της χωρίς κάποια οπτική αλλοίωση. Αυτό θα μπορούσε να αποτρέψει ένα πληροφοριακό σύστημα από κάποιο κόστος αναβάθμισης προκειμένου να διατηρήσει και να διαχειριστεί την επιπλέον πληροφορία. Επί παραδείγματι, επιπλέον πληροφορία μπορεί να εισαχθεί σε μια ακτινογραφία γλυτώνοντας το πληροφοριακό σύστημα από επιπρόσθετη αναβάθμιση του συστήματος για τη διατήρηση της[4].

- **Υδατογραφήματα**

Τα υδατογραφήματα είναι ένας τρόπος να τοποθετηθεί επιπλέον πληροφορία κρυμμένη μέσα σε ψηφιακό περιεχόμενο από τους δημιουργούς του όπως σε βιβλία, αρχεία ήχου και ταινίες που σκοπό έχει να προστατέψει τα πνευματικά δικαιώματα του αρχείου ή να δηλώσει τους όποιους περιορισμούς προς τους χρήστες όσον αφορά στη χρήση τους. Στην εικόνα 1 μπορεί να φανεί πιο καθαρά ένα παράδειγμα προστασίας και περιορισμού κατά της μη νόμιμης χρήσης.



Εικόνα 1: Προστασία μέσω του Υδατογραφήματος, Downloaded 10/2015, <http://www.digitalwatermarkingalliance.org/app_avcontent.asp>

- **Αναγνώριση-Ταυτοποίηση νόμιμου κατόχου**

Σε περίπτωση διανομής του ψηφιακού υλικού σε μη εξουσιοδοτημένα άτομα ό νόμιμος κάτοχος του αρχείου να κινηθεί εναντίων τους μέσω εργαλείων που θα μπορούν να ανιχνεύσουν την κρυφή πληροφορία που θα είναι εισαγμένη ανάμεσα στα δεδομένα του αρχείου (π.χ. επιχειρήσεις ή σώματα ασφαλείας για την ανιχνευσιμότητα των αρχείων)[4].

- **Αυθεντικοποίηση αρχείων**

Η κρυφή πληροφορία μπορεί να εισαχθεί ως μια ψηφιακή υπογραφή προκειμένου να μπορεί να πιστοποιηθεί ότι το σωστό ή το νόμιμο πρόσωπο το έχει υπογράψει[4].

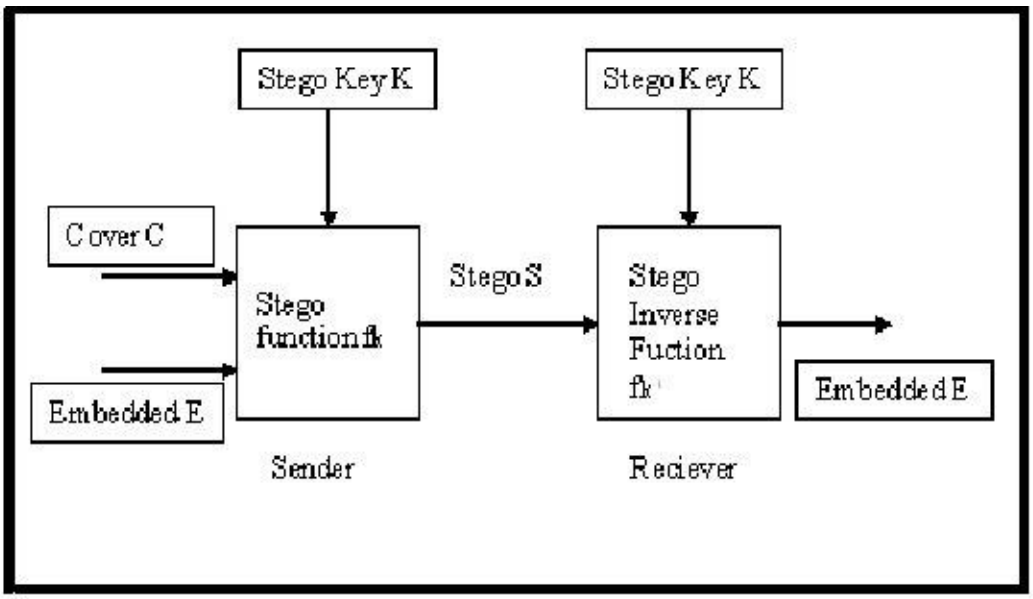
- **Ιδιωτικότητα στην επικοινωνία**

Στην ιδιωτικότητα της επικοινωνίας η Στεγανογραφία μπορεί να αποτελέσει το μέσω για την υλοποίηση και διασφάλιση της σε συνθήκες αυστηρούς επιτήρησης από κυβερνήσεις ή κατασκοπείας στο Διαδίκτυο αλλά και σε άλλα ψηφιακά μέσα επικοινωνίας. Η επικοινωνία αυτή μπορεί να περιλαμβάνει την ανταλλαγή προσωπικών ή και ευαίσθητων δεδομένων ανάμεσα σε δύο μέρη.

5.1 Ψηφιακή Στεγανογραφική Επικοινωνία και Πρωτόκολλα

Μια τέλεια ψηφιακά Στεγανογραφική επικοινωνία συνεπάγεται μια απολύτως κρυφή ψηφιακή επικοινωνία μεταξύ δυο μερών. Στην πραγματικότητα αυτό είναι πολύ δύσκολο να υπάρξει. Αν υποθέσουμε ότι αυτό συμβαίνει τότε όπως περιγράφεται από την εικόνα 2 ο αποστολέας του κρυφού μηνύματος E στέλνει το μήνυμα αυτό στον παραλήπτη κωδικοποιώντας το μέσα σε ένα αρχείο C χρησιμοποιώντας κάποιο κλειδί K σε συνδυασμό με έναν αλγόριθμο fk. Το αποτέλεσμα είναι το Στεγανογραφημένο αντικείμενο-αρχείο S. Ο παραλήπτης προκειμένου να εξάγει τα δεδομένα του κρυφού μηνύματος πρέπει να γνωρίζει το κλειδί σε συνδυασμό με την αντίστροφη διαδικασία

fk΄ του αλγορίθμου που χρησιμοποιήθηκε κατά την κωδικοποίηση. Το αποτέλεσμα είναι το κρυφό μήνυμα του εισήγαγε το αποστολέας, Στην τέλεια επικοινωνία αν κάποιος άνθρωπος ή υπολογιστής ελέγξει τα δύο αρχεία C και S, το αρχείο C δεν θα πρέπει να διαφέρει σε σύγκριση με το τροποποιημένο αρχείο S.



Εικόνα 2: Το Στεγανογραφικό Σύστημα, Downloaded 10/2015, < https://bitsofbinary.wordpress.com/2011/10/18/an-introduction-to-lexical-steganography/>

Πού στηρίζεται όμως η Στεγανογραφική αυτή επικοινωνία; Στηρίζεται στα πλεονάζοντα(redundant) δεδομένα ή στον θόρυβο που υπάρχει μέσα στα ψηφιακά δεδομένα του αρχείο C που λειτουργεί ως κάλυψη για τη μεταφορά του μηνύματος E. Κι αυτό γιατί κατά τη διαδικασία της στεγανογράφησης γίνεται αντικατάσταση των πλεοναζόντων δεδομένων ή του θορύβου με τα δεδομένα του κρυφού μηνύματος. Τα πλεονάζοντα δεδομένα είναι τα δεδομένα αυτό που προσφέρουν μεγαλύτερη ακρίβεια στην περιγραφή ενός αρχείου από όση χρειάζεται για να χρησιμοποιηθεί και να προβληθεί. Αυτό σημαίνει ότι τα δεδομένα αυτά μπορούν να αλλοιωθούν ή και να παραληφθούν χωρίς το αρχείο να υποστεί σημαντικές αλλοιώσεις και χωρίς αυτές να μπορούν να ανιχνευθούν εύκολα[5].

Όμως, η φύση της διαδικασίας αυτής μας οδηγεί στο συμπέρασμα ότι υπάρχουν περιορισμοί στην ποσότητα των δεδομένων που μπορούν να σταλούν.

Η Στεγανογραφική αυτή επικοινωνία συντελείται στην πράξη με τη χρήση ενός από τα παρακάτω πρωτόκολλα[2]:

- Στεγανογραφία χωρίς χρήση κλειδιού
- Στεγανογραφία με χρήση μυστικού κλειδιού
- Στεγανογραφία με χρήση δημόσιου κλειδιού

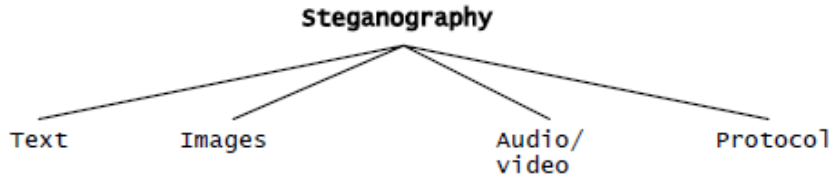
Η Στεγανογραφία χωρίς τη χρήση κλειδιού είναι η λιγότερο ασφαλής μέθοδος και προϋποθέτει ότι μόνο τα δύο μέρη που επικοινωνούν γνωρίζουν την ύπαρξη του κρυφού μηνύματος.

Η Στεγανογραφία με τη χρήση μυστικού κλειδιού απαιτεί την ανταλλαγή μεταξύ των μερών ενός κοινού μυστικού κλειδιού. Το κλειδί χρησιμοποιείται στη διαδικασία κωδικοποίησης του μηνύματος μέσα στο αρχείο ή μήνυμα που χρησιμοποιείται ως κάλυψη και είναι απαραίτητο προκειμένου να αντιστραφεί η διαδικασία. Αυτό κάνει τη διαδικασία πιο ασφαλή αφού ακόμα κι αν η μυστική επικοινωνία γίνει αντιληπτή και υποκλαπεί το αρχείο ή μήνυμα που λειτουργεί ως κάλυψη, να μην μπορεί να εξαχθεί το κρυφό μήνυμα χωρίς τη γνώση του κλειδιού.

Στην περίπτωση της **Στεγανογραφίας δημόσιου κλειδιού**, ο αποστολέας χρησιμοποιεί το δημόσιο κλειδί του στη διαδικασία απόκρυψης του μηνύματος ενώ ο παραλήπτης εξάγει το μήνυμα με το ιδιωτικό του κλειδί. Η μέθοδος αυτή έχει όλα τα πλεονεκτήματα της Στεγανογραφίας με χρήση κλειδιού μαζί με τα πλεονεκτήματα που προσφέρει η μέθοδος του δημοσίου-ιδιωτικού κλειδιού που χρησιμοποιεί η κρυπτογραφία.

5.2 Εφαρμογές Στεγανογραφίας σε ψηφιακά αρχεία

Σχεδόν όλα τα ψηφιακά αρχεία μπορούν να χρησιμοποιηθούν για τη Στεγανογραφία καθώς σε όλα υπάρχουν πλεονάζοντα bit δεδομένων όμως όπως αναφέρθηκε και πιο πάνω υπάρχουν περιορισμοί στη «χωρητικότητα» του αρχείου που θα λειτουργήσει ως κάλυψη. Τα αρχεία που ενδείκνυνται περισσότερο για χρήση είναι οι εικόνες και τα αρχεία ήχου χωρίς να αποκλείεται η χρήση της και σε άλλα όπως παρουσιάζεται στην ενότητα αυτή. Στην εικόνα 3 μπορούμε να δούμε τις κυριότερες κατηγορίες των αρχείων όσον αφορά τη χρήση της Στεγανογραφίας[5].



Εικόνα 3: Κατηγορίες χρήσεις αρχείων στη Στεγανογραφία, Χρήση από [5]

- **Κωδικοποίηση σε αρχεία κειμένου**

Η κωδικοποίηση μηνυμάτων σε αρχεία κειμένου είναι η λιγότερο ενδεδειγμένη μέθοδος για την εφαρμογή Στεγανογραφίας για το λόγο ότι τα αρχεία αυτά έχουν μικρό αριθμό πλεοναζόντων δεδομένων που θα μπορούσαν να αντικατασταθούν με τα δεδομένα της εισηγμένης κρυφής πληροφορίας. Επίσης τα αρχεία αυτά μπορούν να αλλοιωθούν εύκολα το κείμενο είτε τη διαμόρφωση των αρχείων αυτών π.χ. από .TXT σε .DOC ή .PDF στα οποία και βασίζεται η εισαγωγή του κρυφού μηνύματος. Παρά ταύτα είναι εφικτή η χρήση τους και μπορεί να υλοποιηθεί με διάφορους τρόπους όπως Ολίσθηση κατά γραμμή (χρησιμοποίηση κάθετων κενών) ή κατά γράμμα (χρησιμοποίηση οριζόντιων κενών) τα οποία χρησιμοποιούνται για τη αντιστοίχηση και «μετάφραση» αυτών των σημείων με τα σύμβολα του κρυφού μηνύματος. Επίσης η κωδικοποίηση μπορεί να επιτευχθεί μέσα από την τροποποίηση ιδιοτήτων στη διαμόρφωση του κειμένου των αρχείων αυτών όπως την κάθετο ή οριζόντιο μήκος των γραμμάτων που χρησιμοποιούνται. Ένα άλλα πρόβλημα που υπάρχει στις μεθόδους αυτές είναι ότι απαιτείται ότι ο παραλήπτης γνωρίζει αρχική διαμόρφωση των αρχείων αυτών πριν την κωδικοποίηση για να αντιληφθεί τις αλλαγές και να εξάγει το κρυφό μήνυμα[2].

- **Κωδικοποίηση σε αρχεία εικόνας**

Ένας από τους πιο ενδεδειγμένους και αποτελεσματικούς τύπους αρχείων για χρήση Στεγανογραφίας είναι τα αρχεία εικόνας. Αυτό λόγω τις περιορισμένης ικανότητας της ανθρώπινης όρασης να αντιληφθεί σχεδόν ανεπαίσθητες ψηφιακές αλλαγές στα δεδομένα των εικόνων. Αυτό τις καθιστά ιδανικούς «ξενιστές» για κείμενα, εικόνες, και άλλους τύπους αρχείων που μπορούν να περιγραφούν ως μια σειρά από ψηφιακά δεδομένα. Ενδεικτικά η κωδικοποίηση μπορεί να γίνει είτε στο **πεδίο του χώρου** μιας δισδιάστατης εικόνας χρησιμοποιώντας αλγορίθμους αντικατάστασης LSB(Least Significant Bit) ο οποίος χρησιμοποιείται στην πράξη και από την

εφαρμογή που παρουσιάζεται σε αυτήν την εργασία και μελετηθεί εκτενέστερα αργότερα, είτε στο **πεδίο των συχνοτήτων** χρησιμοποιώντας αλγορίθμους μετασχηματισμού της εικόνας όπως ο αλγόριθμος DCT (Discrete Cosine Transformation) αλλά και άλλες τεχνικές που είναι έκτος της σκοπιάς αυτής της εργασίας. Άλλες τεχνικές μπορεί να είναι πιο είτε περισσότερο είτε λιγότερο ανθεκτικές σε σφάλματα και αλλαγές στην εικόνα τα οποία μπορεί να επηρεάσουν την ορθή εξαγωγή του κρυφού μηνύματος αλλά και στην ανίχνευση της μυστικής επικοινωνίας ελέγχοντας την εικόνα οπτικά αλλά και στατιστικά για αλλοιώσεις[5].

- **Κωδικοποίηση σε αρχεία ήχου**

Στην περίπτωση αυτή το κρυφό μήνυμα εισάγεται στον ψηφιοποιημένο ήχο με αλλαγή τις δυαδικής ακολουθίας των bits του αρχείου. Ενδεικτικά κάποιες από τις τεχνικές που χρησιμοποιούνται είναι

- LSB: Εδώ γίνεται αντικατάσταση των LSBs τις δυαδικής ακολουθίας των bits που προκύπτουν κατά την ψηφιοποίηση[6].

-Κωδικοποίηση Φάσης: Σε αυτήν την τεχνική γίνεται αντικατάσταση της αρχικής φάσης (συχνότητας) ενός τμήματος από κάποιο αρχείο ήχου με μία άλλη αντίστοιχη φάση που αναπαριστά το κρυφό μήνυμα. Αυτό επιτυγχάνεται με τη χρήση του αλγόριθμου μετασχηματισμού Discrete Fourier Transform(DFT)[2].

-Εύρος Φάσματος(Spread Spectrum): Γίνεται κωδικοποίηση σχεδόν σε όλο το φάσμα των συχνοτήτων πολλαπλασιάζοντας το σήμα της πληροφορίας που θέλουμε να σταλεί με μια ψευδό-τυχαία ακολουθία θορύβου. Με τη διαμόρφωση αυτή, η ισχύς του σήματος πληροφορίας παραμένει χαμηλότερα από την ισχύ του θορύβου με αποτέλεσμα αυτό να μην γίνεται ανιχνεύσιμο. Στη συνέχεια το παραγόμενο σήμα εισάγεται στο σήμα κάλυψης με τη μορφή θορύβου. Μια άλλη προσέγγιση είναι η διαμόρφωση της συχνότητας του σήματος κάλυψης με σκοπό να ταιριάξει και να δημιουργηθεί μια ομοιόμορφη παραγωγή συχνότητας σε σχέση με τη συχνότατα της μεταδιδόμενης κρυφής πληροφορίας[6]. Το εύρος φάσματος είναι η πιο ασφαλής μέθοδος όσον αφορά τον ήχο όμως μπορεί να εισάγει τυχαίο θόρυβο στον ήχο με συνέπεια την πιθανότητα να χαθούν δεδομένα[2].

- **Κωδικοποίηση Βίντεο**

Τα αρχεία βίντεο αποτελούνται αποτελούνται από καρέ εικόνων και ήχο και άρα οι πιο πάνω χαρακτηριστικές μέθοδοι μπορούν να εφαρμοστούν και σε αυτής την περίπτωση.

- **Κωδικοποίηση Πρωτοκόλλων επικοινωνίας**

Σε αυτήν την μέθοδο, η κρυφή πληροφορία εισάγεται μέσα στα πακέτα επικοινωνίας του πρωτοκόλλου OSI. Για παράδειγμα μπορεί να εισαχθεί επιπλέον πληροφορία στις κεφαλίδες των πακέτων του TCP/IP πρωτοκόλλου μέσα σε παιδία που μεταφέρουν εναλλακτικά πληροφορία(optional) ή που δεν χρησιμοποιούνται καθόλου[5].

6 Χρήση στο πλαίσιο της εφαρμογής

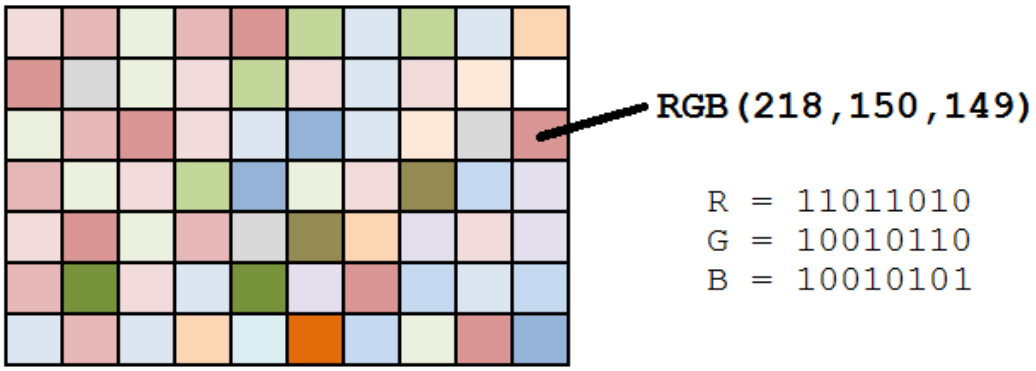
Στην εργασία αυτή επιλέγεται, όπως προαναφέρθηκε, η χρήση Στεγανογραφίας στο πεδίο του χώρου και συγκεκριμένα με τη χρήση του αλγορίθμου LSB. Ο λόγος όσον αφορά την επιλογή αρχείων εικόνας είναι προφανής μιας και όπως περιγράφηκε νωρίτερα αποτελούν πρόσφορο έδαφος για να φιλοξενήσουν κρυφή πληροφορία. Η επιλογή του αλγορίθμου έγινε με κριτήριο ότι αποτελεί ένα καλό, ενδεικτικό και κατανοητό παράδειγμα χωρίς υψηλή πολυπλοκότητα πράγμα που διευκολύνει την υλοποίηση του στα πλαίσια της εφαρμογής μας σε περιβάλλον κινητής συσκευής Android.

Ο σκοπός της ανάπτυξης και υλοποίησης της εφαρμογής είναι η δημιουργία ενός εργαλείου – μέσου ανταλλαγής συμμετρικού κλειδιού μεταξύ δυο μερών θέλοντας να προστατέψουμε την ιδιωτικότητα στην επικοινωνία της ευαίσθητης αυτής πληροφορίας.

Αμέσως μετά γίνεται μια γενική και θεωρητική περιγραφή της τεχνικής του αλγορίθμου LSB για την κωδικοποίηση εικόνας, ενώ η περιγραφή της εφαρμογής και η υλοποίηση του αλγορίθμου σε Android περιγράφεται αργότερα σε επόμενη ενότητα.

