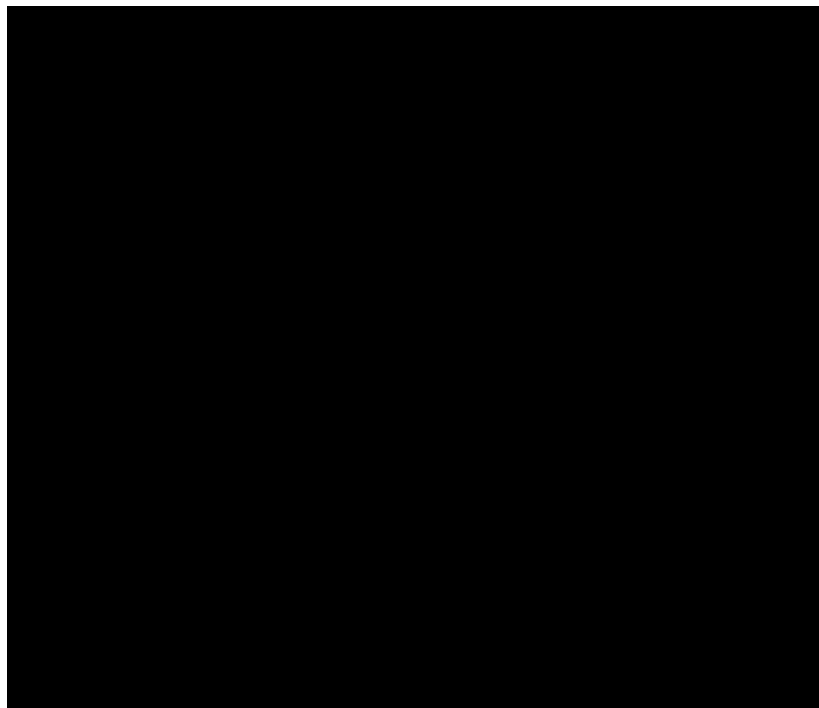




Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Δ.Π.Μ.Σ. Επιστήμης Δεδομένων
και Μηχανικής Μάθησης



February 6, 2021

1. Εισαγωγή

Τα στοιχεία βρίσκονται στο εξώφυλλο της εργασίας.

2. Κλασικοί Αλγόριθμοι Κρυπτογράφησης

2..1

Για κείμενο θα χρησιμοποιήσουμε τις ωδές 3 και 4 από την Κόλαση του Δάντη (Dante's Inferno, Canto III and IV), που πάρθηκαν από την πηγή:

<https://www.owleyes.org/text/dantes-inferno/read/canto-3>.

Συνολικά και τα δύο μαζί, αθροίζουν σε 2,111 λέξεις, που αντιστοιχούν σε ένα αρχείο 12,090 bytes.

Στο CrypTool, δημιουργούμε ένα καινούριο workspace, όπου θα εφαρμόσουμε στο το ίδιο κείμενο input, τα αντίστοιχα blocks για τους αλγορίθμους που ζητούνται στο ερώτημα.

► Για τον αλγόριθμο αντικατάστασης Καίσαρα, ως κλειδί χρησιμοποιήθηκε ο ακέραιος αριθμός 13, δηλαδή το A αντιστοιχεί στο N.

► Για τον αλγόριθμο Playfair, ως κλειδί επιθυμούμε να εισάγουμε τη φράση "DATASECURITYANDPRIVACY" το οποίο μετατρέπεται σε "DATSECURIYNPNVBFGHKLMOQWXZ", δηλαδή αφαιρούμε τα γράμματα που επαναλαμβάνονται και στο τέλος προσθέτουμε τα υπόλοιπα γράμματα του αλφάβητου, όπως είδαμε και στην αντίστοιχη διάλεξη του μαθήματος.

► Συνεχίζοντας με τον αλγόριθμο Hill, βλέπουμε πως για να λειτουργήσει ο αλγόριθμος πρέπει τα γράμματα να είναι σε Uppercase, και να μην υπάρχουν κενά καθώς και σημεία στίξης, εισαγωγικά κτλ. Επομένως, για να το επιτύχουμε αυτό, χρησιμοποιήσαμε τις παρακάτω online υπηρεσίες:

- Για κεφαλαία: <https://www.dcode.fr/uppercase-lowercase>
- για αφαίρεση κενών: <https://www.dcode.fr/spaces-remover>
- για αφαίρεση σημείων στίξης κτλ: <https://www.browserling.com/tools/remove-punctuation>.

► Για τον αλγόριθμο Vigenere ως κλειδί, χρησιμοποιήθηκε το "DANTEANDVIRGIL".

► Τέλος, για τον αλγόριθμο Vernam, χρησιμοποιούμε ένα τυχαίο κλειδί, όπως φαίνεται και στην εικόνα 2..1, καθώς και το αλφάβητο που χρησιμοποιείται. Σε αυτόν τον αλγόριθμο, εισάγουμε ως Input το κείμενο που χρησιμοποιήσαμε και στον αλγόριθμο

Hill, δηλαδή χωρίς κενά, σημεία στίξης κτλ. Θα μπορούσαμε αντίστοιχα να διευρύνουμε το αλφάβητο ώστε να περιέχει τους επιπλέον χαρακτήρες που του λείπουν, όμως είχαμε ήδη έτοιμο το απλοποιημένο κείμενο, οπότε χρησιμοποιήσαμε αυτό.

