ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόγραμμα Μεταπτυχιακών Σπουδών
Ασφάλεια Ψηφιακών Συστημάτων

Ανάλυση αποτυπωμάτων μνήμης εφαρμογών Android οι οποίες εφαρμόζουν κρυπτογράφηση AES βασιζόμενη σε μυστικό κωδικό.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΡΑΔΑΜΑΝΘΥΣ Κ. ΔΕΡΕΔΑΚΗΣ

Επιβλέπων : Χρήστος Ξενάκης

Αναπληρωτής Καθηγητής Πανεπιστημίου Πειραιώς.

Πειραιάς , Ιούνιος 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πρόγραμμα Μεταπτυχιακών Σπουδών
Ασφάλεια Ψηφιακών Συστημάτων

Ανάλυση αποτυπωμάτων μνήμης εφαρμογών Android οι οποίες εφαρμόζουν κρυπτογράφηση AES βασιζόμενη σε μυστικό κωδικό.

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΡΑΔΑΜΑΝΘΥΣ Κ. ΔΕΡΕΔΑΚΗΣ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την .....η Ιουνίου 2015.

.....	.....	.....
Χ.Ξενάκης	Κ.Λαμπρινουδάκης	Σ. Κάτσικας
Αναπληρωτής .Καθηγητής	Αναπληρωτής.Καθηγητής	Καθηγητής
Παν. Πειραιά	Παν. Πειραιά	Παν. Πειραιά

Πειραιάς , Ιούνιος 2015

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της διπλωματικής εργασίας, κ. Χρήστο Ξενάκη , καθώς και τον επιστημονικό συνεργάτη του Πανεπιστήμιου Πειραιώς Χριστόφορο Νταντογιάν για την πολύτιμη βοήθεια, καθοδήγηση και υπομονή τους σε όλη την πορεία ανάπτυξης και συγγραφής της διπλωματικής αυτής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την υποστήριξη που μου προσέφερε, γεγονός το οποίο συνέβαλλε στην ολοκλήρωση των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Στην παρούσα διπλωματική εργασία, γίνεται μελέτη των ευρημάτων τα οποία προκύπτουν από την ανάλυση αποτυπωμάτων μνήμης εφαρμογών Android οι οποίες χρησιμοποιούν κρυπτογράφηση AES βασιζόμενες σε ένα μυστικό κωδικό που εισάγει ο χρήστης.

Για την ανάλυση αυτή δημιουργήθηκε μια εφαρμογή Android με ονομασία "AES Android Forensics" η οποία προσομοίαζε τις υπό εξέταση εφαρμογές. Κύριος σκοπός της συγκεκριμένης εφαρμογής ήτανε η εμφάνιση του κλειδιού κρυπτογράφησης και η εύρεσή του στο αποτύπωμα μνήμης.

Με βάση τα συμπεράσματα που προέκυψαν από την ανάλυση της συγκεκριμένης εφαρμογής έγινε η εξέταση δέκα παρόμοιων εφαρμογών Android. Σε πέντε από αυτές βρέθηκαν όλες οι παράμετροι κρυπτογράφησης του AES, συμπεριλαμβανομένου του κλειδιού κρυπτογράφησης.

Για την αυτοματοποίηση της παραπάνω διαδικασίας δημιουργήθηκαν τρία διαφορετικά προγράμματα σε Java.

Το πρώτο προσομοιάζει τις πέντε παραπάνω Android εφαρμογές σε περιβάλλον Java.

Το δεύτερο αναλύει τα κλειδιά κρυπτογράφησης τα οποία παράγονται από τις πέντε αυτές εφαρμογές.

Τέλος τρίτο αναζητά τα κλειδιά κρυπτογράφησης με τη χρήση δυαδικής εντροπίας, σε αποτυπώματα μνήμης τα οποία παράγονται από Android εφαρμογές οι οποίες χρησιμοποιούν κρυπτογράφηση AES βασιζόμενες σε ένα μυστικό κωδικό.

Λέξεις Κλειδιά: αποτύπωμα μνήμης, κρυπτογράφηση με χρήση κωδικού , κλειδί κρυπτογράφησης, κρυπτογράφηση AES, παράμετροι κρυπτογράφησης,
δυαδική εντροπία.

ABSTRACT

Subject matter of this study is, the memory dump analysis of Android applications that use AES Password Based Encryption (PBE).

For the implementation of this analysis a custom android application named "AES Android Forensics", was created which simulated the applications that use AES Password Based Encryption. Main purpose of this application is to show the AES encryption key, and it's location to the memory dump.

Based on the analysis and findings of the AES Android Forensics application, ten similar android AES PBE applications were examined. In five of them, all of the AES cipher parameters were found including the encryption key.

For further automation of the above stated process three more Java programs were developed.

The first one emulates the five android applications in Java environment.

The second one analyzes the pattern of the encryption keys created by these android applications.

Finally the third one, searches in memory dumps for AES PBE encryption keys, based on binary entropy.

Key Words: memory dumps, Password Based Encryption, PBE, encryption key AES encryption, cipher parameters, binary entropy, android, Java program,

Στους γονείς μου, Κώστα και Ιωάννα..

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΥΡΕΤΗΡΙΟ ΣΧΗΜΑΤΩΝ-ΠΙΝΑΚΩΝ

ΚΕΦΑΛΑΙΟ 1

Εικόνες

ΚΕΦΑΛΑΙΟ 2

Εικόνες

Πίνακες

ΚΕΦΑΛΑΙΟ 3

Εικόνες

Πίνακες

ΚΕΦΑΛΑΙΟ 4

Εικόνες

ΠΑΡΑΡΤΗΜΑ Α

ΠΑΡΑΡΤΗΜΑ Β

ΠΑΡΑΡΤΗΜΑ Γ

ΕΙΣΑΓΩΓΗ

Με την ολοένα μεγαλύτερη αύξηση της χρήσης του διαδικτύου και την ταυτόχρονη αύξηση της χρήσης κινητών τηλεφώνων, η ανάγκη για τη διατήρηση της ιδιωτικότητας και της προστασίας των προσωπικών δεδομένων οδήγησε στην ανάπτυξη εφαρμογών οι οποίες χρησιμοποιούν κρυπτογράφηση AES βασιζόμενες σε ένα μυστικό κωδικό που εισάγει ο χρήστης.
Σκοπός της συγκριμένης εργασίας είναι, το κατά πόσον αυτές οι εφαρμογές είναι όντως σε θέση να προστατεύσουν σε αξιόπιστο βαθμό τα προσωπικά δεδομένα του χρήστη. Αυτό γίνεται με την εξαγωγή συμπερασμάτων από την ανάλυση των αποτυπωμάτων μνήμης που αυτές παράγουν.

Η διάρθρωση της διπλωματικής σε κεφάλαια είναι:

Κεφάλαιο 1: Γίνεται εισαγωγή στο λειτουργικό σύστημα του Android, καθώς και στην κρυπτογράφηση με χρήση κωδικού, καθώς επίσης και στον αλγόριθμο κρυπτογράφησης του AES. Τα τρία παραπάνω αποτελούν τα δομικά στοιχεία των android εφαρμογών οι οποίες εξετάστηκαν.

Κεφάλαιο 2: Στο δεύτερο κεφάλαιο γίνεται ανάλυση του τρόπου λειτουργίας της εφαρμογής η οποία δημιουργήθηκε (AES Android Forensics), πάνω στη οποία βασίστηκε η ανάλυση, των παρόμοιων android εφαρμογών.

Κεφάλαιο 3: Στο κεφάλαιο αυτό γίνεται η ανάλυση δέκα εφαρμογών οι οποίες χρησιμοποιούν κρυπτογράφηση AES με τη χρήση κωδικού τον οποίο εισάγει ο χρήστης.

Κεφάλαιο 4: Στο τέταρτο κεφάλαιο γίνεται η ανάλυση των τριών Java προγραμμάτων τα οποία δημιουργήθηκαν βασιζόμενα στις προηγούμενες αναλύσεις. Ιδιαίτερη έμφαση δίνεται στο πρόγραμμα της αναζήτησης κλειδιού κρυπτογράφησης σε αποτυπώματα μνήμης με τη χρήση δυαδικής εντροπίας.

ΚΕΦΑΛΑΙΟ 1

ΚΡΥΠΤΟΓΡΑΦΗΣΗ ΜΕ ΒΑΣΗ ΚΩΔΙΚΟ ΧΡΗΣΤΗ ΣΤΟ ΛΕΙΤΟΥΡΓΙΚΟ ΣΥΣΤΗΜΑ ANDROID

Στις ενότητες του κεφαλαίου αυτού, εξετάζεται, ο αλγόριθμος κρυπτογράφησης AES, καθώς επίσης και η κρυπτογράφηση με βάση ένα μυστικό κωδικό.
Επίσης αναλύεται η αρχιτεκτονική του λειτουργικού Android, καθώς και ο τρόπος με τον οποίο διεξάγεται η κρυπτογράφηση σε αυτό.

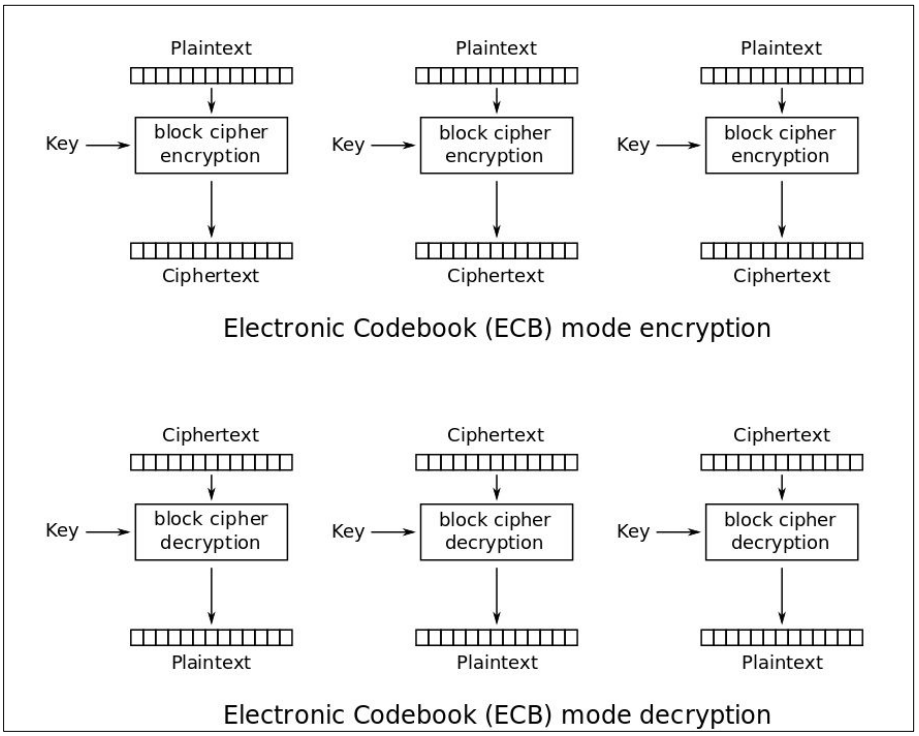
1.1 Ο αλγόριθμος κρυπτογράφησης AES

Ο αλγόριθμος κρυπτογράφησης AES (Advanced Encryption Standard) πιστοποιήθηκε το 2001 από Εθνικό Ινστιτούτο Πιστοποίησης και Τεχνολογίας των Ηνωμένων Πολιτειών (National Institute of Standards and Technology-NIST), σαν αποδεκτός αλγόριθμος κρυπτογράφησης και έκτοτε θεωρείται από τους πλέον ασφαλείς, και είναι από τους πιο διαδεδομένους παγκοσμίως.
Ο AES ανήκει στην κατηγορία των συμμετρικών αλγορίθμων κρυπτογράφησης, δηλαδή το ίδιο κλειδί με το οποίο γίνεται η κρυπτογράφηση (encryption), χρησιμοποιείται και για την αποκρυπτογράφηση (decryption).
Οι συμμετρικοί αλγόριθμοι κρυπτογράφησης διακρίνονται σε δύο κατηγορίες, στους αλγόριθμους ροής (stream ciphers) και στους αλγόριθμους τμήματος (block ciphers). Στην πρώτη κατηγορία το κάθε byte κρυπτογραφείται ξεχωριστά, ενώ στη δεύτερη τα bytes τα οποία πρόκειται να κρυπτογραφηθούν (plaintext) ομαδοποιούνται σε τμήματα συγκεκριμένου μεγέθους.
Στην περίπτωση όπου τα bytes αυτά δεν είναι ακριβώς στο ίδιο μέγεθος με το τμήμα του αλγόριθμου, χρησιμοποιείται η τεχνική της συμπλήρωσης (padding) των bytes του κειμένου έτσι ώστε αυτά να είναι ακριβές πολλαπλάσιο του τμήματος.
Ο AES είναι αλγόριθμος τμήματος και το κάθε τμήμα του έχει μήκος 16 bytes.
Το κλειδί κρυπτογράφησης το οποίο παράγεται δεν είναι σταθερό αλλά μπορεί να έχει μήκος 128, 192, ή 256 bits.
Καθώς ανήκει στην κατηγορία των αλγορίθμων τμήματος, υποστηρίζει και διαφορετικούς τρόπους δημιουργίας του κλειδιού οι οποίοι παρουσιάζονται παρακάτω:

Electronic Codebook-ECB Mode

το κλειδί κρυπτογραφεί το κάθε block ανεξάρτητα από τα υπόλοιπα, στο αντίστοιχο κρυπτόγραμμα. Δύο ίδια μηνύματα οδηγούν πάντα στο ίδιο κρυπτόγραμμα (αν τους εφαρμοστεί το ίδιο κλειδί).

Άρα, δεν συνίστανται σε εφαρμογές όπου υπάρχουν επαναλαμβανόμενα μοτίβα δεδομένων στο αρχικό μήνυμα. Ένα λάθος στη λήψη επηρεάζει το συγκεκριμένο block μόνο. [3]



Εικόνα 1: Λειτουργία Electronic Codebook-ECB.

Cipher block chaining (CBC) mode

Το αρχικό μήνυμα υποβάλλεται σε αποκλειστική διάζευξη (XOR) με το προηγούμενο κρυπτόγραμμα, πριν την κρυπτογράφηση. Συνεπώς, το κάθε κρυπτόγραμμα εξαρτάται από όλα τα προηγούμενα μηνύματα άρα, δύο ίδια μοτίβα δεν οδηγούν σε ίδια κρυπτογράμματα.

Λάθος κατά τη κρυπτογράφηση σε ένα απλό bit του αρχικού επηρεάζει έπειτα όλα τα υπόλοιπα μηνύματα και άρα οδηγεί σε λανθασμένο κρυπτόγραμμα.

Λάθος κατά τη μετάδοση σε ένα απλό bit του κρυπτογράμματος (που μπορεί να οφείλεται είτε σε σφάλμα μετάδοσης είτε σε παρεμβολή ενός «επιτιθέμενου») προκαλεί λάθος σε δύο τμήματα του αποκρυπτογραφημένου κειμένου.

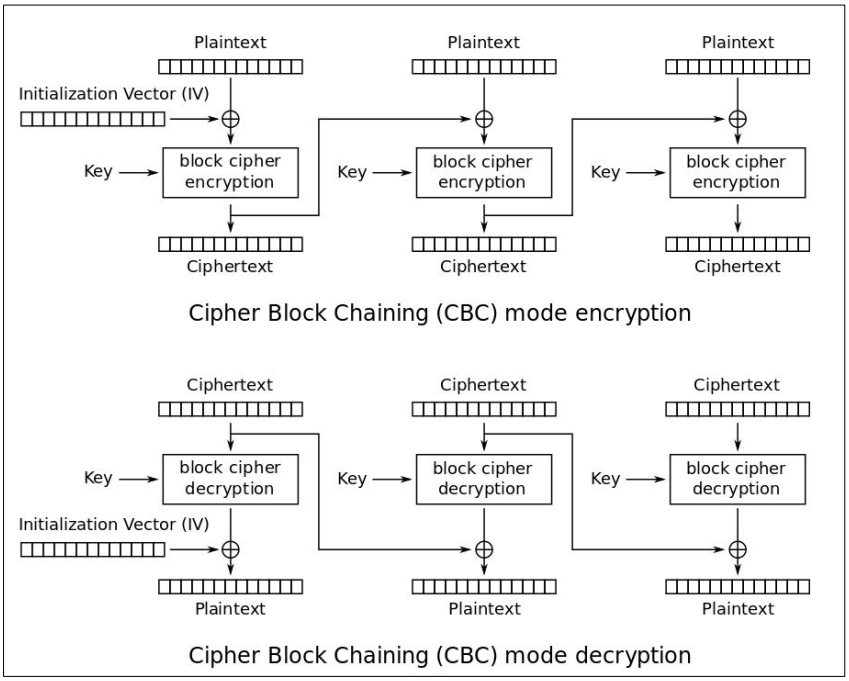
Διάνυσμα αρχικοποίησης (Initialization Vector- IV).

Επίσης όπως φαίνεται και από την Εικόνα 2 ο συγκεκριμένος τρόπος λειτουργίας χρησιμοποιεί και ένα διάνυσμα αρχικοποίησης.

Ο λόγος είναι ότι το πρώτο μήνυμα δεν έχει κάποιο προηγούμενο κρυπτόγραμμα να δεχτεί σαν είσοδο στη συνάρτηση XOR και για το λόγο αυτό μόνο στο πρώτο μήνυμα εφαρμόζεται το διάνυσμα αρχικοποίησης.

Το διάνυσμα αυτό έχει σταθερό μήκος, συνήθως ίδιο με το μήκος τμήματος του κάθε αλγόριθμου. Το διάνυσμα πρέπει να το γνωρίζουν τόσο ο πομπός όσο και ο δέκτης για τη σωστή κρυπτογράφηση αποκρυπτογράφηση του μηνύματος.

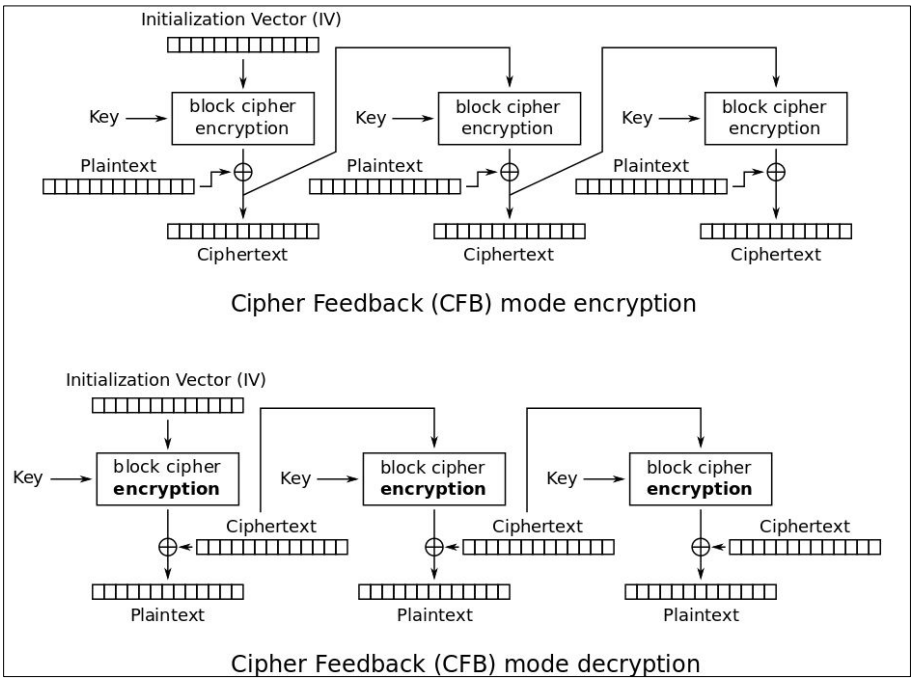
Το IV μεταδίδεται από τον έναν στον άλλον κρυπτογραφημένο με ECB, και εφαρμόζεται και συνάρτηση κατακερματισμού για έλεγχο της ακεραιότητάς του. Αν κάποιος εχθρός το μεταβάλλει κατά τη μετάδοσή του, ο παραλήπτης θα ανιχνεύσει τη μεταβολή αυτή. Για τον AES είναι ίδιο με το μήκος τμήματος, 16 bytes. Τέλος δεν είναι απαραίτητο να είναι κρυφό καθώς δεν προσδίδει κάποια πληροφορία για το κλειδί ή για το κρυπτόγραμμα.



Εικόνα 2: Λειτουργία Cipher block chaining-CBC

Cipher Feedback-CFB

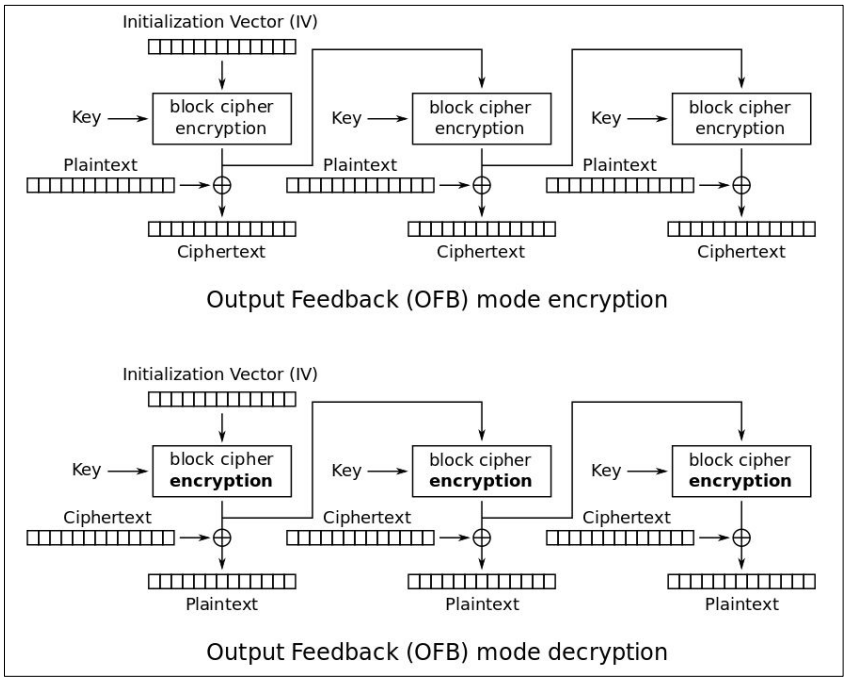
Η λειτουργία αυτή μοιάζει αρκετά με αυτή του CBC, και μετατρέπει έναν αλγόριθμο τμήματος σε έναν αυτό- συγχρονιζόμενο αλγόριθμο ροής. Για τη λειτουργία του απαιτείται διάνυσμα αρχικοποίησης. Η λειτουργία του φαίνεται στην παρακάτω εικόνα.



Εικόνα 3: Λειτουργία Cipher Feedback-CFB

Output Feedback (OFB)

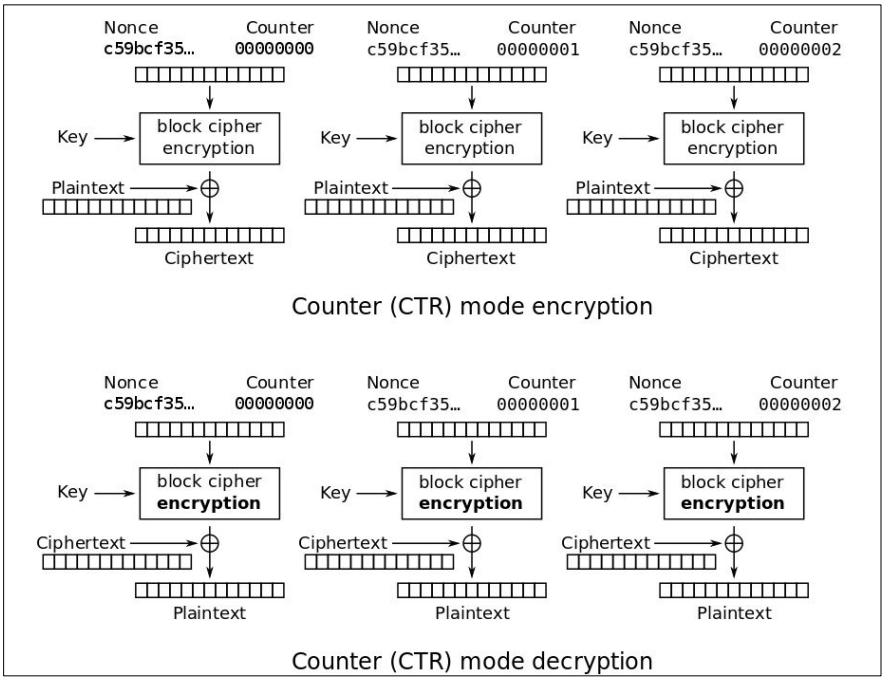
Η λειτουργία αυτή είναι παρόμοια με αυτή του CTR με τη μόνη διαφορά ότι η έξοδος της συνάρτησης κρυπτογράφησης χρησιμοποιείται στην ανάδραση και όχι το κρυπτόγραμμα. Για τη λειτουργία του απαιτείται διάνυσμα αρχικοποίησης. Η λειτουργία του φαίνεται στην παρακάτω εικόνα.



Εικόνα 4: Λειτουργία Output Feedback-OFB

Counter (CTR)

Χρησιμοποιείται μετρητής μήκους n, όπου n το μέγεθος του τμήματος του αλγόριθμου κρυπτογράφησης. Ο αλγόριθμος τμήματος κρυπτογραφεί κάθε φορά το περιεχόμενο του αθροιστή, και η έξοδός του γίνεται XOR με το τμήμα του μηνύματος. Ο μετρητής ξεκινά τη μέτρηση από μία τυχαία αρχική τιμή. Η λειτουργία του φαίνεται στην παρακάτω εικόνα.



Εικόνα 5: Λειτουργία Counter (CTR)

1.2 Κρυπτογράφηση βασισμένη σε κωδικό-PKCS#5

Σε αυτή την ενότητα αναλύεται η δημιουργία κλειδιού κρυπτογράφησης η οποία βασίζεται σε ένα μυστικό κωδικό τον οποίο παρέχει ο χρήστης.

Πιο συγκεκριμένα αναλύεται το πρότυπο PKCS#5 v2.0 (Public Key Cryptography Specification) το οποίο περιγράφει αναλυτικά τη δημιουργία του κλειδιού με τη χρήση κωδικού.[4]

Το πρότυπο αυτό παραθέτει συστάσεις για τη σωστή δημιουργία του κλειδιού με τη χρήση κωδικού. Για να γίνει αυτό εκτός τις σταθερές οι οποίες είναι το μήκος του κλειδιού το οποίο παρέχει ο εκάστοτε αλγόριθμος κρυπτογράφησης καθώς επίσης και ο κωδικός που εισάγεται κάθε φορά, το πρότυπο προσθέτει δύο επιπλέον παραμέτρους στη δημιουργία του κλειδιού οι οποίες είναι ο αριθμός των επαναλήψεων (iterations) και το «αλάτι» (salt). Οι δύο αυτές επιπλέον παράμετροι αναλύονται παρακάτω. [4][5]

Αλάτι: Το «αλάτι» είναι μια ακολουθία bytes τα οποία παράγονται τυχαία και προστίθενται στον κωδικό έτσι ώστε το κλειδί να μην δημιουργείται απευθείας από τον κωδικό τον ίδιο. Για κάθε κωδικό δημιουργείται ένα ξεχωριστό αλάτι το οποίο προστίθενται στο κωδικό.

Ο λόγος για τον οποίο γίνεται η προσθήκη του αλατιού στον κωδικό είναι η αποφυγή των "dictionary attacks" όπου ο επιτιθέμενος προσπαθεί να βρει τον κωδικό του χρήστη από μια συγκεκριμένη λίστα κωδικών.

Επίσης αποτρέπει την επίθεση "rainbow attack" κατά την οποία ο κωδικός αναζητείται σε κατακερματισμένη μορφή (hashed) από τον επιτιθέμενο. Με την προσθήκη κάθε φορά τυχαίου «αλατιού» το hash για κάθε κωδικό αλλάζει με αποτέλεσμα ο επιτιθέμενος να αναγκαστεί να δημιουργεί έναν καινούριο πίνακα από hashes για το κάθε νέο αλάτι που δημιουργείται.

Επίσης το «αλάτι» μπορεί είτε να είναι μια ακολουθία χαρακτήρων σταθερού μήκους, είτε κάθε φορά να δημιουργείται από μια ψευδο-τυχαία συνάρτηση. Το μήκος του «αλατιού» θα πρέπει να είναι μεγαλύτερο από 8 bytes.

Τέλος το «αλάτι» δεν χρειάζεται να είναι κρυφό καθώς δεν προσθέτει κάποια πληροφορία για τον κωδικό του χρήστη ή το κλειδί κρυπτογράφησης.

Επαναλήψεις: Ο αριθμός των επαναλήψεων αυξάνει το κόστος της δημιουργίας του κλειδιού, αλλά ταυτόχρονα αυξάνει τη δυσκολία της επίθεσης. Αυτό συμβαίνει γιατί όσο πιο πολλές είναι οι επαναλήψεις για τη δημιουργία του κλειδιού τόσες πιο πολλές δοκιμές θα χρειαστεί να κάνει ο επιτιθέμενος για την ανάκτηση του κλειδιού.

Το συγκεκριμένο πρότυπο συστήνει να γίνονται κατ' ελάχιστο **1000** επαναλήψεις για τη δημιουργία του κλειδιού.

Το πρότυπο αυτό καθορίζει δύο διαδικασίες για τη δημιουργία του κλειδιού, τις διαδικασίες PBKDF1, PBKDF2 (Password Based Key Derivation Function).

Η PBKDF1 χρησιμοποιεί για τη δημιουργία του κλειδιού μια συνάρτηση κατακερματισμού (MD2, MD5, SHA-1) και το μήκος του κλειδιού εξαρτάται από αυτή τη συνάρτηση. Πιο συγκεκριμένα για τις συναρτήσεις MD2, MD5 το κλειδί έχει μήκος 16 bytes, ενώ με τη χρήση της SHA-1 το κλειδί έχει μήκος 20 bytes.

Αντίθετα η PBKDF2 χρησιμοποιεί μια ψευδο-τυχαία συνάρτηση (pseudorandom function) για τη δημιουργία του κλειδιού και το μήκος του κλειδιού δεν έχει κάποιο περιορισμό.

Στο συγκεκριμένο πρότυπο η συνάρτηση που χρησιμοποιείται είναι η `HMAC-SHA-1 (Hash-based Message Authentication Code-HMAC)`. Η συγκεκριμένη συνάρτηση έχει σαν κύριο σκοπό την πιστοποίηση ενός κωδικού πιστοποίησης σε ένα μήνυμα (Message Authentication Code) . Σαν εισόδους δέχεται ένα κλειδί κρυπτογράφησης και το μήνυμα το οποίο πρόκειται να πιστοποιηθεί, και σαν έξοδο παράγει το MAC το οποίο έχει μήκος όσο η συνάρτηση κατακερματισμού η οποία χρησιμοποιείται.(20 bytes για SHA-1.)

Αντίστοιχα όταν εφαρμόζεται στην συνάρτηση PBKDF2 σαν εισόδους δέχεται τον κωδικό του χρήστη και το salt.

Η PBKDF2 για την παραγωγή του κλειδιού δέχεται σαν εισόδους τις εξής παραμέτρους:

1. Ψευδο-τυχαία συνάρτηση (PRF)
2. Κωδικό (P)
3. Αλάτι (S)
4. Αριθμό Επαναλήψεων (c)
5. Μήκος επιθυμητού κλειδιού κρυπτογράφησης. (dkLen, derived key length)

Και σαν έξοδο παράγει το κλειδί κρυπτογράφησης (DK- Derived Key).

Το hLen δηλώνει την έξοδο σε bytes το οποίο παράγει η ψευδο-τυχαία συνάρτηση που χρησιμοποιείται.

Στα επόμενα βήματα εξηγείται η δημιουργία του κλειδιού κρυπτογράφησης: [4]

1. Το μέγιστο μήκος του επιθυμητού κλειδιού δεν μπορεί να είναι μεγαλύτερο από $(2^{\wedge 32}-1)*hLen$.
2. Η παράμετρος l συμβολίζει τα τμήματα (blocks) σε bytes στρογγυλεμένα προς τον πρώτο μεγαλύτερο ακέραιο από την τιμή που λαμβάνει το l. Αυτό γίνεται με τη βοήθεια της συνάρτησης ceil όπως φαίνεται παρακάτω.

   `l = CEIL (dkLen / hLen)` (Μήκος Κλειδιού Κρυπτογράφησης/ Μήκος συνάρτησης κατακερματισμού)

   Η παράμετρος r συμβολίζει τον αριθμό των bytes τα οποία περιέχονται στο τελευταίο τμήμα:
   `r = dkLen - (l - 1) * hLen`
   Μήκος Κλειδιού Κρυπτογράφησης –(l-1)*Μήκος συνάρτησης κατακερματισμού.

3. Αφού έχουν προσδιοριστεί τα τμήματα από τα οποία αποτελείται το κλειδί, έπειτα για κάθε ένα από αυτά τα τμήματα (T_1, T_2…T_l), εφαρμόζεται η συνάρτηση F:

   `F(password, salt, iterations, block index)`

```
T_1 = F (P, S, c, 1)
T_2 = F (P, S, c, 2)
...
T_l = F (P, S, c, l)
```

Η συνάρτηση F είναι το άθροισμα της αποκλειστικής διάζευξης (XOR) των επαναλήψεων που εκτελείται η ψευδο-τυχαία συνάρτηση, στον κωδικό, και στη συνένωση του αλατιού με τον αριθμό του τμήματος.
Οι επαναλήψεις αυτές ονομάζονται U_1, U_2…U_c όπως φαίνεται στις παρακάτω σχέσεις:

```
F (P, S, c, i) = U_1 \xor U_2 \xor ... \xor U_c

U_1 = PRF (P, S || INT (i))
U_2 = PRF (P, U_1)
...
U_c = PRF (P, U_{c-1})
```

4. Αφού έχει υπολογιστεί η συνάρτηση F και έχει εφαρμοστεί σε όλα τα τμήματα του κλειδιού (Βήμα 3) το κλειδί που εξάγεται είναι η συνένωση των τμημάτων αυτών, δηλαδή:

```
DK = T_1 || T_2 || ... || T_l<0..r-1>
```

Από τα παραπάνω πρέπει να δοθεί ιδιαίτερη προσοχή στη σχέση:

```
T_1 = F (P, S, c, 1)
T_2 = F (P, S, c, 2)
...
T_l = F (P, S, c, l)
```

καθώς οι επαναλήψεις εξαρτώνται από τα τμήματα (`T_1`, `T_2`, `T_l`) τα οποία θα δημιουργηθούν αφού όπως φαίνεται από τις παραπάνω σχέσεις οι επαναλήψεις της συνάρτησης F θα εφαρμοστούν σε κάθε τμήμα του κλειδιού ξεχωριστά.

Για παράδειγμα για τα κλειδιά του AES 128/192/256 και για τη ψευδο-τυχαία συνάρτηση `HMAC-SHA-1` αν ο χρήστης επιλέξει να γίνουν 1000 επαναλήψεις, τότε θα γίνουν 1000 για το κλειδί των 128 bits, και 2000 για τα κλειδιά 192/256 bits.
Αυτό θα συμβεί γιατί για τα κλειδιά των κλειδιά 192/256 bits θα δημιουργηθούν 2 τμήματα T_1 και T_2 και θα εφαρμοστούν 1000 επαναλήψεις στο κάθε ένα από αυτά, ενώ

στο κλειδί των 128 bits (16 bytes) θα δημιουργηθεί ένα μόνο τμήμα, το T_1 στο οποίο θα εφαρμοστούν οι 1000 επαναλήψεις.