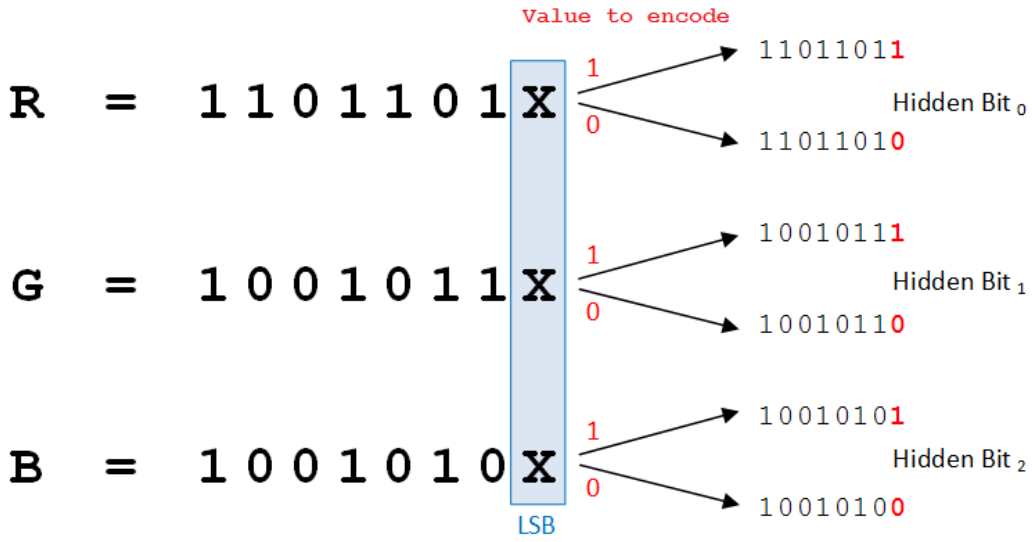
6.1 Κωδικοποίηση εικόνας με τη χρήση LSB

Όσον αφορά την κωδικοποίηση με χρήση LSB για τη δημιουργία Στεγανογραφημένης εικόνας, υπενθυμίζεται ότι πρόκειται για την πιο απλή μέθοδο για την εφαρμογή Στεγανογραφίας σε εικόνα. Σε αυτήν την περίπτωση, όπως δηλώνει και το ακρωνύμιο, το λιγότερο σημαντικό bit από ένα ή και από όλα τα byte από τα οποία αποτελείται μια εικόνα αντικαθίσταται από κάθε ένα από τα bits του κρυφού μηνύματος. Στην πράξη όταν έχουμε μια εικόνα 24bit δηλαδή που το κάθε pixel της αποτελείται από τρία bytes, ένα byte για κάθε χρώμα του το συνθέτει. Ένα για το κόκκινο(red), ένα για το πράσινο(green) και ένα για το μπλε(blue) – RGB όπως φαίνεται στην εικόνα 4.



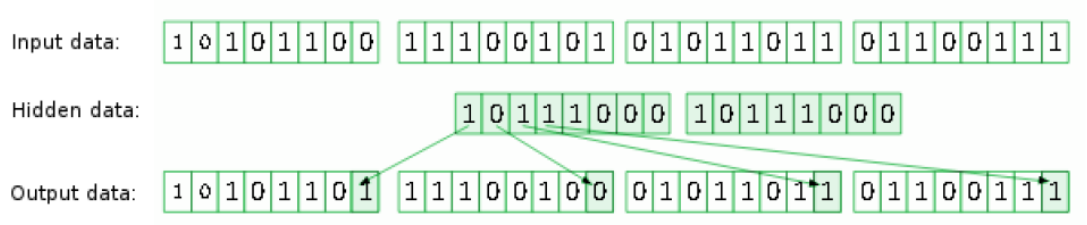
Εικόνα 4: Αναπαράσταση pixel 24bit σε RGB

Έτσι κάθε pixel μπορεί να αποθηκεύσει 3 bits από το κρυφό μήνυμα, Στην εικόνα 5 φαίνεται πώς κωδικοποιείται ανά RGB ενός pixel, 3 bits κρυφής πληροφορίας στην περίπτωση που μπορεί είναι είτε 1, είτε 0.



Εικόνα 5: Κωδικοποίηση LSB ανά RGB

Σε λίγο μεγαλύτερη κλίμακα η εισαγωγή των bit κρυφής πληροφορίας συνεχίζεται για όσο διάστημα τα bit αυτά μπορούν να αντιστοιχιστούν σε ένα LSB άρα σε ένα byte δεδομένων της αρχικής εικόνας όπως φαίνεται στην εικόνα 6



Εικόνα 6: Αντιστοίχηση 1 bit σε κάθε byte

Μια εικόνα με ανάλυση 800 X 600 pixels μπορεί να αποθηκεύσει 3 bits σε κάθε ένα από τα 480.000 pixels, συνολικά 1.440.000 bits ή αλλιώς 180.000 bytes(180KB). Όπως προκύπτει λοιπόν υπάρχει περιορισμός στην χωρητικότητα(Capacity) των bits κρυφού μηνύματος που μπορεί να φιλοξενήσει μια εικόνα που είναι ίση με C = (N^o of Pixels *3) * 1 αν πρόκειται για αντικατάσταση ενός bit ή (N^o of Pixels *3) * 2 για αντικατάσταση δύο bit.

Όπως μπορούμε να παρατηρήσουμε προηγούμενα δεν είναι απαραίτητη η αντικατάσταση των bit σε κάθε byte παρά μόνο αν διαφέρει. Για να δώσουμε ένα παράδειγμα, έστω ότι έχουμε 3 pixels μιας εικόνας 24bit εκφρασμένα σε bytes όπως πιο κάτω

(00101101 00011100 11011100)

(10100110 11000100 00001100)

(11010010 10101101 01100011)

Και θέλουμε να εισάγουμε το byte **10001101** τότε τα αρχικά byte θα μεταβάλλονταν ως εξής

(00101101 00011100 11011100)

(10100110 11000101 00001101)

(11010010 10101101 01100011)

Τα bit που μεταβλήθηκαν σε σχέση με τα αρχικά είναι αυτά που έχουν υπογραμμιστεί, δηλαδή μόνο δύο από τα συνολικά bit χρειάστηκε να αλλάξουν.

Πρέπει να σημειωθεί ότι για κάθε byte και άρα για κάθε χρώμα (RGB) οι διακριτές τιμές άρα και οι αποχρώσεις που μπορεί να πάρει είναι 2^8= 256 , επομένως οι αλλαγές ήταν ανεπαίσθητες για το ανθρώπινο μάτι. Ανάλογα με το μέγεθος και το βάθος χρώματος της εικόνας θα μπορούσαν να επιλεγούν προς αντικατάσταση ακόμα και τα δύο τελευταία bits χωρίς να μπορεί κανείς να δει τη διαφορά[5].

Το μειονέκτημα στο πιο πάνω παράδειγμα είναι ότι τα bits αποθηκεύονται σε συνεχόμενες LSB θέσεις κάτι το οποίο μπορεί να γίνει εύκολα αναγνώσιμο εάν υποπτευθεί κάποιος τη χρήση του LSB στη μυστική επικοινωνία. Μια λύση για ένα πιο ασφαλές σύστημα για τη μέθοδο αντικατάστασης του LSB θα ήταν ή χρήση κάποιου συμμετρικού κλειδιού το οποίο θα όριζε τη μεταβολή μόνο συγκεκριμένων pixel. Έτσι, ακόμα και αν υποπτευόταν μια τέτοια επικοινωνία ο επιτιθέμενος δεν θα μπορούσε να ανακτήσει την πληροφορία, χωρίς να γνωρίζει σε ποιες θέσεις θα πρέπει να διαβάσει, χωρίς να γνωρίζει το κλειδί[5].

6.2 Εφαρμογή σε Συσκευή Android

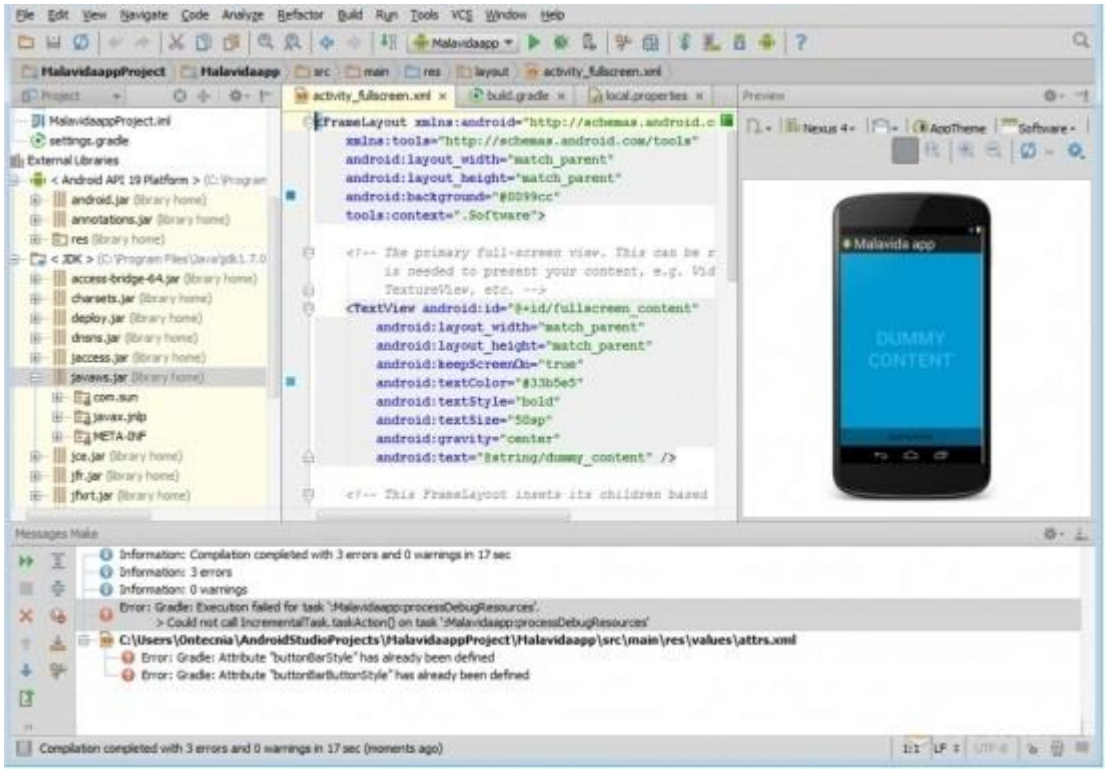
Σκοπός της εργασίας είναι η ανάπτυξη εφαρμογής σε κινητή συσκευή η οποία θα χρησιμοποιεί τη Στεγανογραφία ως εργαλείο για την ασφαλή μετάδοση συμμετρικού (ιδιωτικού) κλειδιού προωθώντας τη χρήση της λιγότερο πολύπλοκης, κατά τη διαδικασία, συμμετρικής[7] κρυπτογραφίας έναντι της ασύμμετρης[7], εξασφαλίζοντας παράλληλα τις αρχές της Εμπιστευτικότητας, της Ακεραιότητας και της Διαθεσιμότητας της πληροφορίας που μεταφέρεται. Στην περίπτωση μας η πληροφορία είναι το συμμετρικό κλειδί. Προτείνεται λοιπόν ένας ασφαλής και εναλλακτικός τρόπος μεταφοράς του κλειδιού κρυπτογράφησης χωρίς τη χρήση ασύμμετρης κρυπτογραφίας και κατά συνέπεια χωρίς την πολυπλοκότητα της χρήσης ενός Public Key Infrastructure(PKI)[8], μιας υποδομής δηλαδή για τη διαχείριση του ζεύγους δημόσιου – ιδιωτικού κλειδιού που χρησιμοποιείται στην συμμετρική κρυπτογράφηση.

Το λειτουργικό σύστημα που επιλέχθηκε να φιλοξενήσει την εφαρμογή είναι το περιβάλλον του Android , ένα «ανοιχτό» σύστημα με πολύ μεγάλη κοινότητα από developers καθώς και έγκυρη και έγκαιρη υποστήριξη από την εταιρία ανάπτυξής

του, τη Google. Ένα ακόμα στοιχείο που συνέτεινε στην επιλογή του Android είναι η υποστήριξη της Java(μιας ευρέως διαδεδομένης και με μεγάλη υποστήριξη γλώσσας προγραμματισμού)[9] ως κύριας γλώσσας προγραμματισμού για την ανάπτυξη της εφαρμογής, μιας γλώσσας προγραμματισμού επίσης με μεγάλη κοινότητα υποστήριξης της.

Η υποστήριξη κάθε επίδοξου developer για Android ενισχύεται από την Google μέσω της διατήρησης επίσημου portal (ιστοσελίδας) με οργανωμένο και κατατοπιστικό εγχειρίδιο ανάπτυξης μιας εφαρμογής από το Α ως το Ω καθώς και ενημέρωσης για τη δομή και λειτουργία του λειτουργικού Android[10].

Η εφαρμογή αναπτύχθηκε με τη χρήση Software Development Kit (SDK) που έχει αναπτυχθεί από την ίδια την Google, του Android Studio[11], και μπορεί να «κατέβει» και να εγκατασταθεί στο PC ελεύθερα και χωρίς καμία χρέωση. Μία άποψη του περιβάλλοντος του Android Studio φαίνεται στη εικόνα 7



Εικόνα 7: Android Studio SDK Interface

Η μελέτη του εγχειριδίου ανάπτυξης εφαρμογής του Android, καθώς και του Android Studio SDK είναι εκτός σκοπιάς αυτής της εργασίας, παρά ταύτα κάποια εισαγωγικά στοιχεία για καθαρά επεξηγηματικούς σκοπούς παρατίθενται στο παράρτημα

Α(Appendix A). Επίσης βασικές γνώσεις Java θεωρούνται δεδομένες και απαραίτητες για την κατανόηση του όποιου κομματιού κώδικα παρατίθεται σε αυτήν την εργασία.

6.2.1 Περιγραφή Τεχνικής και Αλγόριθμου

Η εφαρμογή σχεδιάστηκε για την επεξεργασία μιας δισδιάστατης ψηφιακής εικόνας εισάγοντας επιπλέον πληροφορία (ιδιωτικό κλειδί) εκφρασμένη σε bytes χρησιμοποιώντας τον αλγόριθμο αντικατάστασης LSB όπως αυτός έχει επεξηγηματικά δοθεί στην παράγραφο 5.1 .

Ο αλγόριθμος βασίζεται στο ότι κάθε pixel (εικονοστοιχείο) μιας δισδιάστατης εικόνας, των 32 bit, αποτελείται από 4 byte. Ένα byte για την ψηφιακή αναπαράσταση του alpha (A), ένα byte για το χρώμα κόκκινο(Red (R)), ένα για το πράσινο (Green (G)) και ένα byte για το μπλε (Blue (B))[12]. Η χρήση του alpha σε μία ψηφιακή εικόνα γίνεται για την εισαγωγή επιπλέον πληροφορίας για την διαφάνεια του pixel, εάν δηλαδή καλύπτεται είτε, εν μέρη ή και ολοκληρωτικά, είτε καθόλου από άλλα χρώματα, από άλλα στοιχεία[13]. Στην περίπτωση του αλγορίθμου επιλέγεται η επεξεργασία και εφαρμογή του LSB μόνο στα byte που αναπαριστούν τα RGB. Αυτό γίνεται για τη μη αλλοίωση του επιπέδου της διαφάνειας μιας 32 bit εικόνας, στην προσπάθεια μας να μην υπάρξει οπτικά εμφανής αλλοίωση στον τρόπο παρουσίασης της εικόνας σε σχέση με την εικόνα προ τροποποίησης. Ομοίως για να κρατηθεί όσο το δυνατόν χαμηλά το επίπεδο αλλοίωσης σε σχέση με την αρχική εικόνα λογίζεται ως διαθέσιμο προς αντικατάσταση μόνο το τελευταίο bit από κάθε byte των RGB σε κάθε pixel.

Ένα νέο στοιχείο που παρατίθεται πέρα από την κλασική χρήση της τεχνικής του LSB είναι η αντιμετάθεση που επιλέγεται να γίνει στη σειρά των pixels όπως αυτά αρχικά αναπαρίστανται, πριν την τροποποίηση τους με την εισαγωγή της επιπλέον μυστικής πληροφορίας (ιδιωτικό κλειδί), σε σειρά, το ένα pixel μετά το άλλο. Τα pixels τοποθετούνται πάλι στη σωστή τους σειρά αφού έχουν τροποποιηθεί και εν τέλει αναπαριστούν και πάλι την εικόνα χωρίς εμφανή αλλοίωση, με τα δεδομένα του ιδιωτικού κλειδιού να «φιλοξενούνται» πλέον ανάμεσα στα δεδομένα της εικόνας. Αυτή η αντιμετάθεση γίνεται για να προσδώσουμε ένα επιπλέον επίπεδο

ασφάλειας σε περίπτωση που κάποιος επίδοξος επιτιθέμενος - και αφού θα έχει υποπτευθεί ότι η εικόνα περιέχει επιπλέον ξένη πληροφορία – θελήσει να σαρώσει σειριακά την εικόνα με σκοπό να ανακαλύψει bit ανά bit δεδομένα bytes που να έχουν νόημα ως πληροφορία. Στην περίπτωση της αντιμετάθεσης από-συσχετίζονται με βάση τη θέση τους τα bit που εισάγονται και μπερδεύονται μεταξύ τους αλλοιώνοντας τη συσχέτιση τους με την αρχική θέση των byte της εικόνας. Η μέθοδος της αντιμετάθεσης όπως και ο αλγόριθμος περιγράφονται αργότερα.

Στη συνέχεια ακολουθούν τα βήματα του αλγορίθμου δίνοντας μια αφαιρετική προσέγγιση της λογικής που χρησιμοποιείται στον αλγόριθμο της εφαρμογής.

6.2.2 Κωδικοποίηση και Αποκωδικοποίηση

- **Κωδικοποίηση**

Η είσοδος και η έξοδος της εφαρμογής ως σύστημα κατά τη διαδικασία της κωδικοποίησης είναι:

Είσοδος Συστήματος: Εικόνα κάλυψης, Κείμενο (text) ιδιωτικού κλειδιού, Επιπλέον πληροφορία που είναι απαραίτητη για την αποκωδικοποίηση.

Έξοδος Συστήματος: Στεγανογραφημένη εικόνα που περιέχει το ιδιωτικό κλειδί κωδικοποιημένο.

Η κωδικοποίηση του ιδιωτικού κλειδιού μέσα στην εικόνα στα πλαίσια της εφαρμογής ακολουθεί τα εξής βήματα:

Βήμα 1: Μετατροπή των pixel της εικόνας σε έναν μονοδιάστατο πίνακα ακεραίων(integers). Κάθε pixel(4 byte) αναπαρίσταται από έναν ακέραιο (Στη Java 1 ακέραιος = 4 byte).

Βήμα 2: Μετατροπή του κειμένου του ιδιωτικού κλειδιού σε bytes

Βήμα 3: Αντιμετάθεση των στοιχείων του πίνακα των ακεραίων με σκοπό την αποσυσχέτιση τους σε σχέση με την ορθή σειρά τους.

Βήμα 4: Για κάθε bit της πληροφορίας που εισάγεται γίνεται σύγκριση με τα LSB κάθε byte της εικόνας, το ένα pixel μετά το άλλο. Εάν το bit του μηνύματος είναι ίδιο με το αντίστοιχο LSB τότε δεν γίνεται αντικατάσταση, εάν είναι διαφορετικό, τότε το LSB του αντίστοιχου κατά σειρά byte της εικόνας αντικαθίσταται από το αντίστοιχο bit του μηνύματος (If msg_bit != LSB then LSB = msg_bit). Όταν τελειώσει η αντιστοίχηση και σύγκριση όλων των bit της εισηγμένης πληροφορίας, τότε τα υπόλοιπα byte της εικόνας παραμένουν ως είχαν και συμπληρώνουν τα δεδομένα της εικόνας.

Βήμα 5: Επανατοποθέτηση των στοιχείων του πίνακα των ακεραίων στην ορθή-αρχική τους θέση.

Βήμα 6: Μετατροπή των ακεραίων σε pixels, byte ανά byte και ανασύνθεση της εικόνας ως Στεγανογραφημένης.

- **Αποκωδικοποίηση**

Η είσοδος και η έξοδος της εφαρμογής ως σύστημα κατά τη διαδικασία της αποκωδικοποίησης είναι:

Είσοδος Συστήματος: Στεγανογραφημένη εικόνα.

Έξοδος Συστήματος: Κείμενο (text) ιδιωτικού κλειδιού.

Η αποκωδικοποίηση του ιδιωτικού κλειδιού μέσα από τη Στεγανογραφημένη εικόνα ακολουθεί τα εξής βήματα:

Βήμα 1: Μετατροπή των pixel της εικόνας σε έναν μονοδιάστατο πίνακα ακεραίων(integers). Κάθε pixel(4 byte) αναπαρίσταται από έναν ακέραιο (Στη Java 1 ακέραιος = 4 byte).

Βήμα 2: Αντιμετάθεση των στοιχείων του πίνακα των ακεραίων με σκοπό την αποσυσχέτιση τους σε σχέση με την ορθή τους σειρά, όπως ακριβώς έγινε και κατά την κωδικοποίηση.

Βήμα 3: Ξεκινώντας από τη θέση του pixel-αντίστοιχου byte που ορίζεται, αλλά και για όσο ορίζεται από την επιπλέον πληροφορία που έχει εισαχθεί κατά την

κωδικοποίηση, ο αλγόριθμος διαβάζει τα LSB(bits) των byte της εικόνας κατά σειρά ομαδοποιώντας τα σε byte πληροφορίας.

Βήμα 4: Έλεγχος CRC32 των byte του κλειδιού που εξήχθη σε σχέση με το CRC32 που εισήχθη επίσης κατά την κωδικοποίηση ως επιπλέον πληροφορία.

Βήμα 5: Αν ο έλεγχος του CRC32 συμφωνεί, τότε γίνεται μετατροπή των εξαγόμενων byte πληροφορίας σε χαρακτήρες (Στη Java 1 χαρακτήρας = 1 byte), αλλιώς εμφανίζεται μήνυμα σφάλματος και δεν ολοκληρώνεται η διαδικασία με το βήμα 6.

Βήμα 6: Αποθήκευση των χαρακτήρων που αναπαριστούν το ιδιωτικό κλειδί σε ένα αρχείο κειμένου.

6.2.3 Περιγραφή του Κώδικα και Λειτουργικότητας εφαρμογής

Ο κώδικας της εφαρμογής είναι αυτός που προσδίδει λειτουργικότητα στην εφαρμογή και αποτελείται από **τέσσερα αρχεία java** (.java), τέσσερεις δηλαδή κλάσεις της java.

Το πρώτο αρχείο-κλάση ονομάζεται **Main_InterfaceActivity.java** και χρησιμοποιείται στο περιβάλλον Android για να δηλώσει την πρώτη και κύρια δραστηριότητα(Activity)[14] της εφαρμογής, δηλαδή συνδέεται με το γραφικό περιβάλλον διασύνδεσης χρήστη(Graphical User Interface) που συναντά ο χρήστης ανοίγοντας για πρώτη φορά την εφαρμογή.

Στο περιβάλλον Android ο όρος δραστηριότητα[14] χρησιμοποιείται για να αντικαταστήσει τη λειτουργικότητα που προσδίδει σε άλλα προγραμματιστικά περιβάλλοντα η κύρια μέθοδος main().