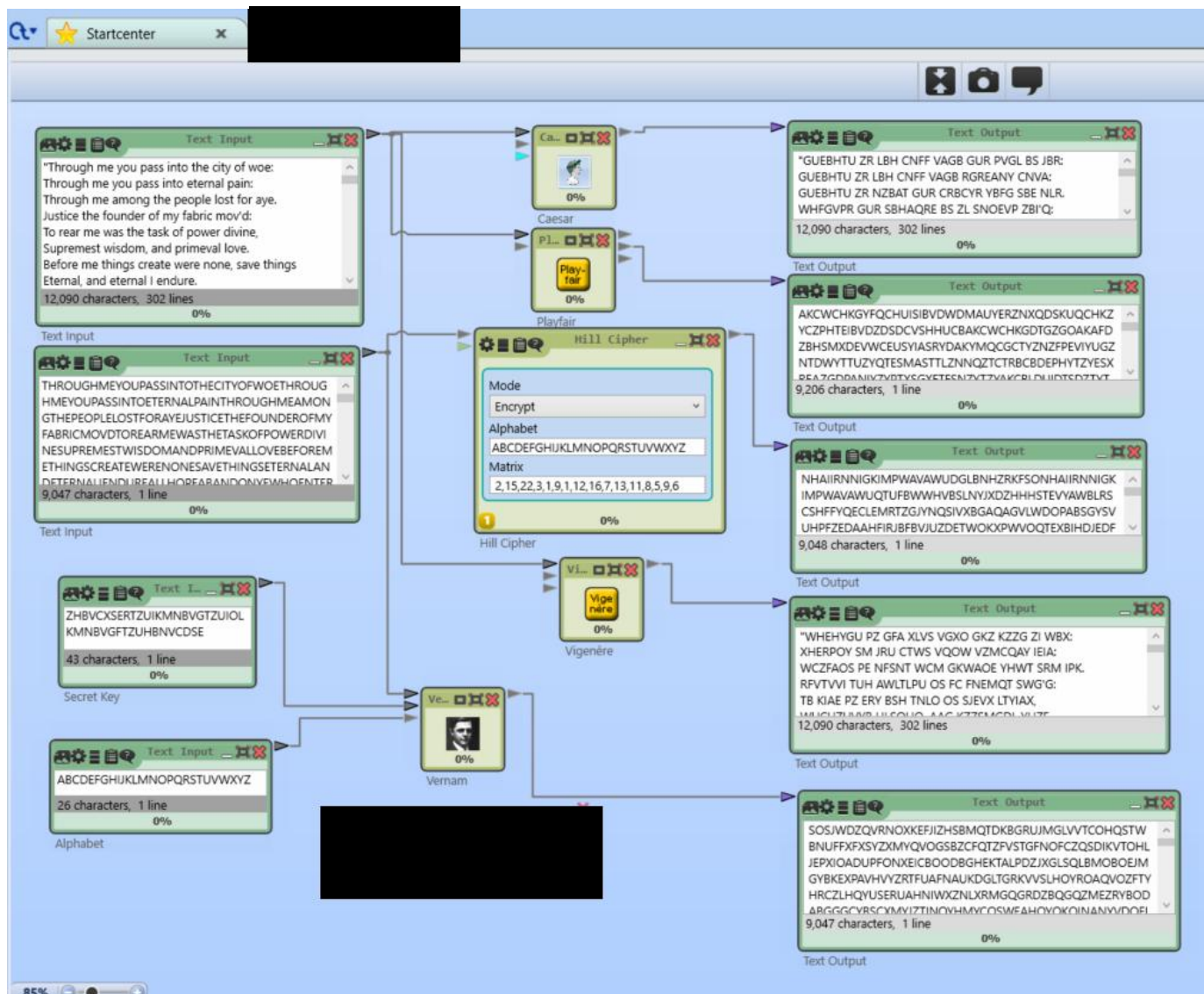


Fig. 1: Classical Cryptography Ciphers

2.2.2

Σε αυτό το ερώτημα, χρησιμοποιούμε το frequency test block του CrypTool προκειμένου να κάνουμε ανάλυση συχνότητας κάθε αλγορίθμου. Καθώς δεν χωράνε όλα τα διαγράμματα σε ένα screenshot, τα παραθέτουμε σε δύο διαφορετικά, με τη σειρά που αναφέρονται οι αλγόριθμοι, δηλαδή: Caesar, Playfair, Hill, Vigenere και Vernam