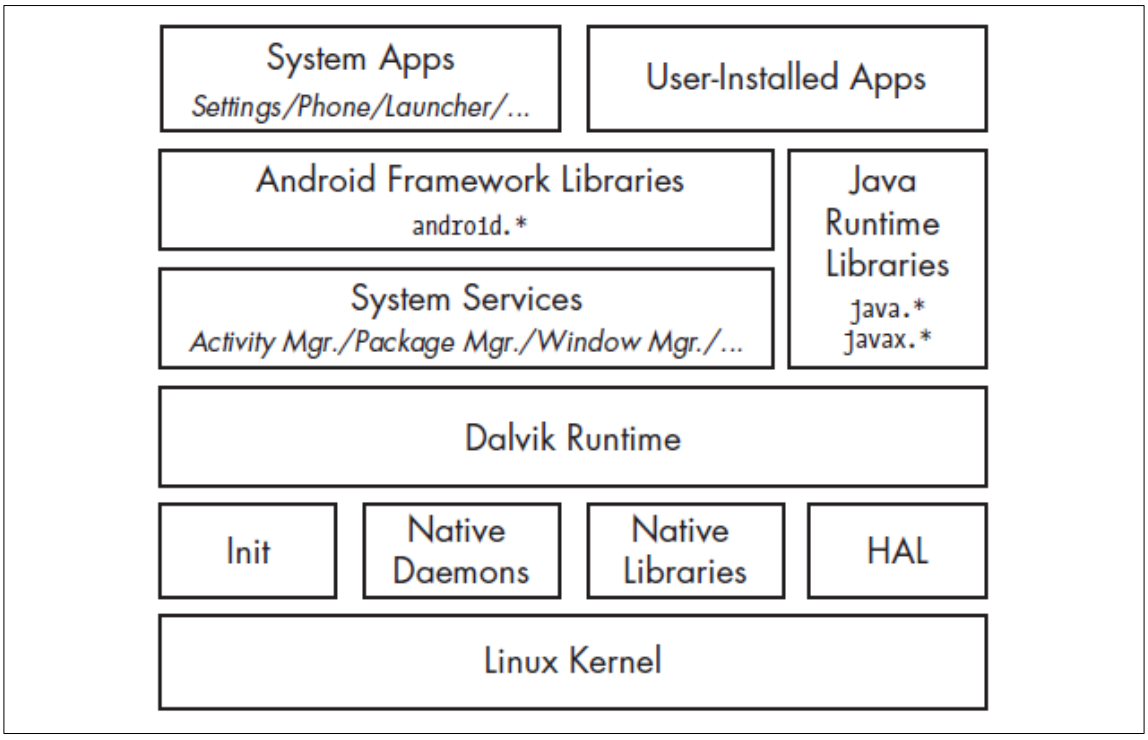
- 128 bits 16 bytes l =16/20=0.2 ceil(0.2)=1 r=16-(1-1)*20 r=16
  Δημιουργία ενός τμήματος T_1=16 bytes

- 192 bits 24 bytes l =24/20=1.2 ceil(1.2)=1 1 r=24-(2-1)*20 r=4
  Δημιουργία δύο τμημάτων T_1=20bytes και T_2=4 bytes

- 256 bits 32 bytes l =32/20=1.6 ceil(1.6)=2 1 r=32-(2-1)*20 r=12
  Δημιουργία δύο τμημάτων T_1=20bytes και T_2=12 bytes

1.3 Δομή του Λειτουργικού Συστήματος Android

Η δομή του λειτουργικού συστήματος Android παρουσιάζεται στο παρακάτω σχήμα:



Εικόνα 6: Αρχιτεκτονική του Android

Η ανάλυση της αρχιτεκτονικής του Android χωρίζεται σε τέσσερα ευρύτερα επίπεδα:, Τον πυρήνα (kernel), το εγγενές επίπεδο (Native Userspace), το επίπεδο εφαρμογών (Application Framework) και τέλος τις ίδιες τις εφαρμογές (Applications).

Ακολουθεί μια σύντομη περιγραφή των τεσσάρων αυτών επιπέδων ξεκινώντας από το χαμηλότερο προς το υψηλότερο:

Linux Kernel: Όπως φαίνεται και στην παραπάνω εικόνα το android είναι βασισμένο σε ένα Linux πυρήνα (kernel) ο οποίος παρέχει όλα τα απαραίτητα προγράμματα οδήγησης (drivers) για τον εξοπλισμό ο οποίος υπάρχει στο εκάστοτε κινητό τηλέφωνο, και είναι υπεύθυνος για την πρόσβαση σε φακέλους, τη σύνδεση στο διαδίκτυο, τη διαχείριση των εφαρμογών κ.α.

Native Userspace: Με τον όρο αυτό περιγράφονται όλα δομικά στοιχεία του android τα οποία δεν περιέχονται στο επίπεδο Dalvik Virtual Machine αλλά ούτε και στον πυρήνα του android. Αυτά είναι το δυαδικό αρχείο init το οποίο είναι και το πρώτο που ξεκινάει κατά την εκκίνηση του λειτουργικού συστήματος, και πάνω σε αυτό βασίζονται όλες οι υπόλοιπες διεργασίες.

Το στοιχείο **HAL** (Hardware Abstraction Layer) προσφέρει επιπλέον δυνατότητες εγκατάστασης των προγραμμάτων οδήγησης τα οποία χάρη σε αυτό δεν είναι απαραίτητο να διαχειρίζονται όλα από τον πυρήνα.

Επίσης κατά την εκκίνηση του λειτουργικού συστήματος φορτώνονται και κάποιες διεργασίες οι οποίες «τρέχουν» στο παρασκήνιο και ονομάζονται **daemons**, οι οποίες παραμένουν ενεργές όση ώρα είναι εν λειτουργία το android. Μια τέτοια διεργασία είναι το adbd (Android Debug Bridge daemon) το οποίο είναι υπεύθυνο για τη διασύνδεση του κινητού τηλεφώνου με κάποια άλλη συσκευή.

Σε αυτό το επίπεδο βρίσκονται και οι βασικές βιβλιοθήκες **(native libraries)** του συστήματος οι οποίες παρέχουν τις βασικές λειτουργικότητες όπως αυτή των γραφικών, του ήχου και άλλες.

Application Framework: Σε αυτό το επίπεδο γίνεται η εκτέλεση των προγραμμάτων του λειτουργικού συστήματος. Καθώς το μεγαλύτερο μέρος των εφαρμογών του android λειτουργούν με τη γλώσσα προγραμματισμού Java, για το λόγο αυτό χρησιμοποιείται για την εκτέλεσή τους μια «εικονική μηχανή» (Java Virtual Machine -JVM).

Στο android η μηχανή αυτή ονομάζεται **Dalvik**, και είναι ειδικά σχεδιασμένη για λειτουργία σε κινητά τηλέφωνα. Επίσης μπορεί να αλληλεπιδρά και με τα υπόλοιπα δομικά στοιχεία του android, συμπεριλαμβανομένων των βασικών βιβλιοθηκών και των δυαδικών αρχείων init.

Εκτός από μια εικονική μηχανή ένα πρόγραμμα υλοποιημένο σε Java χρειάζεται και τις απαραίτητες βιβλιοθήκες για να εκτελεστεί. Αυτές οι βιβλιοθήκες περιέχονται σε αυτό το επίπεδο και ονομάζονται **Java Runtime Libraries** (java.*, javax.*).

Το επίπεδο αυτό περιέχει και τις υπηρεσίες συστήματος (**System Services**) οι οποίες παρέχουν στο λειτουργικό σύστημα του android όλα τα βασικά χαρακτηριστικά του, όπως σύνδεση με το διαδίκτυο, τηλεφωνία, χρήση οθόνης αφής.

Τέλος σε αυτό το επίπεδο περιέχονται και βιβλιοθήκες οι οποίες δεν περιέχονται στις βιβλιοθήκες της Java Runtime Libraries, και περιέχουν τα βασικά στοιχεία για τη δημιουργία εφαρμογών android (android.*)

Applications: Στο ανώτερο αυτό επίπεδο της ιεραρχίας του android περιέχονται όλες οι εφαρμογές με τις οποίες αλληλεπιδρά ο χρήστης. Παρόλο που όλες οι εφαρμογές έχουν την ίδια δομή, διακρίνονται σε εφαρμογές συστήματος **(system-apps)** και εφαρμογές χρήστη **(user-installed applications).**

1.4 Αρχιτεκτονική Κρυπτογράφησης σε περιβάλλον Java

Σε αυτή την ενότητα αναλύονται τα κύρια στοιχεία τα οποία χρησιμοποιούνται σε περιβάλλον Java για την αποτελεσματική κρυπτογράφηση-αποκρυπτογράφηση.

Η αρχιτεκτονική κρυπτογράφησης σε περιβάλλον Java (Java Cryptography Architecture-JCA) παρέχει τις απαραίτητες διεπαφές προγραμματισμού εφαρμογών (Application Programming Interface-API) για τη δημιουργία ενός αλγορίθμου κρυπτογράφησης.

Μερικές από αυτές είναι για παράδειγμα, κρυπτογράφηση (συμμετρική και ασύμμετρη), ψηφιακές υπογραφές, δημιουργία κλειδιού κρυπτογράφησης και διαχείριση αυτού, επαλήθευση πιστοποιητικών κ.α.

Η αρχιτεκτονική αυτή σχεδιάστηκε με γνώμονα τις παραμέτρους, Ανεξαρτησία Υλοποίησης, Διαλειτουργικότητα, και Επεκτασιμότητα.

Οι τρεις αυτές παράμετροι αναλύονται παρακάτω.

Ανεξαρτησία υλοποίησης: Οι εφαρμογές δεν χρειάζονται να υλοποιούν οι ίδιες τους αλγόριθμους κρυπτογράφησης, αλλά ζητούν την εφαρμογή αυτών από τις υπηρεσίες οι οποίες παρέχονται από τη Java.

Η ανεξαρτησία αυτή επιτυγχάνεται με τη χρήση ‘‘παρόχων’’ στην αρχιτεκτονική της Java, ή διαφορετικά Παρόχων Κρυπτογραφικών Υπηρεσιών (Cryptographic Service Provider

–CSP). Ο όρος πάροχος αναφέρεται σε ένα πακέτο (java package) υπηρεσιών το οποίο προσφέρει υπηρεσίες κρυπτογράφησης. Η κλάση (.class) της java η οποία περιλαμβάνει αυτούς τους παρόχους είναι η `java.security.Provider`.

Ένας άλλος τρόπος για την επίτευξη της ανεξαρτησίας υλοποίησης είναι ο ορισμός των κρυπτογραφικών υπηρεσιών οι οποίες αποκαλούνται "engines" στη Java, και η δημιουργία αντίστοιχων κλάσεων οι οποίες υποστηρίζουν τις υπηρεσίες αυτές. Οι κλάσεις αυτές ονομάζονται engine classes. Οι κλάσεις αυτές παρέχουν την διεπαφή για συγκεκριμένου είδους κρυπτογραφικές υπηρεσίες, οι οποίες όμως είναι ανεξάρτητες από κάποιον συγκεκριμένο πάροχο, ή κάποιο αλγόριθμο κρυπτογράφησης.

Οι κλάσεις αυτές είναι σε θέση να παρέχουν τα εξής:

1. Κρυπτογραφικές Υπηρεσίες όπως κρυπτογράφηση, ψηφιακές υπογραφές, συναρτήσεις κατατεμαχισμού κ.α.
2. Γεννήτριες (generators), ή μετατροπείς κρυπτογραφικού υλικού όπως είναι τα κλειδιά κρυπτογράφησης ή το διάνυσμα αρχικοποίησης.
3. Αντικείμενα (java Objects) τα οποία ενθυλακώνουν κρυπτογραφικά δεδομένα όπως ψηφιακά πιστοποιητικά, ή δεδομένα τα οποία περιέχουν κλειδιά κρυπτογράφησης (keystore) ή άλλα πιστοποιητικά ασφαλείας.

Οι κλάσεις οι οποίες είναι διαθέσιμες στην αρχιτεκτονική κρυπτογράφησης της Java είναι οι εξής[5]:

`SecureRandom`: Χρησιμοποιείται για την παραγωγή τυχαίων ή ψευδο-τυχαίων αριθμών.

`MessageDigest`: Χρησιμοποιείται για τον κατακερματισμό ενός μηνύματος.

`Signature`: Αρχικοποιείται με κλειδί κρυπτογράφησης και προορίζεται για την υπογραφή δεδομένων και την πιστοποίηση ψηφιακών υπογραφών.

`Cipher`: Αρχικοποιείται με κλειδί κρυπτογράφησης και προορίζεται για την κρυπτογράφηση-αποκρυπτογράφηση δεδομένων. Υποστηρίζει συμμετρικούς και ασύμμετρους αλγόριθμους κρυπτογράφησης, καθώς επίσης και αλγόριθμους ροής (stream ciphers) , αλγόριθμους δέσμης (block ciphers), και κρυπτογράφηση με βάση κωδικό (PBE encryption)

`Message Authentication Codes (MAC)`: Χρησιμοποιείται όπως και οι συναρτήσεις κατακερματισμού, αλλά πρώτα αρχικοποιείται με κλειδί κρυπτογράφησης για την προστασία της ακεραιότητας του μηνύματος.

`KeyFactory`: Η κλάση αυτή χρησιμοποιείται για τη δημιουργία κλειδιών κρυπτογράφησης με συγκεκριμένα χαρακτηριστικά, ή το αντίστροφο, την ανάκτηση των χαρακτηριστικών ενός κλειδιού κρυπτογράφησης.

SecretKeyFactory: Η διαφορά της κλάσης αυτής με την KeyFactory είναι ότι λειτουργεί μόνο για συμμετρικά κλειδιά κρυπτογράφησης.

KeyPairGenerator: Χρησιμοποιείται για τη δημιουργία ενός ζεύγους δημόσιου και ιδιωτικού κλειδιού κατάλληλο για χρήση με ένα συγκεκριμένο αλγόριθμο κρυπτογράφησης.

KeyGenerator: Χρησιμοποιείται για τη δημιουργία κλειδιού κρυπτογράφησης με συγκεκριμένο αλγόριθμο.

KeyAgreement: Χρησιμοποιείται από δύο η περισσότερα μέρη για τη συμφωνία ενός κλειδιού κρυπτογράφησης για χρήση για μια συγκεκριμένη κρυπτογραφική διεργασία.

AlgorithmParameters: Χρησιμοποιείται για την αποθήκευση των παραμέτρων του αλγόριθμου κρυπτογράφησης.

AlgorithmParameterGenerator: Παράγει ένα σύνολο παραμέτρων οι οποίες είναι κατάλληλες για τον συγκεκριμένο αλγόριθμο που επιλέγεται.

KeyStore: Χρησιμοποιείται για τη δημιουργία και τη διαχείριση μιας βάσης δεδομένων κρυπτογραφικών κλειδιών (keystore).

CertificateFactory: Χρησιμοποιείται για τη δημιουργία ψηφιακών πιστοποιητικών.

CertPathBuilder: Χρησιμοποιείται για τη δημιουργία μιας αλυσίδας ψηφιακών πιστοποιητικών.

CertPathValidator: Χρησιμοποιείται για την πιστοποίηση μιας αλυσίδας ψηφιακών πιστοποιητικών.

CertStore: Χρησιμοποιείται για την ανάκτηση ψηφιακών πιστοποιητικών και μιας αλυσίδας ψηφιακών πιστοποιητικών από τον χώρο που είναι αποθηκευμένα.

Τα πακέτα (packages) που περιέχουν τις παραπάνω και όχι μόνο κλάσεις στην αρχιτεκτονική της Java είναι τα εξής:

1. `java.security`
2. `javax.crypto`
3. `java.security.cert`
4. `java.security.spec`
5. `javax.crypto.spec`
6. `java.security.interfaces`
7. `javax.crypto.interfaces`

Από τα παραπάνω το `javax.crypto` περιλαμβάνει τις κύριες κλάσεις οι οποίες χρειάζονται για την κρυπτογράφηση και είναι οι: `Cipher`, `KeyGenerator`,

`SecretKeyFactory` καθώς και η διεπαφή (interface) για το κλειδί κρυπτογράφησης, `SecretKey`.[7]

Αντίστοιχα στο πακέτο `java.security` περιέχονται οι κλάσεις `AlgorithmParameters`, `MessageDigest` και `SecureRandom` οι οποίες χρησιμοποιούνται και αυτές σε λιγότερο βαθμό για τη δημιουργία ενός κλειδιού.[8]

Τέλος στο πακέτο javax.crypto.spec περιέχονται οι κλάσεις οι οποίες παρέχουν τα χαρακτηριστικά του κλειδιού ή τα χαρακτηριστικά του αλγόριθμου κρυπτογράφησης. Ενδεικτικά αναφέρονται οι εξής κλάσεις:

1. `SecretKeySpec`: Η κλάση αυτή καθορίζει ένα κλειδί κρυπτογράφησης, ανεξάρτητα από την επιλογή παρόχου. Επίσης μπορεί να χρησιμοποιηθεί για τη δημιουργία κλειδιού απευθείας από έναν πίνακα bytes χωρίς τη χρήση της class `SecretKeyFactory`.
2. `PBEKeySpec`: Η κλάση αυτή καθορίζει όλες τις απαραίτητες παραμέτρους οι οποίες πρέπει να δοθούν για να μπορέσει να γίνει σωστά η κρυπτογράφηση με τη χρήση κωδικού.
3. `IvParameterSpec`: Η κλάση αυτή καθορίζει ένα διάνυσμα αρχικοποίησης.

Μια εφαρμογή μπορεί να επιλέξει τη χρήση μιας υπηρεσίας κρυπτογράφησης όπως για παράδειγμα τη δημιουργία μιας ψηφιακής υπογραφής, και αυτή να υλοποιηθεί με έναν από τους διαθέσιμους παρόχους, χωρίς η εφαρμογή να χρειαστεί να ορίσει κάποιον συγκεκριμένο.

Παρόλαυτα δίνεται η δυνατότητα στην εφαρμογή να ορίσει συγκεκριμένο πάροχο με τον οποίο θα γίνει η δημιουργία της ψηφιακής υπογραφής και κατ' επέκταση οποιασδήποτε υπηρεσίας κρυπτογράφησης.

Διαλειτουργικότητα: Μια εφαρμογή μπορεί να επιλέξει την κρυπτογράφηση από διαφορετικούς παρόχους, και το αποτέλεσμα να παραμείνει το ίδιο σε διαφορετικές συσκευές. Επίσης όλοι οι πάροχοι είναι διαθέσιμοι σε όλες τις συσκευές.

Έστω ότι μια εφαρμογή επιλέξει να χρησιμοποιήσει τη συνάρτηση κατατεμαχισμού (hash function) MD5 εκτελώντας τις εντολές:

```
md = MessageDigest.getInstance("MD5");
md = MessageDigest.getInstance("MD5", "ProviderC");
```

Με την πρώτη εντολή δεν επιλέγει κάποιο συγκεκριμένο πάροχο, ενώ με τη δεύτερη επιλέγει να εκτελεστεί η εντολή κατατεμαχισμού με τον πάροχο ProviderC.

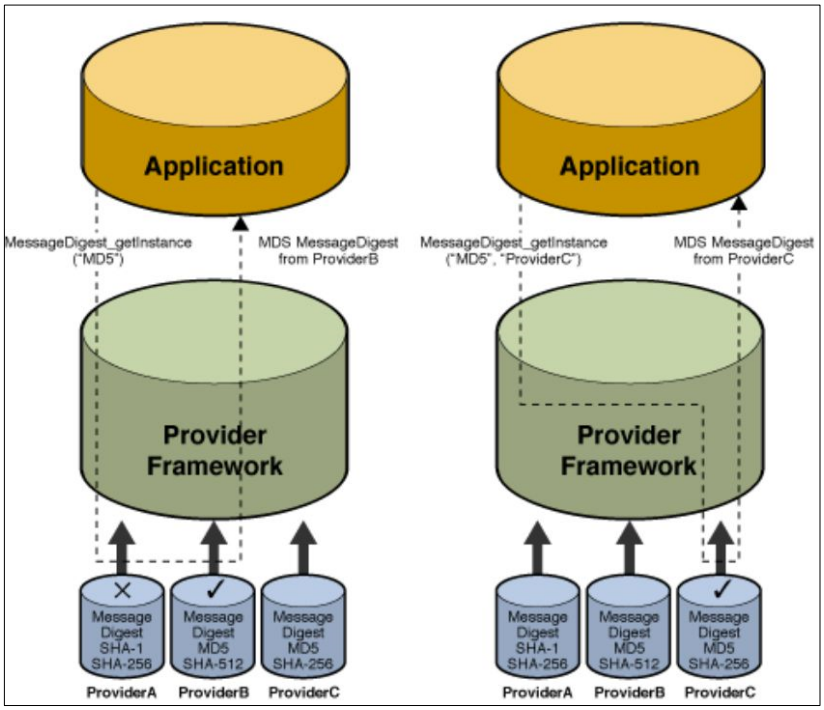
Αν υπάρχουν τρεις διαθέσιμοι πάροχοι για υπηρεσίες κρυπτογράφησης, αυτοί ταξινομούνται με σειρά προτίμησης από τα αριστερά προς τα δεξιά (1-3).

Στην Εικόνα 7 παρατηρούμε ότι με τη πρώτη εντολή όπου δεν επιλέγεται κάποιος συγκεκριμένος πάροχος επιλέγεται τελικά να εκτελέσει αυτή την εντολή ο πάροχος “ProviderB” ο οποίος είναι ο πρώτος σε σειρά προτεραιότητας ο οποίος μπορεί να εκτελέσει με επιτυχία την εντολή αυτή, καθώς ο πάροχος “ProviderA” ναι μεν είναι πρώτος σε σειρά προτίμησης αλλά δεν υποστηρίζει τον αλγόριθμο MD5.

Αντίστοιχα για την δεύτερη εντολή όπου καλείται από το πρόγραμμα να εκτελεστεί η εντολή με τον πάροχο “ProviderC”, αυτό γίνεται άμεσα αν ο συγκεκριμένος πάροχος υποστηρίζει τον συγκεκριμένο αλγόριθμο κρυπτογράφησης μη λαμβάνοντας υπόψη σε ποια σειρά προτίμησης βρίσκεται ο εν λόγω πάροχος.

Στην παρακάτω εικόνα βλέπουμε ότι επιλέγεται ο πάροχος “ProviderC” ο οποίος υποστηρίζει τον MD5 αν και βρίσκεται στη θέση 3 της σειράς προτίμησης και ο “ProviderB” βρίσκεται στη θέση 2 και υποστηρίζει και αυτός τον MD5.

Η διαλειτουργικότητα έγκειται στο γεγονός ότι το αποτέλεσμα της συνάρτησης κατατεμαχισμού MD5 θα είναι το ίδιο είτε εκτελεστεί από τον πάροχο ProviderB είτε από τον πάροχο ProviderC.



Εικόνα 7: Επιλογή παρόχου από την εφαρμογή.

Οι πάροχοι που είναι ενσωματωμένοι στην αρχιτεκτονική της Java είναι οι εξής[6]:

SunPKCS11: Δεν παρέχει κρυπτογραφικές υπηρεσίες, αλλά προσφέρει μια διεπαφή με τους άλλους παρόχους για τυχόν εφαρμογές οι οποίες χρησιμοποιούν το πρότυπο PKCS#11.

SUN: Είναι ο πρώτος πάροχος ο οποίος ενσωματώθηκε στην αρχιτεκτονική της Java και παρέχει κρυπτογραφικές υπηρεσίες.

SunRsaSign: Προσφέρει καλύτερες κρυπτογραφικές υπηρεσίες για ψηφιακές υπογραφές με τον αλγόριθμο RSA συγκριτικά με τον πάροχο JSSE.

SunJSSE: Παρέχει κρυπτογραφικές υπηρεσίες, κυρίως όσον αφορά τη κρυπτογράφηση δεδομένων τα οποία μεταφέρονται στο διαδίκτυο.

SunJCE: Είναι ο πάροχος με τους πλέον διαδεδομένους αλγόριθμους κρυπτογράφησης και ο πιο συχνά χρησιμοποιούμενος σε Java εφαρμογές.

SunJGSS: Δεν παρέχει κρυπτογραφικές υπηρεσίες αλλά τα πρωτόκολλα Kerberos v5 και SPNEGO.

SunSASL: Δεν παρέχει κρυπτογραφικές υπηρεσίες, αλλά αλγόριθμους για το πρωτόκολλο SASL (Simple Authentication and Security Layer).

XMLDSig: Δεν παρέχει κρυπτογραφικές υπηρεσίες.

SunPCSC: Δεν παρέχει κρυπτογραφικές υπηρεσίες.

SunMSCAPI: Δεν παρέχει κρυπτογραφικές υπηρεσίες, αλλά παρέχει τη διεπαφή για τις εφαρμογές ώστε αυτές να αποκτήσουν πρόσβαση σε εγγενείς βιβλιοθήκες, βάσεις δεδομένων κλειδιών, και ψηφιακά πιστοποιητικά τα οποία βρίσκονται αποθηκευμένα στο λειτουργικό σύστημα των Windows.

SunEC: Παρέχει αλγόριθμους για Κρυπτογραφία Ελλειπτικών Καμπυλών. (Elliptical Curve Cryptography).

OracleUcrypto: Δεν παρέχει κρυπτογραφικές υπηρεσίες, αλλά τη διεπαφή μεταξύ της Java και της βιβλιοθήκης Ucrypto η οποία βρίσκεται στα λειτουργικά συστήματα Solaris.

Επεκτασιμότητα: Η Java περιλαμβάνει έναν συγκεκριμένο αριθμό παρόχων για υπηρεσίες κρυπτογράφησης. Αν όμως μια εφαρμογή επιθυμεί την εκτέλεση κάποιου εξειδικευμένου αλγορίθμου κρυπτογράφησης ο οποίος δεν περιλαμβάνεται σε αυτούς που βρίσκονται ενσωματωμένοι στη Java, υπάρχει η επιλογή να προστεθεί ο συγκεκριμένος πάροχος ο οποίος υλοποιεί τον συγκεκριμένο αλγόριθμο.

ΚΕΦΑΛΑΙΟ 2

ΑΝΑΛΥΣΗ ΕΦΑΡΜΟΓΗΣ AES ANDROID FORENSICS

Στο κεφάλαιο αυτό γίνεται ανάλυση της εφαρμογής AES Android Forensics. Εξετάζεται ο τρόπος δημιουργίας της, ο τρόπος λειτουργίας της καθώς και τα ευρήματα που προκύπτουν από την εξέταση των αποτυπωμάτων μνήμης της.

Η εφαρμογή αυτή δημιουργήθηκε ώστε να προσομοιάσει σε όσο το δυνατόν μεγαλύτερο βαθμό παρόμοιες android εφαρμογές οι οποίες χρησιμοποιούν κρυπτογράφηση AES με βάση τη χρήση κωδικού για την κρυπτογράφηση ενός κειμένου του χρήστη.

2.1 Δημιουργία της εφαρμογής AES Android Forensics

Η δημιουργία της συγκεκριμένης εφαρμογής έγινε με τη βοήθεια του Android Software Development Kit (SDK) το οποίο αποτελεί μια ολοκληρωμένη πλατφόρμα εργαλείων τα οποία χρειάζονται για την ανάπτυξη μιας εφαρμογής android.

Τα πιο σημαντικά από αυτά τα εργαλεία είναι ο προσομοιωτής (emulator), το πρόγραμμα εντοπισμού σφαλμάτων (debugger), και οι απαραίτητες βιβλιοθήκες έτσι ώστε να μπορέσει να εκτελεστεί το android στον προσομοιωτή.

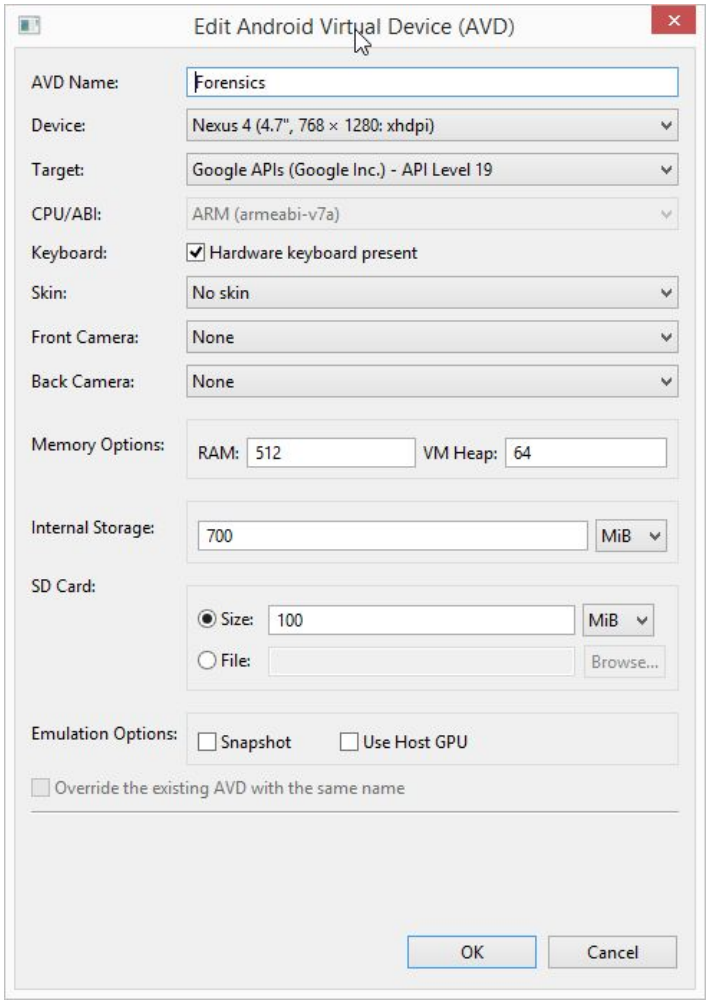
Επίσης παράλληλα με την παραπάνω πλατφόρμα χρησιμοποιήθηκε και το πρόγραμμα Eclipse Juno το οποίο είχε ενσωματωμένα τα εργαλεία για την ανάπτυξη εφαρμογών android (Android Development Tools-ADT).

Τέλος για την λειτουργία και τον έλεγχο σφαλμάτων της εφαρμογής μας δημιουργήθηκε μια εικονική συσκευή android (Android Virtual Device – AVD) με τη βοήθεια του προσομοιωτή. Η ονομασία της εικονικής αυτής συσκευή είναι: “Forensics”.

Όπως φαίνεται και από την Εικόνα 2.1 η συσκευή η οποία επιλέχτηκε είναι η LG Nexus 4, η οποία λειτουργεί με την έκδοση 4.4 Kitkat του λογισμικού Android. Επιλέχθηκε η συγκεκριμένη έκδοση για την προσομοίωση καθώς αποτελεί την πιο δημοφιλή έκδοση του λογισμικού Android.[9]

Επίσης για την CPU επιλέχθηκε η αρχιτεκτονική ARM η οποία είναι και η πιο διαδομένη αρχιτεκτονική στα κινητά τηλέφωνα. [10]

Τέλος επιλέχθηκε για εσωτερική μνήμη η τιμή των 700MB, για μνήμη RAM η τιμή 512MB και για την SD Card η τιμή των 100MB.



Εικόνα 2.1: Χαρακτηριστικά εικονικής συσκευής "Forensics"

Η εφαρμογή αποτελείται από πέντε οθόνες (Activities) με τις οποίες μπορεί να αλληλεπιδράσει ο χρήστης, καθώς επίσης και από δύο κλάσεις (class) της Java, η μία εκ των οποίων περιέχει όλες τις βοηθητικές μεθόδους τις οποίες είναι σε θέση να χρησιμοποιήσουν όλες οι activities, και άλλη μια η οποία ονομάζεται Class_AES_Cipher η οποία περιέχει την υλοποίηση του αλγόριθμου AES.

Η συνοπτική λειτουργία των activities και classes οι οποίες αναφέρθηκαν περιλαμβάνεται στον παρακάτω πίνακα:

Activity/Class	Λειτουργία
`Main.java`	Η αρχική οθόνη που βλέπει ο χρήστης.
`Activity_Encryption_View.java`	Σε αυτή την οθόνη εμφανίζονται το κρυπτόγραμμα και το κλειδί κρυπτογράφησης.
`Activity_Decryption_View.java`	Σε αυτή την οθόνη εμφανίζονται το αποκρυπτογραφημένο κείμενο και το κλειδί αποκρυπτογράφησης.
`Activity_Options_View.java`	Οθόνη με επιλογές για αντιγραφή, επικόλληση, διαγραφή κειμένου, και επιστροφή στην αρχική οθόνη.
`Activity_Information_View.java`	Οθόνη με πληροφορίες για την εφαρμογή.
`Class_Methods.java`	Περιέχει βοηθητικές μεθόδους για την εκτέλεση της εφαρμογής
`Class_AES_Cipher.java`	Περιέχει την υλοποίηση της κρυπτογράφησης του AES.

Πίνακας 2.1: Classes and Activities της εφαρμογής AES Android Forensics

2.1.1 Class_AES_Cipher.java

Από τις παραπάνω αναλύεται η `Class_AES_Cipher.java` καθώς σε αυτή γίνεται η κρυπτογράφηση και αποκρυπτογράφηση του κειμένου το οποίο δίνει ο χρήστης σε συνδυασμό με κάποιο μυστικό κωδικό.

Οι παράμετροι που χρησιμοποιήθηκαν για τη δημιουργία της κλάσης αυτής φαίνονται στον παρακάτω πίνακα:

Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογής AES Android Forensics	
Αλγόριθμος Κρυπτογράφησης	AES
Μήκος Κλειδιού	256/192/128 bits
Συνάρτηση Δημιουργίας κλειδιού (Key Derivation Function-KDF)	PBKDF2WithHmacSHA1
Τρόπος Λειτουργίας	Cipher Block Chaining (CBC)
Διάνυσμα Αρχικοποίησης (IV)	randmstringforiv
«Αλάτι» (salt)	The salt is only 32 bytes length
Αριθμός Επαναλήψεων (Iterations)	1000

Κωδικοποίηση Κρυπτογράμματος-Κλειδιού	Base-64
Java Classes	javax.crypto.SecretKeyFactory javax.crypto.PBEKeySpec javax.crypto.SecretKey javax.crypto.SecretKeySpec javax.crypto.Cipher javax.crypto.IvParameterSpec

Πίνακας 2.2: Παράμετροι αλγόριθμου AES για το πρόγραμμα AES Android Forensics

Ο κώδικας της κλάσης Class_AES_Cipher.java είναι ο εξής:

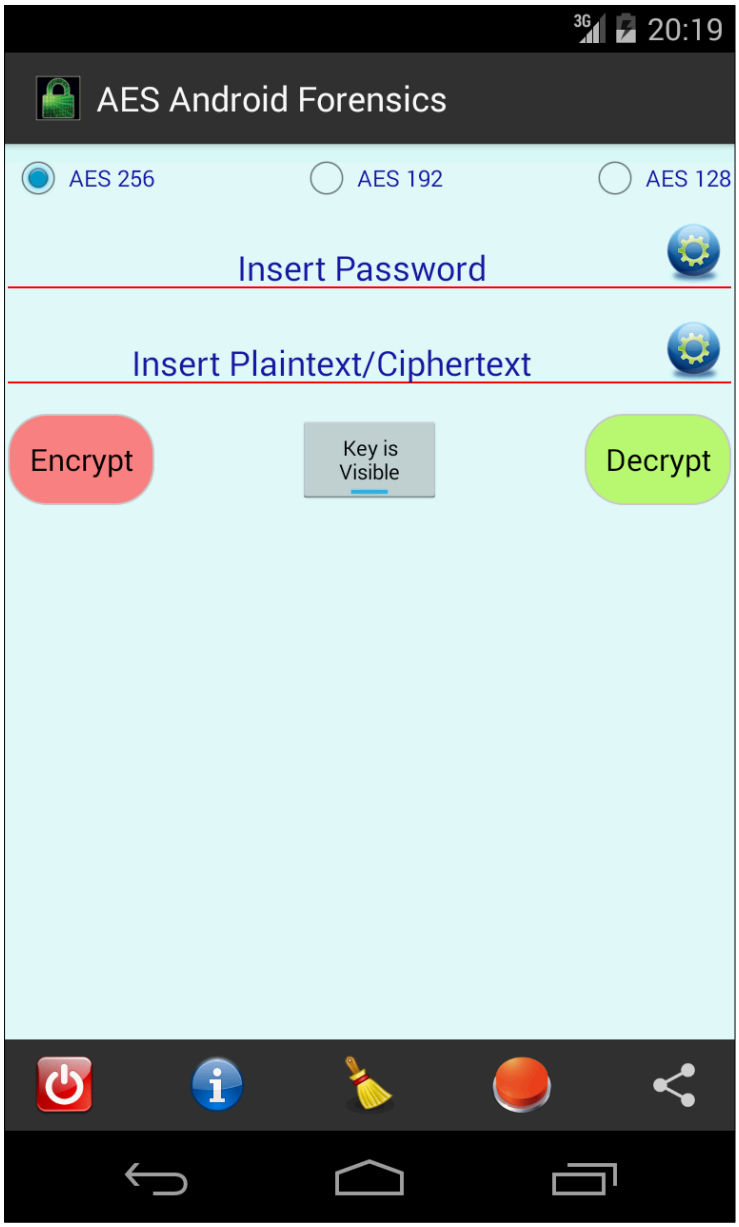
```
package com.unipi.android_forensics;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

import android.os.Build;
import android.util.Base64;



@SuppressWarnings("static-access")
public class Class_AES_Cipher {

Main main = new Main();

Class_Methods methods = new Class_Methods();

public  String encrypt_decrypt (String data_text, String aes_mode) {

          try {

// Αρχικοποίηση Διανύσματος Αρχικοποίησης και «Αλατιού»
byte[] salt_Bytes = main.Salt.getBytes("UTF-8");
IvParameterSpec  iv_specs = new
IvParameterSpec(Main.Init_Vector.getBytes("UTF-8"));

if (main.AES_Key_Length==128)
main.Password_Iterations=1000;
else main.Password_Iterations=500;

// Καταχώρηση των παραμέτρων για κρυπτογράφηση με χρήση μυστικού
κωδικού. (password based encryption)
PBEKeySpec PBE_specs = new PBEKeySpec(
Main.user_password_string.toCharArray(),
salt_Bytes,
main.Password_Iterations,
Main.AES_Key_Length
      );
```

```
// Επιλογή SecretKey Factory ανάλογα με την έκδοση του Android.[11]
SecretKeyFactory factory;
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
//Ο αλγόριθμος PBKDF2WithHmacSHA1And8bit επιλέγεται αν η έκδοση του
Android είναι μεγαλύτερη από την 4.4 και χρησιμοποιεί τα 8 λιγότερα
σημαντικά bits (Least Significant Bits-LSB) ενός χαρακτήρα Unicode.

factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1And8bit");
   }
else {
// Για εκδόσεις του Android μικρότερες της 4.4 επιλέγεται ο αλγόριθμος
PBKDF2WithHmacSHA1, ο οποίος για τις συγκεκριμένες εκδόσεις
χρησιμοποιεί εκ' κατασκευής τα 8 LSB.
// Για εκδόσεις μεγαλύτερες του 4.4 ο αλγόριθμος
PBKDF2WithHmacSHA1χρησιμοποιεί όλα τα bits ενός Unicode χαρακτήρα.

factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
}

// Δημιουργία κλειδιού Κρυπτογράφησης-Αποκρυπτογράφησης.
   SecretKey PBE_Key = factory.generateSecret(PBE_specs);
   SecretKey AES_Key = new SecretKeySpec(PBE_Key.getEncoded(), "AES");
   Main.AES_Key = Base64.encodeToString(AES_Key.getEncoded(),
Base64.DEFAULT);

Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");

//Παραγωγή κρυπτογράμματος-Λειτουργία κρυπτογράφησης
if (aes_mode.matches("Encryption_Mode")){

cipher.init(Cipher.ENCRYPT_MODE, AES_Key, iv_specs);
byte[] ciphertext_bytes = cipher.doFinal(data_text.getBytes("UTF-8"));
String ciphertext = new
String(Base64.encodeToString(ciphertext_bytes,Base64.DEFAULT));
return ciphertext;
      }

//Παραγωγή αποκρυπτογραφημένου κειμένου- Λειτουργία Αποκρυπτογράφησης
else if (aes_mode.matches("Decryption_Mode")){

cipher.init(Cipher.DECRYPT_MODE, AES_Key, iv_specs);
byte[] ciphertext_bytes =  Base64.decode(data_text, Base64.DEFAULT);
byte[] plaintext_bytes = cipher.doFinal(ciphertext_bytes);

String plaintext = new String(plaintext_bytes);
return plaintext;
  }

} catch (BadPaddingException ex) {
methods.warning_message("Correct Symbols and Length.\nWrong Password,
or Key length",1);
ex.printStackTrace();
   }
catch (Exception ex) {
ex.printStackTrace();
    }
return null;
      }

}// End Class_AES_Cipher
```

2.2 Λειτουργία της Εφαρμογής AES Android Forensics

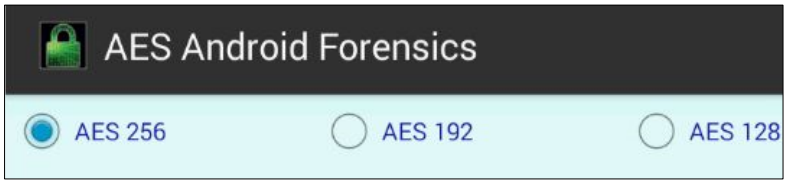
Σε αυτή την ενότητα παρουσιάζεται συνοπτικά η λειτουργία της εφαρμογής, τόσο για τη λειτουργία κρυπτογράφησης όσο και για τη λειτουργία αποκρυπτογράφησης.