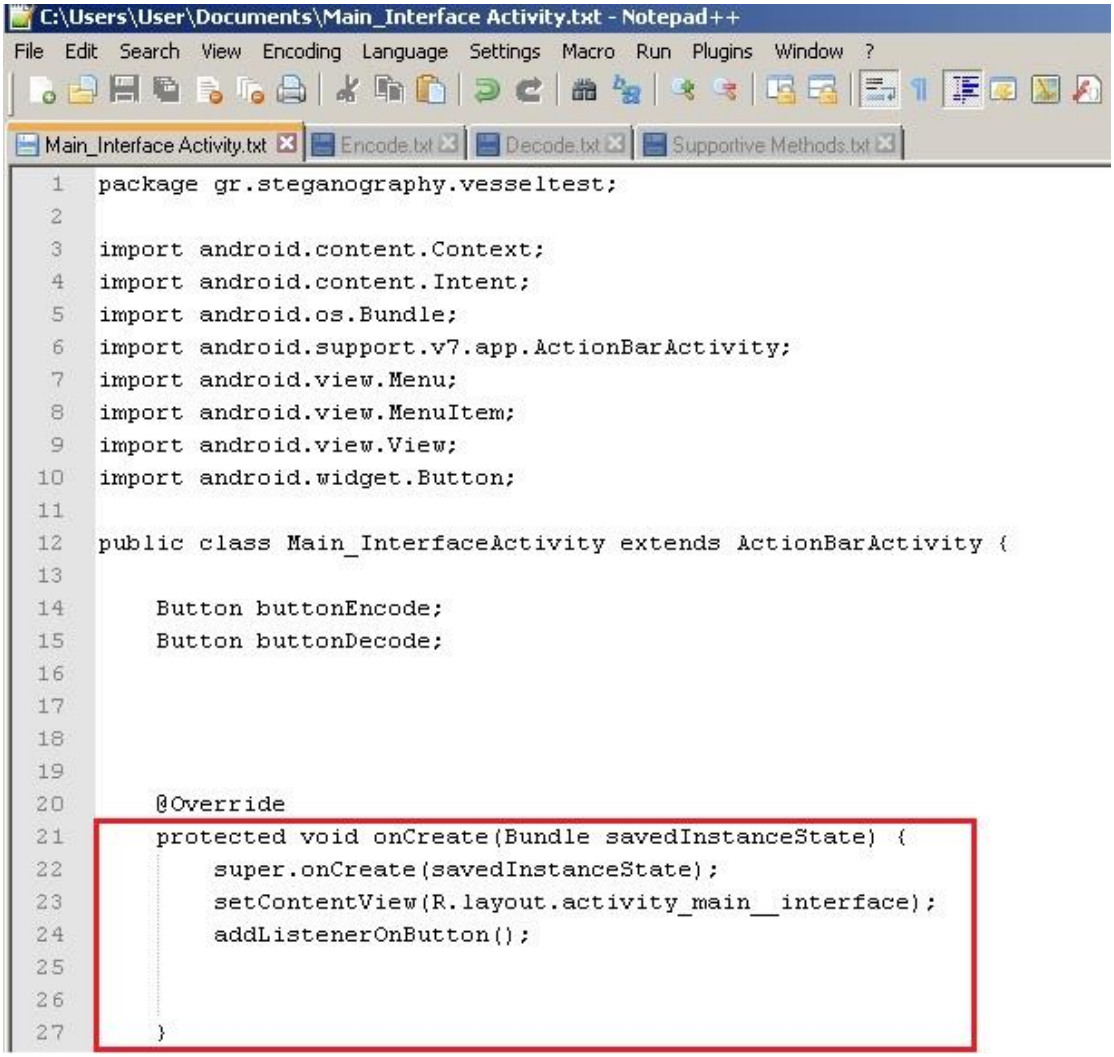
Μέσα από τη δραστηριότητα αυτή ο χρήστης μπορεί να επιλέξει να καλέσει, είτε τη δραστηριότητα της Κωδικοποίησης που συνδέεται με το αρχείο-κλάση **Encode.java**, είτε τη δραστηριότητα της αποκωδικοποίησης που συνδέεται με το αρχείο **Decode.java**. Τα αρχεία αυτά προσδίδουν λειτουργικότητα στο GUI της εφαρμογής που εμφανίζεται κατά την επιλογή της κωδικοποίησης και αποκωδικοποίησης αντίστοιχα.

Τέλος το τέταρτο αρχείο καλείται **Supportive_Methods.java** και περιλαμβάνει μεθόδους οι οποίες καλούνται μέσα από τις κλάσεις Encode και Decode προκειμένου να υλοποιηθεί η λειτουργικότητα που θέλουμε να προσδώσουμε στην εφαρμογή.

Περισσότερες πληροφορίες για την έννοια της δραστηριότητας μιας εφαρμογής Android καθώς και τη χρήση των αρχείων java αλλά και του GUI δίνονται στο Παράρτημα Α (Appendix A) και Παράστημα Β (Appendix B).

- **Κύρια Δραστηριότητα – αρχείο Main_InterfaceActivity.java**

Στην αρχή της κλάσης όπως και σε κάθε κλάση η οποία προσδίδει λειτουργικότητα συναντάμε τη μέθοδο onCreate() η οπόια χρησιμοποιείται για την εκκίνηση της δραστηριότητας και μπορεί συνήθως να υλοποιεί κάποιο GUI ή κάποιες μεταβλητές στη java[14]. Στη δική μας περίπτωση θέτει το αρχικό γραφικό περιβάλλον του χρήστη κατά την εκκίνηση μέσω της μεθόδου setContentView(); με παράμετρο το activity_main__interface, δηλαδή το αρχείο του interface που έχει προηγουμένος σχεδιαστεί κατά την υλοποίηση για να εμφανιστεί και επιπλέον παρακολουθεί το πάτημα κάποιου κουμπιού μέσα από την κλήση της μεθόδου addListenerOnButton(); όπως φαίνεται στην εικόνα 8

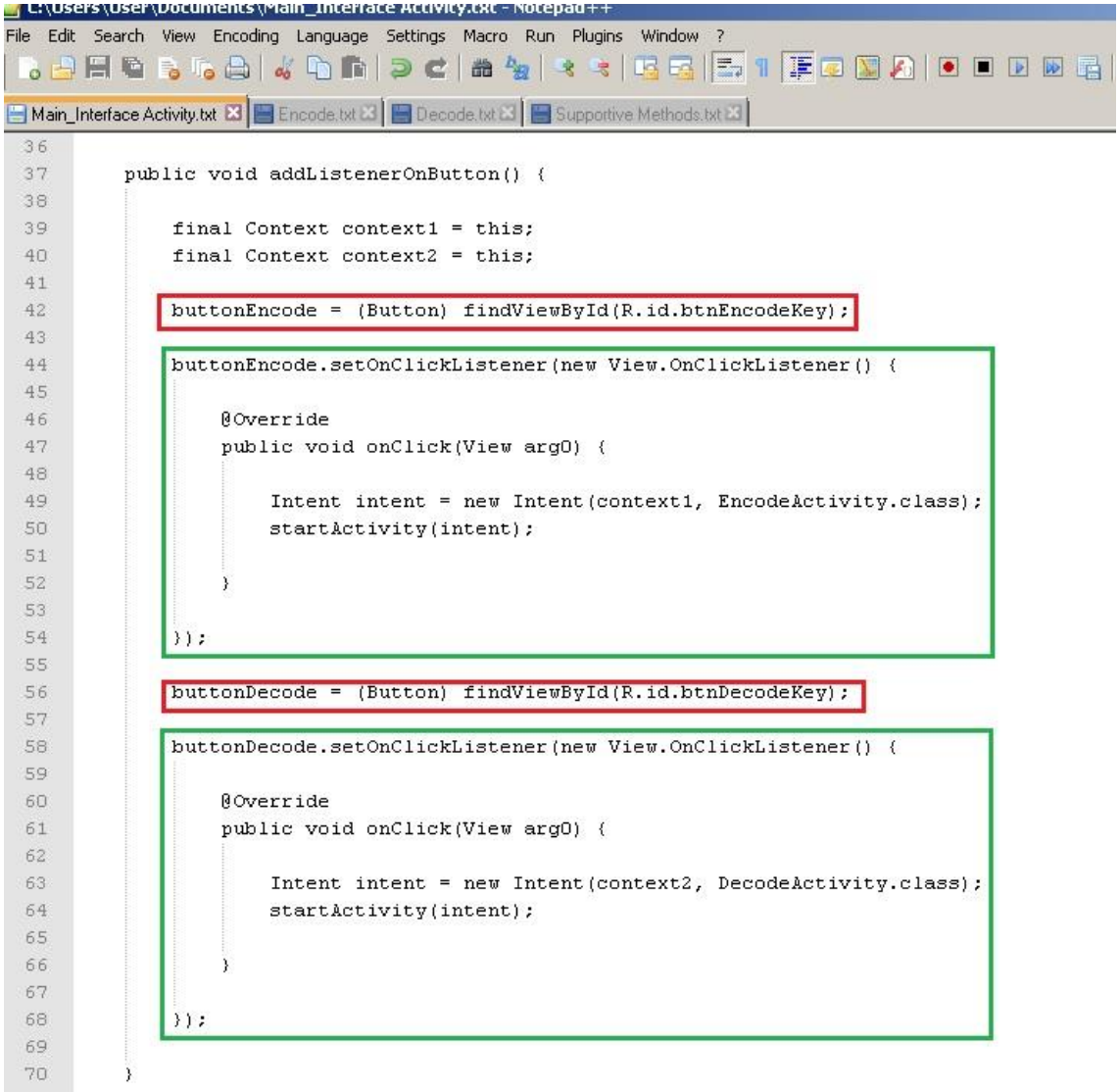


Εικόνα 8: Η μέθοδος onCreate() της εφαρμογής

Στην συνέχεια υλοποιούνται δύο κουμπιά που ορίζουν το περιεχόμενο (content) του αρχικής οθόνης(GUI) και καλούνται ως buttonEncode και buttonDecode προσδίδοντάς τους παράλληλα ιδιότητες οι οποίες ορίζονται από στοιχεία τα οποία καθορίζουν τον τρόπο εμφάνισης τους(btnEncodeKey, btnDecodeKey) όπως φαίνεται στα κόκκινα πλαίσια στην εικόνα 9. Τα btnEncodeKey και btnDecodeKey ορίζονται μέσα στα αρχεία layout (.xml) που είναι υπεύθυνα για την διαμόρφωση του περιεχομένου του GUI.

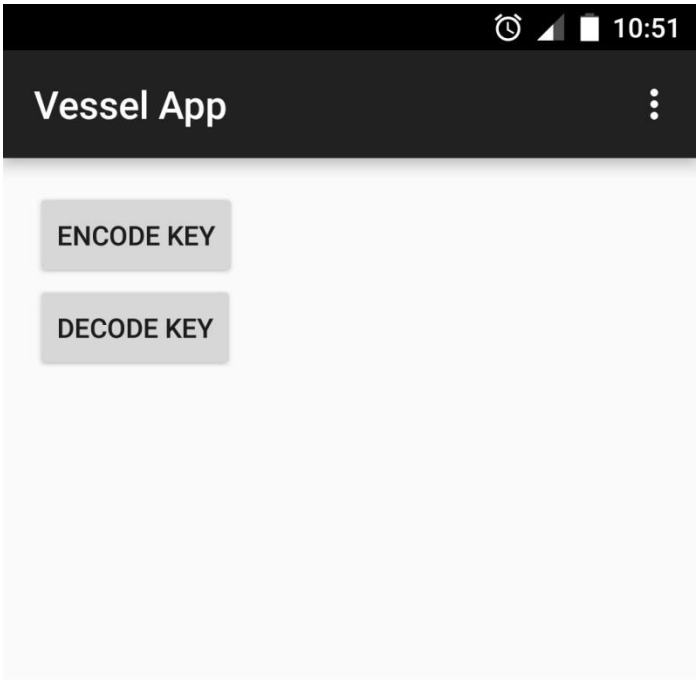
Τέλος η μέθοδος setOnClickListener(); ορίζει τη λειτουργικότητα των κουμπιών αυτών δηλώνοντας έναν Listener της java , μια μέθοδο δηλαδή που αναλαμβάνει να εκτελέσει μια ενέργεια σε περίπτωση που το κουμπί αυτό πατηθεί. Στη δική μας περίπτωση καλεί τις κλάσεις-δραστηριότητες της εφαρμογής EncodeActivity.class και DecodeActivity.class – και των αντίστοιχων GUI που συνδέονται με αυτές -

αντίστοιχα για το κάθε κουμπί της εφαρμογής όπως φαίνεται στην εικόνα 9 μέσα στα πράσινα πλαίσια.



Εικόνα 9: Η μέθοδος setOnClickListener(); και τα αντικείμενα buttonEncode και buttonDecode

Σαν αποτέλεσμα έχουμε το GUI όπως αυτό απεικονίζεται στην εικόνα 10, με τη λειτουργικότητα που περιγράφηκε προηγούμενα, δηλαδή δύο κουμπιά στην κύρια οθόνη της εφαρμογής τα οποία συνδέονται με τις λειτουργίες της κωδικοποίησης (Encode) και αποκωδικοποίησης (Decode) και τα οποία στο πάτημά τους ανοίγουν δύο ξεχωριστές οθόνες απεικόνισης για την κωδικοποίηση και αποκωδικοποίηση του κλειδιού αντίστοιχα.



Εικόνα 10: Πρώτη οθόνη της εφαρμογής κατά την εκκίνηση

- **Δραστηριότητα Κωδικοποίησης – αρχείο Encode.java**

Στην κλάση αυτή ορίζεται η λειτουργικότητα της διαδικασίας κωδικοποίησης. Αρχικά δηλώνεται η εισαγωγή(import) απαραίτητων για την κωδικοποίηση μεθόδων όπως είναι οι convertArray(); , encode_key(); , toBytes(); οι οποίες βρίσκονται στο αρχείο Supportive_Methods.java όπως φαίνεται και στην εικόνα 11 πριν από την έναρξη της κλάσης EncodeActivity. Ακολουθεί η δήλωση σταθερών και μεταβλητών μέσα στην EncodeActivity.

```
25
26  import static gr.steganography.vesseltest.Supportive_Methods.convertArray;
27  import static gr.steganography.vesseltest.Supportive_Methods.encode_key;
28  import static gr.steganography.vesseltest.Supportive_Methods.toBytes;
29
30  public class EncodeActivity extends Activity  {
31
32      private static final int PICK_IMAGE = 1;
33      private static final int STATE1 = 0;
34      private static final int STATE2 = 1;
35      private static final int STATE3 = 2;
```

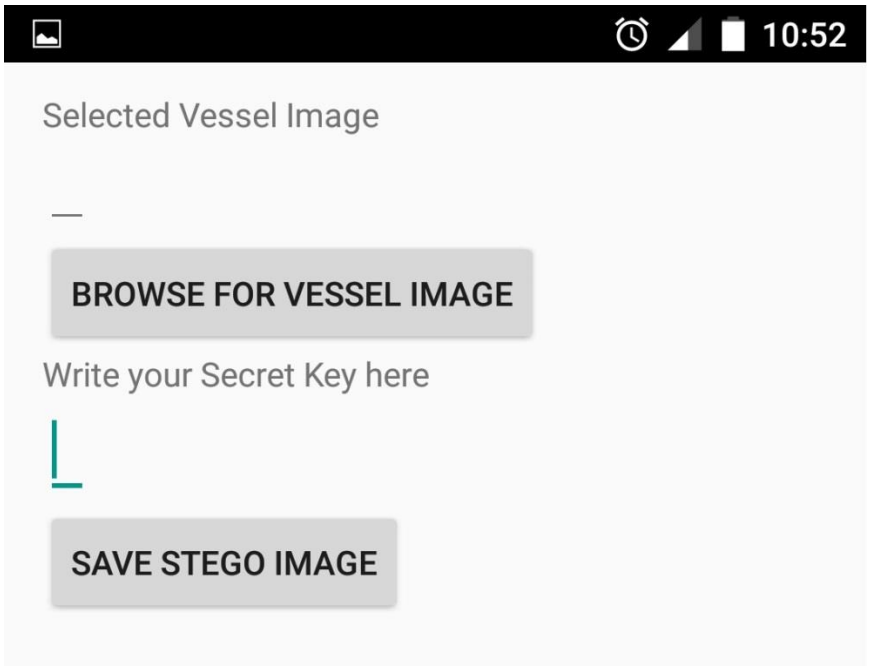
Εικόνα 11: Δήλωση εισαγωγής μεθόδων από την κλάση Supportive_Methods στην EncodeActivity

Με την κλήση της EncodeActivity ορίζεται και το γραφικό περιβάλλον που θα εμφανιστεί και αυτό γίνεται μέσα από τη μέθοδο onCreate(); , όπως και για την προηγούμενη κλάση η setContentView(); , θέτει λοιπόν το περιεχόμενο της οθόνης όπως ορίζεται από την παράμετρο activity_encode και παρουσιάζεται στην εικόνα 12

```
66      protected void onCreate(Bundle savedInstanceState) {
67          super.onCreate(savedInstanceState);
68          setContentView(R.layout.activity_encode);
```

Εικόνα 12: Ορισμός Γραφικού Περιβάλλοντος (GUI) για την EncodeActivity

Ως αποτέλεσμα της παραπάνω κλήσης είναι το γραφικό περιβάλλον που φαίνεται στην εικόνα 13 με δύο πεδία κειμένου και δύο κουμπιά



Εικόνα 13: Γραφικό Περιβάλλον EncodeActivity

Η λειτουργικότητα των δύο κουμπιών ορίζεται από τη μέθοδο onClick();. και αφού προηγουμένως το κάθε κουμπί έχει υλοποιηθεί και έχει οριστεί για αυτό ένας Listener που ελέγχει την περίπτωση του το κουμπί αυτό πατηθεί. Στην περίπτωση του πρώτου κατά σειρά κουμπιού υλοποιείται ένα αντικείμενο Intent που θα δηλώσει την πρόθεση να εκτελεστεί μια ενέργεια στο πάτημα του όπως γίνεται και για κάθε κουμπί. Ο κώδικας αυτής της λειτουργίας φαίνεται στην παρακάτω εικόνα

```
70          Button btnSelectImageVessel = (Button) findViewById(R.id.btnSelectImageVessel);
71          btnSelectImageVessel.setOnClickListener(new Button.OnClickListener() {
72
73              public void onClick(View v) {
74
75                  Intent photoPickerIntent = new Intent(Intent.ACTION_PICK);
76                  photoPickerIntent.setType("image/*");
77                  startActivityForResult(photoPickerIntent, PICK_IMAGE);
78
79              }
80
81          });
```

Εικόνα 14: Εκκίνηση λειτουργίας επιλογής εικόνας κάλυψης(Vessel Image)

Με τη δημιουργία του αντικειμένου αυτού μπορούμε να καλέσουμε τη δραστηριότητα για επιλογή της εικόνας κάλυψης (cover image) στη δική μας περίπτωση καλείται ως vessel image (όχημα). Αυτό συμβαίνει ανοίγοντας για επιλογή τις φωτογραφίες της κινητής συσκευής μέσα από την προεπιλεγμένη εφαρμογή προβολής εικόνων-φωτογραφιών της συσκευής. Η μέθοδος που πυροδοτεί αυτήν την αντίδραση είναι η startActivityForResult(); , με παραμέτρους το ίδιο το αντικείμενο και μια σταθερά – δείκτη PICK_IMAGE που χρησιμοποιείται από τη μέθοδο για την έναρξη της λειτουργίας αυτής.

Μέσα στη μέθοδο startActivityForResult(); συμβαίνουν τρία πράγματα, το πρώτο είναι η αποθήκευση της θέσης και του ονόματος της εικόνας που επιλέγει ο χρήστης μέσα από την κινητή συσκευή. Αυτό συμβαίνει με τη βοήθεια του αντικειμένου photoUri1 και του αλφαριθμητικού path που αποθηκεύουν δεδομένα τα οποία περιγράφουν τη θέση και το όνομα του αρχείου που έχει επιλεγεί. Αυτά περνάνε σαν ορίσματα σε ένα αντικείμενο Cursor του Android μέσω του οποίου τελικά η θέση του αρχείου επιλογής περνά στη μεταβλητή filePathImageVessel όπως φαίνεται στο πράσινο πλαίσιο στην εικόνα 15.

Το δεύτερο είναι ο ορισμός της θέσης του αρχείου ως κείμενο και εμφάνισης του μέσα στο πρώτο πεδίο κειμένου της εφαρμογής που χαρακτηρίζεται με το κείμενο Selected Vessel Image στην εικόνα 13. Ορίζεται λοιπόν μέσω της setText() του αντικειμένου TextView να εμφανιστεί μέσα στο πεδίο κειμένου EditTextImageVessel η θέση της επιλεγμένης εικόνας όπως φαίνεται στα μπλε πλαίσια της εικόνας 15.

Το τρίτο είναι η αποκωδικοποίηση της εικόνας που έχει επιλεγεί μέσω της μεθόδου decodeFile(); , της κλάσης BitmapFactory, ως ένα αντικείμενο Bitmap της java, με όνομα bmpVessel μέσα από το οποία θα γίνει η εξαγωγή αρχικά και έπειτα η

επεξεργασία των pixels της επιλεγμένης εικόνας. Η δημιουργία του Bitmap φαίνεται στο κόκκινο πλαίσιο στην εικόνα 15.

```
protected void onActivityResult(int requestCode, int resultCode,
                                Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);

    if (requestCode == PICK_IMAGE) {

        if (resultCode == RESULT_OK) {

            Uri photoUri1 = intent.getData();

            if (photoUri1 != null) {

                try {
                    TextView t = (TextView) this
                            .findViewById(R.id.EditTextImageVessel);

                    String[] path = {MediaStore.Images.Media.DATA};
                    Cursor cursor = getContentResolver().query(
                            photoUri1, path, null, null, null);

                    cursor.moveToFirst();

                    int idx1 = cursor
                            .getColumnIndex(path[0]);

                    filePathImageVessel = cursor.getString(idx1);
                    t.setText(filePathImageVessel);
                    BitmapFactory.Options opt = new BitmapFactory.Options();
                    opt.inDither = false;
                    opt.inScaled = false;
                    opt.inDensity = 0;
                    opt.inJustDecodeBounds = false;
                    opt.inSampleSize = 1;
                    opt.inScreenDensity = 0;
                    opt.inTargetDensity = 0;

                    bmpVessel = BitmapFactory.decodeFile(filePathImageVessel, opt);
                    selectFlag = true;
                    cursor.close();

                } catch (Exception e) {
                    e.printStackTrace();
                }
            } else {
                Log.v("URI NULL", "URI NOT RETURNED ");
            }
        }
```

Εικόνα 15: Η μέθοδος onActivityResult();

Το δεύτερο κουμπί εκκινεί την κύρια λειτουργία της κωδικοποίησης. Η κωδικοποίηση εκτελείται εφόσον έχει προηγουμένως επιλεγεί εικόνα κάλυψης, έχει συμπληρωθεί ιδιωτικό κλειδί μέσα στο δεύτερο πεδίο κειμένου της εικόνας 13 και η εικόνα που θέλουμε έχει την χωρητικότητα σε διαθέσιμα bytes (3 bytes ανά pixel και άρα 1 bit εισαγόμενων δεδομένων για κάθε διαθέσιμο byte) προκειμένου να

εισάγουμε όλα τα δεδομένα μας. Αυτό επιτυγχάνεται με μια σειρά ελέγχων πριν την εκκίνηση της κωδικοποίησης όπως θα παρουσιαστεί πιο κάτω.