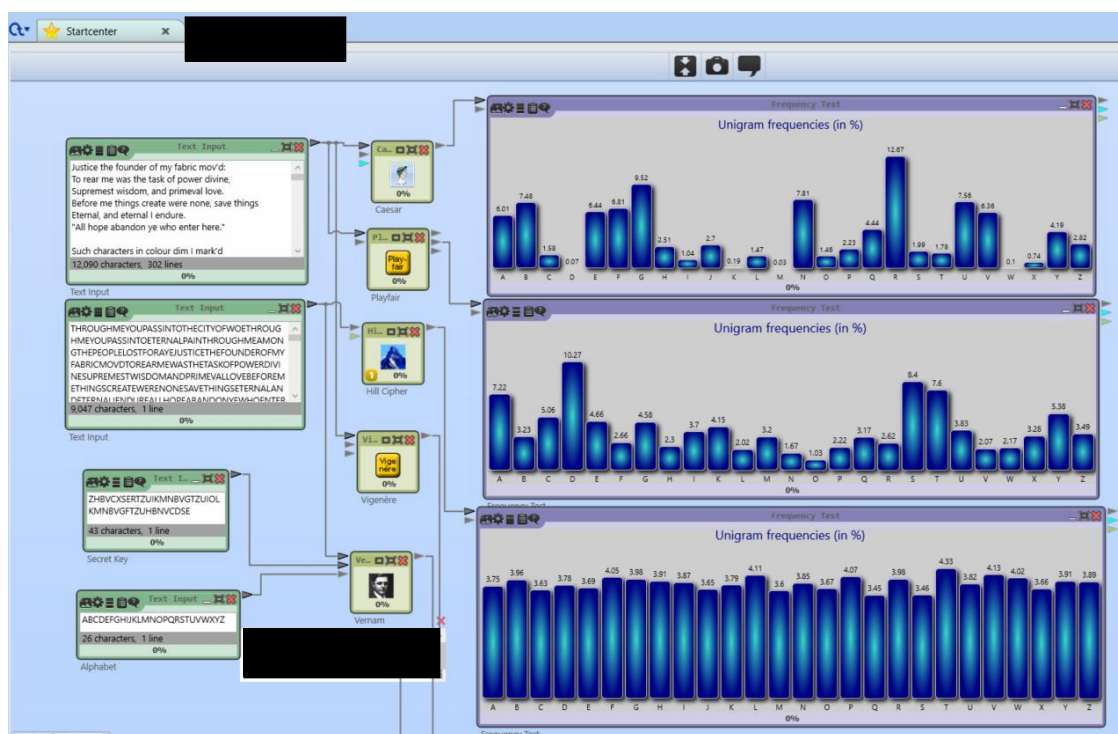


Fig. 2: Caesar, Playfair and Hill ciphers frequency test.

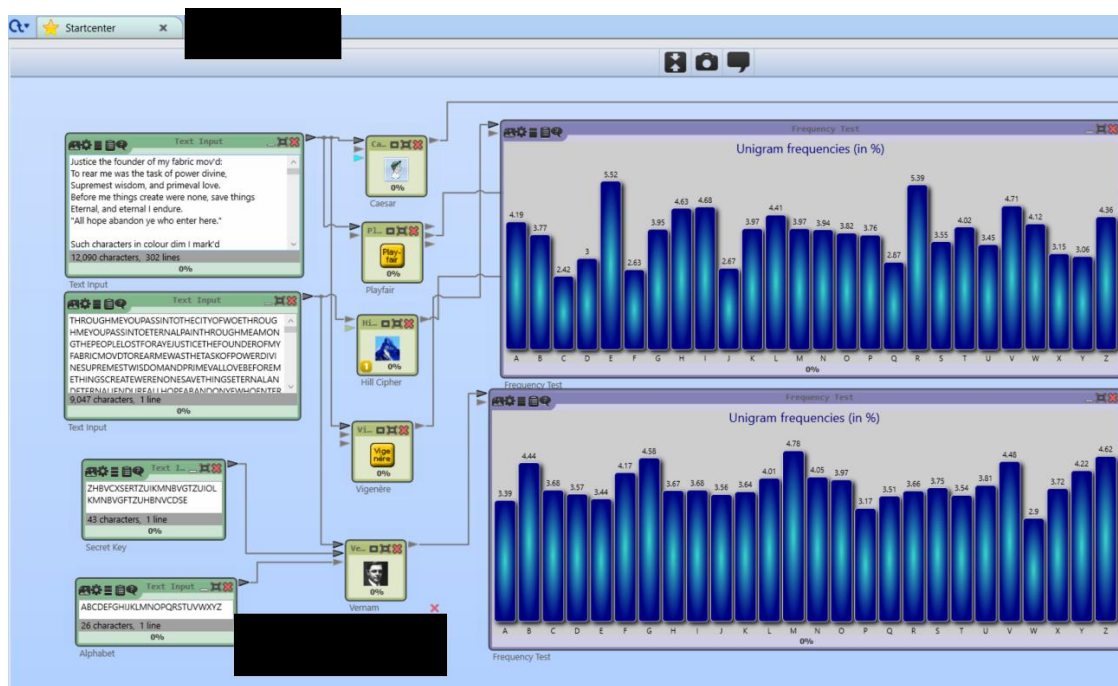


Fig. 3: Vigenere and Vernam ciphers frequency test.

►Όσον αφορά τον **αλγόριθμο του Καίσαρα**, είναι πολύ εύκολο αυτός να σπάσει καθώς υπάρχουν μόνο 25 πρακτικά σενάρια αντιμετάθεσης, επομένως ένα brute-force attack θα ήταν επιτυχής, αρκεί να είμαστε σε θέση να διαβάσουμε το αποκρυπτογραφημένο κείμενο. Εκτός από μια μορφή επίθεσης Brute Force, γνωρίζουμε ότι το γράμμα 'Ε' είναι το πιο συχνό στο αγγλικό αλφάβητο, οπότε χωρίς καμία περαιτέρω γνώση για το αρχικό κείμενο, το μόνο που παίζει ρόλο είναι το ότι έχει μεγάλο μέγεθος, μπορούμε να υποθέσουμε από το αποτέλεσμα της ανάλυσης συχνότητας, ότι το γράμμα 'R' που είναι το πιο συχνό, αντιστοιχεί στο γράμμα 'Ε'. Το γράμμα 'Ε' -> 4 ενώ το 'R' -> 17. Επομένως το κλειδί πρέπει να είναι το 13, το οποίο γνωρίζουμε ότι είναι το σωστό.