Κατά την εκκίνηση της εφαρμογής εμφανίζεται στο χρήστη η κύρια οθόνη (`Main.java`) η οποία φαίνεται στην παρακάτω εικόνα:



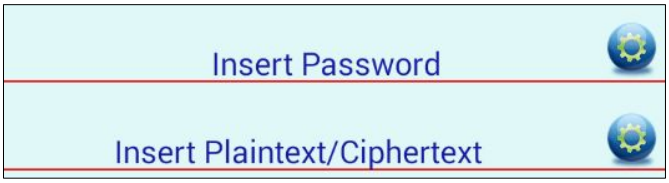
Εικόνα 2.2: Αρχική οθόνη της εφαρμογής AES Android Forensics

Ο χρήστης επιλέγει το μήκος του κλειδιού του AES από τις τρεις επιλογές `(RadioButtons)` οι οποίες βρίσκονται στο πάνω μέρος της οθόνης, και κάθε φορά που επιλέγεται διαφορετικό μήκος εμφανίζεται ένα ενημερωτικό μήνυμα στο κέντρο της οθόνης το οποίο ειδοποιεί το χρήστη για την επιλογή του. `(Toast Message)`



Εικόνα 2.3 Επιλογή μήκους κλειδιού AES

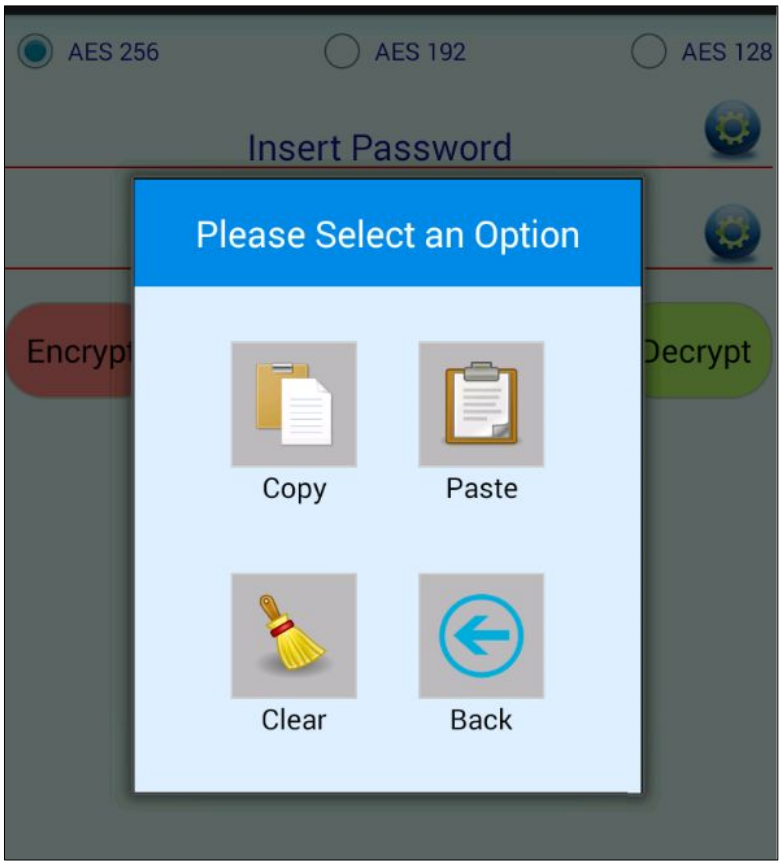
Στην οθόνη αυτή υπάρχουν δύο πεδία τα οποία προτρέπουν το χρήστη να δώσει τον κωδικό (Insert Password), καθώς επίσης και το κείμενο που επιθυμεί να κρυπτογραφήσει/αποκρυπτογραφήσει (Insert Plaintext/Ciphertext).



Εικόνα 2.4: Εισαγωγή κωδικού και απλού Κειμένου ή Κρυπτογράμματος

Δίπλα σε κάθε πεδίο υπάρχει ένα κουμπί το οποίο άμα επιλεγεί εμφανίζει στο χρήση επιπλέον επιλογές για την επεξεργασία κειμένου .`(Activity_Options_View.java)`.

Οι επιλογές που προσφέρονται στο χρήστη είναι η αντιγραφή (Copy), επικόλληση (Paste), και διαγραφή (Clear) του κειμένου το οποίο επιθυμεί, καθώς επίσης και η επιστροφή στην αρχική οθόνη (Back). Σε όλες τις επιλογές πλην της επιστροφής στην αρχική οθόνη εμφανίζεται ένα ενημερωτικό μήνυμα.



Εικόνα 2.5: Επιλογές για επεξεργασία κειμένου.

Ο χρήστης είναι σε θέση να αφήσει κενό το πεδίο “Insert Password” και να προχωρήσει στην κρυπτο-αποκρυπτογράφηση, καθώς σε αυτή την περίπτωση χρησιμοποιείται από την εφαρμογή ο προκαθορισμένος κωδικός

```
Default_Password =”password_by_default_if_no_input_ginven_from_user"
```

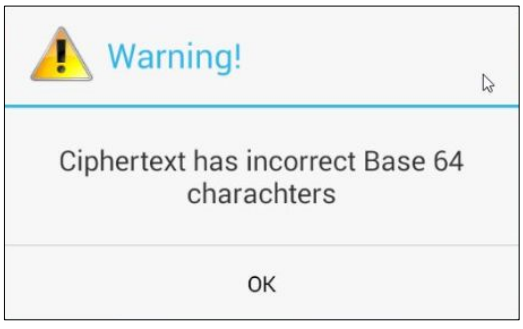
Το πεδίο “Insert Plaintext/Ciphertext” όμως δεν μπορεί να είναι κενό, και αν υπάρξει μια τέτοια περίπτωση, εμφανίζονται τα αντίστοιχα μηνύματα τα οποία ενημερώνουν το χρήστη να εισάγει είτε απλό κείμενο για την κρυπτογράφηση είτε κρυπτόγραμμα για την αποκρυπτογράφηση:

Εικόνα 2.6: Μη εισαγωγή Κρυπτογράμματος, απλού κειμένου.

Όσον αφορά την αποκρυπτογράφηση, για να εκτελεστεί σωστά πρέπει να πληρούνται οι παρακάτω προϋποθέσεις:

1. Το κείμενο που θα δοθεί να είναι σε μορφή Base-64[12] η οποία χρησιμοποιεί για την αναπαράσταση κειμένου τους χαρακτήρες `A-Z, a-z, 0-9, +, -`.
   Επίσης για την κωδικοποίηση του κάθε χαρακτήρα αντίθετα με το UTF-8 το οποίο χρησιμοποιεί 8 bits για κάθε χαρακτήρα, η Base-64 χρησιμοποιεί 6 bits/χαρακτήρα. Οπότε μια ακολουθία χαρακτήρων θα έχει μεγαλύτερο μήκος κωδικοποιημένη σε Base-64 από ότι σε UTF-8.
   Επειδή σε ορισμένες περιπτώσεις όταν η αρχική ακολουθία δεν είναι ακριβές πολλαπλάσιο του 3, ο τελευταίος χαρακτήρας της Base-64 ακολουθίας δεν θα έχει ακριβώς 6 bits, αλλά θα έχει είτε 2 είτε 4.
   Οπότε για να συμπληρωθεί ο αριθμός των 6 bits που απαιτούνται και να μπορέσει να κωδικοποιηθεί σωστά, στην πρώτη περίπτωση προστίθενται 4 bits και 2 bits στην δεύτερη.
   Αυτή η προσθήκη των bits φαίνεται σε μια Base-64 ακολουθία με το σύμβολο '==' στο τέλος μιας ακολουθίας στην οποία έχουν προστεθεί 4 bits στον τελευταίο χαρακτήρα, και με το σύμβολο '=' όταν έχουν προστεθεί 2 bits.
   Ο χαρακτήρας '=' δεν συμβολίζει κάποιον χαρακτήρα της Base-64 κωδικοποίησης, βρίσκεται πάντα στο τέλος μιας ακολουθίας και ο ρόλος τους είναι να δείχνει πόσα bit έχουν προστεθεί στην ακολουθία.
2. Το κείμενο πέραν της μορφής Base-64, πρέπει να έχει το σωστό μήκος το οποίο να αντιπροσωπεύει ένα κρυπτόγραμμα.
   Να είναι δηλαδή πολλαπλάσιο των 16 bytes, όσο είναι δηλαδή το κάθε τμήμα το οποίο κωδικοποιεί ο AES.

Αν ο χρήστης εισάγει για παράδειγμα το κείμενο `"decode!"` το οποίο περιέχει τον μη έγκυρο Base-64 χαρακτήρα " ! ", και επιλέξει να το αποκρυπτογραφήσει πατώντας το κουμπί Decrypt θα του εμφανιστεί το παρακάτω μήνυμα.



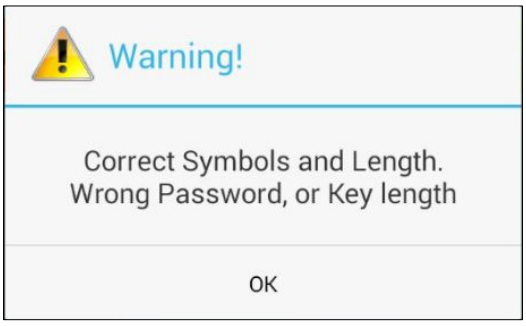
Εικόνα 2.7: Μη έγκυροι Base-64 χαρακτήρες

Αντίστοιχα αν δώσει το κείμενο `wLPT1dsCFEYGeQ7ruiPQLA1==` παρατηρούμε ότι η ακολουθία έχει μήκος 23 χαρακτήρων χωρίς τα σύμβολα "==". Με τα σύμβολα αυτά στο τέλος της ακολουθίας ξέρουμε ότι στην αρχική ακολουθία έχουν προστεθεί 4 επιπλέον bits.

Οπότε σύμφωνα με τα παραπάνω τα bits της Base-64 ακολουθίας χωρίς τα 4 συμπληρωματικά, θα είναι (23*6)-4=134 bits. Άρα τα πραγματικά bytes τα οποία έχουν κωδικοποιηθεί για αυτή την ακολουθία είναι τελικά 134/8=16,75 bytes.
Σε μια τέτοια περίπτωση αν ο χρήστης επιλέξει να αποκρυπτογραφήσει το παραπάνω κείμενο και πατήσει το κουμπί "Decrypt" θα του εμφανιστεί το εξής μήνυμα:

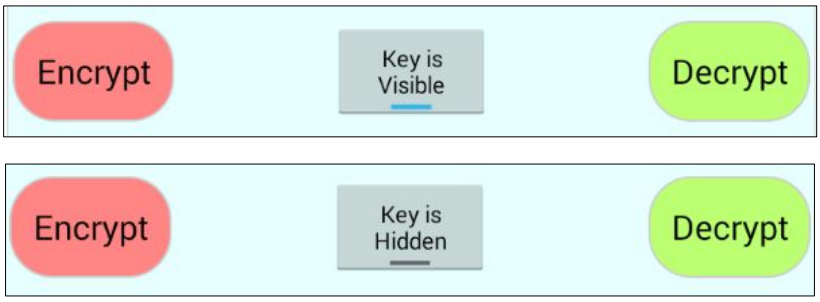
Εικόνα 2.8: Μη έγκυρο μήκος κρυπτογράμματος

Στην περίπτωση που το κρυπτόγραμμα έχει αποδεκτούς Base-64 χαρακτήρες και το μήκος του είναι ακριβές πολλαπλάσιο του 16, υπάρχει η περίπτωση ο χρήστης να δώσει είτε λάθος κωδικό για αποκρυπτογράφηση, είτε τον σωστό κωδικό αλλά το κρυπτόγραμμα να έχει παραχθεί με διαφορετικό μήκος κλειδιού του AES από αυτό που έχει επιλέξει ο χρήστης.
Σε αυτές τις περιπτώσεις αν ο χρήστης επιλέξει να αποκρυπτογραφήσει το παραπάνω κείμενο και πατήσει το κουμπί "Decrypt" θα του εμφανιστεί το εξής μήνυμα:



Εικόνα 2.9: Μη έγκυρος κωδικός ή μήκος κλειδιού.

Η κρυπτογράφηση εκτελείται πατώντας το κουμπί "Encrypt" και η αποκρυπτογράφηση αντίστοιχα με το κουμπί "Decrypt".
Το κουμπί "Key is Visible" δίνει στο χρήστη τη δυνατότητα να εμφανίζει ή όχι το κλειδί της κρυπτογράφησης.

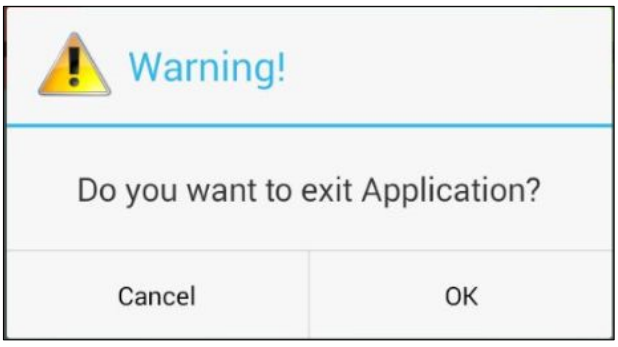


Εικόνα 2.10: Επιλογή ορατότητας κλειδιού Κρυπτογράφησης.

Επίσης στην αρχική οθόνη υπάρχουν και πέντε ακόμα βοηθητικά κουμπιά (action bar buttons) τα οποία εκτελούν διαφορετική λειτουργία το καθένα:

Εικόνα 2.11: Γενικές Επιλογές Κύριας Οθόνης

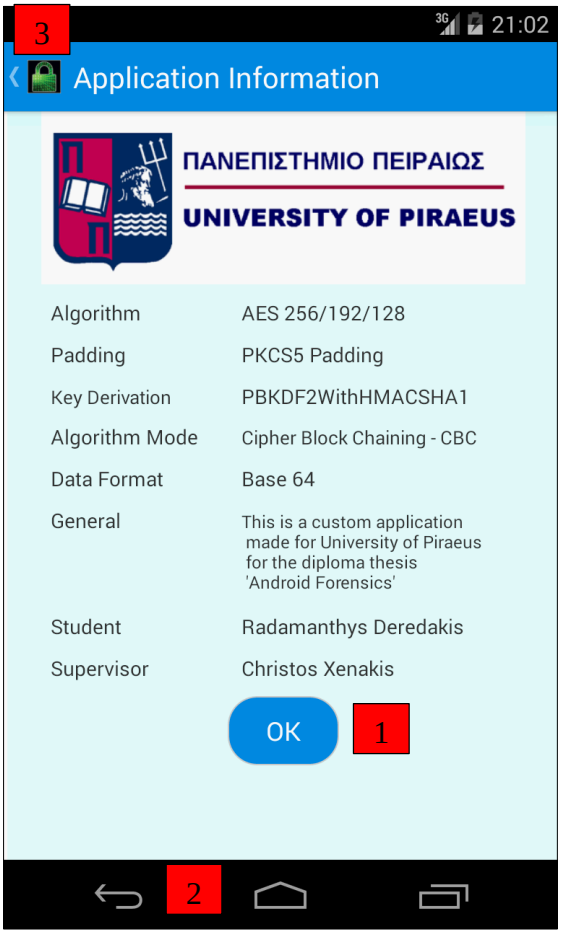
 Με την επιλογή του κουμπιού αυτού εμφανίζεται στο χρήστη ένα μήνυμα επιλογής, για το αν θέλει να τερματίσει την εφαρμογή ή να συνεχίσει. Επιλέγοντας το "OK" η εφαρμογή τερματίζεται, με την επιλογή "Cancel" η εφαρμογή συνεχίζει να εκτελείται.



Εικόνα 2.12: Μήνυμα επαλήθευσης εξόδου

 Με την επιλογή του κουμπιού αυτού εμφανίζεται μια νέα οθόνη στο χρήστη (`Activity_Options_View.java`) η οποία παρέχει όλες τις πληροφορίες λειτουργίας του προγράμματος όπως αυτές εμφανίζονται συνοπτικά στον **Error! Reference source not found.**.

Η επιστροφή στην προηγούμενη οθόνη γίνεται είτε με το κουμπί “OK” (1) είτε με το ενσωματωμένο κουμπί επιστροφής της συσκευής (2), είτε με το κουμπί επιστροφής που βρίσκεται στην action bar της κορυφής (3).



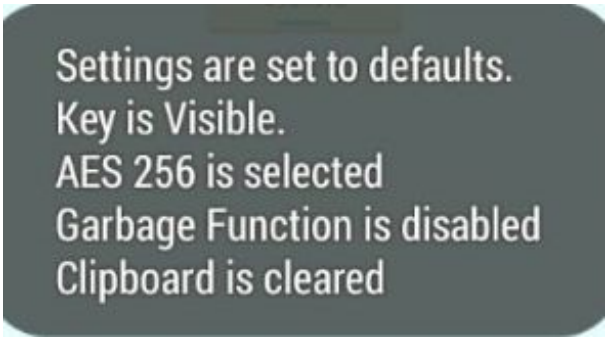
Εικόνα 2.13: Πληροφορίες εφαρμογής

Επιλέγοντας το συγκεκριμένο κουμπί όλες οι παράμετροι της εφαρμογής επιστρέφουν στις αρχικές τους τιμές, και παράλληλα εμφανίζεται και ένα μήνυμα ενημέρωσης προς το χρήστη για τη λειτουργία αυτή.

Οι αρχικές παράμετροι της εφαρμογής είναι οι εξής:

Παράμετρος	**Τιμή**
Μήκος κλειδιού AES	256
Ορατότητα κλειδιού AES	Ορατό
Συνάρτηση Garbage_Function	Απενεργοποιημένη
Πεδίο “Insert Password”	Ορατό
Πεδίο “Insert Plaintext/Ciphertext”	Ορατό
Προσωρινός Χώρος Αποθήκευσης (Clipboard)	Ενεργοποιημένος

Πίνακας 2 .3 Αρχικές παράμετροι της εφαρμογής



Εικόνα 2.14: Μήνυμα ενημέρωσης για την επιστροφή στις αρχικές παραμέτρους του προγράμματος.

Με το πάτημα του κουμπιού αυτού ο χρήστης ενεργοποιεί τη λειτουργία "Garbage Function", και το χρώμα του κουμπιού αλλάζει σε πράσινο για να γνωρίζει ο χρήστης ότι η λειτουργία αυτή παραμένει ενεργή.

Η λειτουργία αυτή κάθε φορά που γίνεται κρυπτογράφηση ή αποκρυπτογράφηση, δημιουργεί ένα τυχαίο διάνυσμα αρχικοποίησης το οποίο όμως δεν χρησιμοποιείται. Επίσης εκτελεί μερικές αριθμητικές πράξεις.

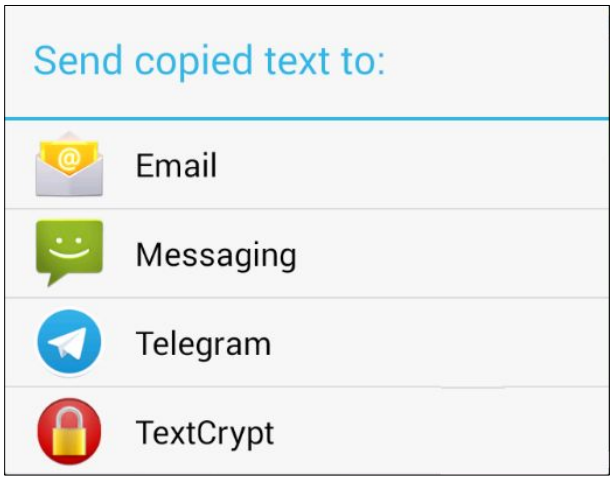
Η μόνη επίπτωση στην εφαρμογή είναι ότι προσθέτει στην εκτέλεση του κώδικα[20] επιπλέον εντολές. Ο λόγος που προστέθηκε αυτή η λειτουργία είναι για να εξεταστεί η επίπτωση που έχουν οι επιπλέον αυτές εντολές που εκτελούνται, στα αποτυπώματα μνήμης της εφαρμογής.

Τέλος εμφανίζεται ένα ενημερωτικό μήνυμα στην οθόνη το οποίο πληροφορεί το χρήστη ότι η λειτουργία είναι ενεργή.

Με το πάτημα του πράσινου κουμπιού σταματάει η λειτουργία της "Garbage Function" , και σταματάει να δημιουργείται το τυχαίο διάνυσμα αρχικοποίησης. Τέλος εμφανίζεται ένα ενημερωτικό μήνυμα στην οθόνη το οποίο πληροφορεί το χρήστη ότι η λειτουργία είναι πλέον ανενεργή και το χρώμα του κουμπιού επιστρέφει στο κόκκινο.

Πατώντας αυτό το κουμπί εμφανίζεται στο χρήστη μια οθόνη με τα διαθέσιμα προγράμματα στην εκάστοτε συσκευή τα οποία μπορούν να στείλουν κείμενο σε κάποια άλλη συσκευή. Επιλέγοντας ένα από αυτά τα προγράμματα ο χρήστης μπορεί να στείλει

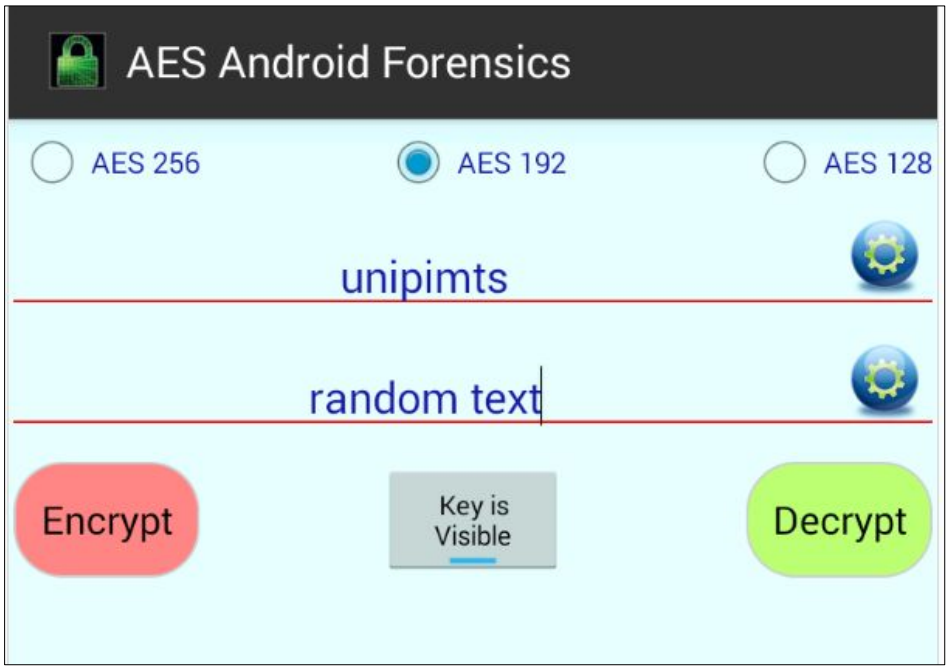
σε κάποιον άλλο χρήστη όποιο κείμενο έχει προηγουμένως αντιγράψει (απλό κείμενο, αποκρυπτογραφημένο κείμενο, κρυπτόγραμμα, κωδικό, ή κλειδί κρυπτογράφησης).
Η οθόνη αυτή παρουσιάζεται στην παρακάτω εικόνα για τον προσομοιωτή που δημιουργήθηκε.



Εικόνα 2.15: Αποστολή Κειμένου

2.*1.1 Κρυπτογράφηση Κειμένου*

Σε αυτή την ενότητα παρουσιάζεται το αποτέλεσμα της κρυπτογράφησης όταν ο χρήστης δώσει σαν είσοδο κωδικό με τιμή `"unipimts"`, απλό κείμενο για κρυπτογράφηση με τιμή `"random text"` και μήκος κλειδιού `192 bits`.
Οι τιμές αυτές φαίνονται στην παρακάτω εικόνα:

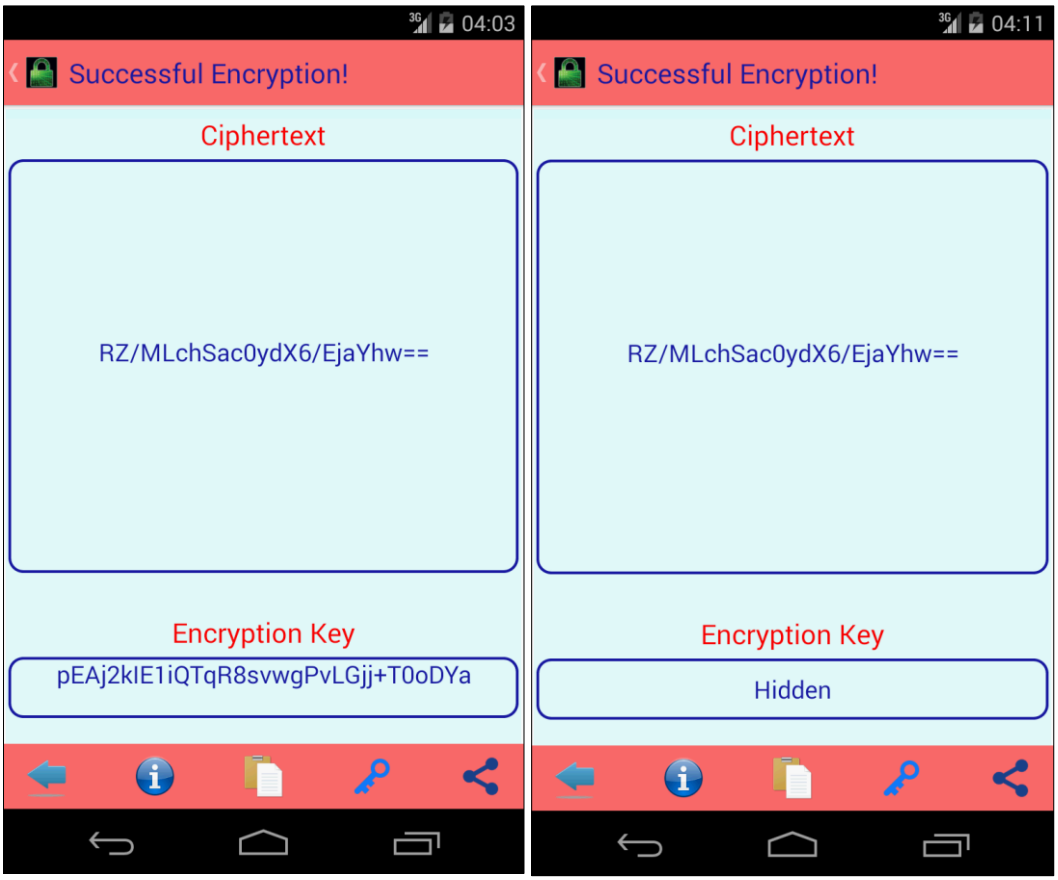


Εικόνα 2.16: Τιμές Κρυπτογράφησης

Με αυτές τις τιμές όταν ο χρήστης πατήσει το κουμπί "Encrypt" θα του εμφανιστεί μια νέα οθόνη (Activity_Encryption_View.java), όπου υπάρχουν δύο πλαίσια, ένα με την ονομασία Ciphertext στην οποία εμφανίζεται το κρυπτόγραμμα, και ένα με την ονομασία Encryption Key στην οποία εμφανίζεται το κλειδί κρυπτογράφησης.

Στην περίπτωση που ο χρήστης επιλέξει να μην εμφανίζεται το κλειδί κρυπτογράφησης στο πλαίσιο αυτό εμφανίζεται το μήνυμα Hidden.

Οι παρακάτω εικόνες δείχνουν το αποτέλεσμα της κρυπτογράφησης με τις τιμές που επιλέχθηκαν τόσο με το κλειδί ορατό όσο και με το κλειδί κρυφό.



Εικόνα 2.17: Οθόνη Κρυπτογράφησης

Στην οθόνη αυτή εμφανίζονται τρία διαφορετικά βοηθητικά κουμπιά στην action bar από αυτά της κύριας οθόνης όπως φαίνεται στην παρακάτω εικόνα.

Εικόνα 2.18: Γενικές Επιλογές Οθόνης Κρυπτογράφησης.

Πατώντας το κουμπί αυτό ο χρήστης επιστρέφει στην αρχική οθόνη.

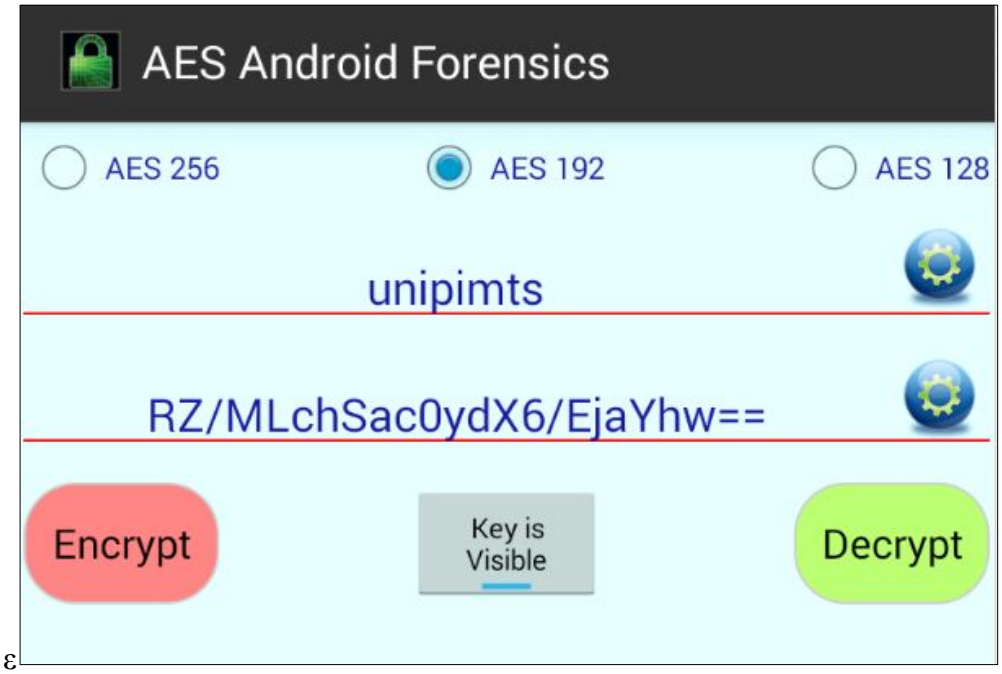
Πατώντας το κουμπί αυτό γίνεται αντιγραφή του κρυπτογράμματος και εμφανίζεται το αντίστοιχο ενημερωτικό μήνυμα για την επιτυχή αντιγραφή του κρυπτογράμματος.

Πατώντας το κουμπί αυτό γίνεται αντιγραφή του κλειδιού κρυπτογράφησης και εμφανίζεται το αντίστοιχο ενημερωτικό μήνυμα για την επιτυχή αντιγραφή του κλειδιού. Στην περίπτωση που το κλειδί είναι κρυφό εμφανίζεται ενημερωτικό μήνυμα ότι δεν μπορεί να γίνει αντιγραφή του κλειδιού καθώς αυτό είναι κρυφό.

2.*1.2 Αποκρυπτογράφηση Κρυπτογράμματος*

Σε αυτή την ενότητα παρουσιάζεται το αποτέλεσμα της αποκρυπτογράφησης όταν ο χρήστης δώσει σαν είσοδο κωδικό με τιμή "unipimts", και κρυπτόγραμμα για αποκρυπτογράφηση με τιμή " RZ/MLchSac0ydX6/EjaYhw==" και μήκος κλειδιού 192 bits.

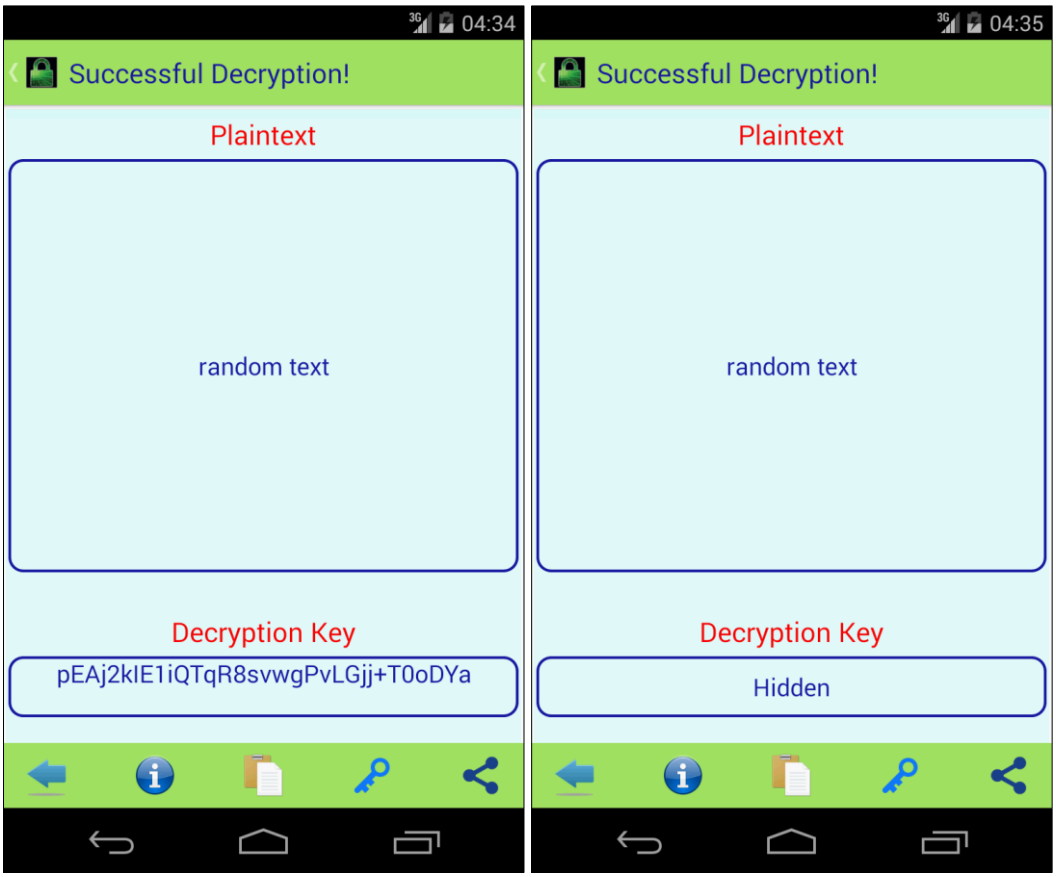
Οι τιμές αυτές φαίνονται στην παρακάτω εικόνα:



ε

Εικόνα 2.19: Τιμές Αποκρυπτογράφησης

Με αυτές τις τιμές όταν ο χρήστης πατήσει το κουμπί "Decrypt" θα του εμφανιστεί μια νέα οθόνη (Activity_Decryption_View.java), όπου υπάρχουν δύο πλαίσια, ένα με την ονομασία Plaintext στην οποία εμφανίζεται το αποκρυπτογραφημένο κείμενο, και ένα με την ονομασία Decryption Key στην οποία εμφανίζεται το κλειδί αποκρυπτογράφησης. Στην περίπτωση που ο χρήστης επιλέξει να μην εμφανίζεται το κλειδί αποκρυπτογράφησης στο πλαίσιο αυτό εμφανίζεται το μήνυμα Hidden.
Οι παρακάτω εικόνες δείχνουν το αποτέλεσμα της αποκρυπτογράφησης με τις τιμές που επιλέχθηκαν τόσο με το κλειδί ορατό όσο και με το κλειδί κρυφό.