Με το πάτημα του δεύτερου λοιπόν κουμπιού Save Stego Image της εικόνας 13 γίνεται αρχικά απόπειρα για λήψη και μετατροπή όλων των δεδομένων που είναι απαραίτητο να εισαχθούν στην εικόνα κάλυψης αλλά και για την επανασύσταση της αφού τελειώσει η διαδικασία της κωδικοποίησης.

Το πρώτο που γίνεται είναι η λήψη της ανάλυσης της εικόνας, δηλαδή το ύψος και το πλάτος της σε αριθμό pixels καθώς και την πυκνότητα της (αριθμός pixels ανά ίντσα ή εκατοστό) για την καλύτερη περιγραφή της ποιότητας της και της κλίμακάς της σε μία οθόνη. Έτσι έχουμε μια πλήρη εικόνα για την ποιότητα της. Το μήκος και πλάτος της εικόνας σε αριθμό pixels χρησιμεύει αρχικά για την δημιουργία ενός πίνακα που θα φιλοξενήσει τις τιμές των pixels που παίρνονται έπειτα μέσω της κλήσης της μεθόδου του Android getPixels(); της κλάσης Bitmap[15] με ορίσματα τον πίνακα που θα αποθηκεύσει τις τιμές και ορίσματα που δηλώνουν από πού θα ξεκινήσει και για πόσο διάστημα θα γίνει η εξαγωγή τους . Στην δική μας περίπτωση ξεκινάμε από το πρώτο pixel μέχρι και το τελευταίο. Τα παραπάνω περιγράφονται στον κώδικα της εικόνας 16.

```
88          public void onClick(View v) {
89
90
91  ////////////////////// Get Pixel Values(Integers) from Bitmap of Selected Image and Store it on an Int Array[] ///////////////
92              try {
93                  width = bmpVessel.getWidth();
94                  height = bmpVessel.getHeight();
95          ////////// An additional Info about Image Bitmap Resolution to describe better the scale of the image on Screen ////////////
96                  density = bmpVessel.getDensity();
97                  oneD = new int[width * height];
98                  bmpVessel.getPixels(oneD, 0, width, 0, 0, width, height);
99              } catch (Exception e) {
100                 // Do Nothing
101             }
```

Εικόνα 16: Η λήψη των pixels και των δεδομένων της εικόνας κάλυψης

Στη συνέχεια γίνεται λήψη του κλειδιού μέσα από το πεδίο κειμένου στο οποίο έχει τοποθετηθεί και αφού μετατραπεί, αποθηκεύεται ως bytes στον πίνακα τύπου byte [] Key όπως φαίνεται στον κώδικα στην εικόνα 17(κόκκινο πλαίσιο). Επίσης το μήκος

του κλειδιού αποθηκεύεται και μετατρέπεται σε πίνακα από bytes(πράσινο πλαίσιο εικόνας 17). Τα δεδομένα αυτά καθώς και άλλα δεδομένα που θα δούμε αμέσως μετά θα κωδικοποιηθούν μαζί με τα bytes του κλειδιού μέσα στην εικόνα και θα εξυπηρετήσουν στην αποκωδικοποίηση του κλειδιού όπως θα παρουσιαστεί αργότερα.

```
102 ////////////////////// Get Key into Bytes from Text Field of the App and keep also the length (Number of Bytes)/////////
103             try {
104                 text = (EditText) findViewById(R.id.message);
105                 key = text.getText().toString();
106                 Key = key.getBytes();
107
108                 int keyLength = Key.length;
109                 byteKeyLength = toBytes(keyLength);
110             } catch (Exception e) {
111                 // Do Nothing
112             }
```

Εικόνα 17: Λήψη Δεδομένων Ιδιωτικού Κλειδιού

Ως τρίτο κατά σειρά γίνεται λήψη του CRC32 checksum του κλειδιού που θα χρησιμεύσει στην ορθή επαλήθευση για την ακεραιότητα του και κατά την εξαγωγή του στη λειτουργία της αποκωδικοποίησης όπως φαίνεται στο κόκκινο πλαίσιο της εικόνας 18. Μαζί γίνεται και η λήψη του αριθμού των bytes του CRC32(πράσινο πλαίσιο εικόνας 18), πράγμα που θα χρησιμεύσει κατά τη λειτουργία της αποκωδικοποίησης όπως περιγράφηκε πιο πριν και για τον αριθμό των bytes του κλειδιού.

```
113 ////////////////// Get CRC32 from Key to be embed converted to bytes and CRC32 length(Number of Bytes)///////////
114             checksum.update(Key, 0, Key.length);
115             checksumValue = checksum.getValue();
116             String checksumString = Long.toString(checksumValue);
117             final byte[] checksumBytes = checksumString.getBytes();
118
119             int crcBytesNum = checksumBytes.length;
120             byte[] crcNum = toBytes(crcBytesNum);
```

Εικόνα 18: Λήψη CRC32 Checksum

Στο τέλος αυτής της πρώτης φάσης γίνεται μετατροπή και λήψη του flag, μιας μεταβλητής String, που θα μας βοηθήσει στην αναγνώρισή μιας εικόνας ως

Στεγανογραφημένης (θα περιέχει κλειδί κωδικοποιημένο) κατά την προσπάθεια να αποκωδικοποιήσουμε μια επιλεγμένη εικόνα όπως θα παρουσιαστεί αργότερα κατά την αποκωδικοποίηση. Επίσης έχουμε τη μετατροπή των Integer τιμών των Pixels της εικόνας σε bytes μέσω της μεθόδου convertArray() που βρίσκεται στο αρχείο Supportive_Methods.java. Αυτό θα εξυπηρετήσει αργότερα στην λειτουργία ελέγχου της χωρητικότητας της εικόνας (αν δηλαδή είναι αρκετά μεγάλη για να χωρέσει όλα τα δεδομένα που θέλουμε να εισάγουμε). Το κομμάτι αυτό του κώδικα φαίνεται στην εικόνα 19.

```
122 ////////////////////////////// String Flag and Int Pixel Values Converted into Bytes ////////////////////
123                 try {
124                     headerFlag = flag.getBytes();
125                     byteArrayVessel = convertArray(oneD);
126                 } catch (Exception e) {
127                     // Do Nothing
128                 }
```

Εικόνα 19: Μετατροπή μεταβλητής flag και τιμών pixels εικόνας σε bytes

Στη συνέχεια μια σειρά ελέγχων ακολουθούν, όπως σημειώθηκε και νωρίτερα, για τη διασφάλιση ότι θα έχουμε διαθέσιμα όλα τα δεδομένα που είναι απαραίτητα για την εφαρμογή προκειμένου να λειτουργήσει η κωδικοποίηση. Γίνεται λοιπόν έλεγχος τόσο για το αν έχει επιλεγεί μια εικόνα ως εικόνα κάλυψης, όσο και για το αν έχει συμπληρωθεί κλειδί στο πεδίο κειμένου και έχει τοποθετηθεί στη μεταβλητή Key όπως φαίνεται στην παρακάτω εικόνα.

```
//////////////////// Conditions and Controls made before starting the encoding process ///////////////

////////////////////// If there is no Image selected then Alert ////////////////

                if (width == 0) {
                    condition = 0;
//////////////////// If there is no Key filled in then Alert /////////////////

                } else if (Key.length == 0) {
                    condition = 1;
```

Εικόνα 20: Έλεγχος επιλογής εικόνας και κλειδιού

Ένας ακόμα έλεγχος είναι αυτός που αφορά τη χωρητικότητα της εικόνας κάλυψης όπως διατυπώνεται στην εικόνα 21.

```
145 ////////////////////// If Selected Image is too small to hide Key, Flag, Key Length, CRC32 and CRC Length and  info inside ///////////
146
147 //One pixel space at the end of each encoded data because while encoding, the process writes based on pixels and not bytes.
148 //We need to make sure each byte previously written remains intact
149
150                     } else if ((byteArrayVessel.length) < ((Key.length + headerFlag.length + byteKeyLength.length + crcNum.length
                    + checksumBytes.length + (2*3)) * 8)){
151                         condition = 3;
```

Χρειάζεται να αφήσουμε διαθέσιμα επιπλέον 2 pixels(2*(3bytes)), όπως φαίνεται στην εικόνα 21, για τα μέρη checksumBytes και Key, τις μεταβλητές δηλαδή που δεν μπορούμε να γνωρίζουμε εκ των προτέρων το πλήθος των byte που θα καταλάβουν μέσα στην εικόνα και στην περίπτωση τους θέλουμε να έχουμε ένα pixel απόστασή κατά ζεύγη μεταξύ checksumBytes και crcNum αλλά και Key και checksumBytes. Δηλαδή να μην υπάρχει περίπτωση να επικαλυφτεί κατά τη διαδικασία της κωδικοποίησης, εκεί που τελειώνουν τα μέρη, το ένα με το άλλο σε κάθε ζευγάρι που αναφέραμε. Αυτό έχει να κάνει με τη λειτουργία της μεθόδου encode(); και το ότι η εγγραφή γίνεται κατά pixels και επαναλαμβάνεται από την αρχή για κάθε μέρος ξεχωριστά (Ξεχωριστώ «σκανάρισμα» της εικόνας και εγγραφή κάθε μέρους). Για τα τρία υπόλοιπα μέρη το μέγεθός τους - και άρα το πλήθος των pixels που απαιτείται - είναι γνωστό μιας και πρόκειται για 3 ακεραίους (integers) όπου στη Java σημαίνει 4 bytes το κάθε ένα.

```
157 ////////////////////////////// Choose between conditions ////////////////////
158                 switch (condition) {
159                     case STATE1:
160                         new Thread() {
161                             public void run() {
162                                 EncodeActivity.this.runOnUiThread(new Runnable() {
163                                     public void run() {
164 
165                                         try {
166 
167                                             final AlertDialog.Builder vesselAlertDialog = new AlertDialog.Builder(EncodeActivity.this);
168 
169                                             vesselAlertDialog.setMessage("Please choose an image...")
170                                                     .setCancelable(false)
171                                                     .setPositiveButton("OK", new DialogInterface.OnClickListener() {
172                                                         public void onClick(DialogInterface dialog, int id) {
173 
174                                                             dialog.dismiss();
175                                                         }
176                                                     });
177                                             vesselAlertDialog.setCancelable(false);
178 
179                                             vesselAlertDialog.show();
180 
181                                         } catch (Exception e) {
182                                             // Do Nothing
183                                         }
184 
185                                     }
186                                 });
187                             }
188                         }.start();
189                         break;
190                     case STATE2:
```

Εικόνα 22: Switch – Case και πλαίσιο διαλόγου

Τον κώδικα ακολουθεί μια σειρά από Switch και Case. Σε κάθε έλεγχο που παρουσιάστηκε πιο πάνω αντιστοιχεί και ένα case το όποιο παρουσιάζει και ένα πλαίσιο διαλόγου ως παρατήρηση(Alert Dialog). Ένα τέτοιο παράδειγμα φαίνεται στην εικόνα 22 όπου εμφανίζεται πλαίσιο διαλόγου για να παρατηρήσει ότι δεν έχει επιλεγεί εικόνα κάλυψης. Αντίστοιχα πλαίσια διαλόγου ανοίγουν σε αντίστοιχους ελέγχους, όπως αν δεν έχει συμπληρωθεί κλειδί ή η εικόνα είναι πολύ μικρή για να φιλοξενήσει τα δεδομένα εισαγωγής.

Εν τέλει και αφού όλοι οι έλεγχοι έχουν ορθή κατάληξη ξεκινά η διαδικασία κωδικοποίησης με ένα πλαίσιο διαλόγου τύπου προόδου (Progress Dialog) να εμφανίζεται καθόλη τη διάρκεια της κωδικοποίησης. Ο κώδικας του πλαισίου αυτού φαίνεται στην εικόνα 23.

```
//////////////////////////////// Progress Dialog Created and Thread for Image Processing//////////////////

                final ProgressDialog ringProgressDialog = ProgressDialog.show(EncodeActivity.this,
                        "Please wait ...", "Encoding Key ...", true);

                ringProgressDialog.setCancelable(false);
                ringProgressDialog.setCanceledOnTouchOutside(false);

                new Thread(new Runnable() {
                    @Override

                    public void run() {
```

Εικόνα 23: Progress Dialog

Για κάθε πλαίσιο διαλόγου παρατήρησης όπως και για όλη τη διαδικασία της κωδικοποίησης, εκκινείται ένα νέο Νήμα(Thread();) το οποίο χρησιμοποιείται από το Android για να εκτελέσει λειτουργίες παράλληλες της λειτουργίας του κύριου Νήματος που εκτελεί την τρέχουσα δραστηριότητα της εφαρμογής (εδώ είναι η EncodeActivity). Τα νήματα είναι απαραίτητα στο Android για την καλύτερη διαχείριση των διεργασιών από το λειτουργικό σύστημα και την αποφυγή επιπλοκών κατά την εκτέλεση παράλληλων λειτουργιών αλλά και λειτουργιών χρονοβόρων ως προς το σύστημα[16] . Η εκκίνηση ενός νήματος φαίνεται τόσο στην εικόνα 22, όσο και στην εικόνα 23.

Η κωδικοποίηση επιτελείται από τη μέθοδο encode_key(); με ορίσματα, κατά σειρά από αριστερά προς τα δεξιά όπως φαίνεται στην εικόνα 24, τα pixels(integers) της εικόνας κάλυψης, τα δεδομένα εισαγωγής με τη μορφή ενός πίνακα από bytes, το πλάτος και το ύψος της εικόνας κάλυψης και τέλος τη θέση του pixel από το οποίο θα αρχίσει η εισαγωγή των bytes. Ως παράδειγμα δίνεται ο κώδικας της εισαγωγής των δεδομένων του flag(headerFlag) στην εικόνα 24.

```
304 ////////////////////////////////////////////////////////// Flag Embedded First  ////////////////////////////////////
305
306                                 int masterIndex;
307                                 Bitmap destBitmap;
308                                 oneDModTempFlag = encode_key(oneD, headerFlag, width, height, (width * height) - 33);
309
310
311                                 destBitmap = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);
312                                 destBitmap.setDensity(density);
313
314                                 masterIndex = 0;
315
316                                 for (int j = 0; j < height; j++) {
317                                     for (int i = 0; i < width; i++) {
318
319                                         destBitmap.setPixel(i, j, Color.argb(0xFF,
320                                                 oneDModTempFlag[masterIndex] >> 16 & 0xFF,
321                                                 oneDModTempFlag[masterIndex] >> 8 & 0xFF,
322                                                 oneDModTempFlag[masterIndex++] & 0xFF));
323
324
325                                     }
326
327                                 }
```

Εικόνα 24: Flag Encoding, κλήση της μεθόδου encode_key

Η μέθοδος encode_key αποτελεί μέρος της κλάσης Supportive_Methods και παρουσιάζεται αμέσως μετά.

- **Η μέθοδος encode_key();**

Όπως έχει ήδη αναφερθεί η μέθοδος αυτή δέχεται ως ορίσματα για την κωδικοποίηση, ένα μονοδιάστατο πίνακα από τιμές ακεραίων που αναπαριστούν τα pixels της εικόνας, ένα πίνακα bytes που περιέχει τα bytes της πληροφορίας που θέλουμε να εισάγουμε, δύο ακεραίους που αναπαριστούν το πλάτος και ύψος της εικόνας (δηλαδή τις γραμμές και τις στήλες των pixels όπως φαίνονται σε μία δισδιάστατη εικόνα) και έναν ακόμα ακέραιο που δηλώνει τη θέση του pixel από το οποίο θα ξεκινήσει η εισαγωγή των bytes της πληροφορίας που εισάγεται. Ο ορισμός αυτός συμβαίνει στην κλάση Supportive_Methods και φαίνεται στον παρακάτω κώδικα

```
62    public static int[] encode_key(int[] oneDPix, byte[] addition, int cols, int rows, int offset) {
63
64        final int blockSize = 3;
65        int[] binary = {16, 8, 0};
66        int[] toShift = {7, 6, 5, 4, 3, 2, 1, 0};
67        int shiftIndex = 8;
68        int element;
69        int[] oneDMod;
70        int k = 0;
71        int channels = 3;
72        byte[] result = new byte[rows * cols * channels];
73        int resultIndex = 0;
74        int addIndex = 0;
75
```

Εικόνα 25: Ορισμός της encode_key και των τοπικών σταθερών και μεταβλητών της

Επόμενο βήμα μέσα στην encode_key, ύστερα από τις δηλώσεις σταθερών και μεταβλητών, είναι η λειτουργία μιας αντιμετάθεσης (Shuffle) των στοιχείων του πίνακα ακεραίων (τιμές pixels) που εισάγονται με σκοπό την αποσυσχέτιση των θέσεων των pixels στα οποία θα εισαχθούν τα bit της πληροφορίας σε σχέση με την πραγματική (ορθή) θέση που αυτά καταλαμβάνουν πάνω στην εικόνα. Αυτό όπως γίνεται αντιληπτό αποσκοπεί στο να προδώσουμε επιπλέον ασφάλεια έτσι ώστε ακόμα κι αν υποπτευθεί η ύπαρξη πληροφορίας στην εικόνα να μην είναι δυνατό να μπορεί να διαβαστεί χωρίς τη γνώση του αλγορίθμου.

Για να δώσουμε ένα παράδειγμα για το πώς λειτουργεί το Shuffle επάνω στον πίνακα ακεραίων ας δούμε το παρακάτω παράδειγμα

Έστω ότι έχουμε τον παρακάτω πίνακα ακεραίων: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Αρχικά ο πίνακας χωρίζεται σε block. Ας υποθέσουμε εδώ ότι το block θα έχει μήκος 5. Η αντιμετάθεση γίνεται στα γειτονικά block ανά ζεύγη block. Έτσι έχουμε τον παρακάτω πίνακα: {6, 7, 8, 9, 10, 1, 2, 3, 4, 5}

Έπειτα γίνεται αντιμετάθεση ανά ζεύγη όλων των στοιχείων του πίνακα που προκύπτει. Δηλαδή το 6 με το 7, το 8 με το 9 κτλ. Έτσι ο τελικός πίνακας που θα σταλεί για κωδικοποίηση θα είναι: {7, 6, 9, 8, 1, 10, 3, 2, 5, 4}

Αφού γίνει η κωδικοποίηση των δεδομένων που θέλουμε να εισάγουμε, καλείται η αντίστροφη λειτουργία έτσι ώστε οι τιμές των pixels να επανέλθουν στην ορθή τους θέση για να επανασυσταθεί η εικόνα.

Η διαδικασία της αντιμετάθεσης ανά block και ανά ζευγάρι ακεραίων γίνεται από τις μεθόδους της κλάσης Supportive_Methods, shuffleBlock(); και shuffleInt(); αντίστοιχα. Η κλήση τους φαίνεται στην παρακάτω εικόνα.

```
77  ////////////////////////////////// Shuffle Block oneDPix ////////////////////////////////////////////////
78
79          int [] oneDPixShuffleBlock = Supportive_Methods.shuffleBlock(oneDPix, blockSize);
80
81  ////////////////////////////////// Shuffle Int oneDPix //////////////////////////////////////////////////////
82          int [] oneDPixShuffleInt = Supportive_Methods.shuffleInt(oneDPixShuffleBlock);
```

Εικόνα 26: Κλήση των μεθόδων shuffleBlock() και shuffleInt()

Στην εικόνα 27 φαίνεται η λειτουργία της κωδικοποίησης. Αυτή επιτελείται ως εξής: Αρχίζοντας από το πρώτο pixel της εικόνας μέχρι και το τελευταίο (πράσινο πλαίσιο) και εφόσον το offset που έχει δοθεί (η θέση από την οποία θα αρχίσει η κωδικοποίηση των byte) είναι 0 ή όχι (μπλε πλαίσιο) τότε για κάθε pixel, δηλαδή για κάθε ένα από τα τρία διαθέσιμα byte – ένα για κάθε «κανάλι χρώματος» RGB – κάθε ένα από τα bit των byte πληροφορίας που εισάγονται (κόκκινο πλαίσιο), κωδικοποιούνται σε κάθε LSB του εκάστοτε byte του εκάστοτε pixel.

Η κωδικοποίηση αυτή γίνεται με τη βοήθεια των πινάκων που ορίζονται ως μεταβλητές στην έναρξη της μεθόδου στην εικόνα 25 και μετά από μια σειρά από Shift(>>) έτσι ώστε να απομονωθεί τόσο το LSB της εικόνας όσο και το κάθε bit των bytes της πληροφορίας, τελικά να γίνει τροποποίηση του μέσω ενός λογικού OR (|) . Τέλος μετά από ένα cast να τροποποιηθεί σε ένα byte (byte) tpm. Όταν ολοκληρωθεί το χτίσιμο ολόκληρου του byte της εικόνας, αποθηκεύεται σε ένα πίνακα από bytes με όνομα result[] και ο αλγόριθμος πηγαίνει στο επόμενο byte πληροφορίας για εισαγωγή (καφέ πλαίσιο). Όταν όλα τα bit πληροφορίας εισηχθούν, τότε η εικόνα συμπληρώνεται με τα byte της ως είχαν.