►Ο **αλγόριθμος Playfair**, είναι κι αυτός ένας αδύναμος αλγόριθμος, καλύτερος από τον αλγόριθμο του Καίσαρα μόνο επειδή αντιμετωπίζει δυάδες γραμμάτων ως μονάδες στο κείμενο, άρα απαιτεί πίνακα συχνοτήτων εμφάνισης digrams $25 \times 25 = 625$ στοιχείων για την κρυπτοανάλυση (έναντι 26 στοιχείων για τους μονοαλφαθητικούς). Επομένως, ανάλυση συχνοτήτων δυάδων θα ήταν αρκετά χρήσιμη, αφού γνωρίζουμε ότι η πιο συχνή δυάδα διαφορετικών γραμμάτων είναι η 'TH'. Οπότε βλέπουμε

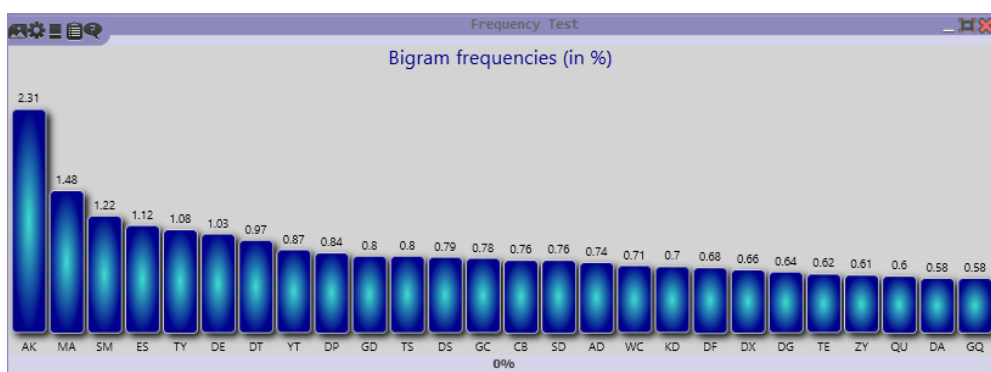


Fig. 4: Playfair Cipher bigram frequency test.

ότι η δυάδα 'AK' είναι το δίγραμμα 'TH'. Το μειονέκτημα αυτού του αλγορίθμου είναι ότι αν αλλάξουμε ένα γράμμα στο plaintext, τότε μόνο ένα γράμμα αλλάζει και στο ciphertext. Άρα, δεδομένου αρκετά μεγάλου ciphertext και ίσως κάποιες γνωστές λέξεις, αυτός ο αλγόριθμος μπορεί να σπάσει γνωρίζοντας ότι το κείμενό μας αποτελείται από >2000 λέξεις, δηλαδή είναι αρκετά μεγάλο. Η απόδοσή του, επομένως, εξαρτάται από το μήκος του κειμένου.

►Όσον αφορά τον **αλγόριθμο Hill**, αυτός βλέπουμε ότι κρύβει τελείως τη συχνότητα εμφάνισης των γραμμάτων. Είναι ισχυρός αλγόριθμος εάν ο attacker γνωρίζει μόνο το κρυπτογραφημένο κείμενο (ciphertext-only) όμως μπορεί να σπάσει εύκολα και να αποκαλυφθεί η δομή του εάν ο attacker γνωρίζει και το αρχικό κείμενο (known-plaintext attack). Παρ'όλα αυτά είναι ένας αλγόριθμος ο οποίος χρησιμοποιείται καθώς ο πολλαπλασιασμός πινάκων που συμβαίνει, ανακατεύει καλά τα γράμματα, οπότε μπορεί να χρησιμοποιηθεί ως μέρος άλλων αλγορίθμων, όπως π.χ. ο AES που μέρος του αλγορίθμου πραγματοποιεί πολλαπλασιασμό πινάκων.

►Για να κρυπτογραφήσουμε ένα plain text με τη χρήση του **αλγορίθμου Vigenere**, χρειαζόμαστε ένα key word, που στην περίπτωσή μας είναι το 'Dante and Virgil'. Είναι σημαντικό εάν θέλουμε να 'σπάσουμε' τον αλγόριθμο Vigenere, να γνωρίζουμε το μέγεθος του κλειδιού, κάτι που μπορεί να επιτευχθεί με μέθοδο

Kasinski. Αν για παράδειγμα βρούμε ένα επαναλαμβανόμενο μοτίβο n χαρακτήρων το οποίο επαναλαμβάνεται κάθε m χαρακτήρες, αυτό υποδηλώνει την ύπαρξη κλειδιού μήκους n ή m χαρακτήρων. Ας δούμε μέσω του Cryptool την ανάλυση Kasinski στο κρυπτοκείμενο του Vigenere: Βλέπουμε ότι υπάρχει μεγάλη συχνότητα ανά

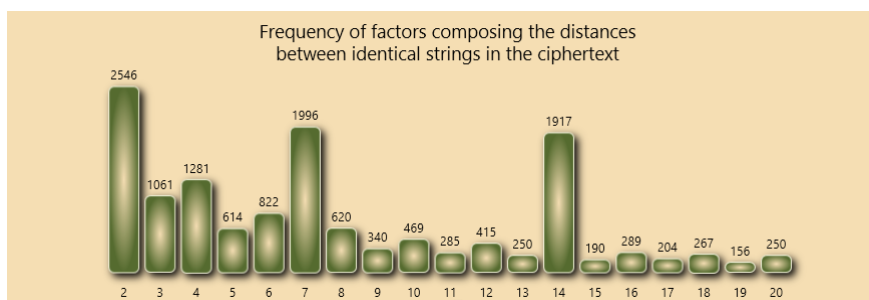


Fig. 5: Kasinski test for Vigenere Cipher.