Εικόνα 2.20: Οθόνη Αποκρυπτογράφησης

Στην οθόνη αυτή τα κουμπιά που εμφανίζονται στην action bar είναι τα ίδια με αυτά της οθόνης κρυπτογράφησης και επιτελούν τις ίδιες λειτουργίες, για την επιστροφή στην αρχική οθόνη, αντιγραφή του απλού κειμένου, και αντιγραφή του κλειδιού αποκρυπτογράφησης αντίστοιχα.

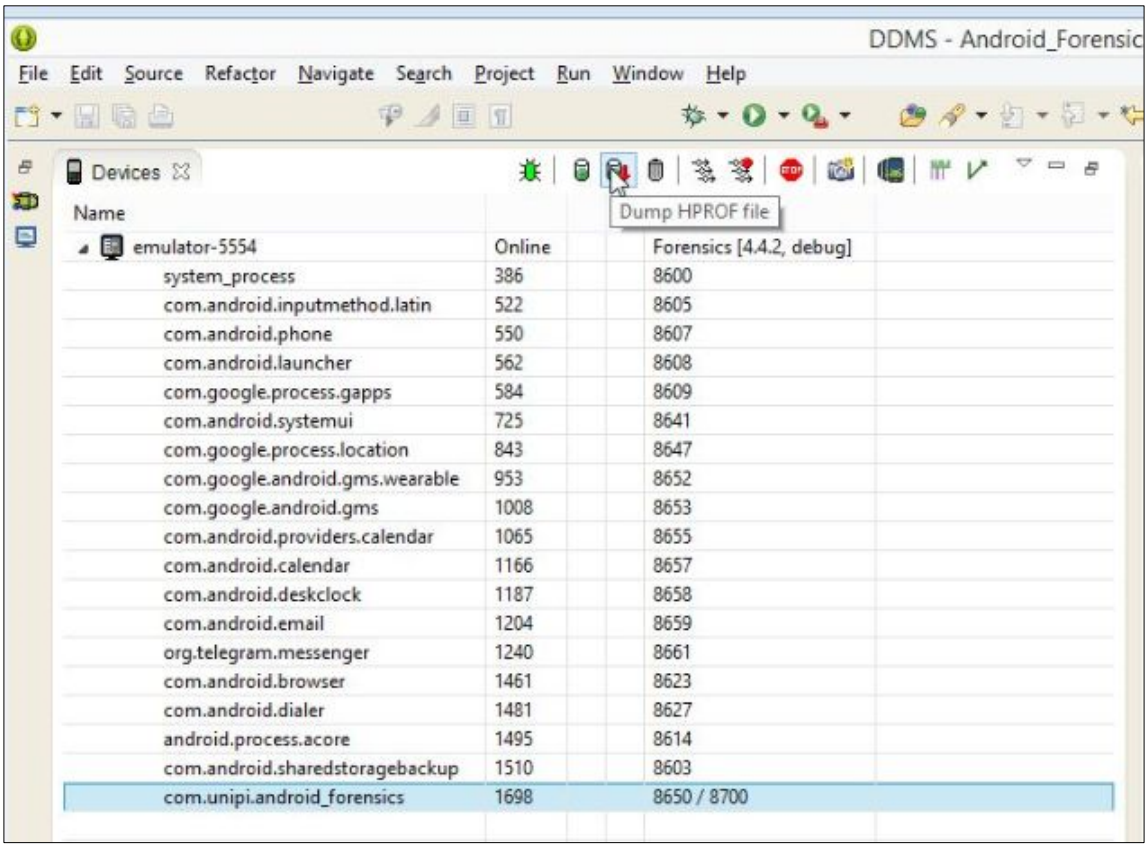
Εικόνα 2.21: Γενικές Επιλογές Οθόνης Αποκρυπτογράφησης

2.3 Δημιουργία και Ανάλυση Αποτυπωμάτων Μνήμης

Για τη δημιουργία των αποτυπωμάτων μνήμης της εφαρμογής από τον προσομοιωτή χρησιμοποιήθηκε το εργαλείο Dalvik Debug Monitor Server-DDMS, το οποίο είναι διαθέσιμο στο Eclipse μέσω του Android Development Tool-ADT.

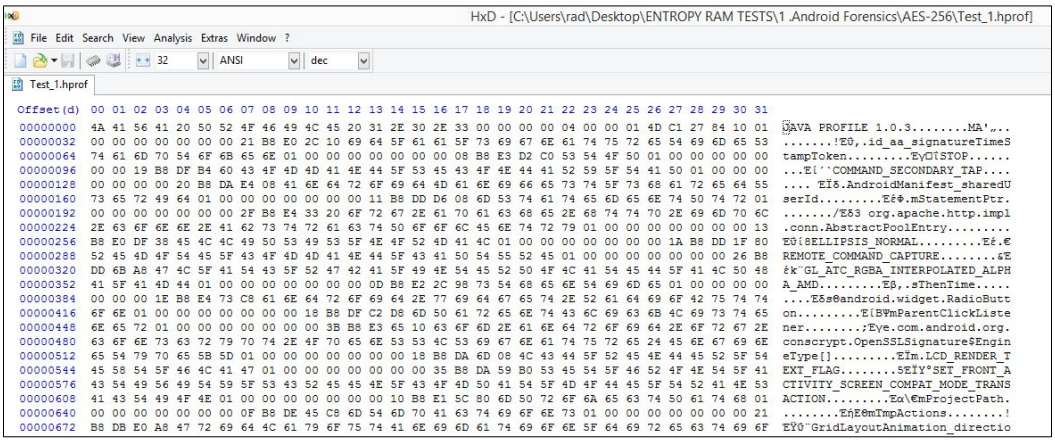
Μια από τις επιλογές που παρέχονται από το συγκεκριμένο εργαλείο είναι η δημιουργία αποτυπωμάτων μνήμης μέσω της επιλογής “Dump Hprof File”.

Η επιλογή αυτή δίνει τη δυνατότητα να αποθηκευτεί το αποτύπωμα μνήμης της εφαρμογής η οποία προσομοιώνεται, με τη μορφή αρχείου `.hprof` , και φαίνεται στην παρακάτω εικόνα:



Εικόνα 2.22: Επιλογή δημιουργίας αποτυπώματος μνήμης DDMS.

Το πρόγραμμα που χρησιμοποιήθηκε για την ανάγνωση και ανάλυση των αποτυπωμάτων αυτών ήταν το HxD - Freeware Hex Editor and Disk Editor[13] , το οποίο δείχνει τόσο το κείμενο σε δεκαεξαδική μορφή (hex) όσο και σε UTF-8 κωδικοποίηση, όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 2.23 Γραφικό περιβάλλον του προγράμματος HxD.

Δημιουργήθηκαν διάφορα αποτυπώματα μνήμης, με χρήση διαφορετικών κωδικών, διαφορετικών κειμένων, υπό διαφορετικές συνθήκες μέχρι να εξαχθούν ασφαλή συμπεράσματα.

Στην παρούσα εργασία αναλύεται ένα αποτύπωμα μνήμης το οποίο δημιουργήθηκε από την εφαρμογή AES Android Forensics, με το χρήστη να εκτελεί κατά σειρά τις εντολές:

1. Επιλογή μήκος κλειδιού 192 bits.
2. Insert Password
3. Insert Plaintext
4. Encrypt
5. Back Button στην οθόνη Κρυπτογράφησης και επιστροφή στην αρχική οθόνη
6. Δημιουργία αποτυπώματος μνήμης με την εντολή Dump Hprof File από τον DDMS.

Οι παράμετροι που χρησιμοποιήθηκαν είναι οι εξής:

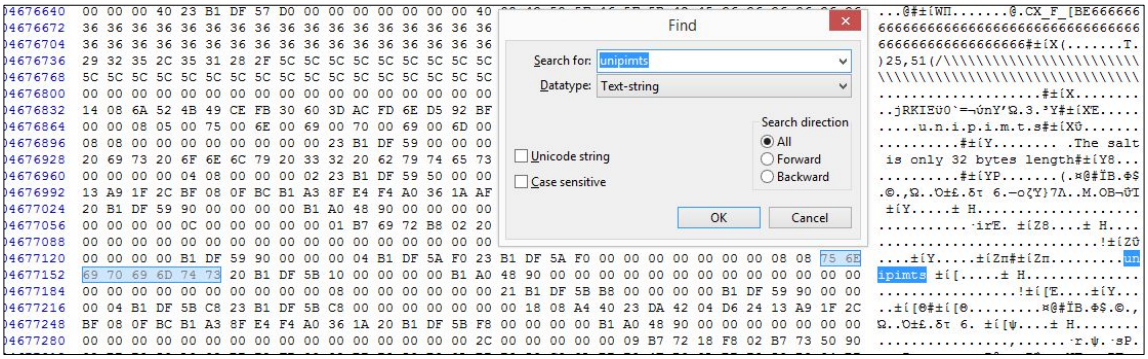
Παράμετροι	**Τιμές**
Κωδικός	`unipimts`
Κείμενο προς κρυπτογράφηση	`random text`
Κρυπτόγραμμα (Base-64)	`RZ/MLchSac0ydX6/EjaYhw==`
Κρυπτόγραμμα (hex)	`45 9F CC 2D C8 52 69 CD 32 75 7E BF 12 36 98 87`
Κλειδί κρυπτογράφησης 192 bits (Base-64)	`pEAj2kIE1iQTqR8svwgPvLGjj+T0oDYa`
Κλειδί κρυπτογράφησης 192 bits (hex)	`A4 40 23 DA 42 04 D6 24 13 A9 1F 2C BF 08 0F BC B1 A3 8F E4 F4 A0 36 1A`

Πίνακας 2.4: Παράμετροι αποτυπώματος μνήμης

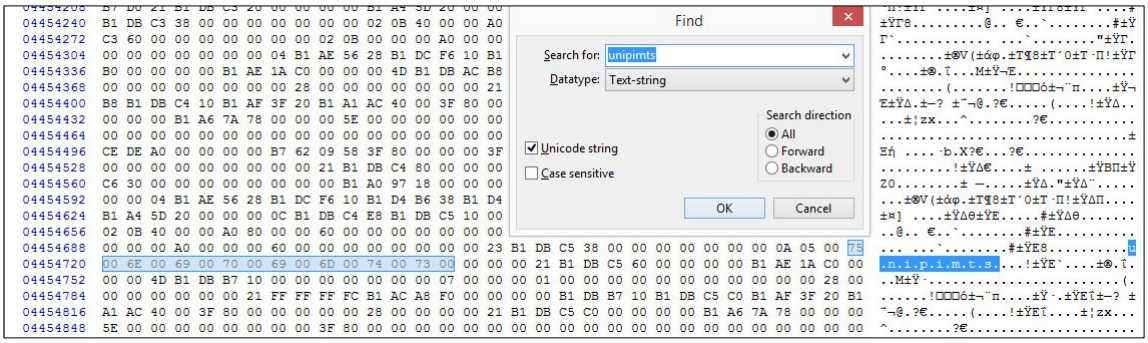
Έπειτα από μια αναζήτηση με τη βοήθεια του προγράμματος HxD προέκυψαν τα εξής ευρήματα:

Ο κωδικός, εμφανίζεται μία φορά σε αναζήτηση απλού κειμένου (UTF-8 κωδικοποίηση) και δύο φορές σε αναζήτηση Unicode (UTF-16).
Το κείμενο προς κρυπτογράφηση εμφανίζεται μία φορά σε αναζήτηση απλού κειμένου και τρεις φορές σε αναζήτηση Unicode.
Το κρυπτόγραμμα σε Base-64, εμφανίζεται μία φορά σε αναζήτηση απλού κειμένου και δύο φορές σε αναζήτηση Unicode.
Το κρυπτόγραμμα σε hex, εμφανίζεται μία φορά στην αναζήτηση, και το κείμενο στο οποίο αποκωδικοποιείται από το HxD (Windows-1253 αποκωδικοποίηση) είναι:
`E.M-ΘRiN2u~Ω.6.‡`
Το κλειδί σε Base-64, εμφανίζεται μία φορά σε αναζήτηση απλού κειμένου και δύο φορές σε αναζήτηση Unicode.
Το κλειδί σε hex, εμφανίζεται μία φορά στην αναζήτηση, και το κείμενο στο οποίο αποκωδικοποιείται από το HxD (Windows-1253 αποκωδικοποίηση) είναι:
`¤@#ΪB.Φ$.©.,Ω..Ό±£.δτ 6.`

Για λόγους συντομίας παρουσιάζονται μόνο τα ευρήματα του κωδικού στις παρακάτω εικόνες:



Εικόνα 2.24: Εύρεση κωδικού σε UTF-8 κωδικοποίηση



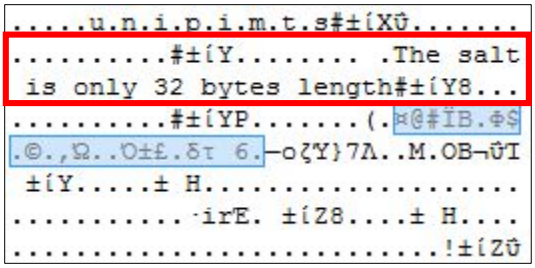
Εικόνα 2.25 Εύρεση κωδικού σε UTF-16 κωδικοποίηση

2.1.1 Ανάλυση Παραμέτρων AES

Για τις παραμέτρους του AES, τόσο σε αυτή τη δοκιμή όσο και σε άλλες οι οποίες διεξήχθησαν με παρόμοιες τιμές, προέκυψαν συμπεράσματα τα οποία παρουσιάζονται συνοπτικά σε αυτή την ενότητα.

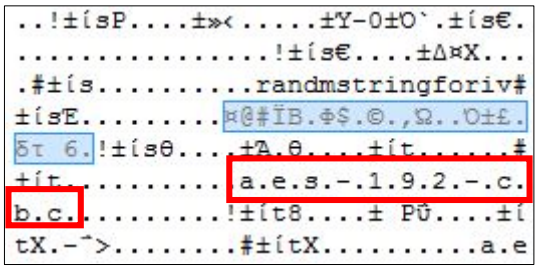
Καθώς οι παρόμοιες εφαρμογές οι οποίες χρησιμοποιούν κρυπτογράφηση κειμένου με βάση κωδικό, δεν εμφανίζουν το κλειδί κρυπτογράφησης σε οποιαδήποτε μορφή, δόθηκε ιδιαίτερη έμφαση στην αναζήτηση του κλειδιού κρυπτογράφησης σε δεκαεξαδική μορφή, και όχι σε Base-64. Τα αποτελέσματα της αναζήτησης αυτής παρουσιάζονται παρακάτω.

1. Το κλειδί κρυπτογράφησης βρίσκεται παραπάνω από δύο φορές στο αποτύπωμα μνήμης.
2. Το πρώτο εύρημά του έχει ιδιαίτερη υψηλή μετατόπιση (offset) καθώς βρίσκεται σε τιμές προς το τέλος των διευθύνσεων μνήμης του αποτυπώματος μνήμης.
3. Η πρώτη εύρεση του κλειδιού είναι στο offset 4676984 bytes από τα συνολικά 4899047 bytes που περιέχονται στο αποτύπωμα μνήμης. 95.46% του μεγέθους του αποτυπώματος μνήμης.
   Ταυτόχρονα, βρίσκεται πολύ κοντά με την τιμή του «αλατιού» (The salt is only 32 bytes length)



Εικόνα 2.26 Κλειδί Κρυπτογράφησης και «αλάτι»

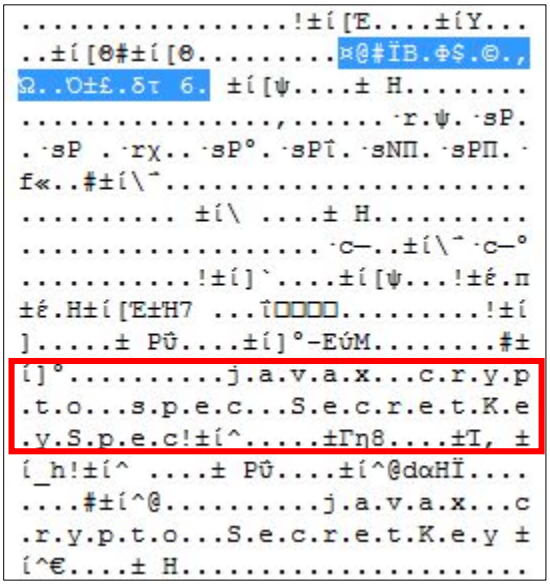
4. Βρίσκεται πολύ κοντά στο διάνυσμα αρχικοποίησης (randmstringforiv), και κοντά στο offset του κλειδιού περιγράφεται και ο τρόπος λειτουργίας του αλγόριθμου (AES-192 CBC)



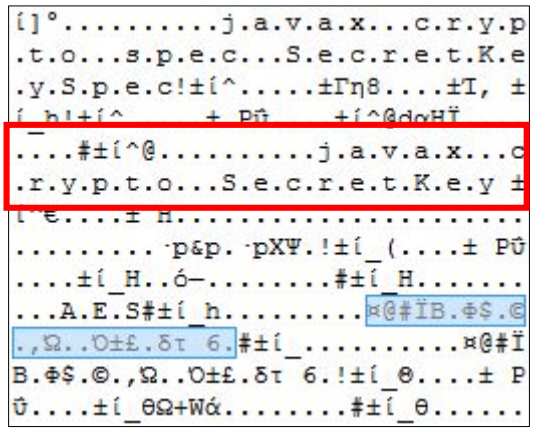
Εικόνα 2.27: Κλειδί Κρυπτογράφησης και διάνυσμα αρχικοποίησης

5. Βρίσκεται σε offsets κοντά τόσο στην `class` `javax.crypto.spec.SecretKeySpec` όσο και στη διεπαφή `javax.crypto.SecretKey`, το οποίο είναι αναμενόμενο καθώς αυτές οι δύο είναι υπεύθυνες για τη δημιουργία του κλειδιού σύμφωνα με την εντολή που εκτελείται στον κώδικα[20] της εφαρμογής:

```
SecretKey AES_Key = new SecretKeySpec(PBE_Key.getEncoded(),
"AES");
```



Εικόνα 2.28: Κλειδί Κρυπτογράφησης και javax.crypto.spec.SecretKeySpec

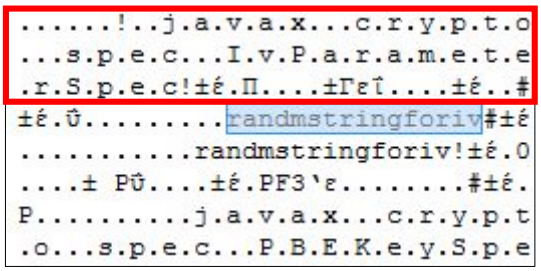


Εικόνα 2.29: Κλειδί Κρυπτογράφησης και javax.crypto.SecretKey

Για το **διάνυσμα αρχικοποίησης** παρατηρήθηκαν τα εξής:

Εμφανίζεται δύο φορές σε offset πολύ κοντινά με τη `class IvParameterSpec` καθώς αυτή είναι υπεύθυνη για τη δημιουργία του:

```
IvParameterSpec  iv_specs = new
IvParameterSpec(Main.Init_Vector.getBytes("UTF-8"));
```

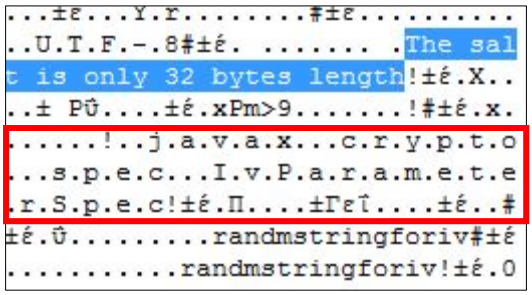


Εικόνα 2.30: Διάνυσμα αρχικοποίησης και javax.crypto.spec.IvParameterSpec

Επίσης ένα άλλο εύρημά του όπως φαίνεται στην Εικόνα 2.27 είναι ότι εμφανίζεται πολύ κοντά στο κλειδί κρυπτογράφησης.

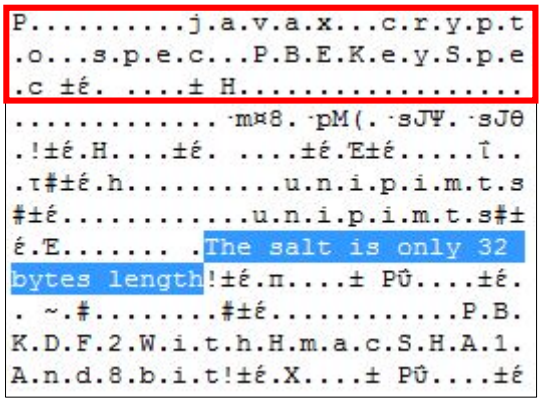
Για το **«αλάτι»** παρατηρήθηκαν τα εξής:

1. Ένα εύρημα του είναι πολύ κοντά στο κλειδί κρυπτογράφησης. (Εικόνα 2.26)
2. Ένα άλλο εύρημα του βρίσκεται κοντά στο διάνυσμα αρχικοποίησης λίγο πριν το offset της `javax.crypto.spec.IvParameterSpec`:



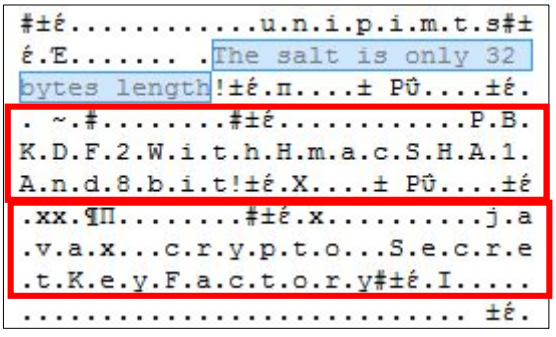
Εικόνα 2. 31: "Αλάτι" και javax.crypto.spec.IvParameterSpec

Ένα εύρημά του βρίσκεται κοντά στην `class PBEKeySpec`, καθώς είναι αυτή που δέχεται τις παραμέτρους για κρυπτογράφηση με βάση κωδικό.



Εικόνα 2.32: "Αλάτι" και javax.crypto.PBEKeySpec

3. Τέλος ένα άλλο εύρημά του βρίσκεται σε κοντινό offset με τον αλγόριθμο που χρησιμοποιείται από τη συνάρτηση PBDFK2 για δημιουργία του κλειδιού. `(PBKDF2WithHmacSHA1And8bit)`.
   Επιπλέον σε κοντινό offset από τον αλγόριθμο βρίσκεται και η class η οποία τον δημιουργεί, δηλαδή η `SecretKeyFactory` καθώς στον κώδικα[20] εκτελείται η εντολή: `factory=SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1And8bit")
   ;`



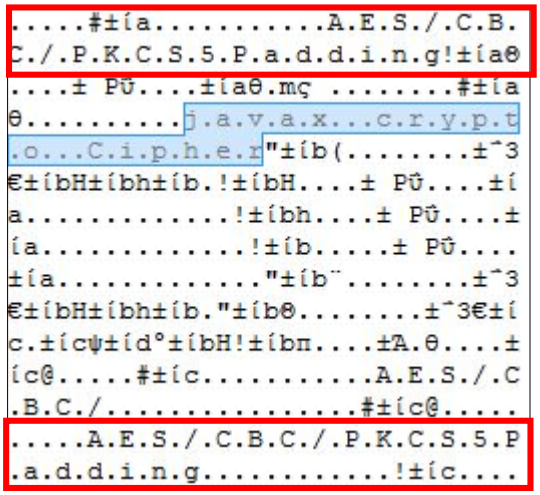
Εικόνα 2.33: «Αλάτι» και PBKDF2WithHmacSHA1And8bit

Όσον αφορά τον **τρόπο λειτουργίας** του αλγόριθμου κρυπτογράφησης AES αυτός δίνεται από την εντολή του προγράμματος:

```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
```

Επίσης όπως φαίνεται στην ενότητα 1.4 η κλάση αυτή ανήκει στο package javax.crypto.

Με αναζήτηση στο αποτύπωμα μνήμης εμφανίζεται η τιμή `javax.crypto.Cipher` σε Unicode μορφή και παρατηρήθηκε ότι σε κοντινό offset από το εύρημα αυτό εμφανίζεται η τιμή που δόθηκε στην παραπάνω εντολή:



Εικόνα 2.34: Τρόπος λειτουργίας AES και javax.crypto.Cipher

Επίσης θα πρέπει να παρατηρηθεί ότι για τη δημιουργία όλων των παραπάνω παραμέτρων του AES οι αντίστοιχες κλάσεις της Java καλούνται μόνο μια φορά, όπως φαίνεται και στον κώδικα[20] του προγράμματος.

Αυτό συμβαίνει γιατί π.χ. με την εντολή

```
iv_specs = new IvParameterSpec(Main.Init_Vector.getBytes("UTF-8"));
```

ορίστηκε ένα διάνυσμα αρχικοποίησης, και άρα δεν υπάρχει κάποιος προφανής λόγος να οριστεί εκ νέου ένα δεύτερο διάνυσμα αρχικοποίησης, είτε το ίδιο είτε διαφορετικό.

Το ίδιο συμβαίνει αντίστοιχα και για τις κλάσεις `PBEKeySPec`, `SecretKeyFactory`, `SecretKey`, `SecretKeySpec`, `Cipher`.

Αυτό έχει σαν αποτέλεσμα στο αποτύπωμα μνήμης οι παραπάνω κλάσεις να εμφανίζονται από μία φορά η κάθε μία, παρόλο που οι τιμές οι οποίες αυτές δημιουργούν (κλειδί κρυπτογράφησης, διάνυσμα αρχικοποίησης, κρυπτόγραμμα) μπορεί να εμφανίζονται περισσότερες της μίας φοράς.

Επίσης παρατηρείται ότι από όλες τις μεταβλητές που χρησιμοποιούνται οι μόνες οι οποίες δεν χρησιμοποιούν κάποια συγκεκριμένη κλάση της java για τη δημιουργία τους είναι το «αλάτι» και ο αριθμός των επαναλήψεων. Οι δύο αυτές μεταβλητές μπορεί να

δηλωθούν σαν απλές μεταβλητές ακολουθίας `(String)` και ακέραιου `(int)` αντίστοιχα, στο εκάστοτε πρόγραμμα, και για το λόγο αυτό είναι αρκετά δύσκολο να εντοπιστούν σε ένα αποτύπωμα μνήμης.

Ο αριθμός των επαναλήψεων δεν βρέθηκε σε κάποιο συγκεκριμένο τμήμα του αποτυπώματος μνήμης.

Τέλος στις δοκιμές που διενεργήθηκαν με την συνάρτηση "Garbage_Function" ενεργοποιημένη δεν παρατηρήθηκε κάποια ιδιαίτερη αλλαγή στα ευρήματα, και η μεταβολή η οποία δημιουργήθηκε στο μέγεθος του αρχείου ήταν αμελητέα.

ΚΕΦΑΛΑΙΟ 3

ΑΝΑΛΥΣΗ ANDROID ΕΦΑΡΜΟΓΩΝ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ ΚΕΙΜΕΝΟΥ ΒΑΣΙΖΟΜΕΝΕΣ ΣΕ ΚΩΔΙΚΟ

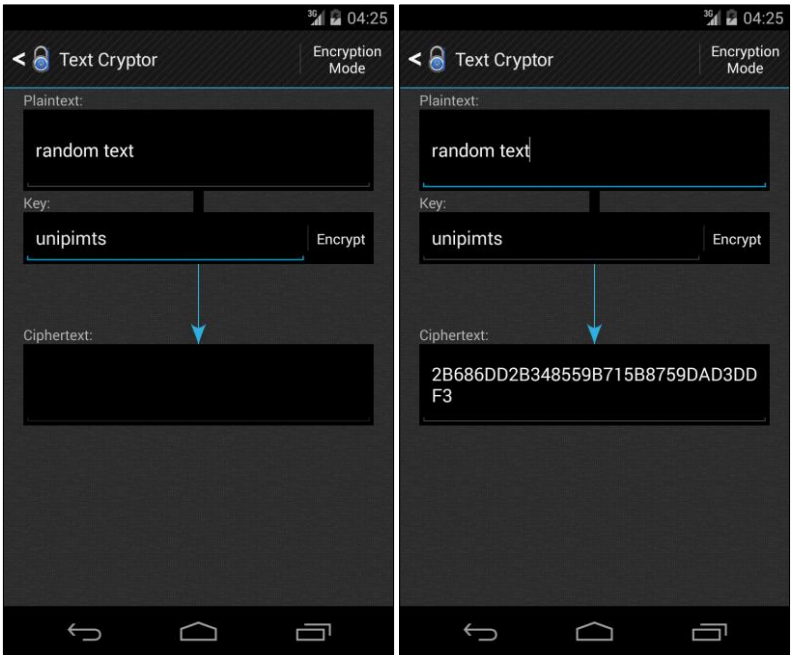
Με βάση τα παραπάνω ευρήματα τα οποία προέκυψαν από την εξέταση της εφαρμογής AES_Android Forensics εξετάστηκαν δέκα παρόμοιες εφαρμογές οι οποίες εφαρμόζουν κρυπτογράφηση βασιζόμενες στη χρήση κωδικού.

Σε όλες τις εφαρμογές έγινε η χρήση του κωδικού "**unipimts**" και σαν κείμενο προς κρυπτογράφηση δόθηκε η τιμή "**random text**".

3.1 Εφαρμογή jCryptor Text Encryption

Η συγκεκριμένη εφαρμογή επιτελεί κρυπτογράφηση/αποκρυπτογράφηση χωρίς να δίνει κάποια επιπλέον επιλογή στο χρήστη, και από την περιγραφή της στον ιστότοπο του Google Play[14] δεν αναφέρεται κάποια πληροφορία για τον τρόπο λειτουργίας του αλγορίθμου κρυπτογράφησης ο οποίος χρησιμοποιείται.

Το γραφικό της περιβάλλον φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.1: Εφαρμογή Jcryptor

Από την εμφάνιση του κρυπτογράμματος `(2B686DD2B348559B715B8759DAD3DDF3)` εξάγεται άμεσα το συμπέρασμα ότι δεν είναι κωδικοποιημένο σε Base-64 αλλά σε δεκαεξαδική μορφή (raw bytes) με μήκος 16 bytes . Άρα ο αλγόριθμος κρυπτογράφησης είναι αλγόριθμος τμήματος.
Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.

Επίσης βρέθηκαν οι εξής κλάσεις της java:

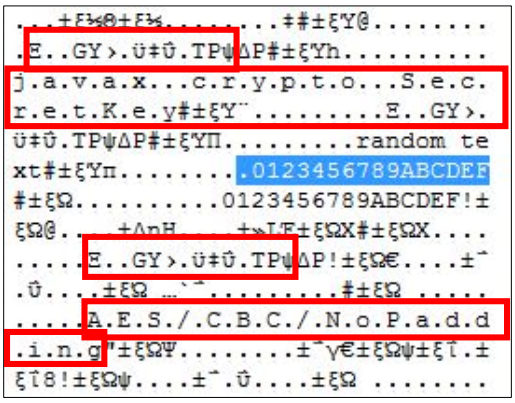
1. `javax.crypto.KeyGenerator`
2. `javax.crypto.SecretKey`
3. `javax.net.ssl.keystore`

Η κλάση `javax.crypto.KeyGenerator` δημιουργεί ένα συμμετρικό κλειδί κρυπτογράφησης και λειτουργεί με συγκεκριμένους αλγόριθμους κρυπτογράφησης οι οποίοι είναι[15]:

1. AES (128)
2. DES (56)
3. DESede (168)
4. HmacSHA1
5. HmacSHA256

Από τους παραπάνω αλγόριθμους που χρησιμοποιεί η κλάση αυτή, ο πιο πιθανός που μπορεί να χρησιμοποιείται στη συγκεκριμένη εφαρμογή είναι ο AES-128 καθώς το κρυπτόγραμμα έχει μήκος 16 bytes, όσο δηλαδή και το μήκος τμήματος του AES.
Επίσης η κλάση `javax.crypto.KeyGenerator` έχει τη δυνατότητα να χρησιμοποιεί για την παραγωγή του κλειδιού και ένα παράγοντα τυχαιότητας ο οποίος δημιουργείται από την κλάση `SecureRandom` με την παροχή εισόδου από το χρήστη (seed).
Αυτή η είσοδος σε περιπτώσεις κρυπτογράφησης με μυστικό κωδικό είναι το «αλάτι».

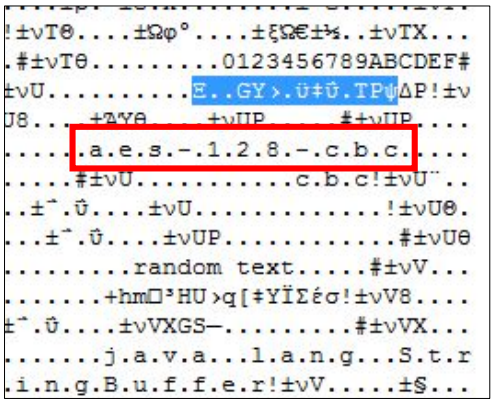
Στο εύρημα `javax.crypto.SecretKey` , παρατηρήθηκε ότι σε κοντινό offset από αυτό υπάρχει δύο φορές η ακολουθία `"0123456789ABCDEF"` με μήκος 16 bytes η οποία πιθανόν να αντιπροσωπεύει ένα διάνυσμα αρχικοποίησης.



Εικόνα 3.2: Διάνυσμα αρχικοποίησης

Επίσης εμφανίζεται άλλη μία φορά σε διαφορετικό offset, όπου παρατηρήθηκε ότι η ακολουθία των 16 bytes " Ξ..GY›.ü‡û.TPψ" εμφανίζεται συνολικά 6 φορές στο αποτύπωμα μνήμης καθώς και δύο φορές κοντά στο offset του `javax.crypto.SecretKey` όπως φαίνεται και στην προηγούμενη εικόνα.

Άρα η ακολουθία αυτή με μεγάλη πιθανότητα αντιπροσωπεύει ένα AES κλειδί μήκους 128 bits. Επίσης σε αυτό το offset τόσο σε αυτή όσο και στις εφαρμογές Encrypt/AES Android Forensics αναφέρεται ο τρόπος λειτουργίας του αλγόριθμου, όπου στη συγκεκριμένη περίπτωση είναι ο AES-128 CBC/NoPadding.



Εικόνα 3.3 Κλειδί κρυπτογράφησης και διάνυσμα αρχικοποίησης

Το κλειδί επιβεβαιώθηκε ότι είναι η παραπάνω ακολουθία " Ξ..GY›.ü‡û.TPψΔP" καθώς σε υλοποίηση της Java κατόρθωσε να γίνει αποκρυπτογράφηση του κρυπτογράμματος "2B686DD2B348559B715B8759DAD3DDF3" στο απλό κείμενο "random text".

Επιπλέον βασιζόμενοι στα προηγούμενα ευρήματα, (IV, AES-192 CBC, `javax.crypto.KeyGenerator`) έγινε προσπάθεια προσομοίωσης του τρόπου λειτουργίας της συγκεκριμένης εφαρμογής σε περιβάλλον Java, αλλά απέτυχε.

Ο λόγος είναι ότι η κλάση `KeyGenerator` έχει τη δυνατότητα να χρησιμοποιεί κάποιο «αλάτι» όπως αναφέρθηκε παραπάνω για τη δημιουργία του κλειδιού.

Έπειτα από εκτενή αναζήτηση για ύπαρξη τυχόν «αλατιού» στο αποτύπωμα μνήμης δεν βρέθηκε κάποιο αξιόπιστο αποτέλεσμα.

Αυτό συνέβη διότι το «αλάτι» δεν είναι απαραίτητο ότι θα είναι σε μορφή απλού κειμένου, αλλά μπορεί προηγουμένως να έχει κατακερματιστεί (hashed), είτε το ίδιο, είτε σε συνδυασμό με κάποια άλλη τιμή (κωδικό, IV, κείμενο) και μετά να χρησιμοποιηθεί από την `KeyGenerator` για τη δημιουργία του κλειδιού.