Κάτι που πρέπει να παρατηρήσουμε είναι η περίπτωση που η τιμή του offset δεν είναι 0, τότε τα byte της εικόνας, μέχρι τη θέση που ορίζει το offset, μένουν ως είχαν και η εισαγωγή γίνεται από το σημείο του offset και μέχρι να ολοκληρωθεί η εισαγωγή των bit πληροφορίας. Αφού εισαχθούν όλα τα bit της πληροφορίας, τα byte της εικόνας συμπληρώνονται ως είχαν όπως φαίνεται στην εικόνα 27.

```
84          for (int i = 0; i < rows; i++) {
85              for (int j = 0; j < cols; j++) {
86                  element = i * cols + j;
87                  byte tmp;
88
89                  if (offset == 0) {
90                      for (int channelIndex = 0; channelIndex < channels; channelIndex++) {
91                          if (k < addition.length) {
92                              tmp = (byte) (((oneDPixShuffleInt[element] >> binary[channelIndex] & 0xFF) & 0xFE) | ((addition[addIndex]
                                >> toShift[(shiftIndex++) % toShift.length]) & 0x1));
93
94                              result[resultIndex++] = tmp;
95
96                              if (shiftIndex % toShift.length == 0) {
97                                  addIndex++;
98                                  k = k + 1;
99                              }
100                         } else {
101                             tmp = (byte) ((((oneDPixShuffleInt[element] >> binary[channelIndex]) & 0xFF)));
102                             result[resultIndex++] = tmp;
103                         }
104                     }
105                 } else {
106
107                     for (int channelIndex = 0; channelIndex < channels; channelIndex++) {
108
109                         if (element >= offset) {
110                             if (k < addition.length) {
111                                 tmp = (byte) (((oneDPixShuffleInt[element] >> binary[channelIndex] & 0xFF) & 0xFE) |
                                    ((addition[addIndex] >> toShift[(shiftIndex++) % toShift.length]) & 0x1));
112
113                                 result[resultIndex++] = tmp;
114
115                                 if (shiftIndex % toShift.length == 0) {
116                                     addIndex++;
117                                     k = k + 1;
118                                 }
119                             } else {
120                                 tmp = (byte) ((((oneDPixShuffleInt[element] >> binary[channelIndex]) & 0xFF)));
121
122                                 result[resultIndex++] = tmp;
123                             }
124                         } else {
125                             tmp = (byte) ((((oneDPixShuffleInt[element] >> binary[channelIndex]) & 0xFF)));
126                             result[resultIndex++] = tmp;
127                         }
128                     }
```

Εικόνα 27: Η Λειτουργία της Κωδικοποίησης

Εν τέλει εξάγεται ένας πίνακας από τα τροποποιημένα bytes της εικόνας κωδικοποιημένα με τα bit της πληροφορίας που εισάγεται.

Για να μπορέσουμε να επεξεργαστούμε στη συνέχεια την εικόνα θα πρέπει να μετατρέψουμε τα byte αυτά σε pixels και στις τιμές των ακεραίων που τα αναπαριστούν όπως και πριν. Αυτό γίνεται με τη βοήθεια της συνάρτησης byteArrayToIntArray();. Τη μετατροπή αυτή ακολουθεί η αντίστροφη λειτουργία της αντιμετάθεσης που περιγράφηκε προηγούμενα. Πρώτα γίνεται αντιμετάθεση κατά ζεύγη ακεραίων και έπειτα κατά block για να επιστρέψουμε στην ορθή σειρά των pixels της εικόνας κάλυψης (εικόνα 28). Τέλος ο πίνακας ακεραίων επιστρέφεται από τη μέθοδο encode_key για χρησιμοποιηθεί από άλλες κλήσεις της για τα άλλα μέρη πληροφορίας που θέλουμε να εισάγουμε αλλάζοντας όμως το offset κάθε φορά.

```
132         oneDMod = byteArrayToIntArray(result);
133
134 ////////////////////////////////////////  UnShuffle to build the image back ////////////////////////
135
136         int[] oneDModIntUnShuffled;
137         oneDModIntUnShuffled = Supportive_Methods.UnShuffleInt(oneDMod);
138         int [] oneDModBlockUnSuffled;
139         oneDModBlockUnSuffled = Supportive_Methods.UnShuffleBlock(oneDModIntUnShuffled, blockSize);
140
141         return oneDModBlockUnSuffled;
142     }
```

Εικόνα 28: Μετατροπή bytes σε integers και Unshuffled

Μετά την επιστροφή της μεθόδου αυτής ο πίνακας των τιμών των pixels θέτει και ορίζει ξανά τα pixels του bitmap της εικόνας μέσω της μεθόδου setPixels(); (εικόνα 24). Το τροποποιημένο πλέον bitmap είναι έτοιμο να χρησιμοποιηθεί και στις επόμενες κλήσεις της encode_key που θα γίνουν με τον ίδιο τρόπο για κάθε ένα από τα μέρη, δηλαδή, του κλειδιού, του flag, του μήκους του κλειδιού, του CRC32 και του μήκους του CRC32. Πέρα από τα ίδια τα δεδομένα που εισάγονται κάθε φορά αλλάζει και η θέση από την οποία θα εισαχθούν(offset). Οι διαφορετικές κλήσεις με τα διαφορετικά offset φαίνονται στην εικόνα 29.

```
oneDModTempFlag = encode_key(oneD, headerFlag, width, height, (width * height) - 33);

oneDModTempWidth = encode_key(oneD, byteKeyLength, width, height, (width * height) - 22);

oneDModTempWidth = encode_key(oneD, crcNum, width, height, (width * height) - 11);

oneDModTempCRC = encode_key(oneD, checksumBytes, width, height, (width * height) - (33 +
crcOffsetFinal));

oneDModFinal = encode_key(oneD, Key, width, height, 0);
```

Εικόνα 29: Οι διαφορετικές κλήσεις της encode_key();

Μετά και την επιστροφή της τελευταίας κλήσης για την κωδικοποίηση των δεδομένων του κλειδιού της εικόνας 29, και τη δημιουργία του τελικού και τροποποιημένου bitmap της εικόνας, απομένει η αποθήκευσή της ως αρχείο και ως μιας κωδικοποιημένης πλέον εικόνας. Αυτό επιτελείται με την κλήση της SaveImage(); της εικόνας 30.

Το format του αρχείου της εικόνας θα είναι .PNG για το λόγο ότι είναι ένα lossless format, δηλαδή ένα format που δεν θα έχει υποστεί συμπίεση και δεν θα έχει χάσει καθόλου πληροφορία. Έτσι είμαστε σίγουροι ότι οι τιμές των pixels του στεγανογραφημένου bitmap που μόλις δημιουργήθηκε θα παραμείνουν αναλλοίωτες και άρα και τα δεδομένα που έχουμε εισάγει μέσα σε αυτά.

```
imagePath = Supportive_Methods.SaveImage(destBitmap);
Log.v("Image Saved", "New Image Saved");
```

Εικόνα 30: Η κλήση SaveImage();

Η διαδικασία ολοκληρώνεται με ένα μήνυμα διαλόγου που ενημερώνει για την επιτυχή αποθήκευση του αρχείου σε μία δεδομένη θέση με ένα ξεχωριστό όνομα όπως φαίνεται στη εικόνα 31.

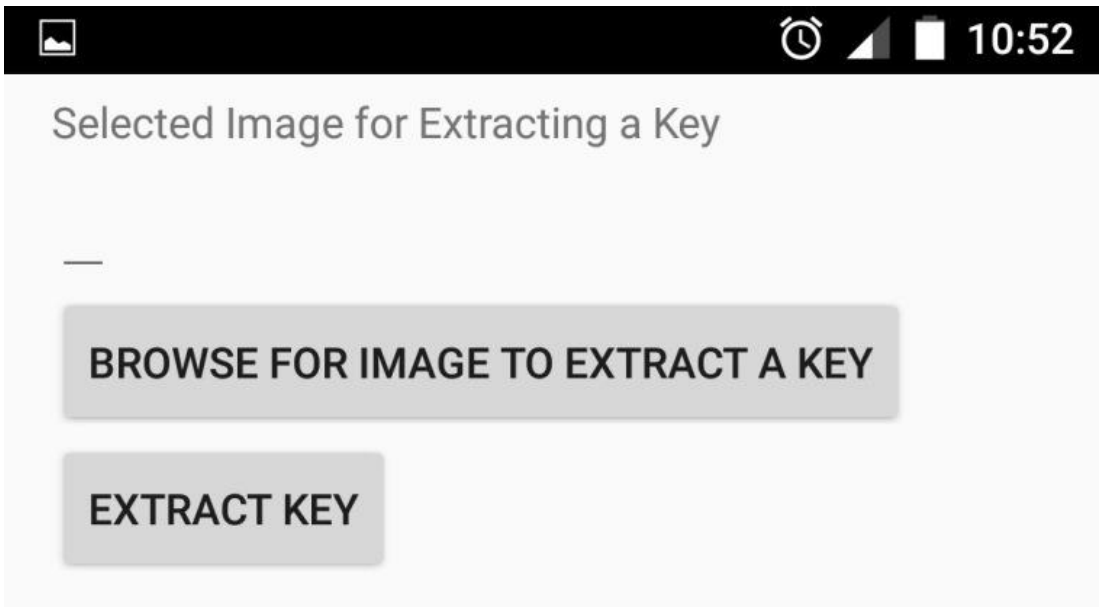
```
////////////////////////////// Creating Alert for Saving Encoded Image ////////////////////////////////////////////
                final AlertDialog.Builder saveImageDialog = new
                AlertDialog.Builder(EncodeActivity.this);

                saveImageDialog.setMessage("Encoded Image has been saved under: /saved_images/" +
                imagePath)
                        .setCancelable(false)
                        .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                            public void onClick(DialogInterface dialog, int id) {

                                dialog.dismiss();
                            }
                        });
                saveImageDialog.setCancelable(false);

                saveImageDialog.show();
```

Εικόνα 31: Το πλαίσιο διαλόγου για την αποθήκευση της εικόνας

- **Δραστηριότητα Αποκωδικοποίησης – αρχείο Decode.java**

Στο αρχείο αυτό ορίζεται η λειτουργία της αποκωδικοποίησης μιας κωδικοποιημένης (Στεγανογραφημένης) εικόνας.

Η δραστηριότητα αυτή προβάλλεται στην οθόνη με το πάτημα του κουμπιού Decode της κύριας δραστηριότητας Main_InterfaceActivity.java. Το γραφικό περιβάλλον που εμφανίζεται φαίνεται στην εικόνα 32.



Εικόνα 32: Γραφικό Περιβάλλον Decode.java

Η λειτουργία του κουμπιού BROWSE FOR IMAGE TO EXTRACT A KEY ελέγχεται από κώδικα παρόμοιο με αυτόν της εικόνας 14 αλλά και 15 με ένα Listener και μία μέθοδο onClick(); αφοσιωμένη στο κουμπί για την περίπτωση που πατηθεί και την μέθοδο onActivityResult(); στην οποία επιλέγεται μέσα από το άνοιγμα της εφαρμογής προβολής εικόνων η εικόνα προς αποκωδικοποίηση. Μαζί επιστρέφεται η θέση και το όνομά της στο πεδίο κειμένου που βρίσκεται πάνω από το κουμπί αυτό.

Η κύρια λειτουργία της δραστηριότητας αυτής επιτελείται από το κουμπί EXTRACT KEY. Αρχικά και πάντα μετά τον ορισμό του κουμπιού, με μια μέθοδο onClick(); ξεκινά η διαδικασία για την εξαγωγή του κλειδιού. Στη μέθοδο αυτή γίνεται πρώτα έλεγχος για το αν έχει επιλεγεί εικόνα προς αποκωδικοποίηση. Αν όχι (if (width==0)) τότε ένα πλαίσιο διαλόγου εμφανίζεται προτρέποντας για την επιλογή μιας εικόνας δημιουργώντας, όπως και για προηγούμενα πλαίσια διαλόγου, ένα νέο Νήμα. Ο κώδικας της λειτουργίας αυτής φαίνεται στην εικόνα 33.

```
59          Button btnExtractImage = (Button) findViewById(R.id.btnExtractAndSaveImage);
60          btnExtractImage.setOnClickListener(new Button.OnClickListener() {
61
62              public void onClick(View v) {
63
64                  try {
65                      width = bmpExtraction.getWidth();
66                      height = bmpExtraction.getHeight();
67                      density = bmpExtraction.getDensity();
68                      extD = new int[width * height];
69                  }catch (Exception e){
70                   // No Nothing
71                  }
72  ////////////////////// Set up Alert Dialog and Create new Thread ////////////////////////////////
73              if (width == 0){
74                  new Thread() {
75                      public void run() {
76                          DecodeActivity.this.runOnUiThread(new Runnable() {
77                              public void run() {
78
79                                  try {
80
81                                      final AlertDialog.Builder extractAlertDialog = new AlertDialog.Builder(DecodeActivity.this);
82
83                                      extractAlertDialog.setMessage("Please choose an image to extract a key...")
84                                              .setCancelable(false)
85                                              .setPositiveButton("OK", new DialogInterface.OnClickListener() {
86                                                  public void onClick(DialogInterface dialog, int id) {
87
88                                                      dialog.dismiss();
89                                                  }
90                                              });
91                                      extractAlertDialog.setCancelable(false);
92
93                                      extractAlertDialog.show();
```

Εικόνα 33: Έναρξη του κουμπιού Extract Key και έλεγχος για την επιλογή εικόνας

Έπειτα ακολουθεί η εξαγωγή των pixels της επιλεγμένης εικόνας σε ένα πίνακα ακεραίων όπως είχε συμβεί και κατά την κωδικοποίηση. Στη συνέχεια είναι απαραίτητη η αντιμετάθεση των στοιχείων του πίνακα όπως και στην διαδικασία της κωδικοποίησης με τη βοήθεια των μεθόδων shuffleBlock(); και shuffleInt();. Πριν την κλήση της μεθόδου αποκωδικοποίησης γίνεται μετατροπή των ακεραίων(pixels) σε bytes (Μια τιμή ακεραίου μετατρέπεται σε 3 bytes(RGB)) και αυτό γιατί η μέθοδος decode_key(); κατά την κλήση της παίρνει ως ορίσματα τα bytes της εικόνας, το πλάτος και το ύψος της σε pixels. Όλα τα παραπάνω φαίνονται στον κώδικα της εικόνας 34.

```
bmpExtraction.getPixels(extD, 0, width, 0, 0, width, height);

//  Shuffle Block and Int(Pixels) to read data(byte) ////////////////////////////

int [] PixShuffleBlock = Supportive_Methods.shuffleBlock(extD, blockSize);
int [] PixShuffleInt = Supportive_Methods.shuffleInt(PixShuffleBlock);

byteArrayExtraction = convertArray(PixShuffleInt);

extractedString = decode_key(byteArrayExtraction, width, height);
```

Εικόνα 34: Αντιμετάθεση των pixels και μετατροπή τους σε bytes πριν την κλήση της decode_key();

- **Η μέθοδος decode_key();**

Η μέθοδος αυτή επιστρέφει ένα String (το κλειδί που θα έχει αποκωδικοποιηθεί) και δέχεται ως ορίσματα ένα πίνακα από bytes (οι τιμές των RGB σε bytes), και δύο ακεραίους (το πλάτος και το ύψος της εικόνας που έχει επιλεγεί για αποκωδικοποίηση). Η εικόνα 35 δείχνει τον ορισμό της μεθόδου καθώς και τις μεταβλητές και σταθερές που δηλώνονται αρχικά μέσα σε αυτήν.

```
145     public static String decode_key(byte[] Pix, int width, int height) {
146
147
148         int k = 0;
149         byte[] andByte = {(byte) 0x80, 0x40, 0x20, 0x10, 0x8, 0x4, 0x2, 0x1};
150         int[] toShift = {7, 6, 5, 4, 3, 2, 1, 0};
151         int shiftIndex = 8;
152         byte[] offsetExtr = new byte[4];
153         byte[] flagExtr = new byte[4];
154         byte[] crcOffset = new byte [4];
155         String flagExtrStr;
156         String CRC32Value;
157         int limit = 0;
158         int j = 0;
159         byte temp = 0x0;
```

Εικόνα 35: Ορισμός της decode_key(); και των σταθερών και μεταβλητών της μέσα από στην Supportive_Methods.java

Η αποκωδικοποίηση των byte γίνεται παρόμοια με την κωδικοποίηση με τη βοήθεια των πινάκων που δηλώνονται στην εικόνα 35 andByte[] και toShift[] και από μία σειρά από Shift (<<) και λογικά AND(&) που εκτελούνται επί των byte της εικόνας προκειμένου να απομονωθούν ανά bit (LSBs) και να αποθηκευτούν σε bytes τα

δεδομένα που αφορούν το κλειδί που έχει κωδικοποιηθεί. Αρχικά γίνεται απόπειρα να εξαχθεί η μεταβλητή-δείκτης, το flag που θα πρέπει να υπάρχει στο προκαθορισμένο σημείο αν αυτή η εικόνα είναι Στεγανογραφημένη. Αν δεν βρεθεί τότε σημαίνει πως η εικόνα δεν έχει κωδικοποιηθεί και η λειτουργία της αποκωδικοποίησης δεν θα συνεχίσει και το String που επιστρέφεται είναι το κενό (null). Τα παραπάνω, καθώς και ο πρώτος έλεγχος του flag, προκειμένου να συνεχιστεί η λειτουργία της αποκωδικοποίησης, φαίνεται στην εικόνα 36.

```
161         for (int n = (width * height * 3) - 99; n < (width * height * 3) - 66; n++) {
162             temp = (byte) (temp | ((Pix[n] << toShift[shiftIndex % toShift.length]) & andByte[shiftIndex++ % toShift.length]));
163
164             if (shiftIndex % toShift.length == 0) {
165                 flagExtr[j] = temp;
166                 j = j + 1;
167                 temp = 0x0;
168             }
169         }
170         flagExtrStr = new String(flagExtr);
171         Log.v("toString", "toString = " + flagExtrStr);
172         temp = 0x0;
173
174         if (flagExtrStr.equals(flag)) {
175             j = 0;
176             shiftIndex = 8;
```

Εικόνα 36: Αποκωδικοποίηση και έλεγχος του flag από το προκαθορισμένο σημείο που θα πρέπει να είχε κωδικοποιηθεί.

Στην περίπτωση που το flag επαληθευτεί, τότε η λειτουργία συνεχίζει με την εξαγωγή του μήκους των byte του κλειδιού το οποίο θα εξυπηρετήσει τον αλγόριθμο στο να γνωρίζει στα πόσα byte θα πρέπει «διαβάσει» για να έχει ολοκληρωμένο το κλειδί. Ο κώδικας φαίνεται στη εικόνα 37.

```
178         for (int n = (width * height * 3) - 66; n < (width * height* 3)-33; ) {
179
180             temp = (byte) (temp | ((Pix[n] << toShift[shiftIndex % toShift.length]) & andByte[shiftIndex++ % toShift.length]));
181
182             n = n + 1;
183             if (shiftIndex % toShift.length == 0) {
184                 offsetExtr[j] = temp;
185                 j = j + 1;
186                 temp = 0x0;
187             }
188         }
189         int offsetExtracted = byteArrayToInt(offsetExtr);
190         byte[] extractedFinal = new byte[offsetExtracted];
```

Εικόνα 37: Εξαγωγή του μήκους του κλειδιού (offsetExtr)

Στον κώδικα ακολουθεί η εξαγωγή toy μήκους των bytes του CRC32(εικόνα 38)

```
196         for (int n = (width * height * 3) - 33; n < width * height* 3; n++) {
197             Log.v("for check ", "for check");
198             temp = (byte) (temp | ((Pix[n] << toShift[shiftIndex % toShift.length]) & andByte[shiftIndex++ % toShift.length]));
199
200             if (shiftIndex % toShift.length == 0) {
201                 crcOffset[j] = temp;
202                 j = j + 1;
203                 temp = 0x0;
204             }
205         }
206         int off = byteArrayToInt(crcOffset);
207         int offFinal = off*8;
```

Εικόνα 38: Εξαγωγή του offset CRC32

Και μετά ακολουθεί η εξαγωγή των byte του ίδιου του CRC32 με βάση το μήκος των byte(πλήθος bit που θα διαβαστούν) του CRC32(εικόνα 39) που εξήχθησαν προηγούμενα.

```
224         for (int n = ((width * height * 3) - (99 + offFinal)); n < (width * height * 3) - 99; n++) {
225
226             temp = (byte) (temp | ((Pix[n] << toShift[shiftIndex % toShift.length]) & andByte[shiftIndex++ % toShift.length]));
227
228             if (shiftIndex % toShift.length == 0) {
229
230                 crcExtr[j] = temp;
231                 j = j + 1;
232                 z = z + 1;
233                 temp = 0x0;
234             }
235         }
```

Εικόνα 39: Εξαγωγή του CRC32

Κατά σειρά έχουμε την εξαγωγή του κλειδιού με τη βοήθεια του offset του κλειδιού - που εξήχθη αμέσως μετά την επαλήθευση του flag - και την εύρεση του CRC32 για τα byte του κλειδιού που μόλις εξήχθησαν. Επίσης γίνεται και η μετατροπή των byte του κλειδιού σε αλφαριθμητικό(String) όπως φαίνεται στην εικόνα 40.

```
240             for (int i = 0; i < (offsetExtracted * 8); ) {
241                 limit = limit + 1;
242
243                 temp = (byte) (temp | ((Pix[i] << toShift[shiftIndex % toShift.length]) & andByte[shiftIndex++ % toShift.length]));
244
245                 i = i + 1;
246                 if ((shiftIndex % toShift.length == 0)) {
247                     extractedFinal[j] = temp;
248                     j = j + 1;
249                     k = k + 1;
250                     temp = 0x0;
251                 }
252             }
253         Checksum checksum = new CRC32();
254         checksum.update(extractedFinal, 0, extractedFinal.length);
255         long checksumValue = checksum.getValue();
256         String checksumString = Long.toString(checksumValue);
257
258         String extractedCRC = new String(crcExtr);
259         String keyString = new String(extractedFinal);
```

Εικόνα 40: Η εξαγωγή και μετατροπή του κλειδιού σε String

Στην περίπτωση που το CRC32 που εξήχθη μέσα από την κωδικοποιημένη εικόνα είναι ίδιο με το CRC32 που μόλις υπολογίστηκε από το εξαγόμενο κλειδί τότε η μεταβλητή keyString κρατά τη προηγούμενη τιμή της και την επιστρέφει προκειμένου να αποθηκευτεί σε ένα αρχείο. Σε αντίθετη περίπτωση παίρνει την τιμή "FALSE" πράγμα το οποίο θα πυροδοτήσει ένα μήνυμα σφάλματος CRC του κλειδιού στην έξοδο. Επίσης όπως προηγούμενα αναφέρθηκε, αν δεν βρεθεί εξαρχής η τιμή του flag κωδικοποιημένη, τότε τίποτα από τα παραπάνω δεν συμβαίνει και επιστρέφεται null, πράγμα που θα πυροδοτήσει ένα άλλο μήνυμα που θα ενημερώνει για τη μη ύπαρξη κωδικοποιημένης εικόνας. Ο κώδικας της επιστροφής του String κατά συνθήκη φαίνεται στη εικόνα 41.

```
264             if (extractedCRC.equals(checksumString)){
265                 // Do Nothing
266             }else{
267                 CRC32Value = "FALSE";
268                 keyString = CRC32Value;
269             }
270             return keyString;
271         } else {
272             Log.v("No Vessel Image", "No Secret Key encoded");
273             return null;
274         }
275     }
```