2 γράμματα, διότι το δίγραμμα 'ΑΝ' υπάρχει δύο φορές μέσα στο κλειδί, μεγάλη συχνότητα ανά 7 γράμματα διότι το γράμμα 'D' υπάρχει στο 1ο και στο 8ο γράμμα οπότε κρυπτογραφείται το ίδιο, και μεγάλη συχνότητα ανά 14 γράμματα που είναι το μήκος του κειμένου. Εμάς θα μας τραβούσαν την προσοχή τα μήκη 7 και 14. Εφόσον γνωρίζουμε το μήκος του κλειδιού, που είναι 14, τότε κάθε 14ο γράμμα θα κρυπτογραφείται με το ίδιο γράμμα. Μπορούμε να χωρίσουμε το cipher text σε 14 κομμάτια, και να ακολουθήσουμε την ίδια διαδικασία με τον αλγόριθμο του Καίσαρα (ανάλυση συχνότητας κτλ). Η απόδοση του αλγορίθμου, εξαρτάται από το μήκος του κλειδιού, επομένως ιδανικό θα ήταν να επιλέξουμε ένα κλειδί μήκους ίσο με του κειμένου, ώστε να μην επαναλαμβάνεται.

► Τέλος, για **τον αλγόριθμο Vernam**, όπως και στον Vigenere, η απόδοση εξαρτάται από το μήκος του κλειδιού, όπου εδώ έχει χρησιμοποιηθεί ένα random. Και εδώ, η αδυναμία είναι οποιοδήποτε είδους επαναλαμβανόμενο Keyword. Επομένως, αυτό λύνεται με το να χρησιμοποιήσουμε κλειδί μήκους ίσο με το αρχικό plain text (one-time pad). Όμως και πάλι, η μέθοδος αυτή δεν διατηρεί την ακεραιότητα του μηνύματος, καθώς με την κατάλληλη αλλαγή key ή ciphertext, μπορεί να αλλάξει το μήνυμα του plaintext και να μεταδοθεί λάθος μήνυμα.

2..3

Για τον αλγόριθμό Transposition, πραγματοποιούμε τα εξής στο CrypTool, χρησιμοποιώντας ως κλειδί το 'DATASECURITYANDPRIVACY', όπως φαίνεται στην εικόνα 2..3. Η ανάλυση συχνότητας του κρυπτοκειμένου δεν μας δίνει κάποια χρήσιμη πληροφορία, καθώς εδώ δεν έχουμε κάποια αντικατάσταση των χαρακτήρων, μόνο πολλαπλές αντιμεταθέσεις. Επομένως, η συχνότητα των χαρακτήρων παραμένει ίδια. Το μήκος του κλειδιού μας είναι 22 χαρακτήρες, επομένως υπάρχουν 22! πιθανοί συνδιασμοί. Όμως, δεν είναι τόσο ασφαλής αυτός ο αλγόριθμος, διότι η γλώσσα έχει συγκεκριμένα χαρακτηριστικά. Θα μπορούσαμε δηλαδή για παράδειγμα να πραγματοποιούμε αντιμεταθέσεις του κρυπτοκειμένου ώσπου να αναγνωρίσουμε κάποιες γνωστές λέξεις, με μεγάλη συχνότητα. Επίσης μπορούμε να εκμεταλλευτούμε

διάφορα χαρακτηριστικά της γλώσσας όπως για παράδειγμα το γεγονός ότι σχεδόν πάντα μετά το γράμμα 'Q' ακολουθεί το γράμμα 'U'. Η απόδοση του αλγορίθμου εξαρτάται από το μήκος του κλειδιού διότι έτσι υπάρχουν εκθετικά μεγαλύτεροι συνδιασμοί λόγω του παραγοντικού, όμως αυτό δεν αυξάνει την ασφάλεια δεδομένου του ότι ο τύπος των επιθέσεων που πραγματοποιούμε για να σπάσουμε αυτόν τον αλγόριθμο δεν είναι Brute force φύσεως.