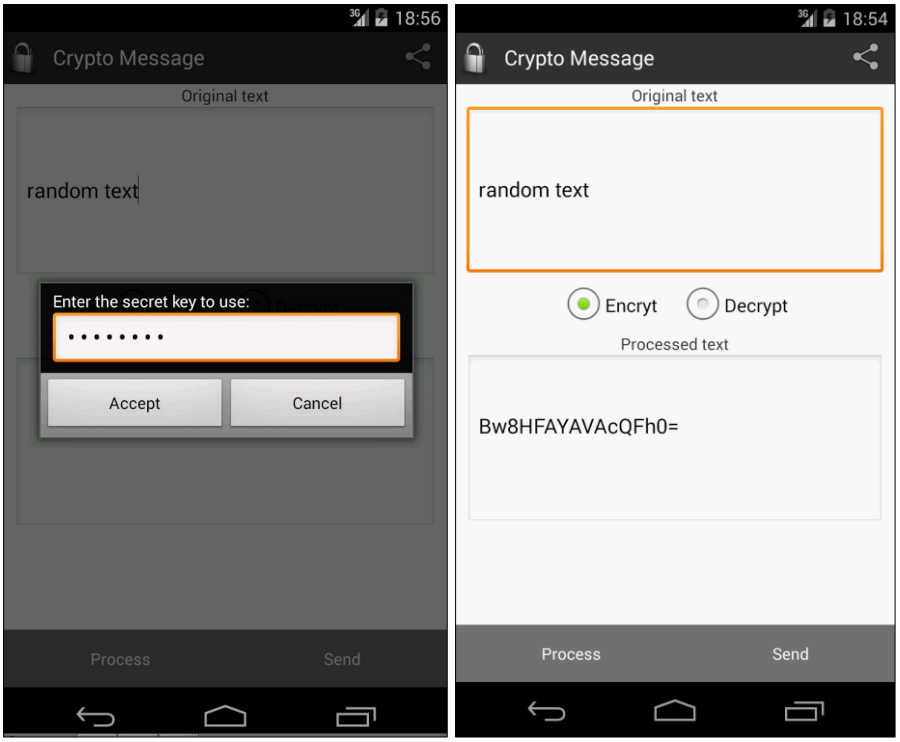
Επίσης δεν είναι υποχρεωτικό ότι θα έχει σταθερό μήκος όπως το IV για παράδειγμα.

Οι παραπάνω παράγοντες οδηγούν στο συμπέρασμα ότι στη συγκεκριμένη λειτουργία κρυπτογράφησης χρησιμοποιείται «αλάτι» αγνώστου τιμής, και μήκους για τη δημιουργία του κλειδιού.

3.2 Εφαρμογή Crypto Message

Η συγκεκριμένη επιτελεί κρυπτογράφηση/αποκρυπτογράφηση χωρίς να δίνει κάποια επιπλέον επιλογή στο χρήστη, και από την περιγραφή της στον ιστότοπο του Google Play[16] δεν αναφέρεται τίποτα για τον τρόπο λειτουργίας της κρυπτογράφησης.

Το γραφικό της περιβάλλον φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.4: Εφαρμογή Crypto Message

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του κρυπτογράμματος και του αρχικού κειμένου, δεν βρέθηκε όμως κάποια class της Java `(javax.crypto)` έτσι ώστε να μπορεί να γίνει περαιτέρω ανάλυση του τρόπου λειτουργίας του αλγορίθμου.

Παρόλαυτα εξετάζοντας ένα ακόμα αποτύπωμα μνήμης με τον ίδιο κωδικό (unipimts) και απλό κείμενο το (random) παρατηρήθηκε το εξής:

Κείμενο	Μήκος (bytes)	Κρυπτόγραμμα (Base-64)	Κρυπτόγραμμα (Bytes)
random text	11	Bw8HFAYAVAcQFh0=	11
random	6	Bw8HFAYA	6

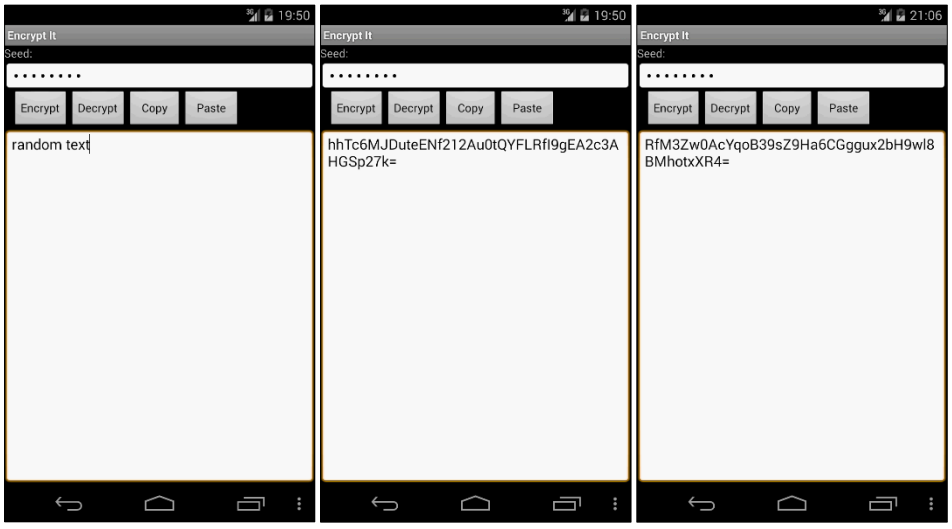
Πίνακας 3.1: Τιμές Κρυπτογράφησης Crypto Message

Το συμπέρασμα το οποίο εξάγεται από το παραπάνω πίνακα είναι ότι ο αλγόριθμος που χρησιμοποιείται είναι αλγόριθμος ροής καθώς παρατηρείται ότι το μέγεθος του κρυπτογράμματος είναι το ίδιο με το μήκος του απλού κειμένου.

3.3 Εφαρμογή Encrypt It

Η συγκεκριμένη επιτελεί κρυπτογράφηση/αποκρυπτογράφηση χωρίς να δίνει κάποια επιπλέον επιλογή στο χρήστη, και από την περιγραφή της στον ιστότοπο του Google Play[17] αναφέρεται ότι χρησιμοποιεί αλγόριθμο κρυπτογράφησης AES-256 καθώς και «αλάτι» για τη δημιουργία του κλειδιού.

Το γραφικό της περιβάλλον φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.5: Εφαρμογή Encrypt It

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, και του αρχικού κειμένου χωρίς να εμφανίζεται η τιμή του κρυπτογράμματος.

Επίσης από διάφορες δοκιμές που διεξήχθησαν (Εικόνα 3.2) παρατηρήθηκε ότι είσοδος απλού κειμένου, με τον ίδιο κωδικό παράγουν διαφορετικό κρυπτόγραμμα.
Αυτό οδηγεί στο συμπέρασμα ότι κάθε φορά που διενεργείται η λειτουργία της κρυπτογράφησης το «αλάτι» το οποίο δημιουργείται είναι και διαφορετικό, το οποίο έχει σαν αποτέλεσμα, την διαφορετική δημιουργία κλειδιού και κατ' επέκταση τη διαφορετική εμφάνιση κρυπτογράμματος.

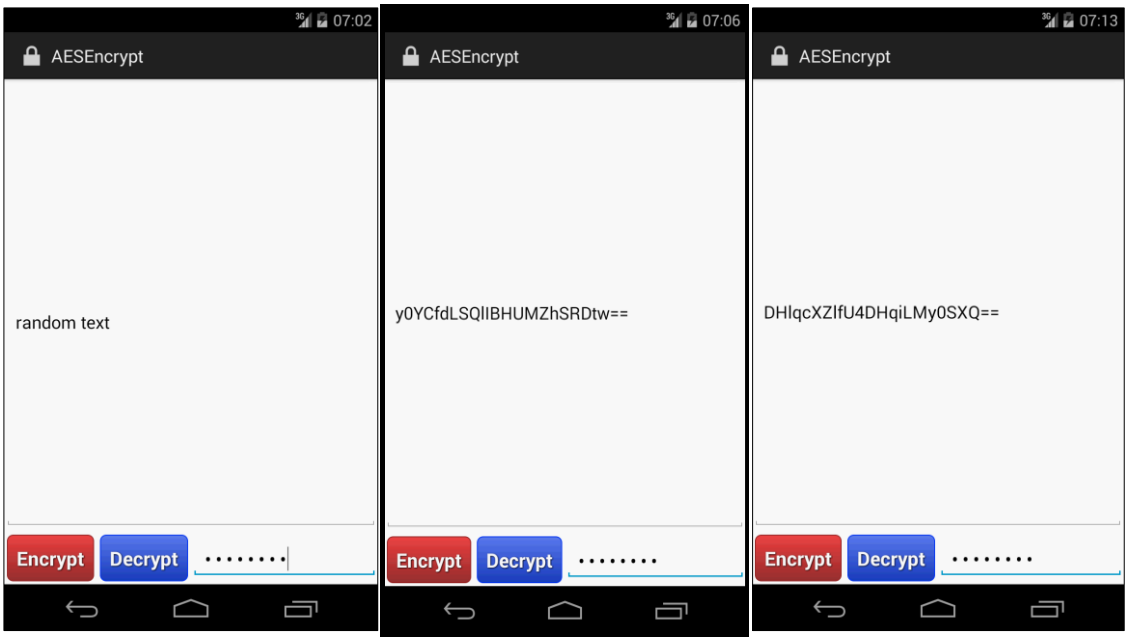
Επίσης βρέθηκαν οι εξής κλάσεις της java:

```
1. javax.crypto.Cipher
2. javax.crypto.SecretKey
3. javax.crypto.SecretKeySpec
4. javax.crypto.KeyGenarator
```

Παρόλαυτα σε καμία από τις παραπάνω κλάσεις δεν βρέθηκε κάποιο εύρημα αντίστοιχο με τα ευρήματα της εφαρμογής AES Android Forensics.
Επίσης παρατηρείται ότι για απλό κείμενο εισόδου 11 bytes (random text) το κρυπτόγραμμα που παράγεται δεν έχει μήκος 16 bytes αλλά 32 bytes το οποίο δεν συμβαδίζει με το μήκος τμήματος του AES.
Αντίστοιχα για είσοδο απλού κειμένου μήκους 23 bytes, ενώ το κρυπτόγραμμα του AES έχει μήκος 32 bytes η συγκεκριμένη εφαρμογή παράγει κρυπτόγραμμα μήκους 48 bytes.

3.4 Εφαρμογή AES Text Encryptor / Decryptor

Η συγκεκριμένη εφαρμογή επιτελεί κρυπτογράφηση/αποκρυπτογράφηση χωρίς να δίνει κάποια επιπλέον επιλογή στο χρήστη, και από την περιγραφή της στον ιστότοπο του Google Play[18] αναφέρεται ότι για τη λειτουργία της χρησιμοποιεί τον αλγόριθμο κρυπτογράφησης AES-256.
Το γραφικό της περιβάλλον φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.6: Εφαρμογή AES Text Encryptor / Decryptor

Από τις παραπάνω εικόνες εξάγεται το συμπέρασμα ότι χρησιμοποιείται κωδικοποίηση Base-64 και το κρυπτόγραμμα έχει μήκος 16 bytes άρα χρησιμοποιείται αλγόριθμος τμήματος 16 bytes.

Επίσης για το ίδιο απλό κείμενο με τον ίδιο κωδικό, παράγονται διαφορετικά κρυπτογράμματα.

Για να συμβαίνει αυτό είτε κάποια από τις μεταβλητές του κλειδιού («αλάτι», επαναλήψεις) μεταβάλλεται σε κάθε κρυπτογράφηση, είτε το διάνυσμα αρχικοποίησης του αλγόριθμου κρυπτογράφησης.

Η μεταβολή αυτή μπορεί να γίνεται με τη χρήση μιας γεννήτριας παραγωγής τυχαίων αριθμών `(SecureRandom)`.

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.

Επίσης βρέθηκαν οι εξής κλάσεις της java:

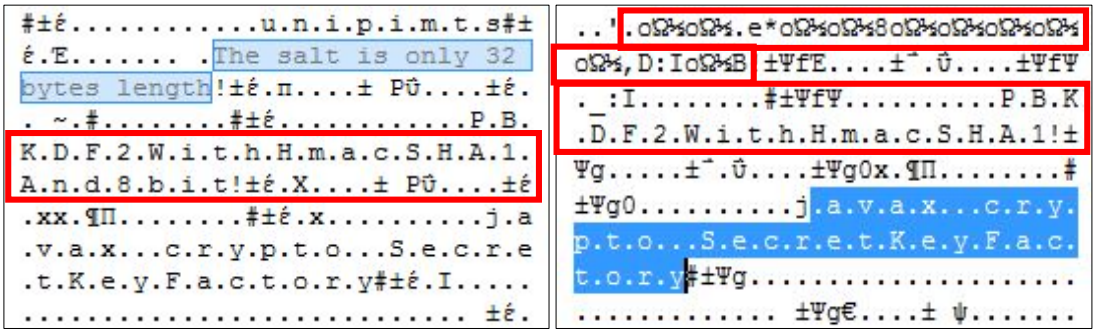
```
1. javax.crypto.SecretKeyFactory
2. javax.crypto.PBEKeySpec
3. javax.crypto.SecretKey
4. javax.crypto.SecretKeySpec
5. javax.crypto.Cipher
6. javax.crypto.IvParameterSpec
```

Οι παραπάνω κλάσεις ταυτίζονται με τις κλάσεις οι οποίες χρησιμοποιούνται στην εφαρμογή AES Android Forensics, και άμεσα προκύπτει το συμπέρασμα ότι η συγκεκριμένη εφαρμογή χρησιμοποιεί «αλάτι» και επαναλήψεις `(PBEKeySpec)`,

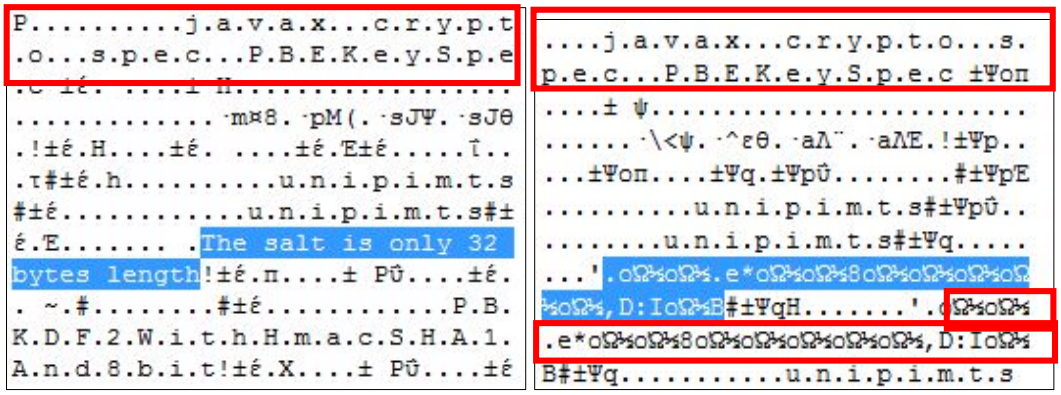
διάνυσμα αρχικοποίησης (IvParameterSpec) και δημιουργεί το κλειδί με συγκεκριμένο αλγόριθμο (SecretKeyFactory).

Με βάση την εφαρμογή AES Android Forensics παρατηρούμε για τη συγκεκριμένη εφαρμογή αντίστοιχα για την κλάση: SecretKeyFactory ότι χρησιμοποιεί και αυτή αντίστοιχα τον αλγόριθμο PBKDF2WithHmacSHA1, και ότι το «αλάτι» της είναι μήκους 40 bytes με τιμή: ".o½o½.e*o½o½8o½o½o½o½o½,D:Io½B".

Στις δύο παρακάτω εικόνες παρατίθενται τα ευρήματα της εφαρμογής δεξιά και στα αριστερά αυτά της AES Android Forensics. Το αλάτι επιβεβαιώνεται ότι είναι αυτό καθώς συναντάται και στην PBEKeySpec.

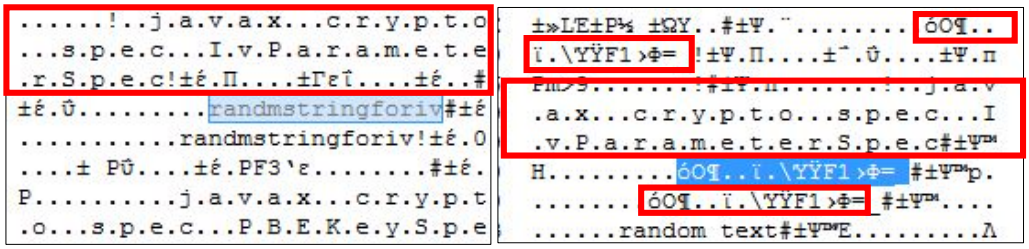


Εικόνα 3.7: «Αλάτι» και SecretKeyFactory



Εικόνα 3.8: «Αλάτι» και PBEKeySpec.

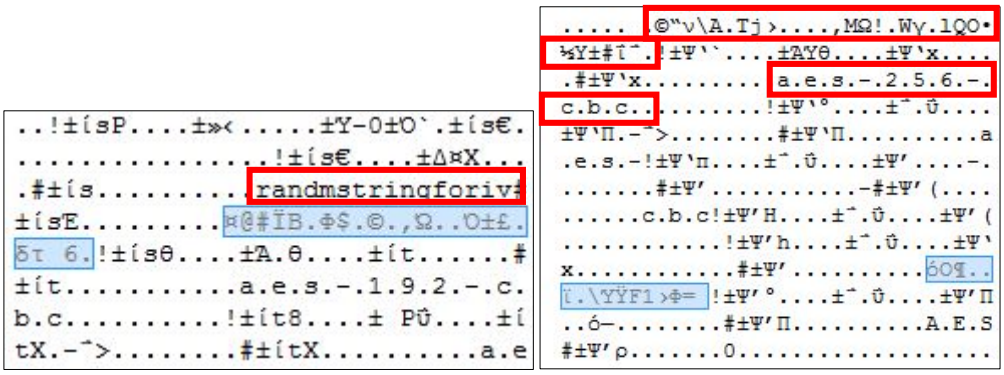
Αντίστοιχα για διάνυσμα αρχικοποίησης βρίσκουμε την τιμή: "óO¶..ï.\YŸF1›Φ=_" μήκους 16 bytes η οποία εμφανίζεται 3 φορές κοντά στην κλάση IvParameterSpec.



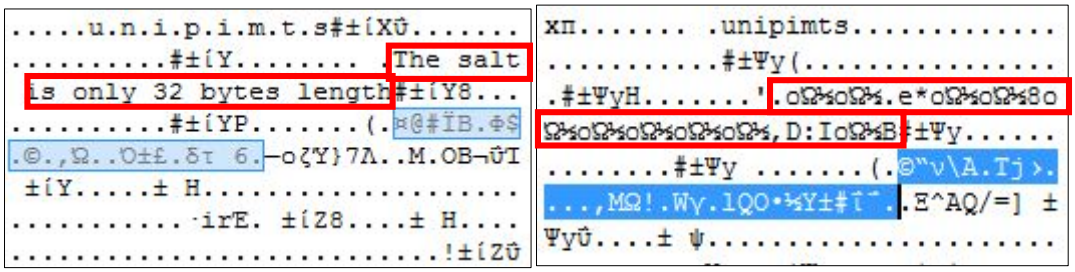
Εικόνα 3.9: Διάνυσμα αρχικοποίησης και IvParameterSpec

Η τέταρτη φορά που εμφανίζεται είναι κοντά στο offset του κλειδιού, το οποίο κλειδί είναι: ©"ν\A.Tj›....,MΩ!.Wγ.lQO•½Y±#î¯. μήκους 32 bytes.

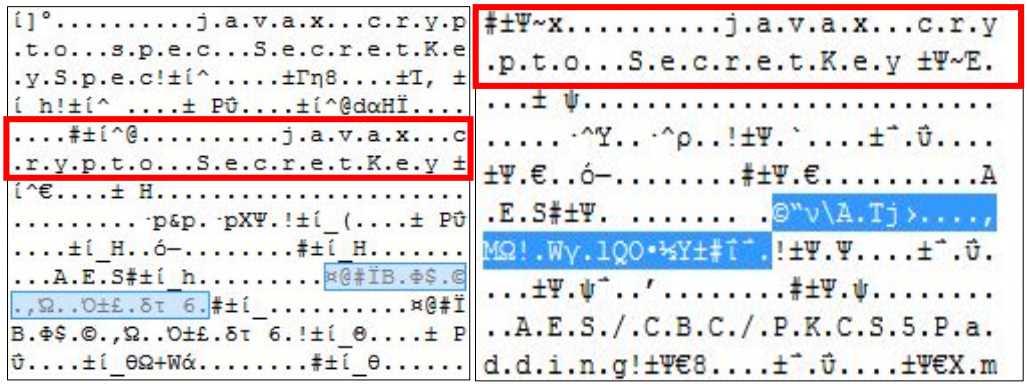
Επίσης και εδώ παρατηρούμε ότι εμφανίζεται ο τρόπος λειτουργίας του αλγόριθμου AES-256 CBC.



Εικόνα 3.10: Κλειδί και διάνυσμα αρχικοποίησης



Εικόνα 3.11: Κλειδί και «αλάτι»



Εικόνα 3.12: Κλειδί Κρυπτογράφησης και javax.crypto.SecretKey.

Από την αναζήτηση στο αποτύπωμα μνήμης βρέθηκαν όλες οι παράμετροι λειτουργίας του αλγόριθμου, αλγόριθμος δημιουργίας κλειδιού, κλειδί κρυπτογράφησης, τρόπος

λειτουργίας, αλάτι, εκτός από τις επαναλήψεις που δεν εμφανίζονται κάπου συγκεκριμένα στο αποτύπωμα μνήμης.

Τέλος για τη συγκεκριμένη εφαρμογή και με τις ίδιες τιμές απλού κειμένου και κωδικού, έγινε ανάλυση σε άλλο ένα αποτύπωμα μνήμης και με τη διαδικασία που ακολουθήθηκε παραπάνω παρατηρήθηκε ότι οι τιμές τόσο του διανύσματος αρχικοποίησης όσο και του «αλατιού» μεταβάλλονται, πιθανόν με τη χρήση της κλάσης `SecureRandom`.

Σε αυτή την εφαρμογή που είναι γνωστές όλες οι παραπάνω παράμετροι λειτουργίας δεν μπορεί να γίνει προσομοίωση σε περιβάλλον Java, καθώς η κλάση `SecureRandom` θα παράγει συνεχώς διαφορετικά IV, salt με αποτέλεσμα να μην μπορεί να ανακτηθεί το κρυπτόγραμμα που δημιουργήθηκε στην εφαρμογή, καθώς η εκεί SecureRandom θα έχει δημιουργήσει διαφορετικές τιμές για τα IV, και salt αντίστοιχα.

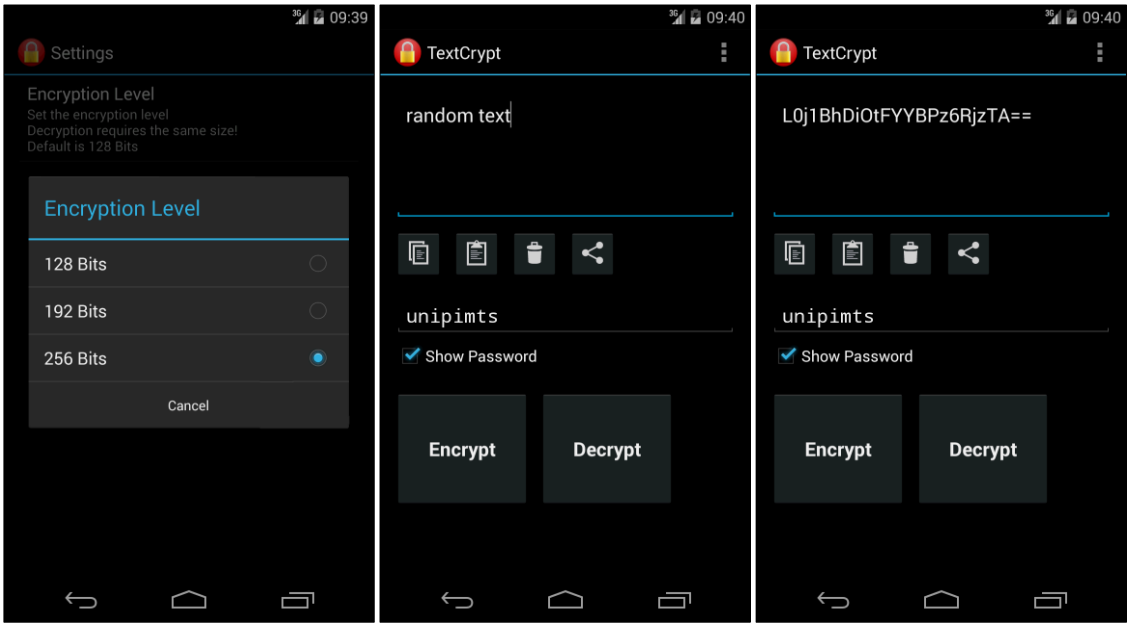
3.5 Εφαρμογή TextCrypt

Από την περιγραφή της συγκεκριμένης εφαρμογής στον ιστότοπο του Google Play[19] αναφέρεται ότι για τη λειτουργία της χρησιμοποιεί τον αλγόριθμο κρυπτογράφησης AES, δίνοντας στο χρήστη την επιλογή να διαλέξει μεταξύ των κλειδιών 256/192/128 bits.

Επιπλέον αναφέρεται ότι γίνεται χρήση «αλατιού» και διανύσματος αρχικοποίησης.

Στη συγκεκριμένη δοκιμή γίνεται επιλογή κλειδιού μήκους 256 bits.

Το γραφικό περιβάλλον της εφαρμογής φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.13: Εφαρμογή TextCrypt

Από τις παραπάνω εικόνες εξάγεται το συμπέρασμα ότι χρησιμοποιείται κωδικοποίηση Base-64 και το κρυπτόγραμμα έχει μήκος 16 bytes άρα χρησιμοποιείται αλγόριθμος τμήματος 16 bytes.
Επίσης για το ίδιο απλό κείμενο με τον ίδιο κωδικό, παράγεται το ίδιο κρυπτόγραμμα, γεγονός το οποίο σημαίνει ότι οι παράμετροι δημιουργίας του κρυπτογράμματος παραμένουν σταθεροί. («αλάτι», διάνυσμα αρχικοποίησης, αριθμός επαναλήψεων).

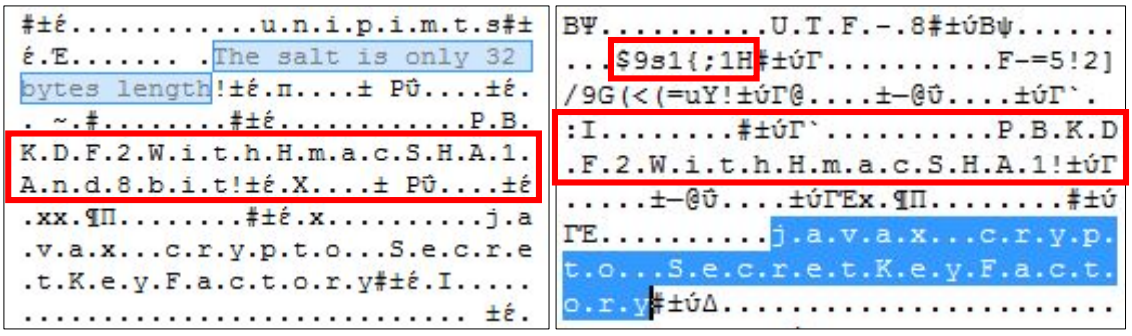
Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.
Επίσης βρέθηκαν οι εξής κλάσεις της java:

1. `javax.crypto.SecretKeyFactory`
2. `javax.crypto.PBEKeySpec`
3. `javax.crypto.SecretKey`
4. `javax.crypto.SecretKeySpec`
5. `javax.crypto.Cipher`
6. `javax.crypto.IvParameterSpec`

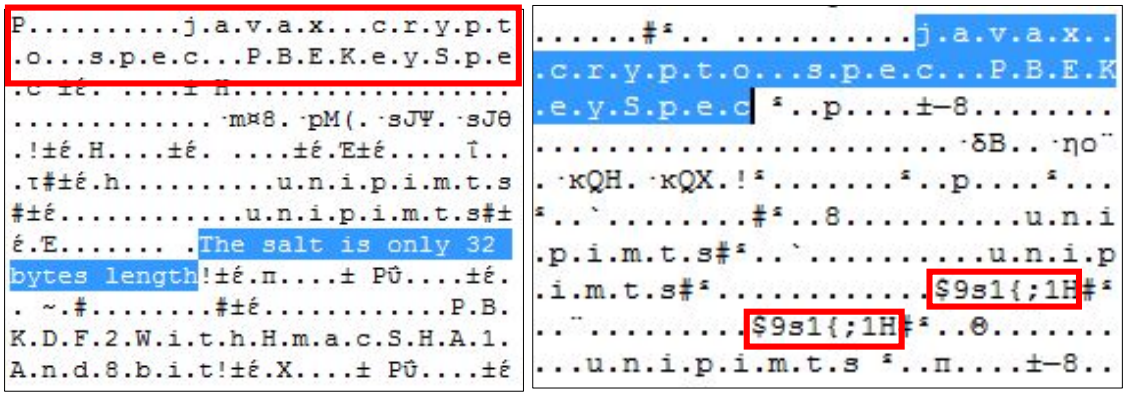
Οι παραπάνω κλάσεις εμφανίζονται τόσο στην εφαρμογή AES Android Forensics όσο και στην εφαρμογή AES Text Encryptor/Decryptor. Πιο συγκεκριμένα πλησιάζει πιο πολύ τον τρόπο λειτουργίας της AES Android Forensics καθώς χρησιμοποιεί σταθερές μεταβλητές για τη δημιουργία του κλειδιού.

Η ανάλυση του αποτυπώματος μνήμης βασίζεται στην εφαρμογή AES Android Forensics όπως αυτή περιγράφηκε στην ενότητα 3.4, και στις εικόνες που ακολουθούν, οι αριστερές παραθέτουν τα ευρήματα από την εφαρμογή AES Android Forensics ενώ οι δεξιές τα ευρήματα από την εφαρμογή TextCrypt.

Για την κλάση `javax.crypto.SecretKeyFactory` η εφαρμογή TextCrypt χρησιμοποιεί τον αλγόριθμο `PBKDF2WithHmacSHA1` όπως φαίνεται στην παρακάτω εικόνα.
Επίσης στην εικόνα αυτή εμφανίζεται και η ακολουθία `"$9s1{;1H"` μήκους 8 bytes η οποία αποδεικνύεται ότι είναι το αλάτι όπως φαίνεται και από την Εικόνα 3.14:

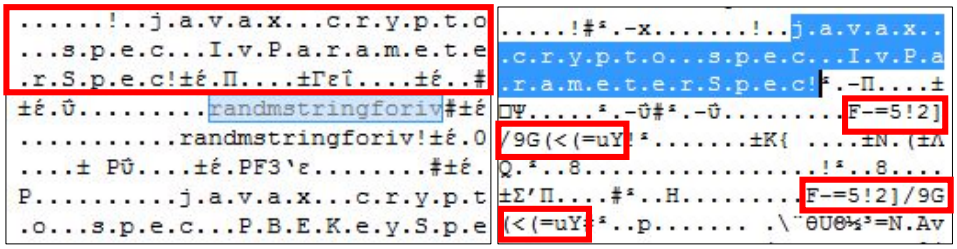


Εικόνα 3.14: «Αλάτι» και SecretKeyFactory



Εικόνα 3.15: «Αλάτι» και PBEKeySpec.

Αντίστοιχα για διάνυσμα αρχικοποίησης βρίσκουμε την τιμή: `"F-=5!2]/9G(<(=uY"` μήκους 16 bytes η οποία εμφανίζεται δύο φορές κοντά στην κλάση `IvParameterSpec`, όπως φαίνεται στην παρακάτω εικόνα, καθώς επίσης και μία φορά κοντά στην `SecretKeyFactory` όπως φαίνεται στην Εικόνα 3.15.



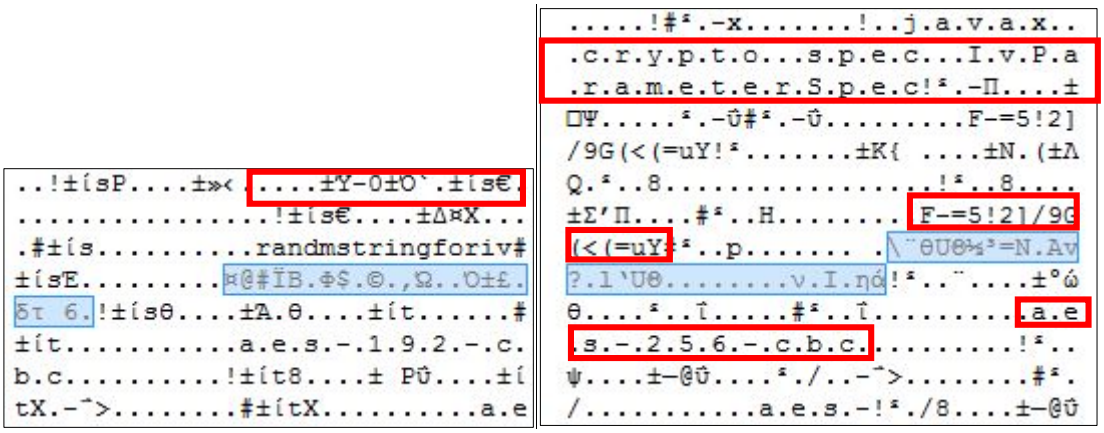
Εικόνα 3.16: Διάνυσμα αρχικοποίησης και IvParameterSpec

Όπως φάνηκε στις προηγούμενες αναλύσεις ένα εύρημα του κλειδιού βρίσκεται πολύ κοντά στο offset ενός διανύσματος αρχικοποίησης.

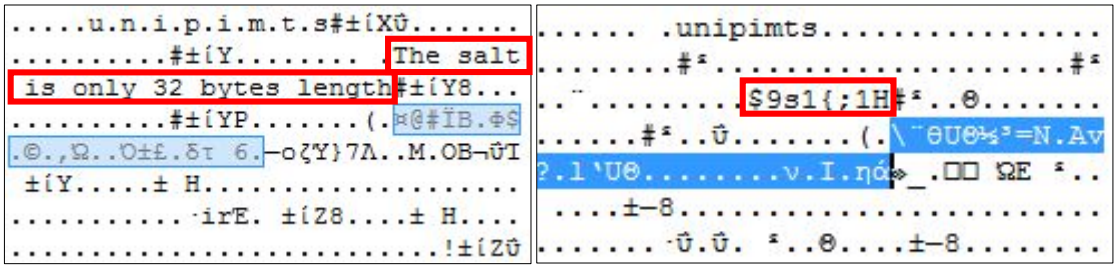
Στην συγκεκριμένη εφαρμογή το κλειδί κρυπτογράφησης το οποίο είναι η ακολουθία "\¨θUΘ½³=N.Av?.l'UΘ.....ν.I.ηά" μήκους 32 bytes βρίσκεται πολύ κοντά με το offset του δεύτερου διανύσματος αρχικοποίησης το οποίο βρίσκεται κοντά στην κλάση `IvParameterSpec`.

Σε συνέχεια των προηγούμενων αναλύσεων εμφανίζεται επίσης, σε κοντινά offset με το αλάτι καθώς επίσης και με τις κλάσεις `SecretKey`, `SecretKeysSpec`.