Εικόνα 41: Επιστροφή εξαγόμενου String κατά συνθήκη

Έξω από τη μέθοδο decode_key(); και μετά την επιστροφή του αποτελέσματος της, μια σειρά ελέγχων ακολουθεί. Αν το String που επιστρέφεται είναι null ένα μήνυμα που ενημερώνει για τη μη ύπαρξη κλειδιού εμφανίζεται και δεν έχουμε τη δημιουργία κανενός αρχείου με κάποιο εξαγόμενο κλειδί (εικόνα 42).

```
if (extractedString == null) {
    new Thread() {
        public void run() {
            DecodeActivity.this.runOnUiThread(new Runnable() {
                public void run() {

                    try {

                        final AlertDialog.Builder nullKeyDialog = new
                        AlertDialog.Builder(DecodeActivity.this);

                        nullKeyDialog.setMessage("No encoded image...")
                                .setCancelable(false)
                                .setPositiveButton("OK", new DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int id) {

                                        dialog.dismiss();
                                    }
                                });
                        nullKeyDialog.setCancelable(false);
                        nullKeyDialog.show();

                    } catch (Exception e) {
                        // Do Nothing
                    }

                }
            });
        }
    }.start();
} else {
```

Εικόνα 42: Έλεγχος για μη ύπαρξη κλειδιού

Στην περίπτωση που έχει επιστραφεί “FALSE” και το CRC δεν επαληθεύτηκε, τότε ένα άλλο μήνυμα που ενημερώνει για τη μη ορθότητα του CRC εμφανίζεται και δεν αποθηκεύεται κανένα κλειδί και πάλι (εικόνα 43).

```
if (extractedString.equals("FALSE")) {

extractedString = null;

    new Thread() {
        public void run() {
            DecodeActivity.this.runOnUiThread(new Runnable() {
                public void run() {

                    try {

                        final AlertDialog.Builder saveKeyDialog = new
                        AlertDialog.Builder(DecodeActivity.this);

                        saveKeyDialog.setMessage("CRC Check Found Errors on the Extracted
                        Key, Stego Image has been damaged...")
                                .setCancelable(false)
                                .setPositiveButton("OK", new
                                DialogInterface.OnClickListener() {
                                    public void onClick(DialogInterface dialog, int id) {

                                        dialog.dismiss();
                                    }
                                });
                        saveKeyDialog.setCancelable(false);
                        saveKeyDialog.show();

                    } catch (Exception e) {
                        // Do Nothing
                    }

                }
            });
        }
    }.start();
```

Εικόνα 43: Πλαίσιο διαλόγου για τη μη ορθή επαλήθευση του CRC32

Τέλος στην περίπτωση που δεν ισχύει τίποτα από τα δύο σφάλματα, τότε το κλειδί που θα έχει εξαχθεί και επαληθευτεί ορθά αποθηκεύεται με τη μορφή αρχείου .txt σε προκαθορισμένη θέση στην κινητή συσκευή με τη βοήθεια της μεθόδου SaveKey();(εικόνα 44).

```
keyPath = SaveKey(extractedString);
```

Εικόνα 44: Η κλήση της μεθόδου SaveKey();

Την αποθήκευση του κλειδιού ακολουθεί ένα μήνυμα διαλόγου που ενημερώνει για την θέση στην οποία αποθηκεύθηκε και το όνομα του αρχείου του κλειδιού. Ο κώδικας του πλαισίου διαλόγου φαίνεται στην εικόνα 45.

```
//////////// Creating New Thread for save key alert dialog /////////////////
                    new Thread() {
                        public void run() {
                            DecodeActivity.this.runOnUiThread(new Runnable() {
                                public void run() {

                                    try {

                                        final AlertDialog.Builder saveKeyDialog = new
                                        AlertDialog.Builder(DecodeActivity.this);

                                        saveKeyDialog.setMessage("Extracted key has been saved under:
                                        /saved_keys/" + keyPath)
                                                .setCancelable(false)
                                                .setPositiveButton("OK", new
                                                DialogInterface.OnClickListener() {
                                                    public void onClick(DialogInterface dialog, int id) {

                                                        dialog.dismiss();
                                                    }
                                                });
                                        saveKeyDialog.setCancelable(false);
                                        saveKeyDialog.show();

                                    } catch (Exception e) {
                                        // Do Nothing
                                    }
                                }
                            });
                        }
                    }.start();
```

Εικόνα 45: Το πλαίσιο διαλόγου αποθήκευσης του κλειδιού

Στο Παράρτημα Β δίνεται μια πιο ολοκληρωμένη εικόνα της λειτουργίας της εφαρμογής με ένα παράδειγμα κωδικοποίησης και αποκωδικοποίησης ενός ιδιωτικού κλειδιού με τη χρήση της εφαρμογής.

6.3 Προτάσεις Βελτίωσης του Συστήματος

- **Χρήση Κλειδιού για την Κωδικοποίηση και Αποκωδικοποίηση**

Όπως έχει προαναφερθεί, η ασφάλεια του Στεγανογραφικού συστήματος μπορεί να ενισχυθεί με τη χρήση ενός κλειδιού το οποίο θα ορίζεται από το χρήστη κατά την κωδικοποίηση και χωρίς το οποίο δεν θα είναι δυνατή η αποκωδικοποίηση, δηλαδή απλώς και μόνο με τη χρήση της εφαρμογής. Θα είναι δηλαδή απαραίτητη μια μυστική πληροφορία (ένα password) η οποία θα είναι γνωστή μόνο μεταξύ των εξουσιοδοτημένων μερών, που θα μπορεί να διαμοιραστεί ασύγχρονα, μέσω π.χ. ενός e-mail ή ακόμα και ενός τηλεφωνήματος, αφού μπορεί να είναι απλώς μια λέξη με ή χωρίς νόημα.

Μια πρόταση για την εφαρμογή του κλειδιού στο σύστημα που περιγράφηκε ήδη σε αυτήν την εργασία είναι η εξής:

Αρχικά θα δίνεται από το χρήστη ένα password κατά την διαδικασία της κωδικοποίησης και το οποίο θα προαπαιτείται προκειμένου να ξεκινήσει η λειτουργία εισαγωγής των δεδομένων του ιδιωτικού κλειδιού μέσα στην εικόνα με τη χρήση του γνωστού αλγορίθμου LSB όπως παρουσιάστηκε στα πλαίσια της εφαρμογής. Η δια φορά στη λειτουργία γίνεται με τη χρήση του password. Αρχικά οι χαρακτήρες του password θα μετατρέπονται σε ένα πίνακα από bytes τα οποία γίνονται XOR μεταξύ τους ξεκινώντας από τα αριστερά του πίνακα (πρώτη θέση) προς τα δεξιά. Με τον τρόπο αυτό θα προκύπτει ένα μόνο byte(8 bit) το οποίο τελικά θα μετατρέπονται στην αριθμητική τιμή ενός ακεραίου (0 - 255).

Ο ακέραιος αυτός θα χρησιμεύει ως δείκτης θέσης σε ένα προκαθορισμένο πίνακα ακεραίων από τον οποίο θα προκύπτει το μέγεθος του block στην πρώτη φάση της αντιμετάθεσης, η οποία περιγράφηκε νωρίτερα στην εργασία. Έτσι το μέγεθος του block θα επιλέγεται «τυχαία» από το δείκτη που σχηματίστηκε από το password. Ένας πίνακας από τιμές block θα μπορεί να έχει το πολύ 256 τιμές μιας και ο δείκτης θα μπορεί να πάρει μόνο αυτές τις τιμές(2^8). Οι τιμές του 0 και 1 απορρίπτονται ως τιμές του πίνακα επειδή χρειαζόμαστε να έχουμε blocks που να περιέχουν τουλάχιστον 2 pixels. Μπορούμε για παράδειγμα να έχουμε τον παρακάτω πίνακα με 99 τιμές επιλογής

[2,3,4…,99, 100]

Για τιμές δείκτη από 0-99 η μεταβλητή δείκτης ακεραίου που προκύπτει από το password θα γίνεται αποδεκτή ως έχει για την επιλογή της θέσης του εκάστοτε στοιχείου του πίνακα.

Για τιμές που θα προκύπτουν μεταξύ 100-255 ή τιμή αυτή θα διαιρείται με το 3 προκειμένου να παραμείνει ο δείκτης του πίνακα εντός αποδεκτών ορίων (0-99).

Έτσι μπορεί να επιλέγεται τόσο για την κωδικοποίηση, όσο και για την αποκωδικοποίηση, ψευδοτυχαία το μέγεθος του block που αντιμετατίθεται.

- **Χρήση Εναλλακτικού Αλγορίθμου DCT στο Πεδίο Συχνοτήτων (Frequency Domain)**

Ένας εναλλακτικός αλγόριθμος που μπορεί να αντικαταστήσει τον αλγόριθμό LSB και την μέθοδο που λειτουργεί στο πεδίο του χώρου (Spatial Domain) είναι ο αλγόριθμος DCT. Όπως έχει προαναφερθεί, η χρήση του αλγορίθμου αυτού είναι πιο ασφαλής μέθοδος όσον αφορά την **ανθεκτικότητα** απέναντι στην **αντίληψη** από τρίτους της Στεγανογραφημένης εικόνας τόσο με γυμνό μάτι, όσο και σε μεθόδους στεγανάλυσης (η διαδικασία για την ανακάλυψη χρήσης Στεγανογραφίας και αποκάλυψης των κρυφών δεδομένων) με στατιστικές μεθόδους. Επίσης είναι μια ποιο ανθεκτική μέθοδος όσον αφορά την **ανάγνωση** των δεδομένων καθώς αυτά δεν κωδικοποιούνται απευθείας στα pixels της εικόνας στο πεδίο του δυσδιάστατου χώρου, αλλά στο πεδίο των συχνοτήτων και επί των τιμών αυτών τις εικόνας. Πρόκειται λοιπόν για μια πιο πολύπλοκη διαδικασία. Τέλος ένα ακόμα πλεονέκτημα σε σχέση με τη μέθοδο του LSB είναι η ανθεκτικότητά του σχετικά με την δυνατότητα ανάκτησης των στεγανογραφημένων δεδομένων απέναντι σε **αλλοιώσεις** που μπορεί να υποστεί η εικόνα (π.χ. Συμπίεση)[17].

Παρόλα τα θετικά δεν θα μπορούσε να μην υπάρχει και κάτι **αρνητικό** έναντι στη μέθοδο του LSB. Αυτό είναι η **περιορισμένη χωρητικότητα** που αφήνει στην εικόνα σε σχέση με τη μέθοδο LSB[17].

Η μέθοδος του DCT χρησιμοποιείται και από τη πολύ γνωστή συμπίεση εικόνας το format JPEG. Η Στεγανογράφηση στηρίζεται στους «συντελεστές» συχνοτήτων, όπως

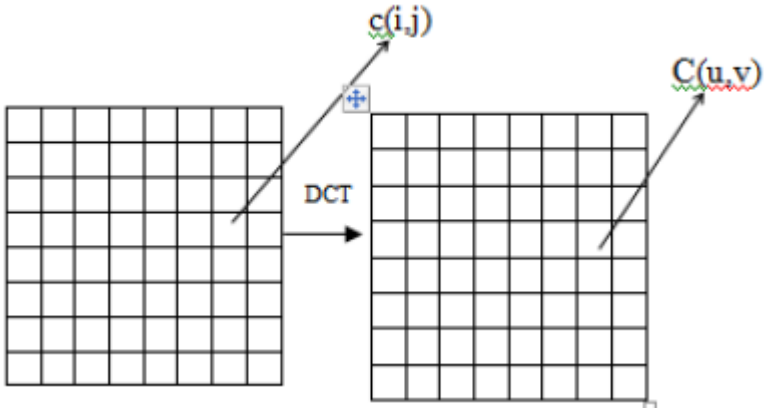
ονομάζονται, που δεν είναι άλλο παρά τιμές που εξάγονται από το κάθε pixel της εικόνας.

Για μία δισδιάστατη εικόνα N * M οι τιμές αυτές προκύπτουν από τον παρακάτω τύπο:

$$C(u,v) = a(v)\sum_{i=0}^{N-1}[\,a(u)\sum_{i=0}^{N-1} x_i \cos\left(\frac{(2i+1)u\pi}{2N}\right)] \times \cos\left(\frac{(2i+1)v\pi}{2N}\right)$$

Όπου u, v = 0, 1, 2….N-1

Για μία τέτοια εικόνα κάθε τιμή pixel(ακέραιος) μετατρέπεται σε μία τιμή DCT που δηλώνει τη μετατροπή της εικόνας από το πεδίο του χώρου, στο πεδίο της συχνότητας (εικόνα 46)[17].



Εικόνα 46: Μετατροπή DCT μιας εικόνας[18]

Έτσι η εικόνα μπορεί να χωριστεί σε τιμές υψηλών, μεσαίων και χαμηλών συχνοτήτων. Οι τιμές που έχουν τη μεγαλύτερη σημασία και αναπαριστούν τα κύρια οπτικά μέρη της εικόνας (χαμηλότερες συχνότητες) μένουν ως έχουν, ενώ οι τιμές με τις υψηλότερες συχνότητες θα αφαιρεθούν κατά τη διαδικασία συμπίεσης που θα υποστεί η εικόνα καθώς είναι αυτές με τη λιγότερη σημασία και οι οποίες δεν γίνονται αντιληπτές από το ανθρώπινο μάτι. Τα δεδομένα θα εισαχθούν στις τιμές με τις μεσαίες συχνότητες έτσι ώστε να μην υπάρξει σημαντική οπτική αλλοίωση της εικόνας[17].

Κατά την έναρξη της διαδικασίας για την εξαγωγή των DCT τιμών η εικόνα χωρίζεται σε pixel blocks των 8*8 = 64 pixels το καθένα. Για κάθε ένα από τα pixels του κάθε block εκτελείται η εξίσωση και εξαγωγή των αντίστοιχων τιμών που αποθηκεύονται σε ένα δισδιάστατο πίνακα τιμών όπως φαίνεται στο παράδειγμα της εικόνας 47

$$F = \begin{bmatrix} 162 & 40 & 20 & 72 & 30 & 2 & -1 & -1 \\ 30 & 108 & 10 & 32 & 27 & 5 & 8 & -2 \\ -94 & -60 & 12 & -43 & -31 & 6 & -3 & 7 \\ -38 & -83 & -5 & -22 & 3 & 5 & -1 & 3 \\ -31 & 17 & -5 & -1 & 4 & -6 & 1 & -6 \\ 0 & -1 & 2 & 0 & 2 & 2 & 8 & 2 \\ 4 & -2 & 2 & 6 & 8 & -1 & 7 & 2 \\ -1 & 1 & 7 & 6 & 2 & 0 & 5 & 0 \end{bmatrix}$$

Εικόνα 47: Πίνακας τιμών συντελεστών DCT[19]

Οι τιμές που βρίσκονται στην πάνω αριστερή γωνία του πίνακα είναι οι συντελεστές χαμηλότερης συχνότητας, ενώ οι τιμές στην κάτω δεξιά γωνία είναι αυτές με την ψηλότερη συχνότητα.

Κάθε τέτοιο block θα υποστεί συμπίεση μέσω κβαντοποίησης(quantization). Αυτό γίνεται με τη διαίρεση κάθε μιας τιμής του πίνακα DCT με την αντίστοιχη τιμή ενός πίνακα κβαντοποίησης (εικόνα 48) και το αποτέλεσμα στρογγυλοποιείται στον κοντινότερο ακέραιο. Ο βασικός πίνακας κβαντοποίησης φαίνεται στην εικόνα 48 και μπορεί να τροποποιηθεί προκειμένου να επιτευχθεί διαφορετική ποιότητα εικόνας[18].

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

Εικόνα 48: Βασικός πίνακας κβαντοποίησης[19]

Οι υψηλότερες συχνότητες που δεν είναι εύκολα αντιληπτές από το ανθρώπινο μάτι όπως φαίνεται στην εικόνα 49 στρογγυλοποιούνται στην τιμή 0 στον πίνακα που προκύπτει.

$$P = \begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Εικόνα 49: Πίνακας DCT που έχει υποστεί κβαντοποίηση[19]

Τελευταία ενέργεια στη διαδικασία της μετατροπής-μετασχηματισμού είναι η σάρωση των τιμών του πίνακα P κατά ζιγκ-ζαγκ ακολουθία και η τοποθέτηση τους με αυτή τη σειρά σε ένα μονοδιάστατο πίνακα προκειμένου την αποσυσχέτιση των θέσεων των γειτονικών τιμών.

Μετά και από αυτήν την διαδικασία ακολουθεί η κωδικοποίηση των bit των δεδομένων αλλάζοντας τα LSB των τιμών του πίνακα ζιγκ-ζαγκ ως εξής

Αν το bit που θέλουμε να εισάγουμε είναι 0 τότε η τιμή του πίνακα γίνεται ζυγός αριθμός, ενώ αν εισάγεται το 1 , τότε η τιμή του πίνακα γίνεται μονός.

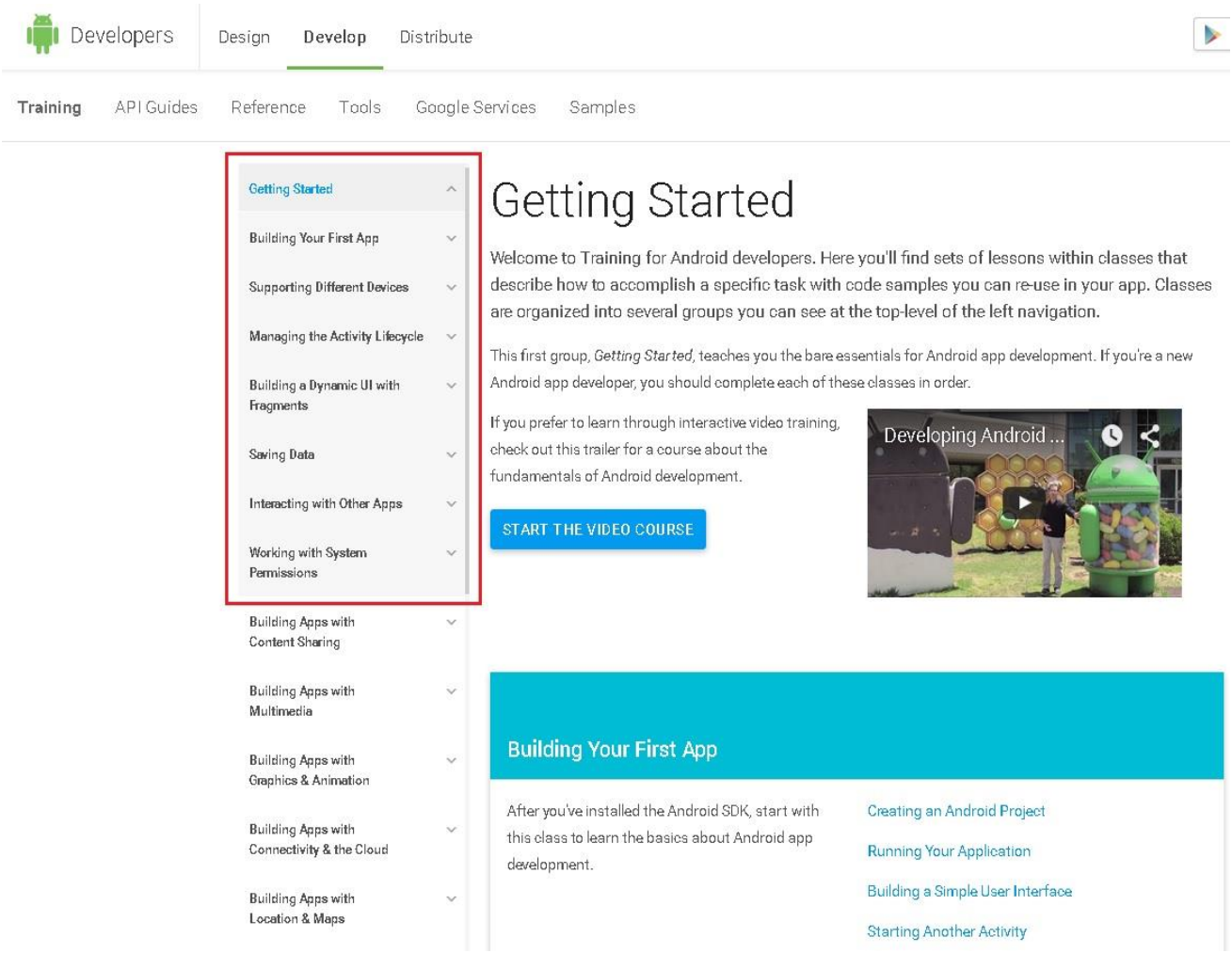
Η όλη διαδικασία της κωδικοποίησης ολοκληρώνεται για το block με τη μετατροπή του μονοδιάστατου πίνακα ζιγκ-ζαγκ πίνακα τιμών 8*8 και την αντίστροφη διαδικασία του DCT για κάθε block και το σχηματισμό της Στεγανογραφημένης πλέον εικόνας.

Η ανάγνωση του μηνύματος ακολουθεί την διαδικασία της κβαντοποίησης και της δημιουργίας του πίνακα ζιγκ-ζαγκ όπως και προηγούμενα κατά την κωδικοποίηση και ολοκληρώνεται με την εξαγωγή των bit ως εξής, αν η τιμή του πίνακα είναι ζυγός αριθμός τότε το bit που εξάγεται είναι το 0, ενώ αν είναι μονός αριθμός, τότε το bit είναι 1. Τα bit από κάθε block συνενώνονται και προκύπτει το μυστικό μήνυμα[18].

Μία εναλλακτική πρόταση για την ενίσχυση της ασφάλειας της μεθόδου DCT στη Στεγανογραφία στα πλαίσια της εφαρμογής είναι η χρήση ενός password με αντίστοιχο τρόπο όπως προτάθηκε και για τη χρήση του στη μέθοδο LSB, με τη μόνη διαφορά τον καθορισμό με τη βοήθεια του password ψευδοτυχαία του μεγέθους του block (N).