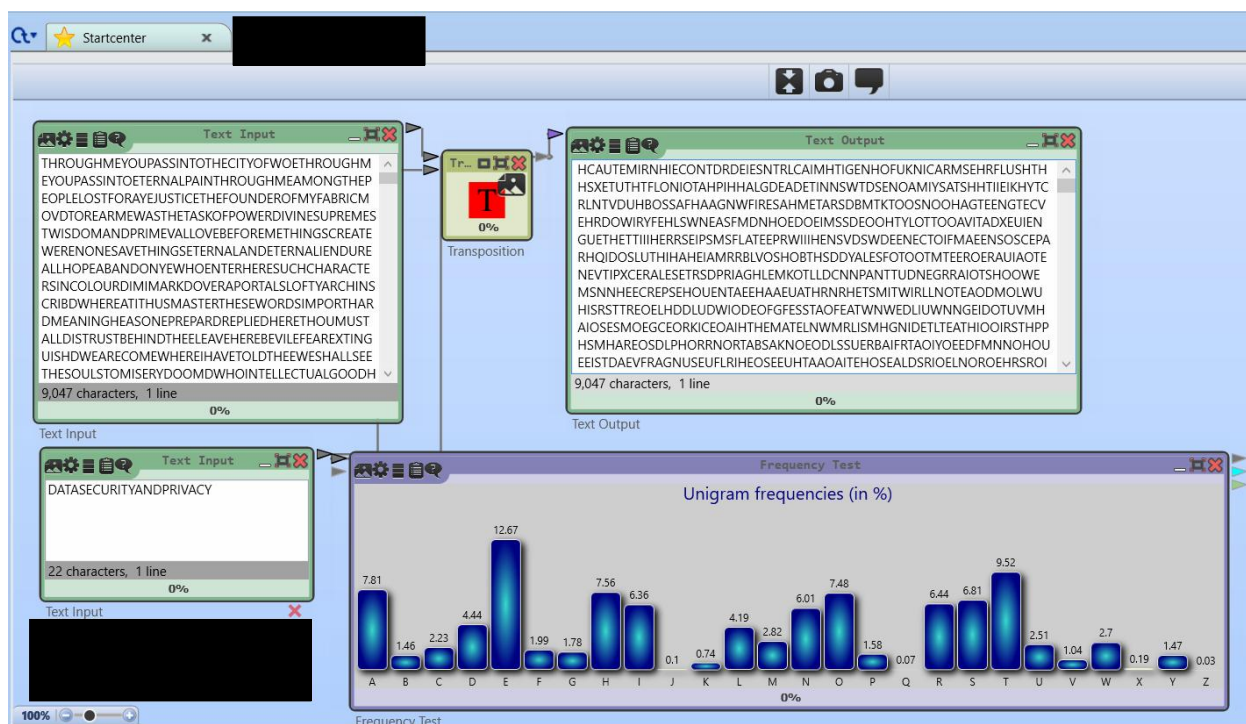


Fig. 6: Transposition Cipher

3. Σύγχρονοι Αλγόριθμοι Κρυπτογράφησης

3..1

Ως αρχικό plain text σε αυτήν την ενότητα, θα χρησιμοποιήσουμε τα πρώτα 16 κεφάλαια από το βιβλίο 'The Great Gatsby'. Το μέγεθος του αρχείου είναι 24 kB.

► Ξεκινάμε με τον XOR cipher, όπου ως κλειδί χρησιμοποιούμε ένα τυχαίο string από 50 χαρακτήρες (50 Bytes), και την έξοδο την περνάμε από ένα string encoder, καθώς το Output του XOR είναι μορφής hex, ώστε να μας εμφανίσει τους ascii χαρακτήρες. Όμως επειδή μπορεί να υπάρχουν διαφορές στην κωδικοποίηση κάποιοι χαρακτήρες δεν εμφανίζονται, όμως αυτό το κάναμε ώστε να έχουμε μια καλύτερη αίσθηση απ'ότι θα είχαμε αν παρουσιάζαμε μορφή hexadecimal χαρακτήρων.

► Συνεχίζουμε με τον αλγόριθμο DES, όπου το plaintext το περνάμε από ένα string decoder καθώς αυτός απαιτεί είσοδο σε μορφή Hex, στην έξοδο όπως και πριν μετατρέπουμε σε ascii. Ως κλειδί χρησιμοποιούμε ένα τυχαίο, μεγέθους 8 bytes