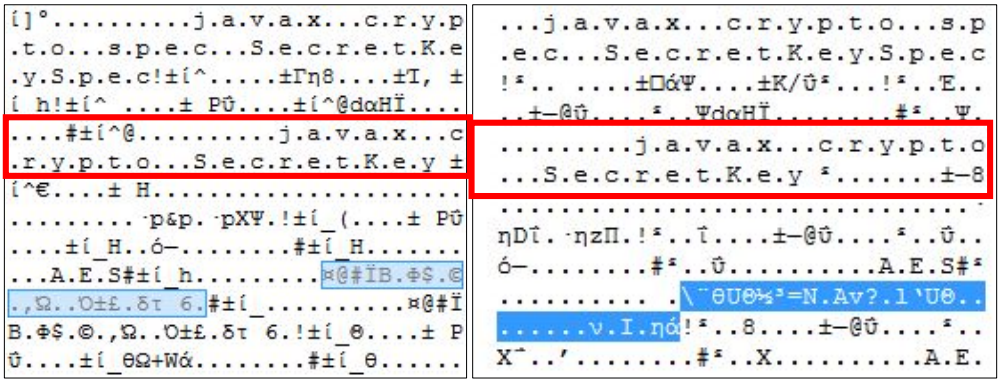
Τα ευρήματα του κλειδιού φαίνονται στις παρακάτω Εικόνες:



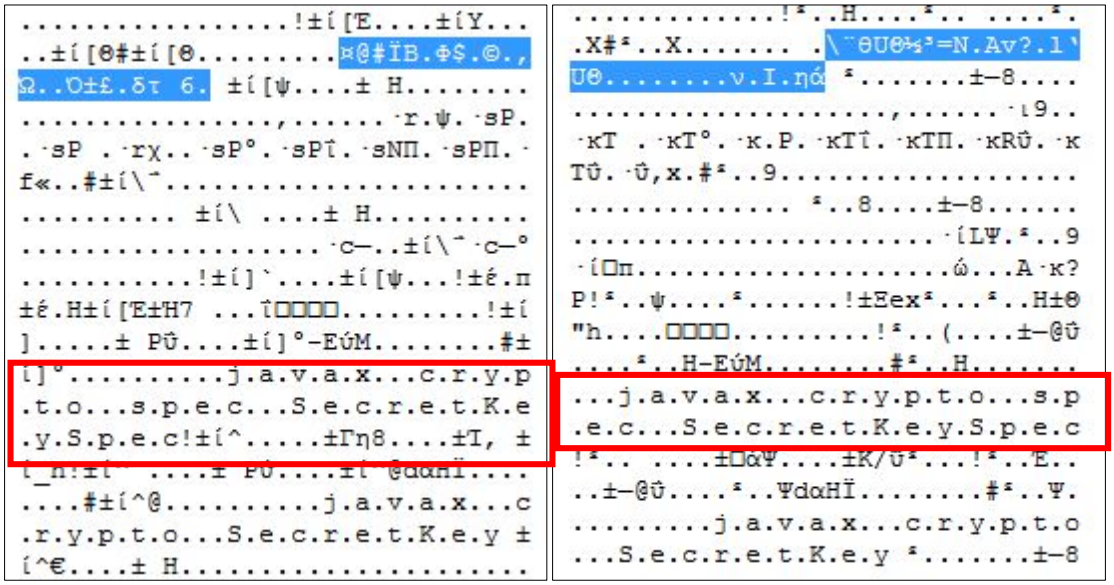
Εικόνα 3.17: Κλειδί και διάνυσμα αρχικοποίησης



Εικόνα 3.18: Κλειδί και «αλάτι»

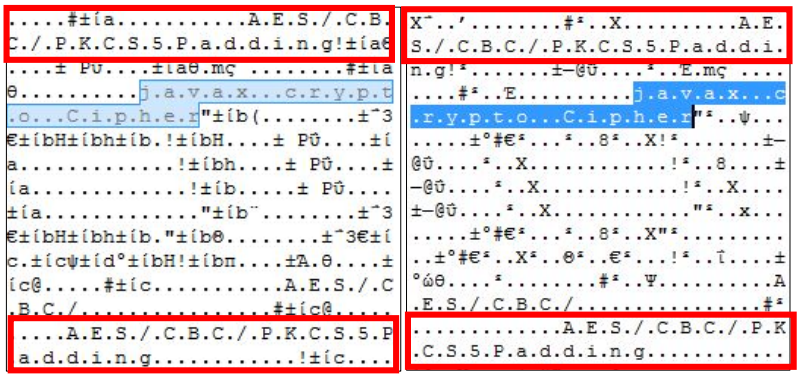


Εικόνα 3.19: Κλειδί και javax.crypto.SecretKey.



Εικόνα 3.20: Κλειδί και javax.crypto.SecretKeySpec

Τέλος από το εύρημα του `javax.crypto.Cipher` επιβεβαιώνεται ο τρόπος λειτουργίας του αλγόριθμου ο οποίος είναι `AES/CBC/PCKS5Padding` όπως φαίνεται στην παρακάτω εικόνα:



Εικόνα 3.21: Τρόπος λειτουργίας AES και javax.crypto.Cipher

Αντιστοίχως με την εφαρμογή AES Android Forensics η μόνη παράμετρος η οποία δεν μπόρεσε να βρεθεί στο αποτύπωμα μνήμης ήταν ο αριθμός των επαναλήψεων.

Για τον λόγο αυτό δημιουργήθηκε μια υλοποίηση σε περιβάλλον Java η οποία προσομοίωνε όλες τις παραμέτρους κρυπτογράφησης της υπό εξέταση εφαρμογής, και με τη μέθοδο δοκιμών διαπιστώθηκε ότι ο αριθμός επαναλήψεων ο οποίος χρησιμοποιείται είναι 5.

Η υλοποίηση αυτή παράγει τα ίδια κρυπτογράμματα/κλειδιά όταν δέχεται τον ίδιο κωδικό και απλό κείμενο με την εφαρμογή TextCrypt και παρουσιάζεται στο Παράρτημα Α.2.

Οι παράμετροι κρυπτογράφησης της εφαρμογής TextCrypt παρουσιάζονται στον παρακάτω πίνακα:

Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογής Encrypt	
Αλγόριθμος Κρυπτογράφησης	AES
Μήκος Κλειδιού	256/192/128 bits
Συνάρτηση Δημιουργίας κλειδιού (Key Derivation Function-KDF)	PBKDF2WithHmacSHA1
Τρόπος Λειτουργίας	Cipher Block Chaining (CBC)
Διάνυσμα Αρχικοποίησης (IV)	F-=5!2]/9G(<(=uY
«Αλάτι» (salt)	$9s1{;1H
Αριθμός Επαναλήψεων (Iterations)	5
Κωδικοποίηση Κρυπτογράμματος	Base-64

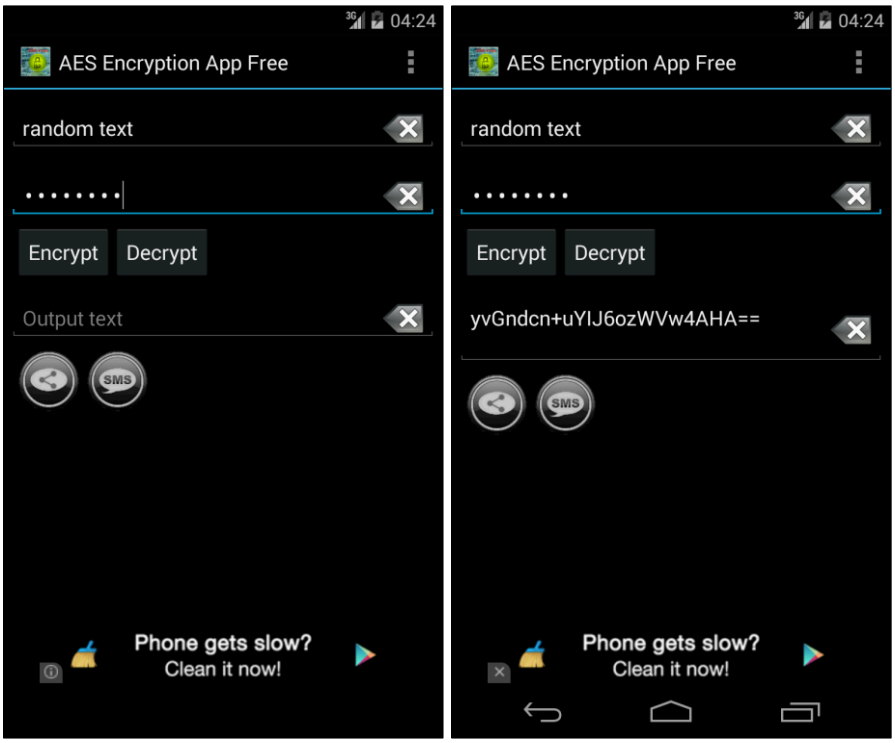
Java Classes	`javax.crypto.SecretKeyFactory`
	`javax.crypto.PBEKeySpec`
	`javax.crypto.SecretKey`
	`javax.crypto.SecretKeySpec`
	`javax.crypto.Cipher`
	`javax.crypto.IvParameterSpec`

Πίνακας 3.2 Παράμετροι Λειτουργίας Αλγόριθμου Κρυπτογράφησης Εφαρμογής TextCrypt.

3.6 Εφαρμογή AES Encryption App FREE

Από την περιγραφή της συγκεκριμένης εφαρμογής στον ιστότοπο του Google Play[20] αναφέρεται ότι για τη λειτουργία της χρησιμοποιεί τον αλγόριθμο κρυπτογράφησης AES, με τρόπο λειτουργίας CBC/PKCS5Padding, καθώς επίσης και ότι το κλειδί δημιουργείται από κατακερματισμό του κωδικού του χρήστη με τη βοήθεια της συνάρτησης SHA-256.

Το γραφικό περιβάλλον της εφαρμογής φαίνεται στις παρακάτω εικόνες:



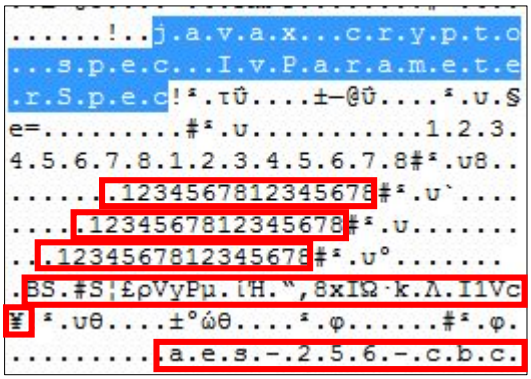
Εικόνα 3.22: Εφαρμογή AES Encryption App FREE

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.

Επίσης βρέθηκαν οι εξής κλάσεις της java:

1. `javax.crypto.IvParameterSpec`
2. `javax.crypto.SecretKeySpec`
3. `javax.crypto.Cipher`
4. `java.security.MessageDigest`
5. `java.security.SecureRandom`

Ακολουθώντας τις προηγούμενες αναλύσεις, σε κοντινό offset από αυτό που εμφανίζεται η κλάση `javax.crypto.IvParameterSpec` παρατηρούμε ότι εμφανίζεται η ακολουθία "`1234567812345678`" συνολικά τρεις φορές και με μήκος 16 bytes η οποία αντιπροσωπεύει και το διάνυσμα αρχικοποίησης.

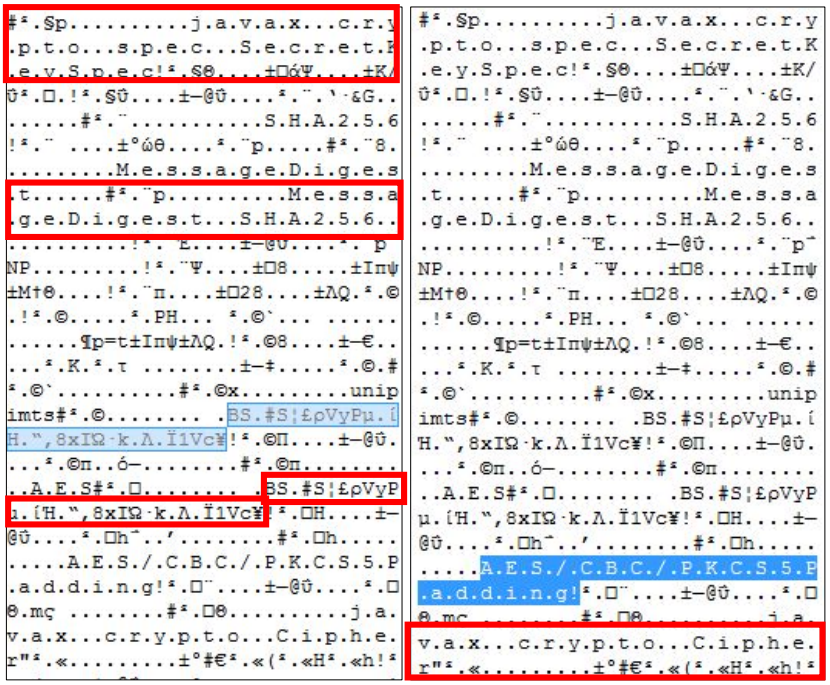


Εικόνα 3.23: Διάνυσμα αρχικοποίησης

Επίσης σε πολύ κοντινό offset με το τελευταίο διάνυσμα αρχικοποίησης στην παραπάνω εικόνα εμφανίζεται το κλειδί " BS.#S¦£ρVyPμ.ί'H.",8xIΩ·k.Λ.Ï1Vc¥" μήκους 32 bytes καθώς και ο τρόπος λειτουργίας του αλγόριθμου και το μήκος του κλειδιού που χρησιμοποιείται "aes-256-cbc".

Επίσης το κλειδί εμφανίζεται άλλες δύο φορές κοντά στην κλάση `javax.crypto.SecretKeySpec`, όπου σε κοντινό offset εμφανίζεται και η ακολουθία `MessageDigestSHA256` όπου επιβεβαιώνει ότι το κλειδί δημιουργείται από τη συνάρτηση κατακερματισμού SHA-256. Επίσης επιβεβαιώνεται και ο τρόπος λειτουργίας του αλγόριθμου από την κλάση `javax.crypto.cipher` καθώς σε κοντινό offset με αυτό το εύρημα εμφανίζεται η ακολουθία: `AES/CBC/PCKS5Padding`.

Τα ευρήματα αυτά φαίνονται στην παρακάτω εικόνα:



Εικόνα 3.24 Κλειδί και Τρόπος Λειτουργίας

Τα παραπάνω ευρήματα επαληθεύθηκαν με υλοποίηση των παραμέτρων αυτών σε περιβάλλον Java. Η υλοποίηση αυτή παράγει τα ίδια κρυπτογράμματα/κλειδιά όταν δέχεται τον ίδιο κωδικό και απλό κείμενο με την εφαρμογή AES Encryption App FREE, και παρουσιάζεται στο Παράρτημα Α.2.

Οι παράμετροι κρυπτογράφησης της εφαρμογής TextCrypt παρουσιάζονται στον παρακάτω πίνακα:

Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογής AES Encryption App FREE	
Αλγόριθμος Κρυπτογράφησης	AES
Μήκος Κλειδιού	256bits
Συνάρτηση Δημιουργίας κλειδιού (Key Derivation Function-KDF)	SHA-256
Τρόπος Λειτουργίας	Cipher Block Chaining (CBC)
Διάνυσμα Αρχικοποίησης (IV)	1234567812345678
Κωδικοποίηση Κρυπτογράμματος	Base-64

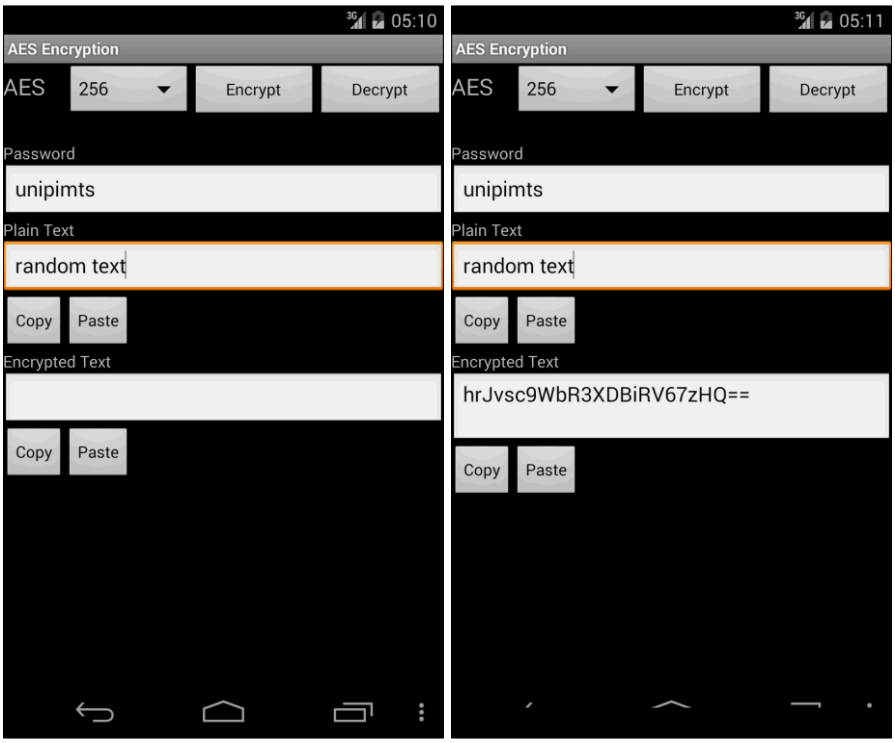
Java Classes	`javax.crypto.IvParameterSpec`
	`javax.crypto.SecretKeySpec`
	`javax.crypto.Cipher`
	`java.security.MessageDigest`
	`java.security.SecureRandom`

Πίνακας 3.3 Παράμετροι Λειτουργίας Αλγόριθμου Κρυπτογράφησης Εφαρμογής AES Encryption App FREE.

3.7 Εφαρμογή AES Message Encryptor/Aescryption

Από την περιγραφή της συγκεκριμένης εφαρμογής στον ιστότοπο του Google Play[21] αναφέρεται ότι γίνεται χρήση του αλγορίθμου κρυπτογράφησης AES, και δίνεται στο χρήστη η επιλογή κλειδιού μήκους 256/192/128 bits. Στη συγκεκριμένη δοκιμή γίνεται επιλογή του κλειδιού μήκους 256 bits.

Το γραφικό περιβάλλον της εφαρμογής φαίνεται στις παρακάτω εικόνες:



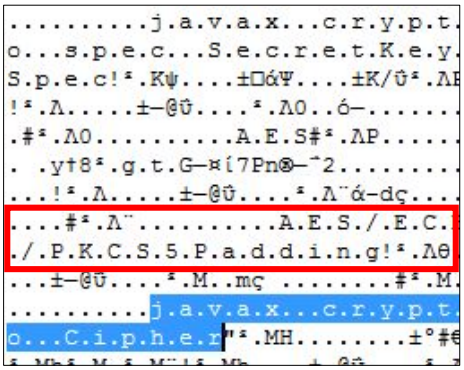
Εικόνα 3.25 Εφαρμογή AES Message Encryptor

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.

Επίσης βρέθηκαν οι εξής κλάσεις της java:

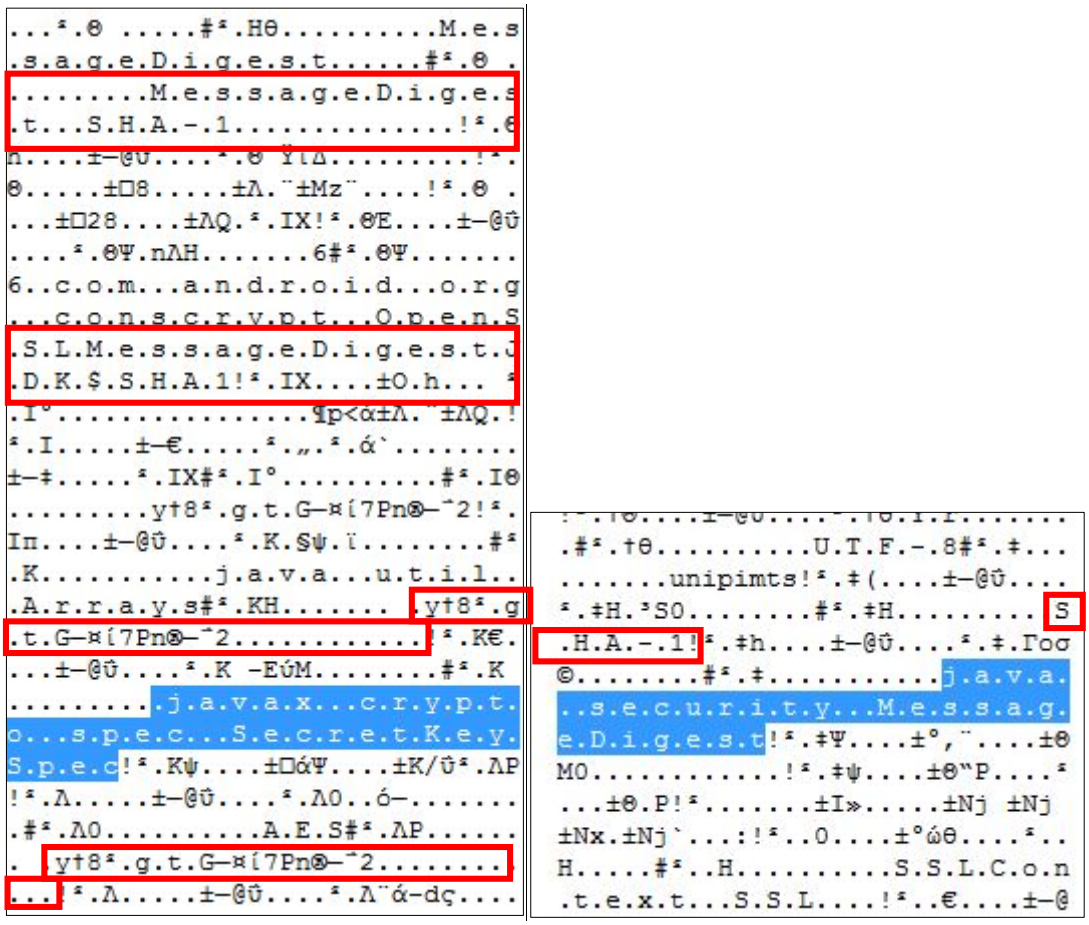
1. `javax.crypto.SecretKeySpec`
2. `javax.crypto.Cipher`
3. `java.security.MessageDigest`

Ξεκινώντας την ανάλυση από το εύρημα `javax.crypto.Cipher` παρατηρούμε ότι σε κοντινή απόσταση από αυτό εμφανίζεται η ακολουθία AES/ECB/PCKS5Padding, γεγονός το οποίο συμβαδίζει με την μη ύπαρξη της κλάσης `IvParameterSpec` καθώς η λειτουργία ECB δεν χρησιμοποιεί διάνυσμα αρχικοποίησης.



Εικόνα 3. 26

Όσον αφορά το εύρημα `javax.crypto.SecretKeySpec` παρατηρούμε ότι σε κοντινή απόσταση από αυτό υπάρχουν δύο ακολουθίες οι οποίες αναφέρουν τη χρησιμοποίηση συνάρτησης κατακερματισμού SHA-1. Το γεγονός αυτό ενισχύεται από την ύπαρξη της ακολουθίας `SHA-1` σε κοντινό offset με το εύρημα `java.Security.MessageDigest`.
Τα ευρήματα αυτά φαίνονται στις παρακάτω εικόνες:

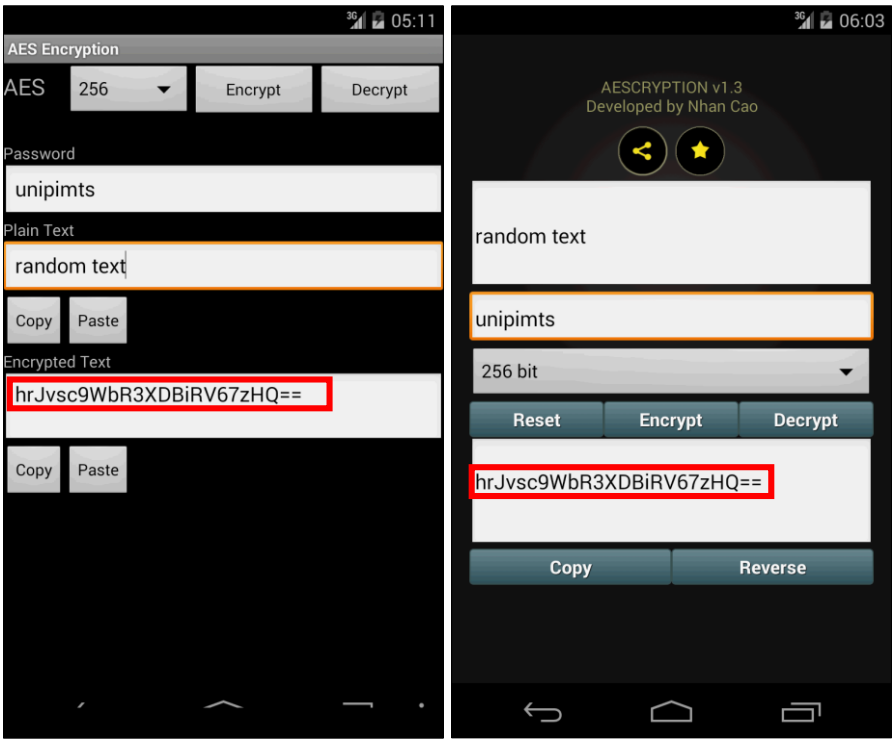


Εικόνα 3.27: Χρησιμοποίηση SHA-1

Με χρήση της SHA-1 σε περιβάλλον Java έγινε κατακερματισμός του κωδικού του χρήστη και προέκυψε η τιμή: "`y†8².g.t.G–¤{7Pn®–¨2`" μήκους 20 bytes.
Η συγκεκριμένη τιμή βρίσκεται τρεις φορές στο αποτύπωμα μνήμης, όπου και στις τρεις αυτές φορές ακολουθείται από 12 μηδενικά, για να δημιουργηθεί η ακολουθία `y†8².g.t.G–¤{7Pn®–¨2.....` μήκους 32 bytes, όσο είναι δηλαδή το κλειδί κρυπτογράφησης που επιλέχθηκε για τη συγκεκριμένη δοκιμή.
Επιπλέον τα δύο από αυτά τα ευρήματα βρίσκονται σε κοντινά offset με το εύρημα `javax.crypto.SecretKeySpec` όπως φαίνεται στην παραπάνω εικόνα.

Αυτό μας οδηγεί στο συμπέρασμα ότι αυτό είναι και το κλειδί κρυπτογράφησης το οποίο χρησιμοποιείται στη συγκεκριμένη εφαρμογή.

Επίσης εξετάζοντας την εφαρμογή AesCryption[22], με τον τρόπο ο οποίος περιγράφεται σε αυτή την ενότητα αποδείχτηκε ότι ο τρόπος λειτουργίας της είναι ακριβώς ο ίδιος με αυτή της AES Message Encryptor.
Αυτό φαίνεται και στις παρακάτω εικόνες όπου με ίδιες τιμές απλού κειμένου, μήκος κλειδιού και κωδικού το κρυπτόγραμμα που παράγεται είναι το ίδιο και στις δύο εφαρμογές!



Εικόνα 3.28: Εφαρμογή Aescryption

Τέλος τα παραπάνω ευρήματα επαληθεύθηκαν με υλοποίηση των παραμέτρων αυτών σε περιβάλλον Java. Η υλοποίηση αυτή παράγει τα ίδια κρυπτογράμματα/κλειδιά όταν δέχεται το ίδιο απλό κείμενο, το ίδιο μήκος κλειδιού, και τον ίδιο κωδικό τόσο για την εφαρμογή AES Message Encryptor όσο και για την εφαρμογή Aescryption.

Οι παράμετροι κρυπτογράφησης για τις δύο εφαρμογές παρουσιάζονται στον παρακάτω πίνακα, και η υλοποίησή τους παρουσιάζεται στο Παράρτημα Α.2.

Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογών AES Message Encryptor / Aescryption	
Αλγόριθμος Κρυπτογράφησης	AES
Μήκος Κλειδιού	256/192/128bits
Συνάρτηση Δημιουργίας κλειδιού (Key Derivation Function-KDF)	SHA-1
Τρόπος Λειτουργίας	Electronic Codebook (ECB)
Κωδικοποίηση Κρυπτογράμματος	Base-64

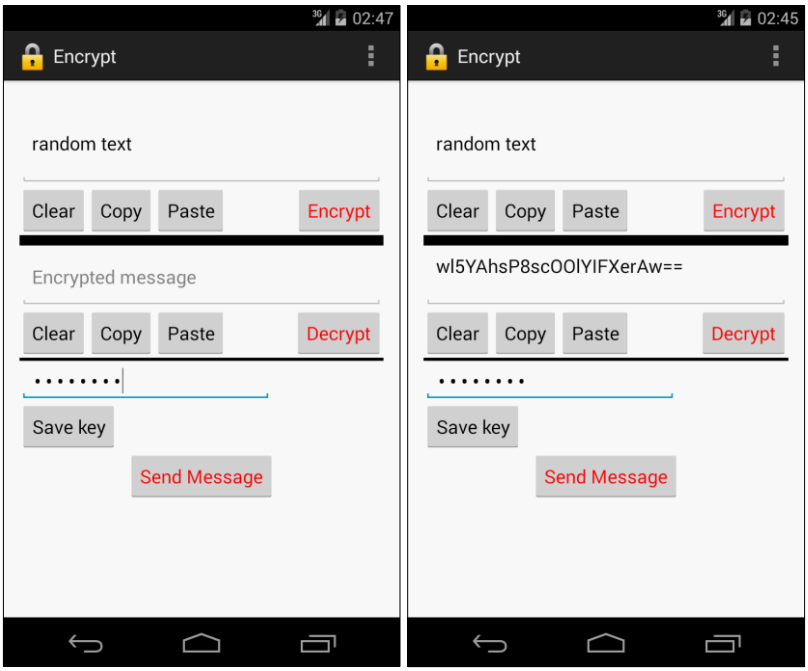
Java Classes	`javax.crypto.SecretKeySpec`
	`javax.crypto.Cipher`
	`java.security.MessageDigest`

Πίνακας 3.4 Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογών AES Message Encryptor / Aescryption

3.8 Εφαρμογή Encrypt

Η συγκεκριμένη επιτελεί κρυπτογράφηση/αποκρυπτογράφηση χωρίς να δίνει κάποια επιπλέον επιλογή στο χρήστη, και από την περιγραφή της στον ιστότοπο του Google Play[23] το μόνο που αναφέρεται είναι ότι ο αλγόριθμος κρυπτογράφησης που χρησιμοποιείται είναι συμμετρικός.

Το γραφικό της περιβάλλον φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.29: Εφαρμογή Encrypt

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.

Επίσης βρέθηκαν οι εξής κλάσεις της java:

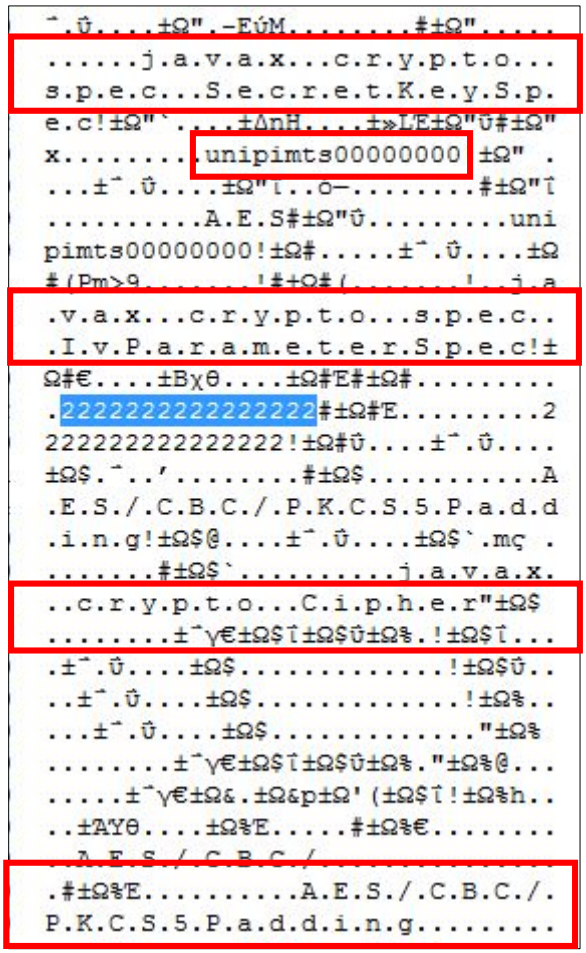
1. `javax.crypto.Cipher`
2. `javax.crypto.SecretKeySpec`
3. `javax.crypto.IvParameterSpec`
4.

Στην Εικόνα 3.30 παρατηρούμε ότι και οι τρεις αυτές κλάσεις οι οποίες εμφανίζονται από μία φορά η κάθε μια στο αποτύπωμα μνήμης, βρίσκονται σε πολύ κοντινά offset μεταξύ τους.

Για το εύρημα `javax.crypto.IvParameterSpec` παρατηρείται ότι σε πολύ κοντινό offset με αυτό βρίσκεται μια ακολουθία μήκους 16 bytes και η οποία είναι η «`2222222222222222`».
Οπότε μπορεί να εξαχθεί το συμπέρασμα ότι αυτό είναι το διάνυσμα αρχικοποίησης.

Το γεγονός αυτό ενισχύεται καθώς σε κοντινό offset από την `javax.crypto.Cipher` εμφανίζεται η ακολουθία `AES_CBC_PKCS5_Padding` από την οποία συμπεραίνεται ότι ο αλγόριθμος χρησιμοποιεί τον τρόπο λειτουργίας CBC ο οποίος για την εφαρμογή του χρησιμοποιεί διάνυσμα αρχικοποίησης.

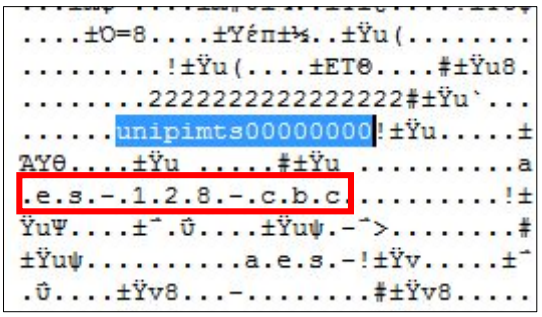
Επίσης παρατηρείται ότι ο κωδικός που δόθηκε έχει συμπληρωθεί με 8 επιπλέον bytes για να φτάσει το μέγεθος των 16 bytes, "`unipimts00000000`" όπως φαίνεται και στην παρακάτω εικόνα:



Εικόνα 3.30: Java Classes στην εφαρμογή Encrypt

Επίσης όπως φαίνεται και στην Εικόνα 2.27 για την εφαρμογή AES Android Forensics παρατηρείται ότι το κλειδί βρίσκεται σε πολύ κοντινό offset με ένα εύρημα του διανύσματος αρχικοποίησης.

Αντίστοιχα στην εφαρμογή Encrypt παρατηρούμε στην παρακάτω εικόνα ότι σε κοντινό offset από το διάνυσμα αρχικοποίησης βρίσκεται ο κωδικός "`unipimts00000000`".



Εικόνα 3.31 Κλειδί Κρυπτογράφησης και Διάνυσμα αρχικοποίησης

Αν το εύρημα αυτό συνδυαστεί με το γεγονός ότι ο κωδικός βρίσκεται και σε κοντινό offset με το `javax.crypto.SecretKeySpec` όπως φαίνεται στην Εικόνα 3.30 τότε μπορεί να εξαχθεί το συμπέρασμα ότι το κλειδί του AES είναι ο κωδικός του χρήστη συμπληρωμένος με μηδενικά μέχρι να αποκτήσει μήκος 16 bytes όσο δηλαδή ένα κλειδί του AES 128 bits.

Επίσης το συμπέρασμα αυτό ενισχύεται περαιτέρω καθώς στην εφαρμογή Encrypt ο χρήστης δεν μπορεί να δώσει κενό κωδικό αλλά ούτε και κωδικό με μήκος μεγαλύτερο από 16 bytes.

Τέλος τα παραπάνω ευρήματα επαληθεύθηκαν με υλοποίηση των παραμέτρων αυτών σε περιβάλλον Java. Η υλοποίηση αυτή παράγει τα ίδια κρυπτογράμματα/κλειδιά όταν δέχεται το ίδιο απλό κείμενο, και τον ίδιο κωδικό τόσο για την εφαρμογή Encrypt, και παρουσιάζεται στο Παράρτημα Α.2.

Οι παράμετροι κρυπτογράφησης παρουσιάζονται στον παρακάτω πίνακα:

Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογής Encrypt	
Αλγόριθμος Κρυπτογράφησης	AES
Μήκος Κλειδιού	128 bits
Συνάρτηση Δημιουργίας κλειδιού (Key Derivation Function-KDF)	Κωδικός του χρήστη συμπληρωμένος με μηδενικά μέχρι την τιμή των 16 bytes
Τρόπος Λειτουργίας	Cipher Block Chaining (CBC)
Διάνυσμα Αρχικοποίησης (IV)	2222222222222222
Κωδικοποίηση Κρυπτογράμματος	Base-64

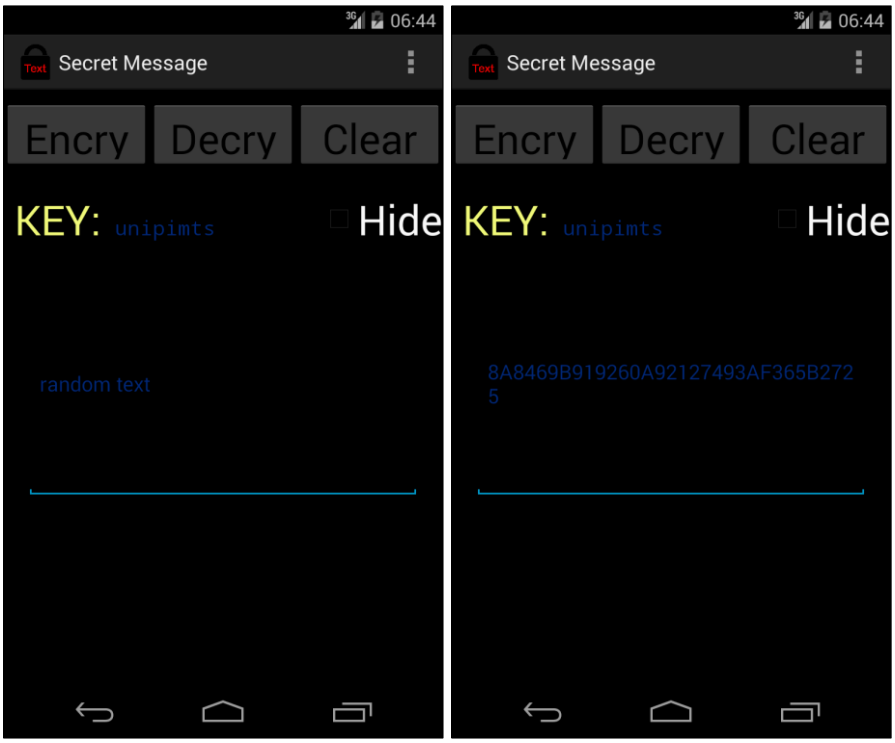
Java Classes	`javax.crypto.Cipher`
	`javax.crypto.SecretKeySpec`
	`javax.crypto.IvParameterSpec`

Πίνακας 3.5: Παράμετροι Λειτουργίας Αλγόριθμου Κρυπτογράφησης Εφαρμογής Encrypt

3.9 Εφαρμογή Secret Message

Η συγκεκριμένη επιτελεί κρυπτογράφηση/αποκρυπτογράφηση χωρίς να δίνει κάποια επιπλέον επιλογή στο χρήστη, και από την περιγραφή της στον ιστότοπο του Google Play[24] αναφέρεται είναι ότι ο αλγόριθμος κρυπτογράφησης που χρησιμοποιείται είναι ο AES.