7 Παράρτημα Α (Appendix A)

Στην επίσημη ιστοσελίδα του Android[10] μπορεί κανείς να βρει όλες τις πληροφορίες που χρειάζεται για την ανάπτυξη μιας εφαρμογής σε Android. Η σελίδα φιλοξενεί οδηγίες και επεξηγήσεις σε κείμενο αλλά και οπτικοακουστικό υλικό. Η πρώτη σελίδα που αφορά στην εκπαίδευση για την ανάπτυξη εφαρμογής φαίνεται στην εικόνα 50

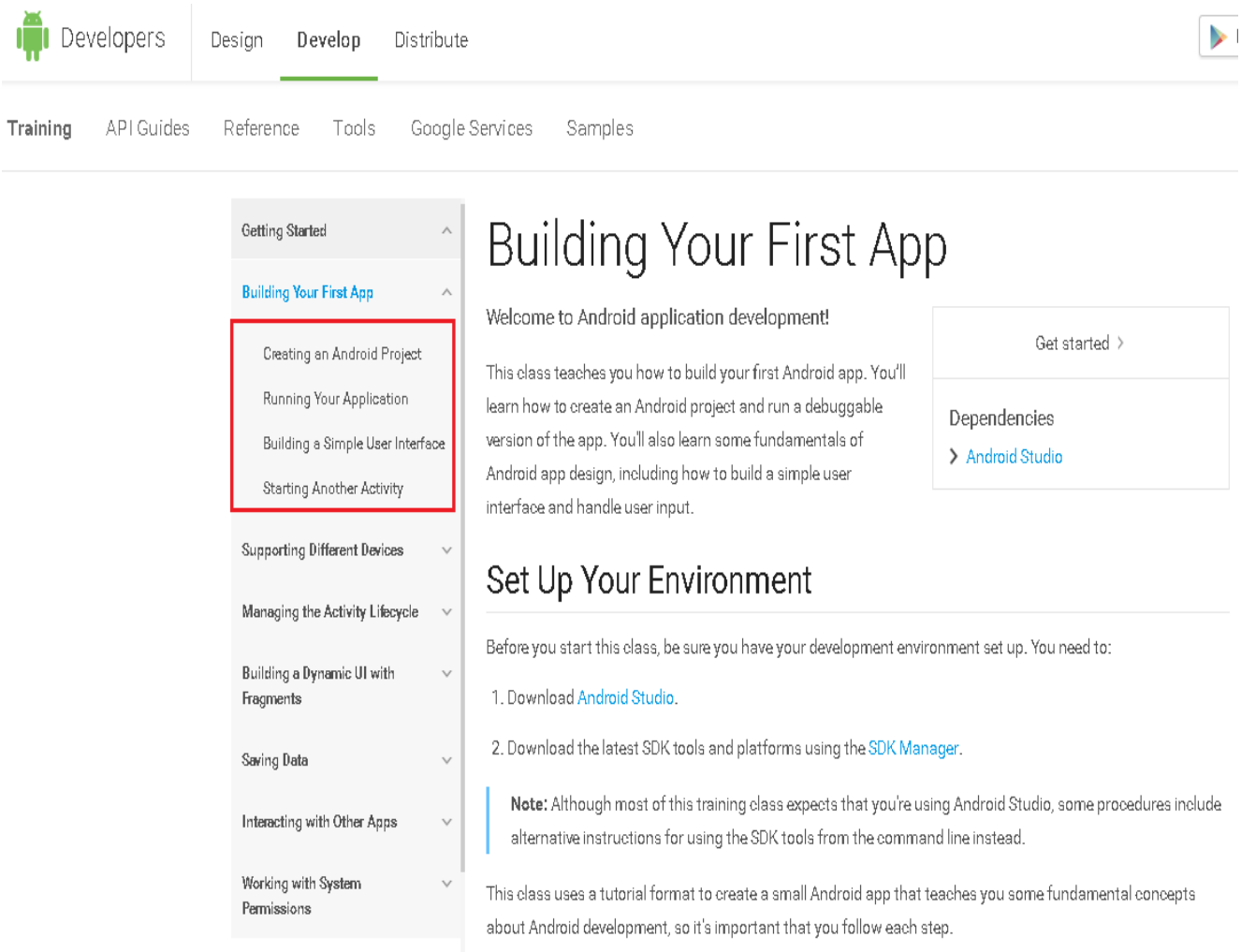


Εικόνα 50: Η εκπαιδευτική σελίδα της Google για την ανάπτυξη της εφαρμογής[10], Downloaded 10/11/2015

Από τα πιο σημαντικά στοιχεία που καταγράφονται στην ιστοσελίδα είναι ένας κατάλογος από θέματα που αναλύουν τα βασικά στοιχεία για το ξεκίνημα μιας εφαρμογής. Ανάμεσα στα άλλα είναι το πώς να ξεκινήσει κανείς το «χτίσιμο» μιας εφαρμογής, την έννοια της δραστηριότητας(Activity) σε μια εφαρμογή και το χτίσιμο

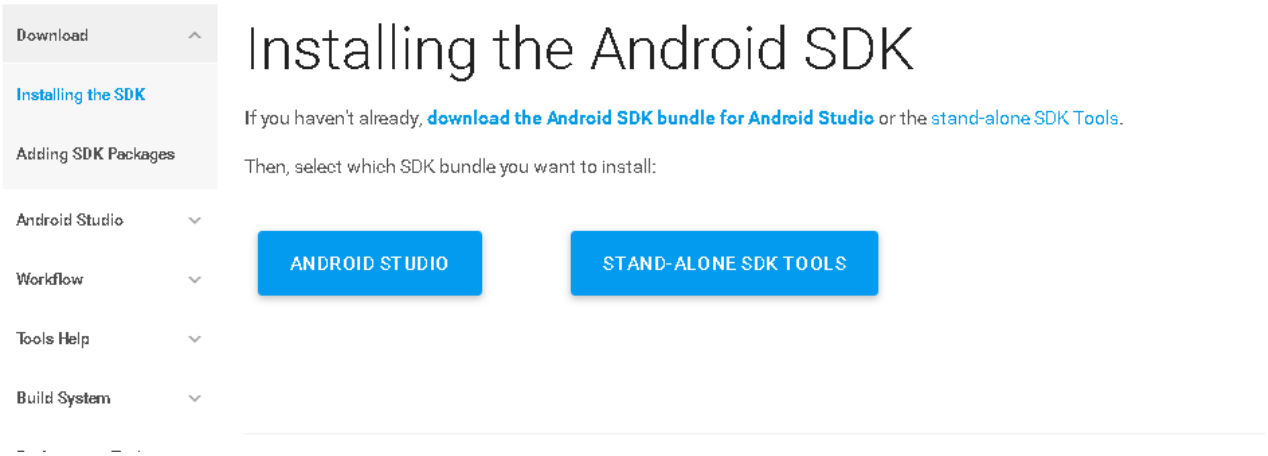
του γραφικού περιβάλλοντος χρήστη και η αποθήκευση δεδομένων όπως φαίνεται και στην εικόνα 50.

Ξεκινώντας από τα βασικά στοιχεία της ανάπτυξης στη σελίδα για το χτίσιμο μιας πρώτης βασικής εφαρμογής, τη δημιουργία ενός πρώτου project και την έναρξή του, τη δημιουργία ενός βασικού γραφικού περιβάλλοντος και την έναρξη και εκτέλεση άλλον δραστηριοτήτων όπως φαίνεται στην εικόνα 51.

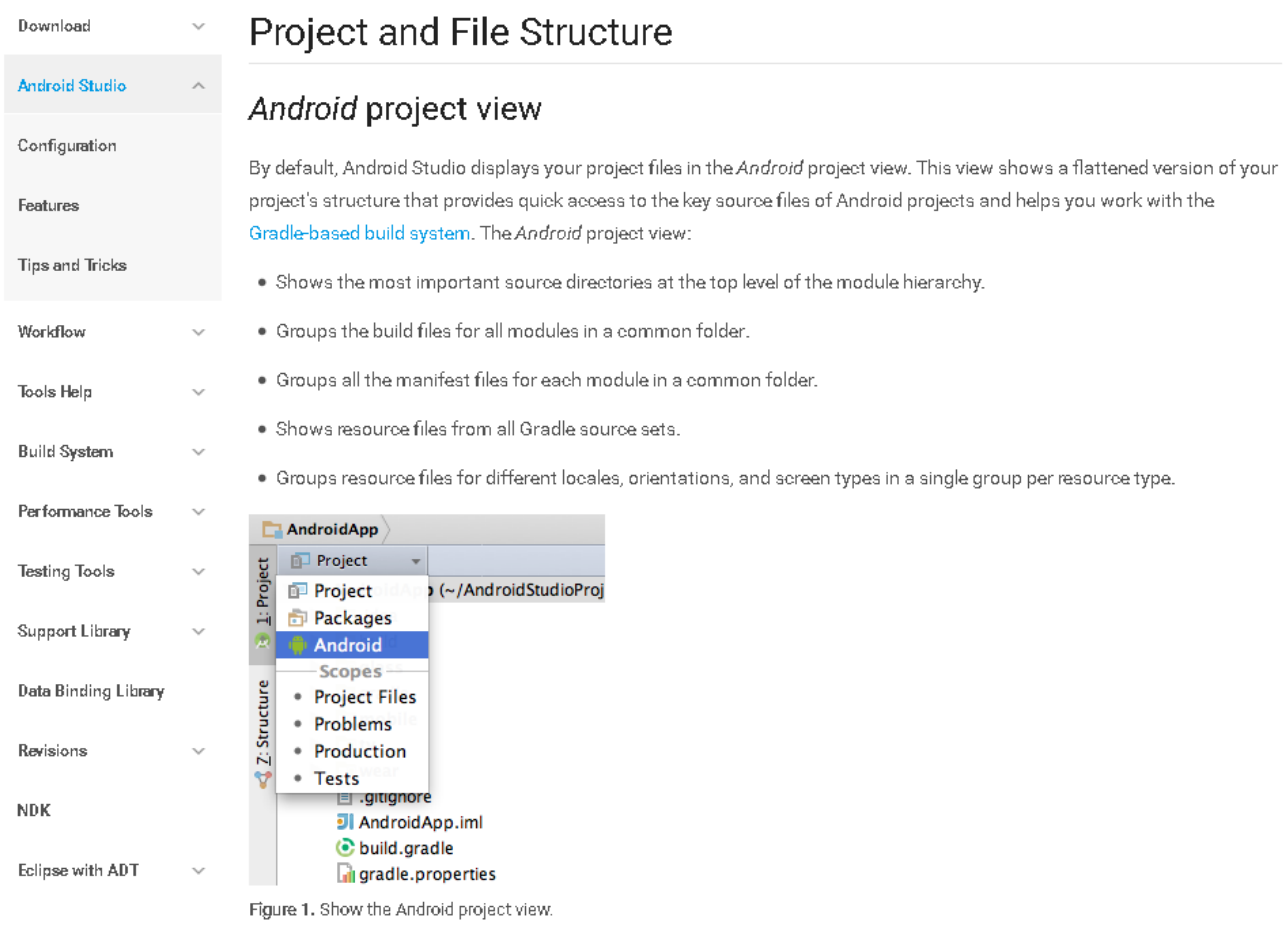


Εικόνα 51: Οδηγίες για την ανάπτυξη μιας της πρώτης εφαρμογής[10], Downloaded 10/11/2015

Η υλοποίηση της εφαρμογής γίνεται με τη χρήση ενός Software Development Kit του Android Studio. Στην πιο πάνω σελίδα δίνεται ο σύνδεσμος (link) για την αποθήκευση του. Στη σελίδα του Android Studio δίνονται και οδηγίες για την εγκατάσταση και τη χρήση του[11] όπως φαίνεται και στην εικόνα 52 και 53.

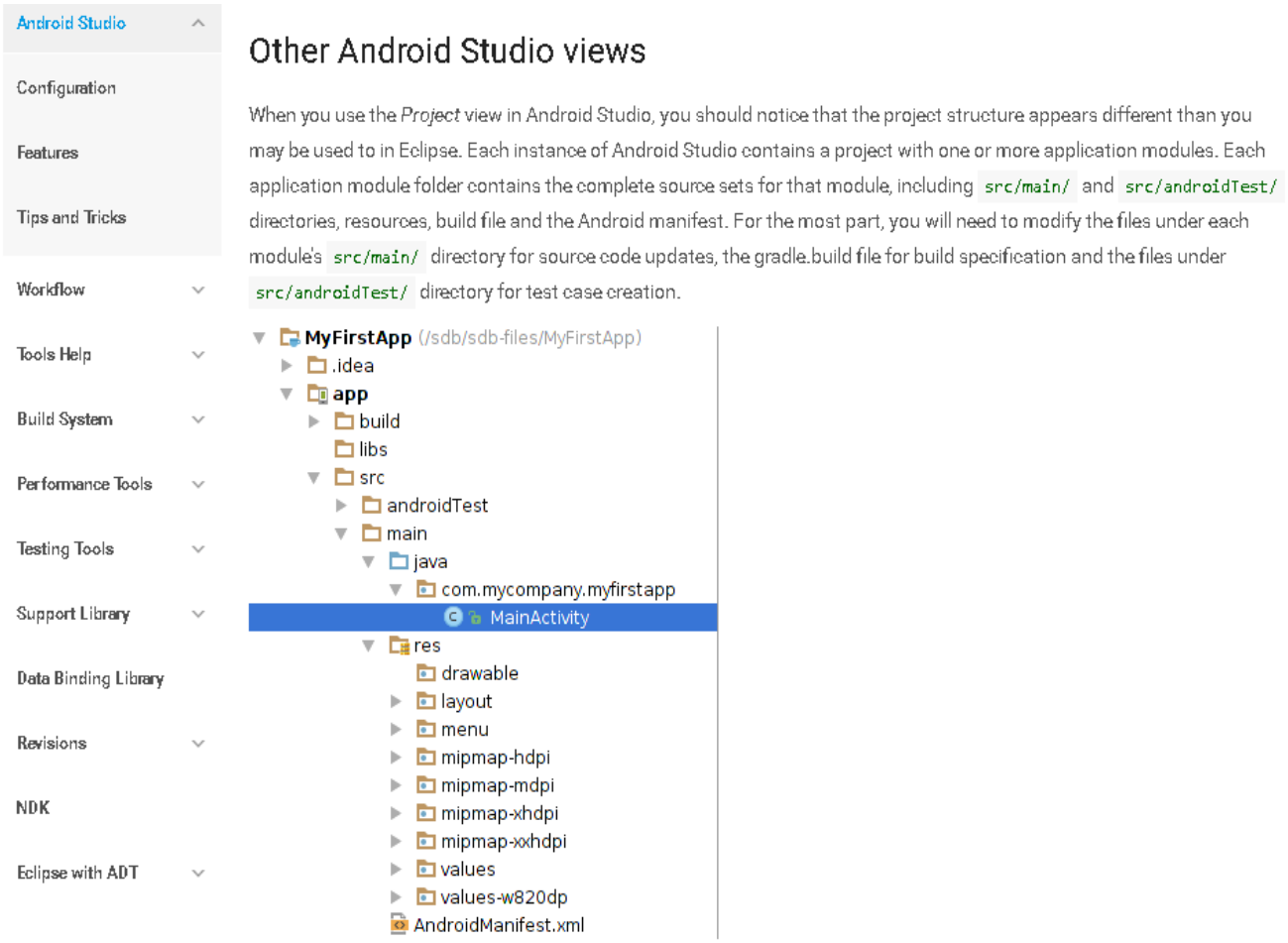


Εικόνα 51: Οδηγίες Εγκατάστασης του Android Studio[11], Downloaded 10/11/2015



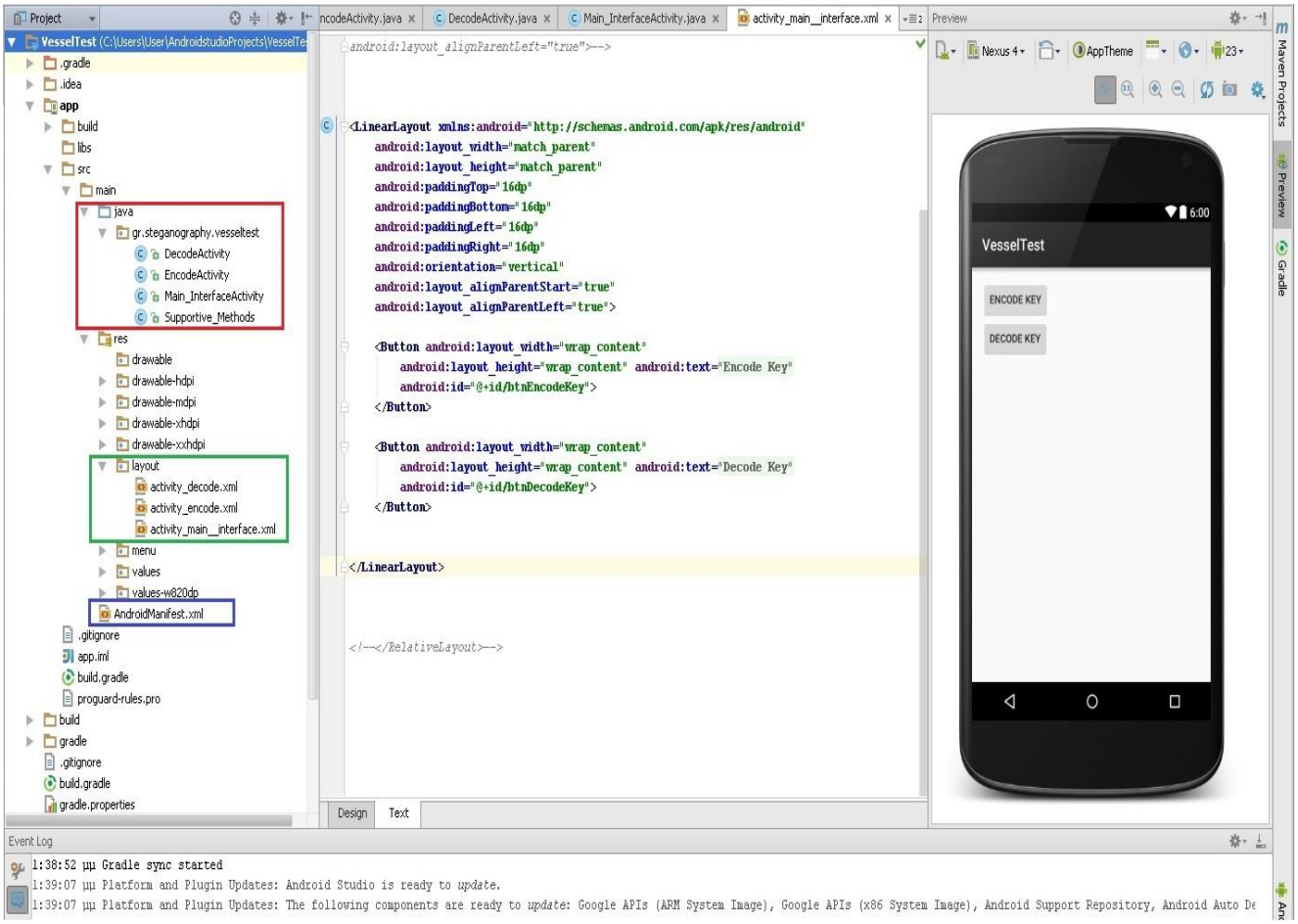
Εικόνα 52: Οδηγίες για τη χρήση του Android Studio[11], Downloaded 10/11/2015

Στη σελίδα μπορούμε να βρούμε περισσότερες πληροφορίες για το GUI του Android Studio όσο και για τις βασικές δομές μιας εφαρμογής όπως φαίνεται στις εικόνες 53 και 54 αντίστοιχα.



Εικόνα 53: Η δομή των αρχείων μιας εφαρμογής[11], Downloaded 10/11/2015

Για την περίπτωση της εφαρμογής που αναπτύχθηκε, μια πιο ολοκληρωμένη άποψη του Android Studio δίνεται στην εικόνα 54.



Εικόνα 54: Άποψη της εφαρμογής VesselApp(VesselTest) μέσα από το Android Studio

Στο αριστερό πλαίσιο του Android Studio φαίνεται η δομή των αρχείων της εφαρμογής. Στο κεντρικό πλαίσιο φαίνεται ο κώδικας του εκάστοτε αρχείου που έχει επιλεγεί για επεξεργασία (εδώ είναι το activity_main_interface.xml), στα δεξιά φαίνεται μια άποψη σε εικονική συσκευή η εμφάνιση του GUI της εφαρμογής που ορίζεται από τον κώδικα του κεντρικού παραθύρου.

Αυτό που αξίζει να αναφέρουμε είναι η περιγραφή για κάποια από τα κύρια αρχεία της εφαρμογής όπως αυτά εμφανίζονται μέσα στο κόκκινο, το πράσινο και το μπλε πλαίσιο, μέσα στην δομή που εμφανίζεται στα αριστερά της εικόνας 54.

Τα αρχεία .java, κλάσεις της Java τα οποία προσδίδουν λειτουργικότητα στην εφαρμογή είτε με τον ορισμό μεθόδων όπως η κλάση Supportive_Methods της εφαρμογής, είτε με τον προσδιορισμό δραστηριοτήτων, αυτό που σε άλλες γλώσσες προγραμματισμού αλλά και στη Java σε διαφορετικά περιβάλλοντα καλείται ως κύρια μέθοδος main() και που σε κάθε περίπτωση είναι υπεύθυνο για την εκτέλεση μιας

κύριας λειτουργίας σε ένα πρόγραμμα. Στο περιβάλλον του android η δραστηριότητα ανοίγει ένα νέο παράθυρο – μια νέα οθόνη – με το οποίο μπορεί να διασυνδέεται ένα περιβάλλον χρήστη (User Interface)[19].

Τα αρχεία .xml που χρησιμοποιούνται για την περιγραφή, σε μορφή κώδικα XML, των διαφορετικών οθονών-περιβαλλόντων χρήστη(UI) όπως δημιουργούνται κατά βούληση από τον προγραμματιστή και οργανώνονται όπως στο πράσινο πλαίσιο της εικόνας 54. Στο κέντρο της εικόνας φαίνεται και ένα παράδειγμα κώδικα XML όπως ορίζεται για την οθόνη της κύριας δραστηριότητας Main_InterfaceActivity.java και περιγράφεται στο αρχείο activity_main_interface.xml.