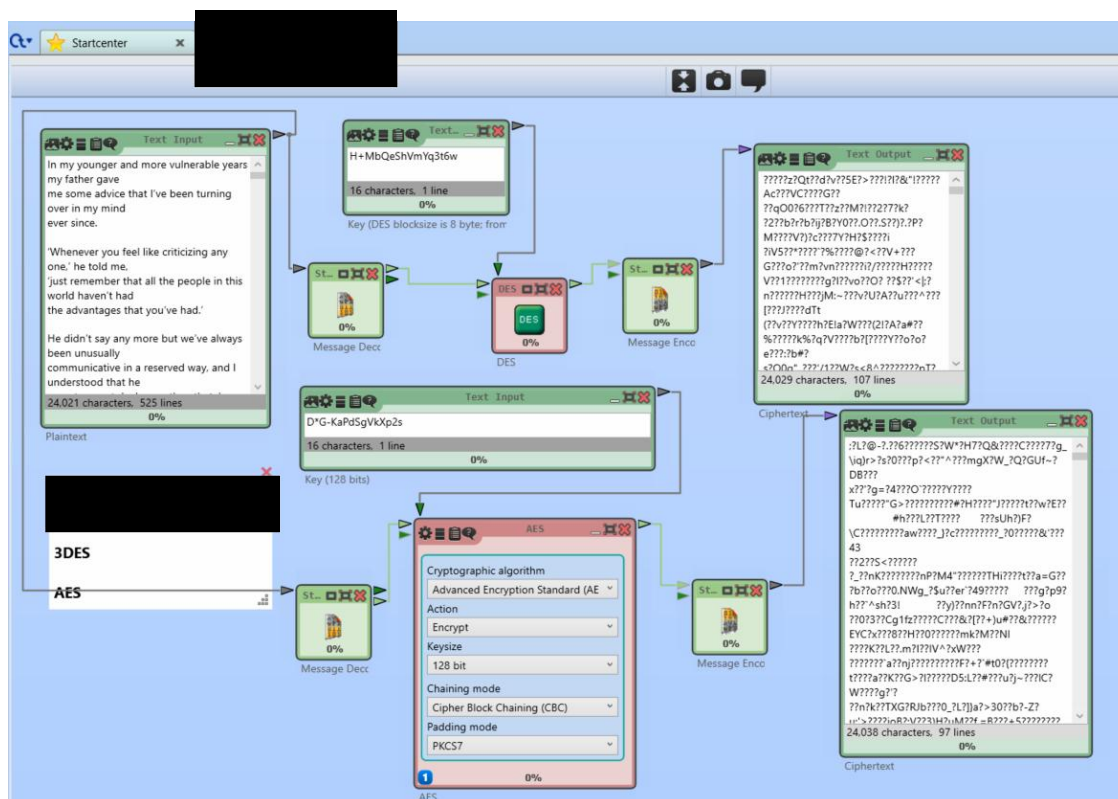


Fig. 8: Triple DES and AES algorithms

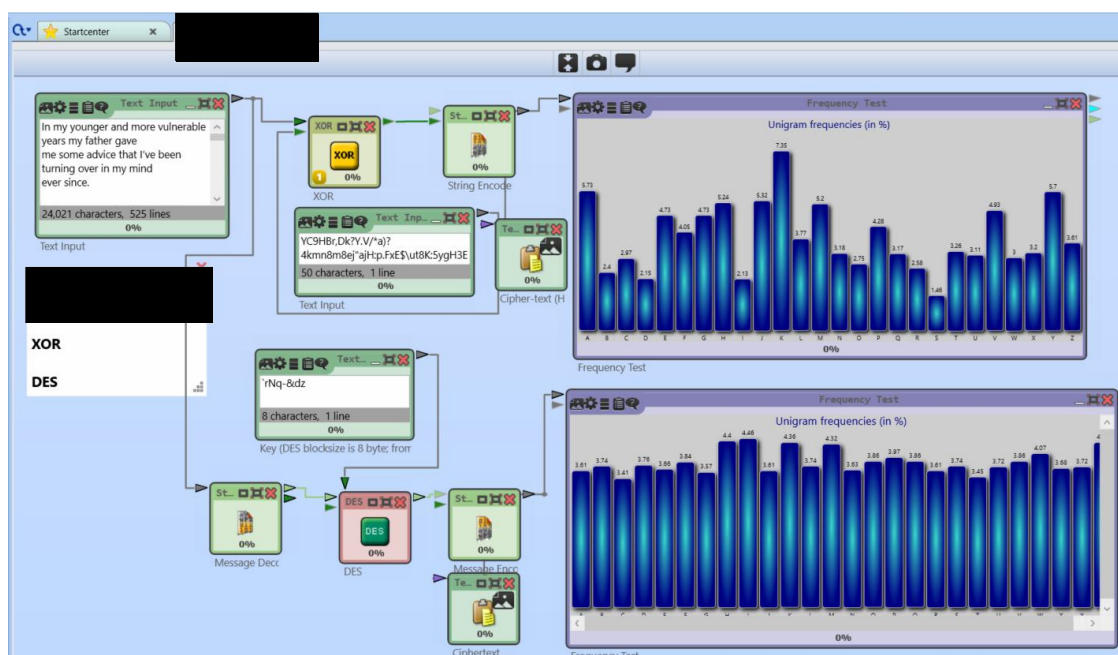


Fig. 9: XOR and DES algorithms frequency analysis

Εξαιτίας των λόγων που σχολιάσαμε παραπάνω, η ανάλυση των συχνοτήτων δεν έχει πολύ νόημα, παρόλα αυτά βλέπουμε πως όλοι οι αλγόριθμοι κρύβουν καλά την κατανομή των συχνοτήτων εάν χρησιμοποιήσουμε ένα τυχαίο κλειδί. Το μόνο που αξίζει να σχολιάσουμε είναι πως ο αλγόριθμος XOR, αν χρησιμοποιήσουμε ένα απλούστερο κλειδί, η ανάλυση συχνότητας δείχνει κορυφές στα γράμματα που