Το γραφικό της περιβάλλον φαίνεται στις παρακάτω εικόνες:



Εικόνα 3.32: Εφαρμογή Secret Message

Από την εξέταση του αποτυπώματος μνήμης η οποία έγινε βρέθηκαν οι τιμές του κωδικού, του αρχικού κειμένου και του κρυπτογράμματος.

Επίσης βρέθηκαν οι εξής κλάσεις της java:

1. `javax.crypto.Cipher`
2. `javax.crypto.SecretKeySpec`
3. `java.security.MessageDigest`

Από το εύρημα `javax.crypto.Cipher` παρατηρείται ότι σε κοντινό offset βρίσκεται η ακολουθία AES/ECB/PKCS5Padding από το οποίο εξάγεται η λειτουργία του αλγόριθμου.

Επίσης σε κοντινό offset από το εύρημα `javax.crypto.SecretKeySpec` παρατηρείται ότι εμφανίζεται ο κωδικός που δόθηκε συμπληρωμένος με μηδενικά, `"unipimts00000000000000000000000000"` και έχει μήκος 32 bytes, όσο δηλαδή το μήκος ενός 256 bit κλειδιού του AES. Επίσης το εύρημα αυτό εμφανίζεται και άλλη μια φορά στο αποτύπωμα μνήμης όπου σε κοντινό offset εμφανίζεται το μήκος του κλειδιού και ο τρόπος λειτουργίας του αλγόριθμου κρυπτογράφησης.

Τα ευρήματα φαίνονται στην παρακάτω εικόνα:

Εικόνα 3.33: Κλειδί Κρυπτογράφησης και τρόπος λειτουργίας

Τα παραπάνω ευρήματα επαληθεύθηκαν με υλοποίηση των παραμέτρων αυτών σε περιβάλλον Java. Η υλοποίηση αυτή παράγει τα ίδια κρυπτογράμματα/κλειδιά όταν δέχεται το ίδιο απλό κείμενο, και τον ίδιο κωδικό τόσο για την εφαρμογή Secret Message, και παρουσιάζεται στο Παράρτημα Α.2.

Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογής Secret Message	
Αλγόριθμος Κρυπτογράφησης	AES
Μήκος Κλειδιού	256 bits
Συνάρτηση Δημιουργίας κλειδιού (Key Derivation Function-KDF)	Κωδικός του χρήστη συμπληρωμένος με μηδενικά μέχρι την τιμή των 32 bytes
Τρόπος Λειτουργίας	Cipher Block Chaining (CBC)
Κωδικοποίηση Κρυπτογράμματος	Δεκαεξαδική Μορφή
Java Classes	`javax.crypto.Cipher` `javax.crypto.SecretKeySpec`

Πίνακας 3.6: Παράμετροι Αλγόριθμου Κρυπτογράφησης εφαρμογής Secret Message

ΚΕΦΑΛΑΙΟ 4

ΑΝΑΛΥΣΗ ΑΠΟΤΥΠΩΜΑΤΩΝ ΜΝΗΜΗΣ ΜΕ ΤΗ ΧΡΗΣΗ ΔΥΑΔΙΚΗΣ ΕΝΤΡΟΠΙΑΣ

Στο προηγούμενο κεφάλαιο το εκάστοτε αποτύπωμα μνήμης το οποίο δημιουργήθηκε, αναλύθηκε γνωρίζοντας εκ των προτέρων τα εξής δεδομένα:

1. Την εφαρμογή από την οποία δημιουργείται
2. Τον κωδικό που δίνει ο χρήστης.
3. Το απλό κείμενο/Κρυπτόγραμμα που δίνεται
4. Το μήκος του κλειδιού. (σε ορισμένες εφαρμογές)

Επίσης για την εύρεση των παραμέτρων λειτουργίας του αλγόριθμου κρυπτογράφησης (διάνυσμα αρχικοποίησης, «αλάτι» κτλ.) κάθε εφαρμογής έπρεπε να ακολουθηθεί μια διαδικασία η οποία περιελάβανε δύο στάδια:

1. Την εύρεση των παραμέτρων έπειτα από εξέταση του αποτυπώματος μνήμης.
2. Την πιστοποίηση της ορθότητας των παραπάνω παραμέτρων με προσομοίωση της λειτουργίας της εκάστοτε εφαρμογής σε περιβάλλον Java.

Καθώς η παραπάνω διαδικασία είναι αρκετά χρονοβόρα, δημιουργήθηκε ένα πρόγραμμα σε περιβάλλον Java `(Entropy_AES_Key)`, βασιζόμενο στην δυαδική εντροπία καθώς επίσης και στα συμπεράσματα τα οποία προέκυψαν από την ανάλυση των αποτυπωμάτων μνήμης των εφαρμογών του Κεφαλαίου 3.

Κύριος σκοπός του συγκεκριμένου προγράμματος είναι η εύρεση του κλειδιού κρυπτογράφησης (αν αυτό υπάρχει) σε ένα αποτύπωμα μνήμης χωρίς να υπάρχει κάποια πληροφορία για το κλειδί ή για τη λειτουργία κρυπτογράφησης που επιτελείται.

Στις επόμενες ενότητες παρουσιάζεται ο τρόπος λειτουργίας του συγκεκριμένου προγράμματος.

4.1 Δυαδική εντροπία

Στη θεωρία πληροφορίας η δυαδική εντροπία ($H_b(p)$) μας δίνει το μέγεθος της αβεβαιότητας ή της «αταξίας» το οποίο υπάρχει σε ένα μήνυμα και δίνεται από τον τύπο[25]:

$$H(X) = H_b(p) = -p \log_2 p - (1-p) \log_2 (1-p).$$

Εικόνα 4.1: Υπολογισμός Δυαδικής Εντροπίας.

Στον παραπάνω τύπο το p συμβολίζει την πιθανότητα δημιουργίας των ψηφίων (bits) 0 (p_0),1 (p_1) τα οποία υπάρχουν σε μια δυαδική ακολουθία. Στην περίπτωση που η πιθανότητα δημιουργίας του ψηφίου 1 σε μια δυαδική ακολουθία λαμβάνει την τιμή 0 (p_1=0, p_0=1) τότε αυτό σημαίνει ότι στο μήνυμα το οποίο θα δημιουργηθεί (0000) δεν θα εμφανιστεί το ψηφίο 1 καμία φορά οπότε δεν υπάρχει και κάποια αβεβαιότητα, και άρα η τιμή της εντροπίας θα λάβει την τιμή 0.

Το ίδιο θα συμβεί αν η πιθανότητα δημιουργίας του ψηφίου 1 λαμβάνει την τιμή 1 (p_1=1) . Σε αυτή την περίπτωση το μήνυμα (1111) που θα δημιουργηθεί δεν θα περιλαμβάνει αντίστοιχα το ψηφίο 0, και πάλι δεν θα υπάρχει κάποια αβεβαιότητα στην ακολουθία, με αποτέλεσμα η εντροπία να λαμβάνει την τιμή 0.

Στην περίπτωση όμως όπου η πιθανότητα δημιουργίας του ψηφίου 1 είναι 0.5, και αντίστοιχά του 0 είναι και αυτή 0.5 τότε η αβεβαιότητα λαμβάνει την μέγιστη τιμή (1.0) της καθώς η δημιουργία είτε του ψηφίου 1 ή 0 είναι ισοπίθανη, και μπορεί να οδηγήσει στην δημιουργία ενός μηνύματος στο οποίο τα ψηφία 1 και 0 θα είναι ισόποσα (1010).

Όσον αφορά τα κλειδιά κρυπτογράφησης η εντροπία είναι εξίσου σημαντική με το μήκος του κλειδιού. Αν τα bits του κλειδιού δεν δημιουργούνται με ένα μεγάλο βαθμό τυχαιότητας, τότε το κλειδί θα είναι πιο ευάλωτο σε επιθέσεις καθώς θα υπάρχει κάποιος πιο πιθανός τρόπος δημιουργίας τους, από τον οποίο ο επιτιθέμενος θα μπορεί να εξάγει συμπεράσματα για την τιμή του κλειδιού.

Χαρακτηριστικό παράδειγμα είναι το κλειδί του DES (Data Encryption Standard) όπου ενώ έχει μήκος 64 bits, τα 8 από αυτά χρησιμοποιούνται για έλεγχο ισοτιμίας (parity bits) και καθώς δεν δημιουργούνται τυχαία το μήκος του κλειδιού του του DES μειώνεται στα 56 bits.

Αντίστοιχα για ένα κλειδί του AES μήκους 128 bits, το οποίο δημιουργείται με τυχαίο τρόπο, δεν μπορεί να προβλεφθεί κανένα από τα bits του, καθώς τα bits 0, και 1 έχουν την ίδια πιθανότητα δημιουργίας.

4.2 Δυαδική εντροπία κλειδιού AES

Για τη δημιουργία του συγκεκριμένου προγράμματος πρώτα έγινε ανάλυση των τιμών της εντροπίας που λαμβάνουν τα κλειδιά του AES. Για τον λόγο αυτό δημιουργήθηκε μια υλοποίηση σε Java με την ονομασία `Check_AES_Entropy` της οποίας τα αποτελέσματα παρουσιάζονται συνοπτικά σε αυτή την ενότητα.

Το συγκεκριμένο πρόγραμμα δημιουργεί κλειδιά του AES 256/192/128 είτε με PBDFK2 είτε με SHA-256, είτε με SHA-1 με βάση έναν τυχαίο κωδικό ο οποίος παράγεται κάθε φορά από μια ψευδοτυχαία συνάρτηση (SecureRandom). Ο κωδικός επιλέχθηκε να έχει μήκος 8 χαρακτήρων[27] και αποτελείται είτε από λατινικούς χαρακτήρες είτε από ελληνικούς.

Έπειτα για ένα εύρος κλειδιών τα οποία δημιουργούνται, υπολογίζει τις εξής τιμές για την εντροπία, βασιζόμενη στον τύπο της εντροπίας που αναφέρθηκε στην ενότητα 4.1.

1. Ελάχιστη τιμή Εντροπίας
2. Μέγιστη Τιμή Εντροπίας
3. Μέσος Όρος Τιμών Εντροπίας
4. Κλειδιά με Τιμή Εντροπίας 1.0

Επίσης πέραν των αποτελεσμάτων των τιμών της εντροπίας διενεργεί και άλλους ελέγχους όσον αφορά τους δεκαεξαδικούς χαρακτήρες από τους οποίους αποτελείται ένα κλειδί του AES. Πιο συγκεκριμένα τους διαχωρίζει σε UTF-8[26] και non UTF-8.

Αυτό γίνεται διότι μια ακολουθία η οποία περιέχει μόνο έγκυρους χαρακτήρες UTF-8 οι οποίοι είναι τα δεκαεξαδικά bytes από `00-7F`, μπορεί να παράγει υψηλές τιμές εντροπίες, και να ληφθεί υπόψη σαν κλειδί του AES.

Για παράδειγμα η ακολουθία μήκους 16 bytes

`"4E 54 45 52 50 4F 4C 41 54 45 44 5F 41 4C 50 48" ("NTERPOLATED_ALPH")`

`η` οποία περιέχει μόνο UTF-8 bytes λαμβάνει τιμή εντροπίας `0.95443`

Το ίδιο συμβαίνει και για την ακολουθία

`"0F` **`B7`** `73 1D` **`C8`** `6D 54 6D 70 41 63 74 69 6F 6E 73" (.·s.`**`Θ`**`mTmpActions)`

η οποία όμως περιέχει τα μη έγκυρα UTF-8 bytes, `B7,C8` και δίνει εξίσου υψηλή τιμή εντροπίας `0.9971`

Οι παραπάνω τιμές σε ένα πρόγραμμα το οποίο θα βασιζόταν μόνο στην εντροπία μιας ακολουθίας πιθανόν να οδηγούσε σε ένα πλαστό εύρημα (false positive) ότι οι συγκεκριμένες ακολουθίες είναι ένα έγκυρο κλειδί του AES-128.

Για την αποφυγή των παραπάνω και όχι μόνο, πλαστών ευρημάτων τα οποία μπορεί να προκύψουν κατά την ανάλυση ενός αποτυπώματος μνήμης τα κλειδιά του AES αναλύθηκαν επιπλέον και για τις παρακάτω συνθήκες:

1. Ελάχιστοι non-UTF χαρακτήρες που περιέχονται σε ένα κλειδί
2. Μέγιστοι non-UTF χαρακτήρες που περιέχονται σε ένα κλειδί
3. Μέσος Όρος non-UTF χαρακτήρων που περιέχονται σε όλα τα υπό εξέταση κλειδιά
4. Εύρεση λιγότερων από 10 /6 / 2 non-UTF χαρακτήρων σε κλειδιά του AES με μήκος 256/192/128 bits αντίστοιχα.
5. Εύρεση κλειδιών τα οποία εμφανίζουν περισσότερους από τρία ίδια δεκαεξαδικά bytes στη σειρά π.χ. AA **BB BB BB** CC DD
6. Εύρεση κλειδιών τα οποία περιέχουν περισσότερα από 5 ίδια δεκαεξαδικά bytes. Π.χ **BB** CC **BB** DD **BB BB** AA **BB** CC DD **BB.**

Στις παρακάτω εικόνες φαίνεται το αποτέλεσμα που παράγει το πρόγραμμα στο Eclipse, για την ανάλυση κλειδιών τα οποία παράγονται από τις εφαρμογές AES Android Forensics, TextCrypt, AES Application Free, AES Message Encryptor.

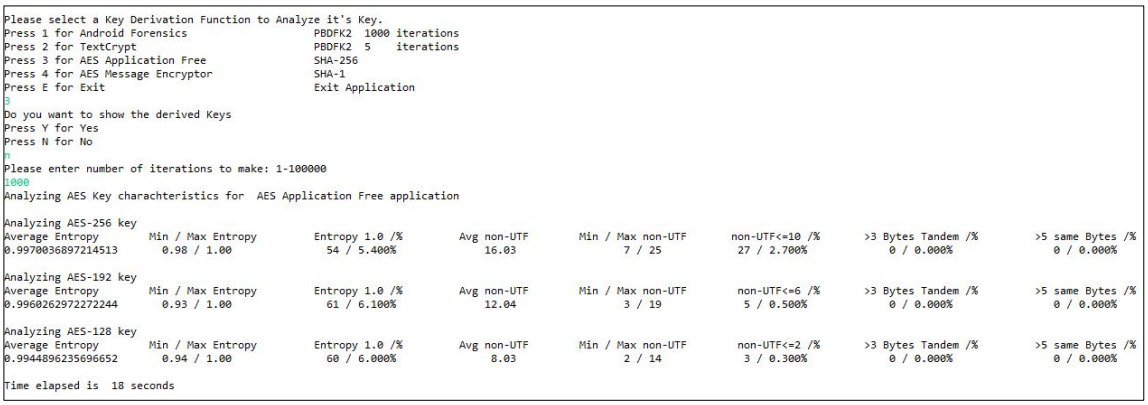
Ο αριθμός των κλειδιών τα οποία ελέγχονται κάθε φορά είναι 1000.

```
Checking AES Key characteristics

Please select a Key Derivation Function to Analyze it's Key.
Press 1 for Android Forensics                   PBDFK2  1000 iterations
Press 2 for TextCrypt                           PBDFK2  5    iterations
Press 3 for AES Application Free                SHA-256
Press 4 for AES Message Encryptor               SHA-1
Press E for Exit                                Exit Application
1
Do you want to show the derived Keys
Press Y for Yes
Press N for No
n
Please enter number of iterations to make: 1-100000
1000
Analyzing AES Key charachteristics for Android Forensics application

Analyzing AES-256 key
Average Entropy        Min / Max Entropy        Entropy 1.0 /%       Avg non-UTF       Min / Max non-UTF      non-UTF<=10 /%      >3 Bytes Tandem /%        >5 same Bytes /%
0.997430210090943       0.97 / 1.00              53 / 5.300%             15.92              8 / 29           24 / 2.400%           0 / 0.000%              1 / 0.100%

Analyzing AES-192 key
Average Entropy        Min / Max Entropy        Entropy 1.0 /%       Avg non-UTF       Min / Max non-UTF      non-UTF<=6 /%       >3 Bytes Tandem /%        >5 same Bytes /%
0.9961755579386203      0.96 / 1.00              55 / 5.500%             12.06              5 / 19           14 / 1.400%           0 / 0.000%              0 / 0.000%

Analyzing AES-128 key
Average Entropy        Min / Max Entropy        Entropy 1.0 /%       Avg non-UTF       Min / Max non-UTF      non-UTF<=2 /%       >3 Bytes Tandem /%        >5 same Bytes /%
0.994245524485133       0.95 / 1.00              69 / 6.900%             8.17               1 / 14           3 / 0.300%            0 / 0.000%              0 / 0.000%

Time elapsed is  25 seconds
```

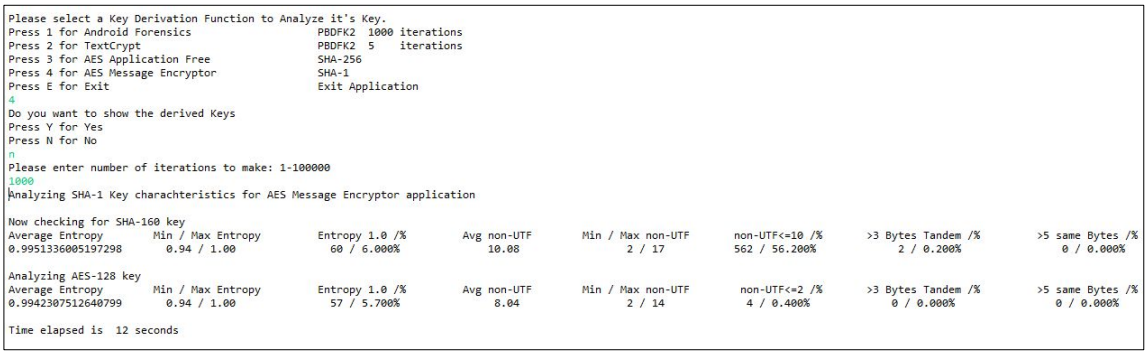
Εικόνα 4.2 Ανάλυση κλειδιών εφαρμογής AES Android Forensics

```
Checking AES Key characteristics

Please select a Key Derivation Function to Analyze it's Key.
Press 1 for Android Forensics                   PBDFK2  1000 iterations
Press 2 for TextCrypt                           PBDFK2  5    iterations
Press 3 for AES Application Free                SHA-256
Press 4 for AES Message Encryptor               SHA-1
Press E for Exit                                Exit Application
2
Do you want to show the derived Keys
Press Y for Yes
Press N for No
n
Please enter number of iterations to make: 1-100000
1000
Analyzing AES Key charachteristics for TextCrypt application

Analyzing AES-256 key
Average Entropy        Min / Max Entropy        Entropy 1.0 /%       Avg non-UTF       Min / Max non-UTF      non-UTF<=10 /%      >3 Bytes Tandem /%        >5 same Bytes /%
0.9970569184893538      0.97 / 1.00              52 / 5.200%             15.95              6 / 25           32 / 3.200%           0 / 0.000%              1 / 0.100%

Analyzing AES-192 key
Average Entropy        Min / Max Entropy        Entropy 1.0 /%       Avg non-UTF       Min / Max non-UTF      non-UTF<=6 /%       >3 Bytes Tandem /%        >5 same Bytes /%
0.9962674658851125      0.95 / 1.00              47 / 4.700%             12.06              4 / 19           10 / 1.000%           0 / 0.000%              0 / 0.000%

Analyzing AES-128 key
Average Entropy        Min / Max Entropy        Entropy 1.0 /%       Avg non-UTF       Min / Max non-UTF      non-UTF<=2 /%       >3 Bytes Tandem /%        >5 same Bytes /%
0.9944310769805743      0.94 / 1.00              78 / 7.800%             7.91               2 / 14           4 / 0.400%            0 / 0.000%              0 / 0.000%

Time elapsed is  20 seconds
```

Εικόνα 4.3: Ανάλυση κλειδιών εφαρμογής TextCrypt



Εικόνα 4.4: Ανάλυση κλειδιών εφαρμογής AES Application Free



Εικόνα 4.5: Ανάλυση κλειδιών εφαρμογής AES Message Encryptor

Επίσης στην παρακάτω εικόνα παρουσιάζεται μια επιλογή του συγκεκριμένου προγράμματος κατά την οποία εμφανίζονται τα κλειδιά κρυπτογράφησης για την εφαρμογή AES Android Forensics σε τρεις διαφορετικές μορφές, σε συνδυασμό με τον κωδικό, από τον οποίο δημιουργούνται.

Εικόνα 4.6: Εμφάνιση κλειδιών και κωδικού

Από τους παραπάνω ελέγχους και για τις τέσσερις εφαρμογές που εξετάστηκαν και για όλα τα μήκη κλειδιών προκύπτουν τα παρακάτω συμπεράσματα:

1. Η ελάχιστη τιμή της εντροπίας δεν είναι μικρότερη της τιμής 0.94
2. Η μέγιστη τιμή της εντροπίας λαμβάνει την τιμή 1.0 σε μικρό ποσοστό ≈ 6%
3. Ο μέσος όρος των χαρακτήρων που χρησιμοποιούνται είναι μισοί UTF και μισοί non UTF.
   Για παράδειγμα στην εφαρμογή TextCrypt ο μέσος όρος των non UTF χαρακτήρων που χρησιμοποιούνται σε κλειδί των 256/192/128 bits είναι 15,95/12,06/7,91 χαρακτήρες αντίστοιχα.
4. Οι ελάχιστοι non-UTF χαρακτήρες βρίσκονται σε κλειδιά των 128 bits και η τιμή τους είναι 2.
5. Οι μέγιστοι non-UTF χαρακτήρες βρίσκονται αντίστοιχα σε κλειδιά των 256 bits και η τιμή τους είναι 25.
6. Η πιθανότητα δημιουργίας κλειδιών οι οποίοι περιέχουν λιγότερους από 10/6/2 non UTF χαρακτήρες σε κλειδιά των 256/192/128 bits αντίστοιχα είναι πολύ μικρή ≈3%.
7. Δεν δημιουργείται κλειδί του AES το οποίο να περιέχει τρία δεκαεξαδικά bytes στη σειρά
8. Δεν δημιουργείται κλειδί του AES το οποίο να περιέχει πάνω από 5 φορές το ίδιο δεκαεξαδικό byte.

Επίσης παρατηρείται ότι η πιο χρονοβόρα εκτέλεση (25”) του προγράμματος `Check_AES_Keys` γίνεται κατά τον έλεγχο των κλειδιών της εφαρμογής AES Android Forensics.

Αυτό συμβαίνει διότι η συγκεκριμένη υλοποίηση απαιτεί να γίνονται 1000 επαναλήψεις για τη δημιουργία του κλειδιού, γεγονός το οποίο καθυστερεί την περάτωση του προγράμματος. Αντίθετα για την εφαρμογή TextCrypt η οποία χρησιμοποιεί μόλις 5 επαναλήψεις, το πρόγραμμα τερματίζεται σε 20”. Για τις εφαρμογές AES Application Free και AES Message Encryption οι οποίες χρησιμοποιούν αλγόριθμο κατακερματισμού για τη δημιουργία του κλειδιού, οι χρόνοι περάτωσης του προγράμματος μειώνονται περαιτέρω.

Ο πλήρης κώδικας υλοποιημένος σε Java, για το πρόγραμμα `Check_AES_Keys` βρίσκεται στο Παράρτημα Β.

4.3 Ανάλυση κλειδιών AES στο αποτύπωμα μνήμης

Για την επιπλέον μείωση των πλαστών ευρημάτων, καθώς και για την μείωση εκτέλεσης χρόνου του προγράμματος, έγινε περαιτέρω ανάλυση των κλειδιών κρυπτογράφησης σε ένα αποτύπωμα μνήμης

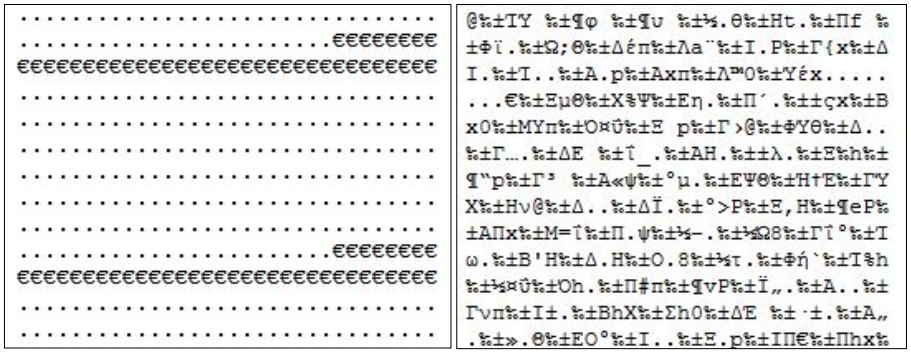
Η ανάλυση αυτή πραγματοποιήθηκε σε βάθος χρόνου κυρίως για τη λειτουργία της κρυπτογράφησης, και εξετάστηκαν αποτυπώματα μνήμης, τα οποία δημιουργήθηκαν με την είσοδο διαφορετικών κωδικών, διαφορετικού κειμένου προς κρυπτογράφηση, και για διαφορετικές εφαρμογές.

Από την ανάλυση αυτή προέκυψαν τα εξής συμπεράσματα για όλες τις εφαρμογές που αναλύονται στο Κεφάλαιο 3 και τα οποία ενσωματώθηκαν στο πρόγραμμα `Entropy_AES_Key`:

Offset Κλειδιού: Όλα τα ευρήματα του κλειδιού σε ένα αποτύπωμα μνήμης είναι μετατοπισμένες προς το τέλος του αρχείου (υψηλές περιοχές μνήμης). Πιο συγκεκριμένα το κλειδί ξεκινάει να εμφανίζεται σε περιοχές μνήμης μεγαλύτερες συνήθως από το 90% του αποτυπώματος μνήμης.

Αριθμός Εμφανίσεων: Το κλειδί κρυπτογράφησης εμφανίζεται σε ένα αποτύπωμα μνήμης από δύο έως έξι φορές. Επίσης οι πιο πιθανές τοποθεσίες για την εύρεσή του είναι κοντά στις κλάσεις `javax.crypto.SecretKey`, `javax.crypto.SecretKeySpec`, καθώς αυτές είναι οι πλέον συχνά χρησιμοποιούμενες κλάσεις στην Java με τις οποίες δημιουργείται το κλειδί. Αντίστοιχα οι κλάσεις αυτές εμφανίζονται σε υψηλές περιοχές της μνήμης.

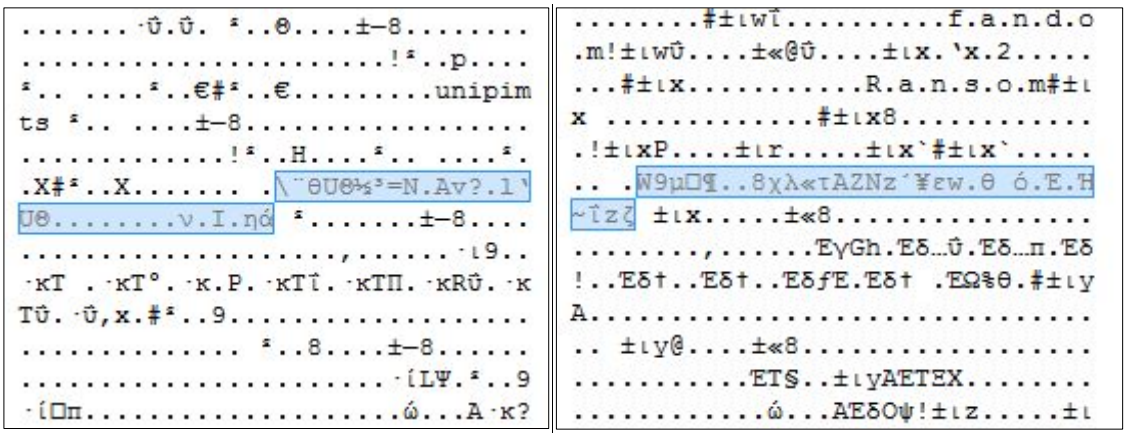
Εντροπία Περιοχής Κλειδιού: Το κλειδί δεν εμφανίζεται σε περιοχές της μνήμης ούτε με πολύ χαμηλή εντροπία αλλά ούτε σε περιοχές της μνήμης με πολύ υψηλή εντροπία. Στις παρακάτω εικόνες φαίνεται στα αριστερά εμφανίζεται μια περιοχή της μνήμης με χαμηλή εντροπία (≈0) ενώ στα δεξιά μία με υψηλή (≈1) αντίστοιχα:



Εικόνα 4.7: Περιοχές μνήμης με χαμηλή και υψηλή εντροπία

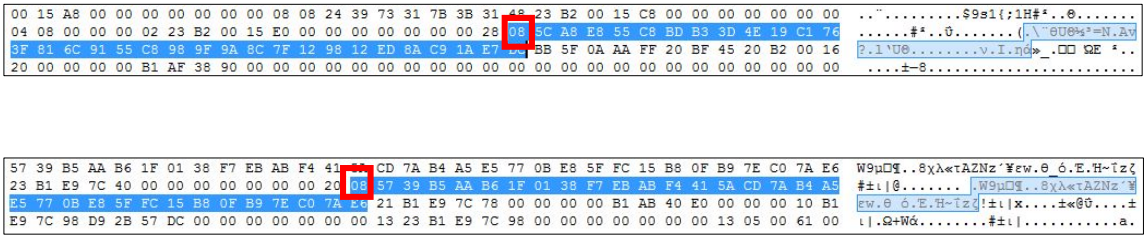
Έπειτα από εξέταση των αποτυπωμάτων μνήμης διαπιστώθηκε ότι τα ευρήματα του κλειδιού εμφανίζονται σε περιοχές της μνήμης όπου πριν ή μετά από αυτό δεν υπάρχουν ακολουθίες με χαμηλή και υψηλή εντροπία.

Οι περιοχές αυτές φαίνονται στην παρακάτω εικόνα για δύο διαφορετικά κλειδιά μήκους 32 bytes.



Εικόνα 4.8 Περιοχή της μνήμης εύρεσης κλειδιού

Μοτίβο εύρεσης κλειδιού: Στα περισσότερα από τα αποτυπώματα μνήμης τα οποία εξετάστηκαν, πριν την ακολουθία του κλειδιού εμφανιζόταν το δεκαεξαδικό byte "08" το οποίο στο UTF-8 συμβολίζει το χαρακτήρα του «διαστήματος» (Space). Τα ευρήματα αυτά φαίνονται στις παρακάτω εικόνες για τα δύο προηγούμενα κλειδιά.



Εικόνα 4.9: Εμφάνιση byte 08 πριν το κλειδί

4.4 Εύρεση κλειδιών AES στο αποτύπωμα μνήμης

Στην ενότητα αυτή παρουσιάζεται συνοπτικά ο τρόπος λειτουργίας του προγράμματος `Entropy_AES_Key` με το οποίο γίνεται η ανάκτηση του κλειδιού κρυπτογράφησης από ένα αποτύπωμα μνήμης.
Το πρόγραμμα αυτό για την εύρεση των κλειδιών χρησιμοποιεί σαν κύριο εργαλείο τη δυαδική εντροπία σε συνδυασμό με τα συμπεράσματα που αναφέρθηκαν στις δύο προηγούμενες ενότητες. Η πλήρης υλοποίηση του προγράμματος, σε κώδικα Java βρίσκεται στο Παράρτημα Γ.

Το πρόγραμμα αυτό δέχεται σαν είσοδο αρχεία τα οποία περιέχουν κάποιο αποτύπωμα μνήμης και πιο συγκεκριμένα τα αρχεία που δημιουργούνται από τον Dalvik Debug Monitor Server-DDMS με κατάληξη .hprof.
Αφού το πρόγραμμα έχει δεχτεί σαν είσοδο ένα τέτοιο αρχείο, διαβάζει τα (raw) bytes που υπάρχουν σε αυτό και τα αποθηκεύει σε δεκαεξαδική μορφή.

Έπειτα ακολουθούνται τα εξής βήματα έως ότου βρεθεί μια ακολουθία η οποία να θεωρείται σαν έγκυρο κλειδί του AES.

1. Γίνεται αναζήτηση των τιμών `javax.crypto.SecretKey`, `javax.crypto.SecretKeySpec` στο αποτύπωμα μνήμης. Σε περίπτωση που βρεθεί τουλάχιστον μία από τις δύο αυτές τιμές, το εύρος αναζήτησης ορίζεται στα 4096 bytes. Τα 2048 bytes αφορούν τις ακολουθίες που εξετάζονται πριν από το εκάστοτε εύρημα, και τα υπόλοιπα 2048 αφορούν τις ακολουθίες που εξετάζονται μετά από αυτό.
2. Δημιουργείται ένα νέο δεκαεξαδικό αρχείο με βάση το αρχικό, με μέγεθος ίσο με το εύρος αναζήτησης, δηλαδή 4096 bytes.
3. Αυτό το αρχείο διαιρείται σε ακολουθίες ίσες με το μήκος κλειδιού που θα δοθεί για αναζήτηση.
4. Έπειτα στην κάθε μία από αυτές διεξάγονται κάποιοι έλεγχοι για να διαπιστωθεί αν μια ακολουθία ίσου μήκους με το κλειδί αναζήτησης μπορεί να αντιπροσωπεύει ένα έγκυρο κλειδί του AES.
   Ο πρώτος έλεγχος είναι η υπό εξέταση ακολουθία να έχει εντροπία μεγαλύτερη από το κατώφλι **0.9.**
   Επίσης η προηγούμενη και επόμενη από αυτήν ακολουθίες πρέπει να έχουν μικρότερη τιμή από αυτή που εξετάζεται, καθώς επίσης και η τιμή εντροπίας τους να είναι μεγαλύτερη από ένα κατώφλι, εντροπίας το οποίο αντιπροσωπεύει ότι δεν βρίσκονται σε περιοχές της μνήμης με χαμηλή εντροπία. Το κατώφλι αυτό της εντροπίας λαμβάνει την τιμή **0.5.**
5. Έπειτα βρίσκεται η ακολουθία με τη μέγιστη εντροπία που μπορεί να προκύψει δημιουργώντας ακολουθίες με μήκος ίσο με του κλειδιού από την αρχική. Η πρώτη από

τις ακολουθίες που δημιουργείται έχει σαν τελευταίο byte το πρώτο της αρχικής, και η τελευταία έχει σαν πρώτο byte, το τελευταίο της αρχικής.
Για παράδειγμα για μήκος κλειδιού ίσο με 4 η ακολουθία EFGH ελέγχεται ως εξής:
ABCD **EFGH** IJKL

- BCDE
- *CDEF*
- DEFG
- **EFGH**
- FGHI
- GHIJ
- HIJK

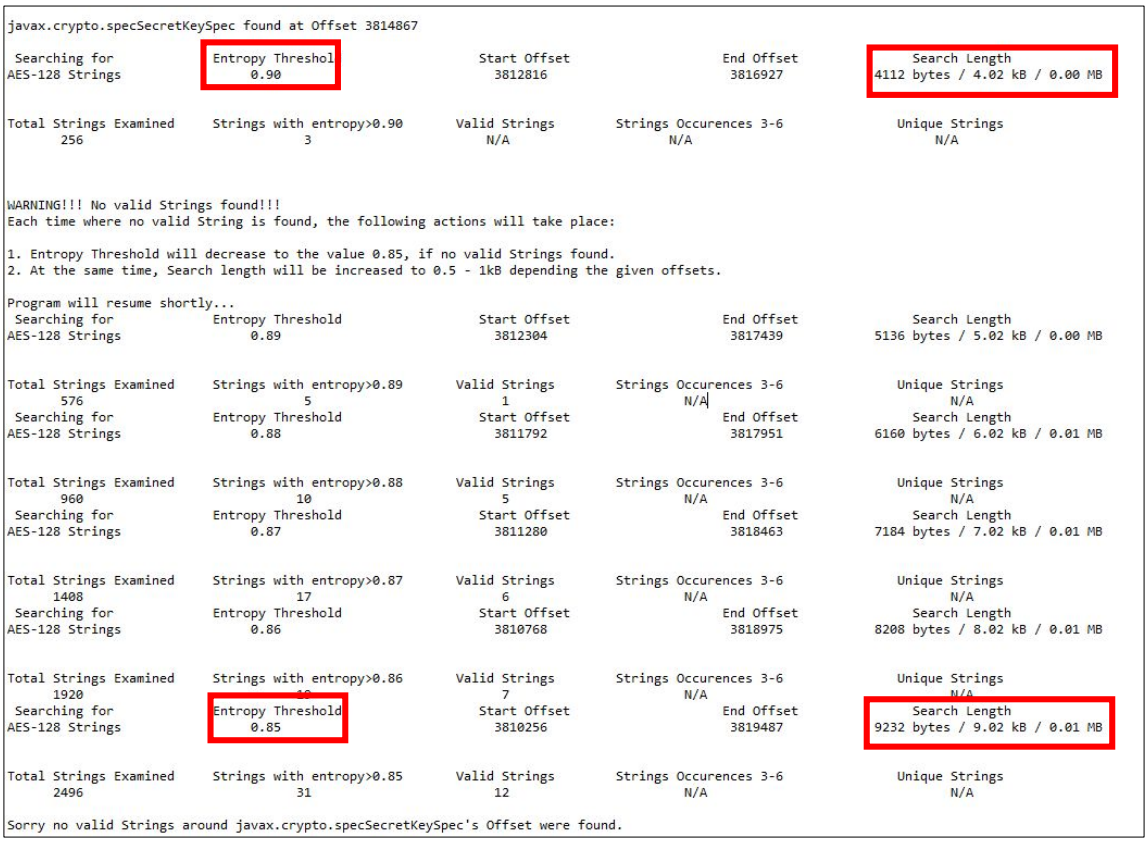
Αν κάποια από τις παραπάνω ακολουθίες έχει μεγαλύτερη εντροπία από την EFGH, τότε την αντικαθιστά, διαφορετικά παραμένει για έλεγχο η ακολουθία EFGH.
Πιο συγκεκριμένα για μήκος κλειδιού 256 ελέγχονται 62 ακολουθίες, για μήκος κλειδιού 192 ελέγχονται 46, και για μήκος κλειδιού 128 ελέγχονται 30 διαφορετικές ακολουθίες από την αρχική.