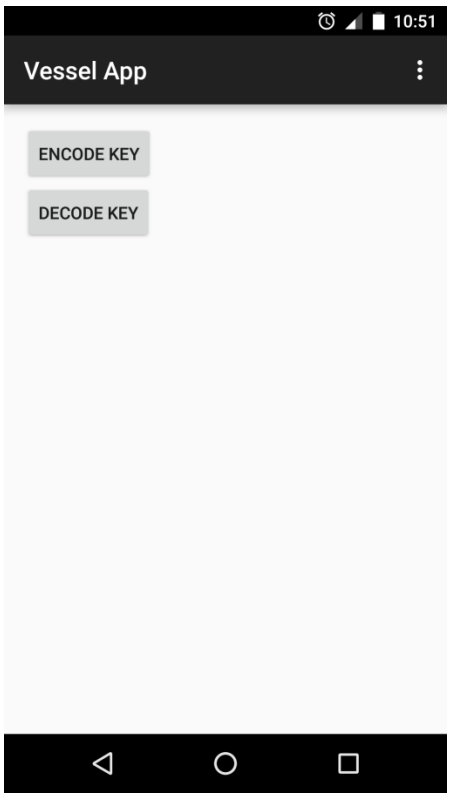
Για το τέλος αφήσαμε το αρχείο AndroidManifest.xml στο μπλε πλαίσιο της εικόνας 54. Κάθε εφαρμογή πρέπει να έχει ένα τέτοιο αρχείο. Το αρχείο αυτό παρουσιάζει σημαντικές πληροφορίες για την εφαρμογή προς το λειτουργικό σύστημα του android όπως μεταξύ άλλων είναι[20]:

- Το όνομα του πακέτου της Java που δημιουργείται κατά τη δημιουργία του project για την εφαρμογή.
- Η περιγραφή των περιεχομένων της εφαρμογής όπως δραστηριότητες, υπηρεσίες κτλ.
- Ο καθορισμός των δικαιωμάτων που θα πρέπει να έχει η εφαρμογή προκειμένου να έχει πρόσβαση σε διάφορα - περιορισμένης πρόσβασης – προστατευόμενα μέρη του λειτουργικού αλλά και κατά τη αλληλεπίδραση με άλλες εφαρμογές
- Τον ορισμό του ελάχιστου επιπέδου API – έκδοσης του λειτουργικού που απαιτείται - για την εκτέλεση της εφαρμογής.

8 Παράρτημα Β (Appendix B)

Στην παράγραφο αυτή περιγράφεται η λειτουργία της εφαρμογής από την οπτική του χρήστη, μέσα από μια σειρά εικόνων που έχουν ληφθεί σε πραγματικό χρόνο κατά τη διάρκεια λειτουργίας της εφαρμογής για τη διαδικασία της α) κωδικοποίησης και β) αποκωδικοποίησης.

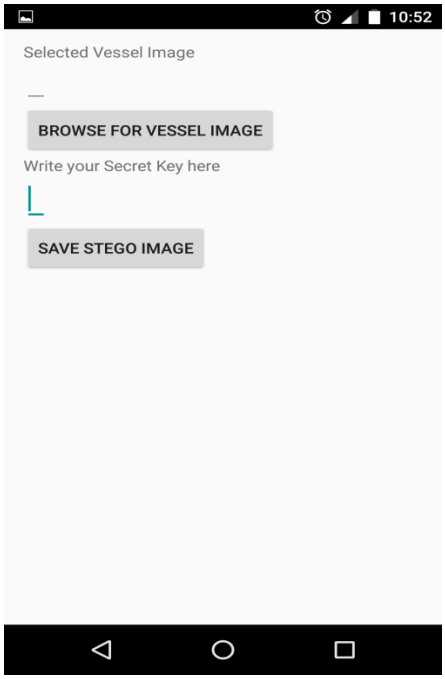
Αρχικά η πρώτη οθόνη που βλέπει ο χρήστης ανοίγοντας την εφαρμογή φαίνεται στην εικόνα 55



Εικόνα 55: Πρώτη οθόνη της εφαρμογής

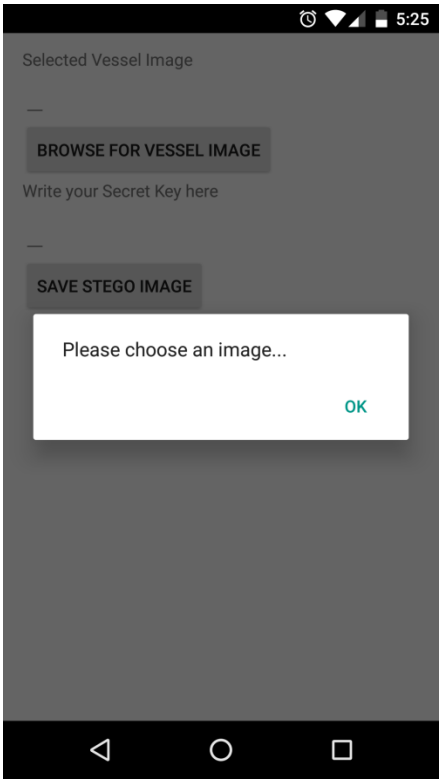
α) <u>Κωδικοποίηση</u>

Επιλέγοντας το κουμπί της κωδικοποίησης(Encode) ο χρήστης μεταβαίνει στην αντίστοιχη οθόνη όπως στην εικόνα 56.

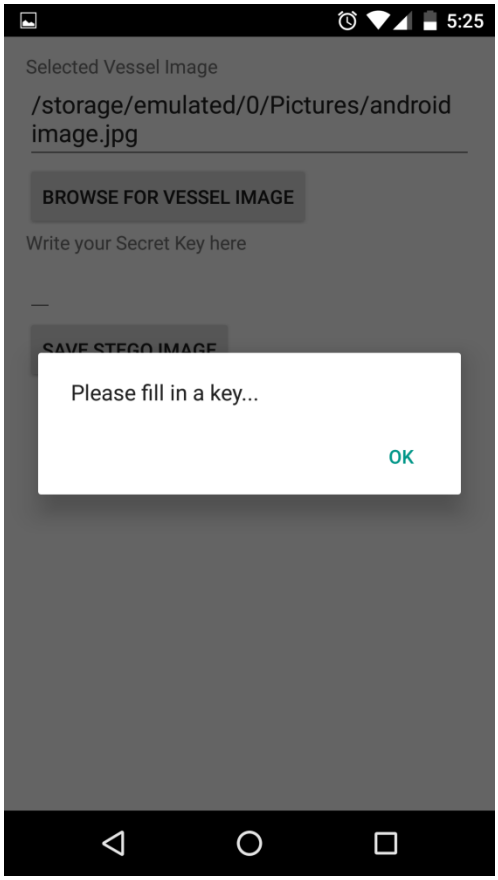


Εικόνα 56: Η οθόνη της κωδικοποίησης

Εάν ο χρήστης επιχειρήσει να ξεκινήσει τη διαδικασία κωδικοποίησης είτε χωρίς να έχει επιλεγεί εικόνα κάλυψης, είτε χωρίς να έχει συμπληρωθεί κλειδί για να κωδικοποιηθεί εμφανίζονται στην οθόνη του αντίστοιχα πλαίσια διαλόγου που τον προτρέπουν να το κάνει. Τα πλαίσια διαλόγου φαίνονται στις παρακάτω εικόνες 57 και 58.

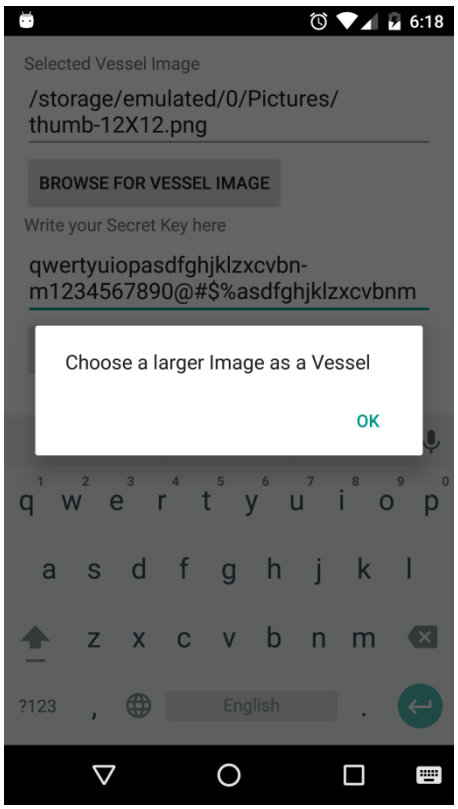


Εικόνα 57: Πλαίσιο διαλόγου για την επιλογή εικόνας



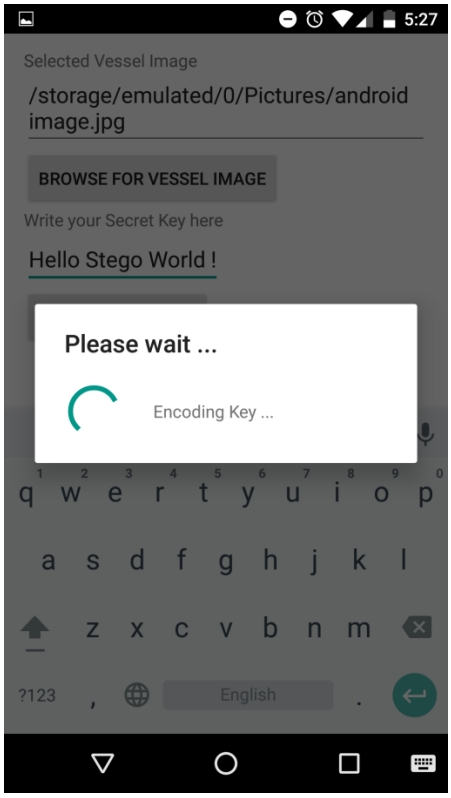
Εικόνα58: Πλαίσιο διαλόγου για τη συμπλήρωση κλειδιού

Στην περίπτωση που οι πιο πάνω συνθήκες έχουν καλυφτεί αλλά η εικόνα που έχει επιλεγεί είναι πολύ μικρή για να φιλοξενήσει τα δεδομένα, τότε ένα άλλο πλαίσιο διαλόγου ενημερώνει για την ανάγκη επιλογής μεγαλύτερης εικόνας όπως φαίνεται στην εικόνα 59.



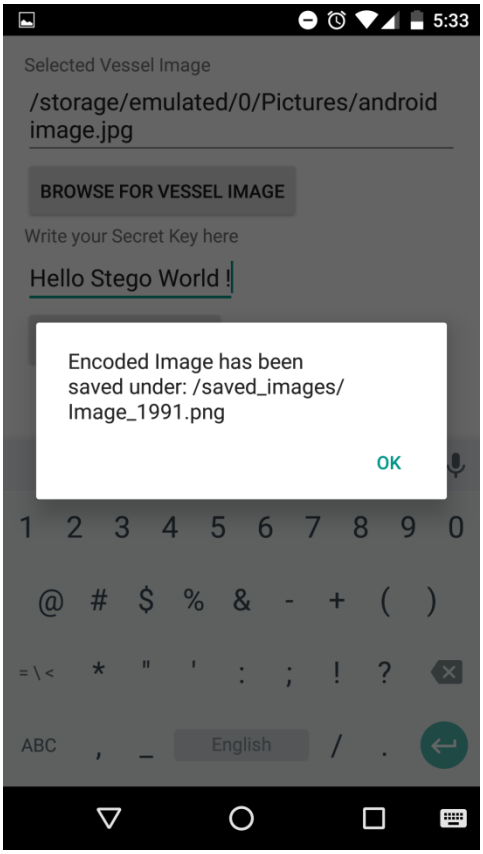
Εικόνα 59: Πλαίσιο διαλόγου για μεγαλύτερη εικόνα

Αν όλες οι συνθήκες έχουν καλυφτεί, τότε ξεκινά η διαδικασία της κωδικοποίησης με ένα πλαίσιο διαλόγου προόδου όπως στην εικόνα 60.



Εικόνα 60: Πλαίσιο διαλόγου προόδου

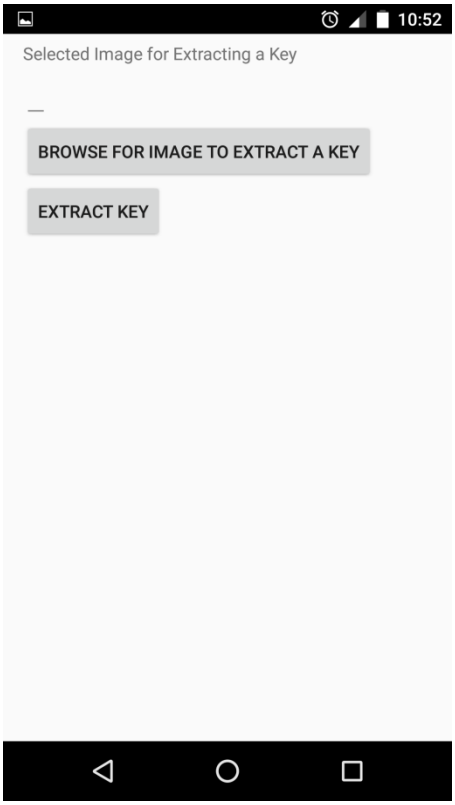
Όταν η διαδικασία της κωδικοποίησης ολοκληρωθεί, ένα άλλο πλαίσιο διαλόγου ενημερώνει για την αποθήκευση της Στεγανογραφημένης εικόνας, τη θέση της και το όνομα της όπως φαίνεται στην εικόνα 61.



Εικόνα 61: Πλαίσιο διαλόγου για την αποθήκευση της Στεγανογραφημένης εικόνας

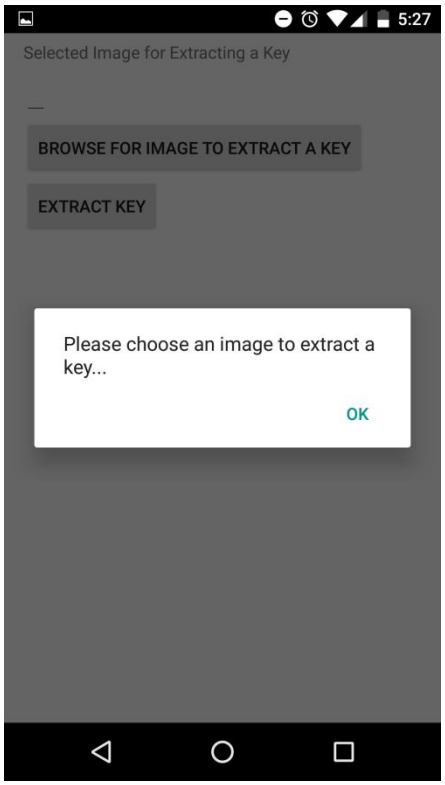
β) <u>Αποκωδικοποίηση</u>

Κατά την αποκωδικοποίηση ο χρήστης μέσω του κουμπιού Decode της πρώτης οθόνης της εφαρμογής μεταβαίνει στην οθόνη αποκωδικοποίησης της εικόνας 62.



Εικόνα 62: Οθόνη αποκωδικοποίησης

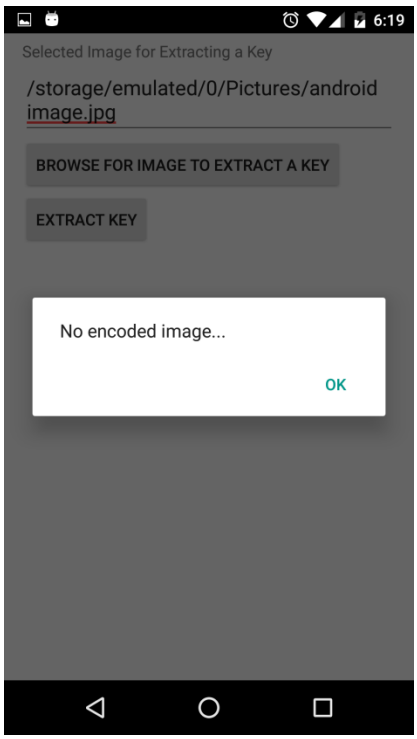
Αν ο χρήστης πατήσει το κουμπί για την αποκωδικοποίηση χωρίς να έχει επιλεγεί εικόνα τότε εμφανίζεται το παρακάτω πλαίσιο διαλόγου



Εικόνα 63: Πλαίσιο διαλόγου για την επιλογή εικόνας προς αποκωδικοποίηση

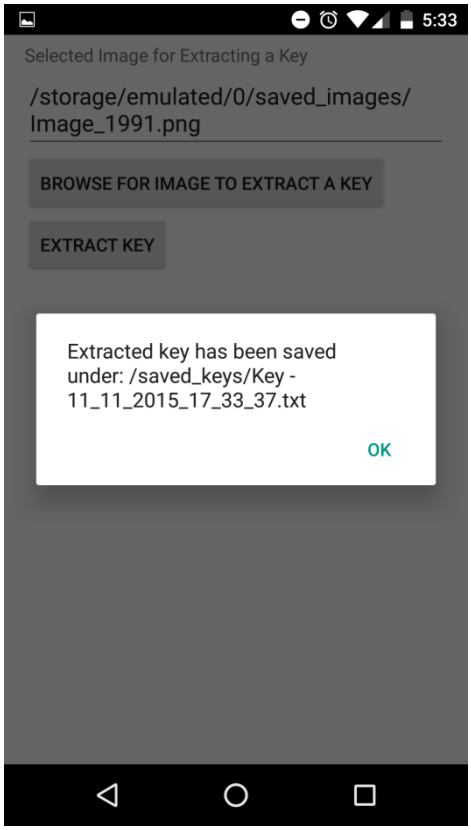
Εφόσον έχει επιλεγεί εικόνα προς αποκωδικοποίηση μέσω του κουμπιού BROWSE FOR IMAGE TO EXTRACT A KEY και μέσα από τη συσκευή, με το πάτημα του EXTRACT KEY ξεκινά η διαδικασία

Σε περίπτωση που εικόνα δεν έχει Στεγανογραφηθεί μέσα από την εφαρμογή, τότε ένα πλαίσιο διαλόγου ενημερώνει για τη μη κωδικοποιημένη εικόνα (εικόνα 64).



Εικόνα 64: Πλαίσιο διαλόγου για μη κωδικοποιημένη εικόνα

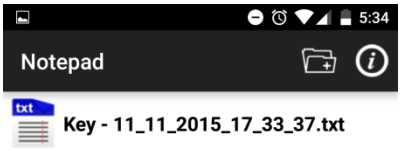
Σε άλλη περίπτωση το κλειδί που εξάγεται αποθηκεύεται στη συσκευή σε ένα αρχείο τύπου .txt και ένα πλαίσιο διαλόγου ενημερώνει για την αποθήκευση του με τη θέση και το όνομά του (εικόνα 65).

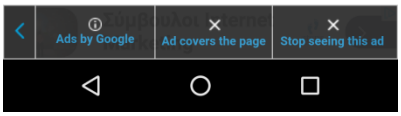


Εικόνα 65: Πλαίσιο διάλογου για την αποθήκευση του κλειδιού που εξήχθη

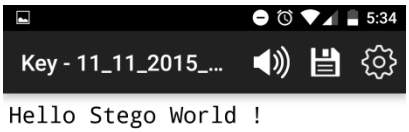
Κατά την εξαγωγή του κλειδιού και πριν την αποθήκευσή του, γίνεται έλεγχος με ένα CRC32 checksum για την ακεραιότητα του κλειδιού. Αν διαπιστωθούν σφάλματα τότε ένα αντίστοιχα πλαίσιο διάλογου εμφανίζεται ενημερώνοντας σχετικά και το κλειδί δεν αποθηκεύεται.

Τελικά αν έχει γίνει αποθήκευση του κλειδιού όπως μπορούμε να δούμε και στις εικόνες 66 και 67 το κλειδί έχει εξαχθεί ακέραιο με επιτυχία.





Εικόνα 66: Το όνομα του αρχείου .txt που έχει δημιουργηθεί με το κλειδί



Εικόνα 67: Το κλειδί που εξήχθη σωστά μέσα στο αρχείο .txt

9 Βιβλιογραφία

1. ISO/IEC 17799: 2000
2. A Detailed look at Steganographic Techniques and their use in an Open-Systems Environment, Bret Dunbar, SANS Institute 2002
3. http://www.cs.virginia.edu/~wm2a/HistorialOverview.html
4. Disappearing Cryptography, Information Hiding: Steganography & Watermarking, Peter Wayner
5. AN OVERVIEW OF IMAGE STEGANOGRAPHY, T. Morkel, J.H.P. Eloff, M.S. Olivier, Information and Computer Security Architecture (ICSA) Research Group Department of Computer Science University of Pretoria, 0002, Pretoria, South Africa
6. An introduction to steganography methods, Masoud Nosrati, Ronak Karimi, Mehdi Hariri, World Applied Programming, Vol (1), No (3), August 2011. 191-195 ISSN: 2222-2510 ©2011 WAP journal.
7. Symmetric and Asymmetric Encryption, GUSTAVUS J. SIMMONS, Sandm Laboratories, Albuquerque, New Mexico 87185
8. Introduction to Public Key Infrastructure, D.Richard Kuhn, Vincent C.Hu, W.Timothy Polk, Shu-Jen Chang, NIST (National Institute of Standards and Technology), 26 February 2001
9. http://www.oracle.com/technetwork/java/index.html
10. https://developer.android.com/training/index.html
11. http://developer.android.com/sdk/index.html#top
12. https://en.wikipedia.org/wiki/RGBA_color_space
13. https://en.wikipedia.org/wiki/Alpha_compositing
14. http://developer.android.com/training/basics/activity-lifecycle/starting.html
15. http://developer.android.com/reference/android/graphics/Bitmap.html
16. http://developer.android.com/reference/java/lang/Thread.html
17. A Review of Comparison Techniques of Image Steganography, Stuti Goel, Arun Rana, Manpreet Kaur, *IOSR Journal of Electrical and Electronics Engineering (IOSR-JEEE) e-ISSN: 2278-1676,p-ISSN: 2320-3331, Volume 6, Issue 1 (May. - Jun. 2013)*

18. Data Security Using LSB & DCT Steganography In Images, Deepak Singla, Rupali Syal, Deepak Singla, Rupali Syal /International Journal Of Computational Engineering Research/ ISSN: 2250–3005
19. https://developer.android.com/reference/android/app/Activity.html
20. http://developer.android.com/guide/topics/manifest/manifest-intro.html