6. Έπειτα η ακολουθία που έχει προκύψει εξετάζεται για το αν μπορεί να είναι ένα έγκυρο κλειδί του AES εφαρμόζοντας τους ελέγχους που αναφέρθηκαν στην ενότητα 4.2.
7. Αν η ακολουθία είναι έγκυρη γίνεται αναζήτηση της στο αποτύπωμα μνήμης. Αν εμφανίζεται 2-6 φορές και οι ακολουθίες πριν ή μετά από αυτήν δεν αντιπροσωπεύουν περιοχές της μνήμης με πολύ χαμηλή εντροπία τότε η ακολουθία θεωρείται έγκυρη.
   Το κατώφλι αυτό της εντροπίας λαμβάνει την τιμή **0.3**.
8. Αν η ακολουθία έχει περάσει τους παραπάνω ελέγχους, γίνεται και ένας ακόμα έλεγχος σύμφωνα με τον οποίο, δημιουργούνται άλλες 14 ακολουθίες, με παρόμοιο τρόπο με αυτόν που περιγράφηκε στο βήμα 5.
   Οι 7 είναι σε υψηλότερο και οι άλλες 7 σε χαμηλότερο offset από το αρχικό byte της έγκυρης αυτής ακολουθίας.
   Από τις 14 αυτές ακολουθίες ελέγχονται μόνο όσες είναι έγκυρες για να βρεθεί η μία ακολουθία η οποία εμφανίζεται τις περισσότερες φορές στο αποτύπωμα μνήμης.
9. Στην ακολουθία που θα βρεθεί διενεργείται ένας τελευταίος έλεγχος για να διαπιστωθεί αν το πρώτο της byte είναι το δεκαεξαδικό “08”. Στην περίπτωση που ισχύει αυτό, η ακολουθία μετατοπίζεται έτσι ώστε να ξεκινάει από το δεύτερο της byte και όχι από το “08”.
10. Μετά από τους παραπάνω ελέγχους αυτή θα είναι και η ακολουθία η οποία έχει τις περισσότερες πιθανότητες να αντιπροσωπεύει ένα έγκυρο κλειδί του AES.

Σε περίπτωση που από τους παραπάνω ελέγχους δεν προκύψει κάποια έγκυρη ακολουθία τότε τα παραπάνω βήματα επαναλαμβάνονται με μειωμένη τιμή εντροπίας κατά 0.01, και ταυτόχρονη αύξηση του εύρους αναζήτησης κατά 1024 bytes. 512 bytes πριν από το εκάστοτε εύρημα, και 512 μετά από αυτό.

Η αναζήτηση για μια έγκυρη ακολουθία σταματάει είτε άμα βρεθεί κάποια τέτοια ακολουθία, είτε όταν το κατώφλι της εντροπίας λάβει την τιμή 0.85 από την αρχική τιμή 0.9 την οποία είχε.

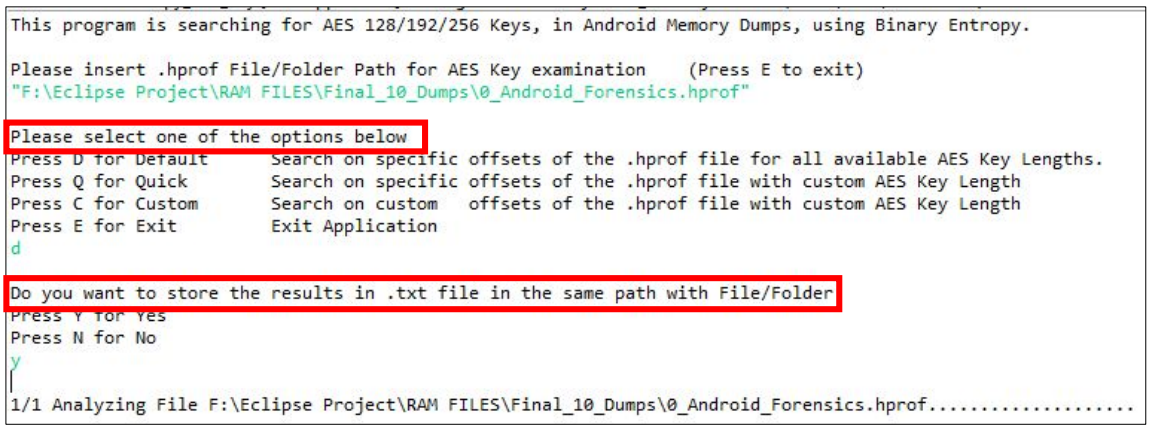
Στην παρακάτω εικόνα παρουσιάζονται οι επαναλήψεις που γίνονται για την εύρεση μιας ακολουθίας μήκους 128 bits.



Εικόνα 4.10: Επαναλήψεις για εύρεση ακολουθίας AES

Επιπλέον ο χρήστης, μπορεί να επιλέξει μεταξύ τριών τρόπων λειτουργίας για την εύρεση του κλειδιού, καθώς επίσης και για το αν θα αποθηκεύσει τα αποτελέσματα της αναζήτησης σε μορφή .txt στον ίδιο φάκελο με αυτόν που βρίσκεται το αποτύπωμα μνήμης.

Οι επιλογές αυτές φαίνονται στην παρακάτω εικόνα:



Εικόνα 4.11: Επιλογές χρήστη

Οι τρόποι λειτουργίας είναι οι εξής:

Εξειδικευμένη Αναζήτηση **(Custom Search)** : Με αυτή τη λειτουργία ο χρήστης εισάγει χειροκίνητα το μήκος του κλειδιού το οποίο θέλει να αναζητήσει, καθώς επίσης και το εύρος σε bytes, στο οποίο θα γίνει η αναζήτηση στο αποτύπωμα μνήμης. (Start Offset-End Offset).

Γρήγορη Αναζήτηση **(Quick Search) :** Με αυτή τη λειτουργία ο χρήστης εισάγει μόνο το μήκος του κλειδιού αναζήτησης, χωρίς να δώσει κάποιο εύρος αναζήτησης.

Αυτό γίνεται διότι το πρόγραμμα άμεσα ξεκινάει την αναζήτηση του κλειδιού σε περιοχές της μνήμης κοντά στα ευρήματα `javax.crypto.SecretKey`, `javax.crypto.SecretKeySpec`.

Αν αυτά δεν υπάρχουν στο αποτύπωμα μνήμης, τότε εμφανίζει το αντίστοιχο ενημερωτικό μήνυμα, και προτρέπει το χρήστη να κάνει εξειδικευμένη αναζήτηση .

```
1/1 Analyzing File C:\Java\Test_21.hprof...

Sorry, no javax.crypto found, please try with Custom Mode

Please select one of the options below
Press C for Custom       Search on custom   offsets of the .hprof file with custom AES Key Length
Press E for Exit         Exit Application
.................
```

Εικόνα 4.12: Αναζήτηση με Custom Mode

Σε περίπτωση που βρεθούν μία η περισσότερες ακολουθίες με το επιθυμητό μήκος αυτές εμφανίζονται σε συνδυασμό και με άλλες παραμέτρους έτσι ώστε να δίνεται μια πλήρη εικόνα στο χρήστη αν αυτό είναι ένα έγκυρο κλειδί του AES ή όχι. Αυτές οι παράμετροι είναι οι εξής:

1. Τιμή εντροπίας ακολουθίας
2. Τα Offset της ακολουθίας στα οποία η εντροπία λαμβάνει τη μέγιστη τιμή της.
3. Συνολικές εμφανίσεις της ακολουθίας στο αποτύπωμα μνήμης
4. Ο αριθμός των UTF-8 και non UTF-8 χαρακτήρων οι οποίες περιέχονται στην ακολουθία.
5. Σε ποιο ποσοστό του αποτυπώματος μνήμης βρίσκεται το πρώτο εύρημα της ακολουθίας.

Οι παραπάνω παράμετροι εμφανίζονται στην Εικόνα 4.13 όπου έγινε αναζήτηση για κλειδί AES-192 στο αποτύπωμα μνήμης της εφαρμογής AES Android Forensics.

```
Total Strings Examined     Strings with entropy>0.90       Valid Strings        Strings Occurences 3-6              Unique Strings
      9490                          19                          4                          3                          1

                24-bytes STRINGS                                        OFFSETS                        TOTAL OCCURRENCES           ENTROPY          NON-UTF    UTF    1st Position (% File)
1    A44023DA4204D62413A91F2CBF080FBCB1A38FE4F4A0361A                 4676984   4677846   4677884                5              0.990508349072055        12        12        95.467

javax.crypto.SecretKey found at Offset 4534781
```

Εικόνα 4.13: Παράμετροι κλειδιού στο αποτύπωμα μνήμης

Προκαθορισμένη Αναζήτηση **(Default Search):** Σε αυτή τη λειτουργία ο χρήστης δεν δίνει ούτε εύρος αναζήτησης, ούτε μήκος κλειδιού.

Αντίστοιχα με την Γρήγορη Αναζήτηση το πρόγραμμα ξεκινάει την εύρεση των κλειδιών κοντά στα ευρήματα `javax.crypto.SecretKey`, `javax.crypto.SecretKeySpec`. Μόνο που σε αυτό τον τρόπο λειτουργίας αυτό το κάνει για όλα τα μήκη κλειδιών του AES (256/192/128).

Επιπλέον στην περίπτωση που το κλειδί του AES έχει δημιουργηθεί από τη συνάρτηση κατακερματισμού SHA-1 (περίπτωση εφαρμογής AES Message Encryption), ελέγχει και για κλειδιά μήκους 20 bytes.

Στη λειτουργία αυτή μπορεί να βρεθούν περισσότερα του ενός κλειδιά με το ίδιο ή διαφορετικό μήκος.

Έπειτα από ελέγχους που πραγματοποιούνται, επιστρέφεται το τελικό κλειδί του AES σε δεκαεξαδική μορφή, όπου σε συνδυασμό με αυτό εμφανίζονται ακόμα το μήκος του, ο τρόπος λειτουργίας του αλγόριθμου, και συνάρτηση δημιουργίας του κλειδιού (αν βρεθεί).

Στις παρακάτω εικόνες εμφανίζεται το αποτέλεσμα αυτής της αναζήτησης στα αποτυπώματα μνήμης για τις εφαρμογές AES Android Forensics, TextCrypt και AES Message Encryption και AES Encryption App Free.

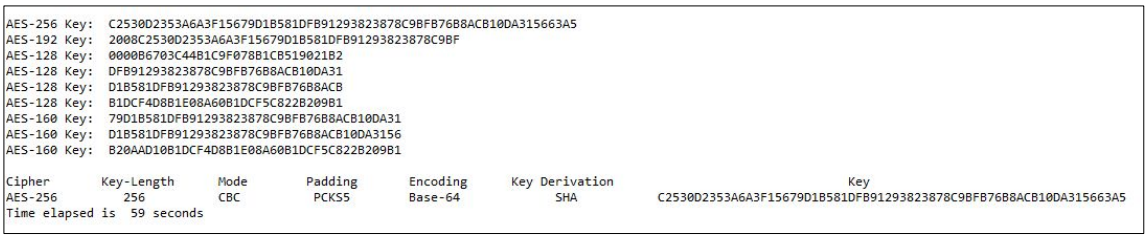
Τα κλειδιά κρυπτογράφησης τα οποία ανακτώνται είναι τα ίδια με αυτά που αναλύθηκαν στις αντίστοιχες ενότητες τους Κεφαλαίου 3 για τις παραπάνω εφαρμογές.

```
AES-192 Key:  A44023DA4204D62413A91F2CBF080FBCB1A38FE4F4A0361A
AES-128 Key:  2413A91F2CBF080FBCB1A38FE4F4A036
AES-128 Key:  13A91F2CBF080FBCB1A38FE4F4A0361A
AES-128 Key:  04D62413A91F2CBF080FBCB1A38FE4F4
AES-160 Key:  4204D62413A91F2CBF080FBCB1A38FE4F4A0361A

Cipher       Key-Length      Mode        Padding       Encoding      Key Derivation     Iterations                    Key
AES-192         192          CBC          PCKS5        Base-64          PBDFK2            N/A          A44023DA4204D62413A91F2CBF080FBCB1A38FE4F4A0361A
Time elapsed is  1 minutes kai 49 seconds
```

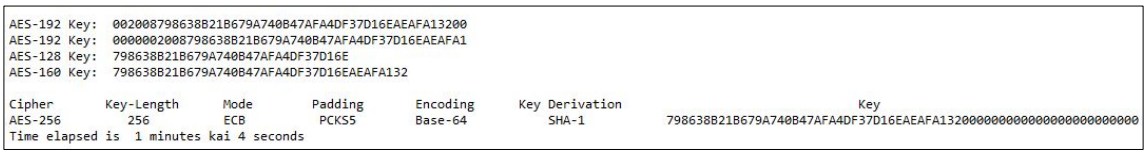
Εικόνα 4.14: Εύρεση κλειδιού εφαρμογής AES Android Forensics

```
AES-256 Key:  5CA8E855C8BDB33D4E19C1763F816C9155C8989F9A8C7F129812ED8AC91AE7DC
AES-192 Key:  4E19C1763F816C9155C8989F9A8C7F129812ED8AC91AE7DC
AES-192 Key:  5CA8E855C8BDB33D4E19C1763F816C9155C8989F9A8C7F12
AES-128 Key:  55C8BDB33D4E19C1763F816C9155C898
AES-128 Key:  0000000181DD21F0B1FDB05823B1FDBA
AES-160 Key:  55C8BDB33D4E19C1763F816C9155C8989F9A8C7F

Cipher       Key-Length      Mode        Padding       Encoding      Key Derivation     Iterations                    Key
AES-256         256          CBC          PCKS5        Base-64          PBDFK2            N/A          5CA8E855C8BDB33D4E19C1763F816C9155C8989F9A8C7F129812ED8AC91AE7DC
Time elapsed is  1 minutes kai 27 seconds
```

Εικόνα 4.15: Εύρεση κλειδιού εφαρμογής TextCrypt



Εικόνα 4.16: Εύρεση κλειδιού εφαρμογής AES Encryption App Free



Εικόνα 4.17: Εύρεση κλειδιού εφαρμογής AES Message Encryptor

Το εν λόγω πρόγραμμα δημιουργήθηκε για την εύρεση κλειδιών υψηλής εντροπίας για τον λόγο αυτό δεν μπορεί να εντοπίσει τα κλειδιά τα οποία παράγονται από τις εφαρμογές Secret_Message και Encrypt καθώς αυτές προσθέτουν μηδενικά στον κωδικό με αποτέλεσμα και τη μείωση της εντροπίας αλλά και την δημιουργία κλειδιού το οποίο δεν ακολουθεί το μοτίβο των χαρακτήρων ενός κλειδιού AES με υψηλή εντροπία.

Επίσης παρ' όλους του ελέγχους οι οποίοι εισήχθησαν υπάρχει η πιθανότητα το αποτέλεσμα της αναζήτησης να δώσει ένα πλαστό εύρημα.

Αυτό συμβαίνει στην περίπτωση της ανάλυσης του αποτυπώματος της εφαρμογής jCryptor όπου ενώ το κλειδί κρυπτογράφησης επιβεβαιώθηκε έπειτα από δοκιμές ότι είναι το " Ξ..GY›.ü‡ű.TPψ" με τη δεκαεξαδική του αναπαράσταση να είναι "CE 01 8A 47 D5 9B 1B FB 87 E0 2E D4 D1 F8 **C4 50**" το πρόγραμμα `Entropy_AES_Key` επιστρέφει την τιμή **10 08** CE 01 8A 47 D5 9B 1B FB 87 E0 2E D4 D1 F8, σαν κλειδί του AES, δηλαδή χάνει τα δύο τελευταία bytes του κλειδιού όπως φαίνεται, τα C4, και 50. Παρόλαυτα έστω και αυτό το πλαστό εύρημα μπορεί να αξιοποιηθεί καθώς αν μετατοπιστεί μόλις κατά 2 bytes μας δίνει το σωστό κλειδί.

Εικόνα 4.18: Πλαστό εύρημα εφαρμογής jCryptor

Επίσης θα πρέπει να τονιστεί ότι το συγκεκριμένο πρόγραμμα παράγει αξιόπιστα αποτελέσματα όταν υπάρχουν οι κλάσεις `javax.crypto.SecretKey`, `javax.crypto.SecretKeySpec` σε ένα αποτύπωμα μνήμης.

Σε διαφορετική περίπτωση όπου και πρέπει να γίνει εξειδικευμένη αναζήτηση σε όλο το αποτύπωμα μνήμης τα ευρήματα είναι και περισσότερα και λιγότερο αξιόπιστα.

Επιπλέον πρέπει να γίνει ανάλυση του καθ' ενός ευρήματος ξεχωριστά για την πιστοποίηση του σαν έγκυρο κλειδί του AES.

Αυτό συνέβη σε ελάχιστα αποτυπώματα μνήμης που ενώ υπήρχαν οι παραπάνω κλάσεις το κλειδί δεν εμφανιζόταν σε κοντινά offset με αυτά, αλλά ήταν μετατοπισμένο σε διαφορετικές περιοχές της μνήμης.

Μια τέτοια αναζήτηση για κλειδί 256 bits παρουσιάζεται στην παρακάτω εικόνα όπου το κλειδί κρυπτογράφησης είναι το:

`" C2530D2353A6A3F15679D1B581DFB91293823878C9BFB76B8ACB10DA315663A5 "`

και έχει γίνει αναζήτηση σε όλο το αρχείο:

```
 Searching for          Entropy Threshold          Start Offset          End Offset        Search Length
AES-256 Strings              0.90                      0                 6745664       6745664 bytes / 6587.56 kB / 6.43 MB

Strings Checked     0/6745664     Progress 0.000%           Time elapsed is  18 seconds
Strings Checked 1000000/6745664   Progress 14.824%          Time elapsed is  20 seconds
Strings Checked 2000000/6745664   Progress 29.649%          Time elapsed is  22 seconds
Strings Checked 3000000/6745664   Progress 44.473%          Time elapsed is  24 seconds
Strings Checked 4000000/6745664   Progress 59.297%          Time elapsed is  27 seconds
Strings Checked 5000000/6745664   Progress 74.122%          Time elapsed is  29 seconds
Strings Checked 6000000/6745664   Progress 88.946%          Time elapsed is  32 seconds
Strings Checked 6745664/6745664   Progress 100.000%         Time elapsed is  34 seconds

Total Strings Examined    Strings with entropy>0.90    Valid Strings    Strings Occurences 3-6        Unique Strings
    210801                       21619                   1884                  22                         9

             32-bytes STRINGS                                  OFFSETS                   TOTAL OCCURRENCES        ENTROPY          NON-UTF   UTF   1st Position (% File)
1    08CA5F050B3C56837505AB35344EFE46787AA054F50FB8FC973845C87B3C7635   3866514  3871840  3872274      3        0.9999559719934967     10       22        57.319
2    8EFC3823186D65E37784C749855E448DD633051E0846118E12E35D7CDCA56875   6052622  6056652  6060852      3        0.9978415729996559     12       20        89.726
3    [illegible]
4    530D2353A6A3F15679D1B581DFB91293823878C9BFB76B8ACB10DA315663A521   6215422  6335104               3        0.9999559719934967     17       15        92.140
5    [illegible]
6    655F636F6E6669726D6174696F6E5F74657874223A225765E280996C6C207265   6408832                        3        0.9999559719934967      3       29        95.007
7    8697CC5B91E45A3A821C426DA98C1CDF5A3D7E97CE309C12C212139A00EF697B   6452695  6466369               3        0.9988990309438097     15       17        95.657
8    54F3A5FCEA005B8E8B0A9F8CE237A0FE123BE5063D422E39D5954C45FA048ECE   6527086  6527916  6550997      3        0.9998238825988729     17       15        96.760
9    53ACBCE3618B740A1FBA4EDC5BB4A2001388F7172759D7C0B20E0AB29AE69AB8   6621164  6625771               3        0.9992954443621547     18       14        97.903
```

Εικόνα 4.19: Αναζήτηση κλειδιού σε όλο το αποτύπωμα μνήμης

Από την παραπάνω αναζήτηση βλέπουμε ότι προέκυψαν 9 ευρήματα, και το κλειδί βρίσκεται στη θέση 4 αλλά έχει χάσει το πρώτο byte καθώς ξεκινάει με το byte 53 και είναι το:

`"530D2353A6A3F15679D1B581DFB91293823878C9BFB76B8ACB10DA31566"`

ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα εργασία έγινε ανάλυση των εφαρμογών android οι οποίες εκτελούν κρυπτογράφηση android βασιζόμενη σε κωδικό. Οι εφαρμογές σαν στόχο έχουν τη διατήρηση της ιδιωτικότητας του χρήστη και την προστασία των προσωπικών του δεδομένων.

Παρόλαυτα από την ανάλυση η οποία έγινε στο κεφάλαιο 3 οι περισσότερες από τις εφαρμογές οι οποίες εξετάστηκαν αποδείχτηκαν ευάλωτες, καθώς δεν επιτυγχάνουν να επιτελέσουν με σωστό τρόπο ούτε την κρυπτογράφηση βασιζόμενη σε κωδικό, αλλά ούτε και την κρυπτογράφηση με τη χρήση του αλγόριθμου AES.

Όσον αφορά την πρώτη κατηγορία, μόλις δύο από τις εφαρμογές οι οποίες εξετάστηκαν χρησιμοποιούν το πρότυπο PKCS5 για τη δημιουργία του κλειδιού κρυπτογράφησης. Ακόμα και σε αυτή την περίπτωση ενώ το πρότυπο υπαγορεύει ότι ένας ικανοποιητικός αριθμός επαναλήψεων που πρέπει να χρησιμοποιείται είναι 1000, η εφαρμογή TextCrypt εφαρμόζει μόλις 5. Παρόλαυτα από όλες τις εφαρμογές που εξετάστηκαν οι οποίες χρησιμοποιούν σταθερές παραμέτρους κρυπτογράφησης είναι η μόνη η οποία δημιουργεί με σωστό τρόπο το «αλάτι» και το διάνυσμα αρχικοποίησης. Η δημιουργία των δύο αυτών παραμέτρων γίνεται με τη χρήση μιας ψευδοτυχαίας συνάρτησης και έτσι δεν εμφανίζονται στο αποτύπωμα μνήμης σε μορφή απλού κειμένου.

Στις υπόλοιπες εφαρμογές στην καλύτερη περίπτωση ο κωδικός απλά δίνεται σαν είσοδος σε μια συνάρτηση κατακερματισμού, είτε στην χειρότερη συμπληρώνεται με μηδενικά. Και στις δύο περιπτώσεις όπως φάνηκε από την κρυπτανάλυση είναι σχετικά εύκολο να ανακτηθεί το κλειδί κρυπτογράφησης.

Επίσης υπάρχουν οι εφαρμογές οι οποίες δεν χρησιμοποιούν σταθερές παραμέτρους για την κρυπτογράφηση όπως η εφαρμογή AES Text Encryptor η οποία για κάθε λειτουργία κρυπτογράφησης χρησιμοποιεί διαφορετικό «αλάτι» και διάνυσμα αρχικοποίησης, με τη χρήση ψευδοτυχαίας συνάρτησης.

Τέτοιου είδους εφαρμογές δεν είναι δυνατόν να κρυπταναλυθούν, αλλά έχουν το μειονέκτημα ότι δεν μπορούν να χρησιμοποιηθούν μεταξύ δύο χρηστών, καθώς το κάθε κρυπτόγραμμα είναι μοναδικό και όταν σταλεί σε έναν άλλο χρήστη δεν θα μπορέσει να αποκρυπτογραφηθεί, καθώς κατά την αποκρυπτογράφηση θα παραχθούν διαφορετικές τιμές για το «αλάτι» και το διάνυσμα αρχικοποίησης.

Επίσης παρατηρήθηκε ότι οι εφαρμογές Secret Message, AES Message Encryptor/Aescryption, χρησιμοποιούν τον τρόπο λειτουργίας Electronic Codebook-ECB για την εφαρμογή του AES. Ο συγκεκριμένος τρόπος λειτουργίας είναι λιγότερο ασφαλής καθώς όπως αναφέρθηκε στο πρώτο Κεφάλαιο με τον ECB ίδια τμήματα του απλού κειμένου δημιουργούν ίδια τμήματα κρυπτογράμματος. Αυτό έχει σαν αποτέλεσμα το τελικό κρυπτόγραμμα το οποίο παράγεται να είναι πιο ευάλωτο σε κάποια επίθεση, καθώς ο επιτιθέμενος είναι σε θέση να αντλήσει πληροφορίες για το αρχικό κείμενο από τη δομή του κρυπτογράμματος.

Για την σωστή λειτουργία των παραπάνω εφαρμογών θα πρέπει να ακολουθούνται οι εξής κανόνες:

1. Η δημιουργία του κλειδιού να γίνεται με βάση το πρότυπο PKCS5
2. Οι επαναλήψεις να είναι τουλάχιστον 1000
3. Οι παράμετροι του αλγόριθμου κρυπτογράφησης είτε σταθερές είτε μεταβλητές να δημιουργούνται από μια ψευδοτυχαία συνάρτηση.
4. Ο τρόπος λειτουργίας του AES να μην είναι ο ECB.

Μια εφαρμογή android βασιζόμενη στους παραπάνω κανόνες έστω και με στατικές παραμέτρους θα ήταν σε θέση να προστατέψει σε μεγάλο βαθμό τα προσωπικά δεδομένα των χρηστών, να δυσκολέψει το έργο της κρυπτανάλυσης, και ταυτόχρονα να είναι και λειτουργική προς το χρήστη.

Τέλος η εφαρμογή που αναπτύχθηκε με την ονομασία `Entropy_AES_Key` βοηθάει προς αυτή την κατεύθυνση καθώς αναλύοντας το αποτύπωμα μνήμης μιας εφαρμογής android βρίσκει με μεγάλη πιθανότητα επιτυχίας τόσο το κλειδί της όσο και τις παραμέτρους κρυπτογράφησης.

Αυτό έχει σαν αποτέλεσμα να εξάγονται άμεσα συμπεράσματα για το πόσο ανθεκτική είναι στην κρυπτανάλυση και το κατά πόσον τηρούνται οι παραπάνω κανόνες για τη σωστή λειτουργία κρυπτογράφησης.

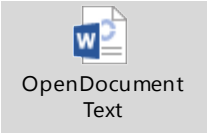
Παράρτημα Α

Κώδικας Java για εφαρμογές Android βασιζόμενες σε κρυπτογράφηση κωδικού

Στο Παράρτημα αυτό παρατίθεται ο κώδικας για τις εφαρμογές Android του 3ου Κεφαλαίου στις οποίες επιβεβαιώθηκε ο τρόπος λειτουργίας του αλγόριθμου κρυπτογράφησής τους.

Ο κώδικας αποτελείται από δύο classes, την `User_Input.java` στην οποία περιέχεται η `main method` και την `AES_Applications.java`.

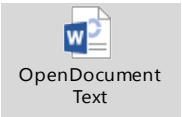
`User_Input.java`: Στην κλάση αυτή ο χρήστης επιλέγει την εφαρμογή Android που θέλει να προσομοιώσει, καθώς επίσης και τις παραμέτρους του αλγόριθμου κρυπτογράφησης. (κρυπτογράφηση/αποκρυπτογράφηση, μήκος κλειδιού, κωδικό, απλό κείμενο/κρυπτόγραμμα).



Παράρτημα Α.1: User_Input.java

`AES_Applications.java`: Οι παραπάνω τιμές αυτές επεξεργάζονται σε αυτή την κλάση και ανάλογα με την επιλογή της Android εφαρμογής που έχει γίνει παράγεται το αντίστοιχο αποτέλεσμα. Οι εφαρμογές που προσομοιώθηκαν είναι οι εξής:

1. AES Android Forensics
2. TextCrypt
3. AES Application Free
4. AES Message Encryptor/Aescryption
5. Encrypt
6. Secret Message



Παράρτημα Α.2: AES_Applications.java

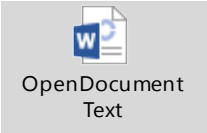
Παράρτημα Β

Κώδικας Java για την ανάλυση εντροπίας κλειδιών AES

Στο Παράρτημα αυτό παρατίθεται ο κώδικας για την ανάλυση της εντροπίας και των UTF χαρακτήρων των κλειδιών του AES 256/192/128 τα οποία δημιουργούνται από τις εφαρμογές AES Android Forensics, TextCrypt, AES Application Free, AES Message Encryptor.

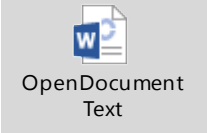
Ο κώδικας αποτελείται από 6 classes, οι οποίες περιγράφονται συνοπτικά παρακάτω:

`User_Input.java`: Στην κλάση αυτή ο χρήστης επιλέγει την αντίστοιχη εφαρμογή από την οποία θα προκύψουν τα υπό εξέταση κλειδιά, τον αριθμό των κλειδιών που θα εξεταστούν καθώς επίσης και το αν θα εμφανίζονται και τα υπό εξέταση κλειδιά στην οθόνη ή όχι.



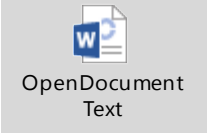
Παράρτημα Β.1: User_Input.java

`Check_AES_Key.java`: Στην κλάση αυτή περιέχεται η `main method` και περιέχονται όλες οι μέθοδοι για τον έλεγχο των κλειδιών του AES.



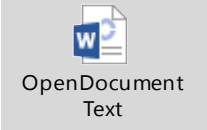
Παράρτημα Β.2: Check_AES_Key.java

`AES_PBDFK2_1000.java`: Η κλάση αυτή δημιουργεί κλειδιά του AES βασιζόμενη στις παραμέτρους κρυπτογράφησης της android εφαρμογής AES Android Forensics.



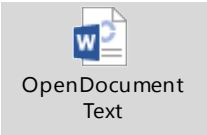
Παράρτημα Β.3: AES_PBDFK2_1000.java

`AES_PBDFK2_5.java`: Η κλάση αυτή δημιουργεί κλειδιά του AES βασιζόμενη στις παραμέτρους κρυπτογράφησης της android εφαρμογής TextCrypt.



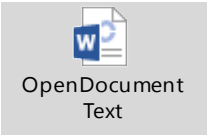
Παράρτημα Β.4: AES_PBDFK2_5.java

`AES_SHA_1.java`: Η κλάση αυτή δημιουργεί κλειδιά του AES βασιζόμενη στις παραμέτρους κρυπτογράφησης της android εφαρμογής AES Message Encryptor.



Παράρτημα Β.5: AES_SHA_1.java

`AES_SHA_256.java`: Η κλάση αυτή δημιουργεί κλειδιά του AES βασιζόμενη στις παραμέτρους κρυπτογράφησης της android εφαρμογής AES Encryption App Free.



Παράρτημα Β.6: AES_SHA_256.java

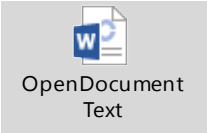
Παράρτημα Γ

Κώδικας Java για την εύρεση κλειδιών AES σε αποτυπώματα μνήμης εφαρμογών Android που χρησιμοποιούν κρυπτογράφηση.

Στο Παράρτημα αυτό παρατίθεται ο κώδικας για την εύρεση του κλειδιού κρυπτογράφησης σε αποτυπώματα μνήμης εφαρμογών Android τα οποία χρησιμοποιούν κρυπτογράφηση βασιζόμενη σε μυστικό κωδικό.

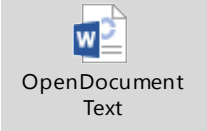
Ο κώδικας αποτελείται από 3 classes, οι οποίες περιγράφονται συνοπτικά παρακάτω:

`User_Input.java`: Στην κλάση αυτή ο χρήστης δίνει τις απαραίτητες εισόδους στο πρόγραμμα για τη λειτουργία του. Αυτές είναι ο κατά σειρά, το αποτύπωμα μνήμης προς εξέταση, ο τρόπος λειτουργίας του προγράμματος, το μήκος του κλειδιού και η εκτύπωση ή όχι των ευρημάτων.



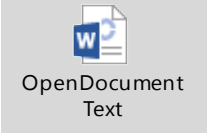
Παράρτημα Γ.1: User_Input.java

`Entropy_AES_Key.java`: Στην κλάση αυτή περιέχεται η `main.method`, και στην κλάση αυτή γίνεται η κύρια επεξεργασία για εύρεση του κλειδιού κρυπτογράφησης στο αποτύπωμα μνήμης.



Παράρτημα Γ.2: Entropy_AES_Key.java

`Support_Methods.java`: Στην κλάση αυτή περιέχονται βοηθητικές μέθοδοι τόσο για τη λειτουργία της `Entropy_AES_Key.java` όσο και για την `User_Input.java`. Μερικές από αυτές είναι ο έλεγχος για την εγκυρότητα μιας ακολουθίας, η εύρεση των παραμέτρων του αλγόριθμου κρυπτογράφησης, η προσθήκη στοιχείων σε πίνακα κ.α.



Παράρτημα Γ.3: Support_Methods.java

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Yury Zhauniarovich Android Security (And Not) Internals (Asani Book)

[2] Nikolay Elenkov (2015) Android Security Internals An In-Depth Guide to Android's Security Architecture

[3] http://users.teilam.gr/~klimn/cryptography/Lec3.pdf

[4] https://tools.ietf.org/html/rfc2898

[5] http://nelenkov.blogspot.gr/2012/04/using-password-based-encryption-on.html http://docs.oracle.com/javase/7/docs/technotes/guides/security/crypto/CryptoSpec.html

[6] http://docs.oracle.com/javase/7/docs/technotes/guides/security/SunProviders.html

[7] http://docs.oracle.com/javase/8/docs/api/javax/crypto/package-summary.html

[8] http://docs.oracle.com/javase/8/docs/api/java/security/package-summary.html

[9] http://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[10] http://www.infoworld.com/article/2610369/processors/intel-vs--arm--two-titans--tangled-fate.html

[11] http://android-developers.blogspot.gr/2013/12/changes-to-secretkeyfactory-api-in.html

[12] http://tenminutetutor.com/base64-encoding

[13] http://mh-nexus.de/en/hxd/

[14] https://play.google.com/store/apps/details?id=js.jcryptor

[15] http://docs.oracle.com/javase/7/docs/api/javax/crypto/KeyGenerator.html

[16] https://play.google.com/store/apps/details?id=com.sperales.crytomessage

[17] https://play.google.com/store/apps/details?id=com.eelcorp.encryptit

[18] https://play.google.com/store/apps/details?id=com.chanyou.aesencrypt

[19] https://play.google.com/store/apps/details?id=com.v_ware.textcrypt

[20] https://play.google.com/store/apps/details?id=com.ucsoftworks.aes_free

[21] https://play.google.com/store/apps/details?id=com.aes.android&hl=en

[22] https://play.google.com/store/apps/details?id=cvnhan.android.aescryption&hl=en

[23] https://play.google.com/store/apps/details?id=com.SteveUngar.encrypt

[24] https://play.google.com/store/apps/details?id=com.unknown.secretmessage

[25] https://en.wikipedia.org/wiki/Binary_entropy_function

[26] https://en.wikipedia.org/?title=UTF-8

[27] https://usmile.at/blog/10-million-passwords-quick-analysis