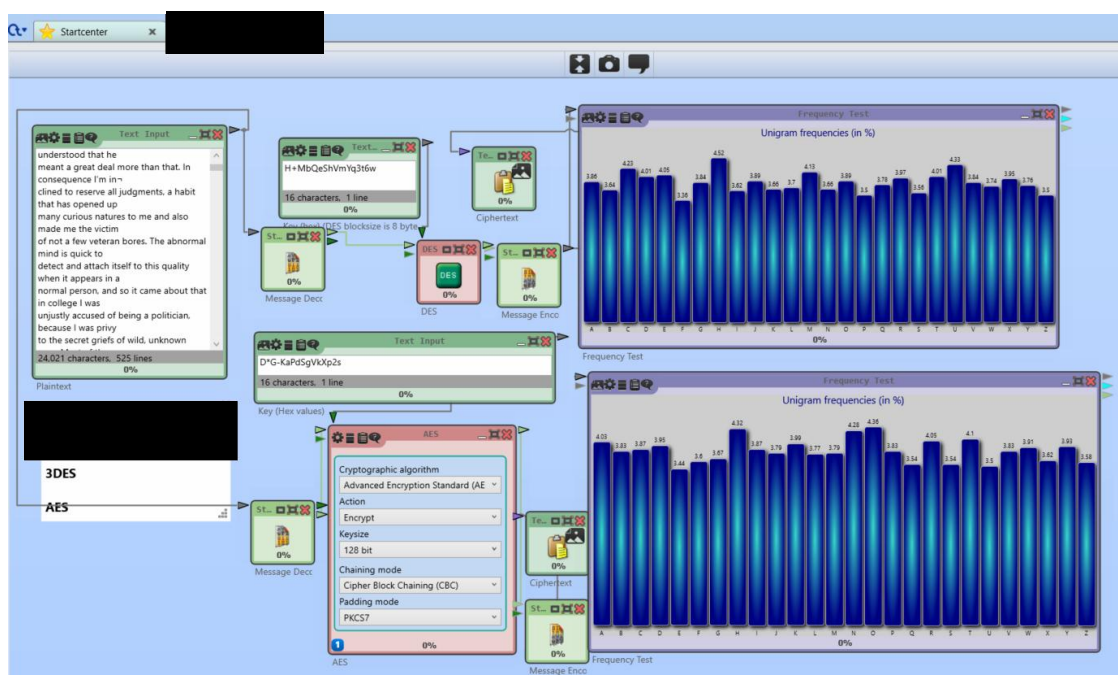


Fig. 10: Triple DES and AES algorithms frequency analysis

εμφανίζονται συχνότερα στο κλειδί.

Ας σχολιάσουμε τα **μεινεκτήματα και πλεονεκτήματα των αλγορίθμων**:

►Όσον αφορά τον αλγόριθμο XOR, μπορεί να είναι ένας σχετικά δυνατός αλγόριθμος, με το σχετικά να εννοούμε ότι δεν μπορεί να σπάσει με τετριμμένες μαθηματικές μεθόδους, αν πληρούνται συγκεκριμένα κριτήρια. Ένα από αυτά τα κριτήρια, είναι το μέγεθος του κλειδιού να είναι ίδιο με το μέγεθος του plaintext, και αν το κλειδί δεν μπορεί να το μαντέψει ο attacker, για παράδειγμα με κάποιο λεξικό ή μαθηματικό τρόπο, δηλαδή πρέπει το κλειδί να είναι truly random. Τότε ο XOR είναι One-time pad και δεν σπάει, όμως αυτό δεν διατηρεί το integrity του κειμένου. Παρόλα αυτά, αν δεν ισχύουν κατ'ανάγκη αυτά, μπορεί κάποιος attacker να εφαρμόσει τον XOR cipher στο κρυπτοκείμενο, με τον εαυτό του μετατοπισμένο κατά κάποιο αριθμό και εξετάζοντας πόσα bytes είναι όμοια. Έτσι μπορεί να εντοπισθεί το μέγεθος του κλειδιού, και αν τα όμοια bytes είναι ίσα κατά κάποιο ποσοστό, τότε έχει μετατοπιστεί κατά ένα ακέραιο πολλαπλάσιο του μεγέθους του κλειδιού. Έτσι, το ελάχιστο ακέραιο πολλαπλάσιο είναι το μέγεθος του κλειδιού. Έτσι μετατοπίζοντας το ciphertext κατά το μέγεθος του κλειδιού, και κάνοντας XOR με τον εαυτό του, έχουμε το Plaintext μετατοπισμένο κατά το μέγεθος του κλειδιού που είναι αρκετό για να πάρουμε μια ιδέα από το περιεχόμενο του plaintext.

►Για τον αλγόριθμο DES, τα πλεονεκτήματά του είναι ότι καθώς έχει μήκος κλειδιού 56 bytes, υπάρχουν 2^{56} δυνατά κλειδιά, που θεωρητικά το καθιστά δυνατό αλγόριθμο ενάντια σε brute force attacks. Επίσης, είναι υπολογιστικά «φθηνός» αλγόριθμος, καθώς χρειάζεται μια συνάρτηση (άρα και συσκευή) τόσο για κρυπτογράφηση όσο και για αποκρυπτογράφηση. Το μόνο που χρειάζεται για την αποκρυπτογράφηση είναι να δοθεί το κλειδί σε αντίστροφη σειρά. Στα μειονεκτήματα, καθώς ο αλγόριθμος σχεδιάστηκε την δεκαετία του 70, για την εποχή ήταν αδύνατο να σπάσει ο αλγόριθμος

με brute force attacks. Όμως, στην εποχή μας, όπου έχει ανέβει εκθετικά η υπολογιστική δύναμη, είναι δυνατό να σπάσει. Συγκεκριμένα βλέπουμε στη wikipedia ότι μια GPU Nvidia GTX 1080Ti που είναι 2 γενιές πίσω από την τελευταία γενιά σήμερα, μπορεί να σπάσει τον DES κατά μέσο όρο σε 15 μέρες με το full search να παίρνεις 30 μέρες, με χρήση του λογισμικού Hashcat. Άλλα μειονεκτήματα του DES είναι ότι υπάρχουν κάποια αδύναμα κλειδιά, καθώς στον διαχωρισμό του κλειδιού σε δύο κομμάτια, αν υπάρχουν συνεχόμενα 1 και 0, μπορεί να προκύψει το ίδιο αποτέλεσμα και να χρησιμοποιείται το ίδιο κλειδί σε όλους τους 16 κύκλους. Γενικότερα, το μέγεθος του κλειδιού στον DES θα μπορούσε να είναι μεγαλύτερο γιαυτό και στην πορεία εφευρέθηκε και ο TripleDES.

►Ο αλγόριθμος TripleDES είναι παρόμοιος λογικής με τον DES, όμως χρησιμοποιούνται τρία διαφορετικά κλειδιά, μήκους 56 bytes, άρα συνολικά το κλειδί είναι μήκους 168 bytes. Φυσικά, αν τα δύο κλειδιά είναι ίδια, τότε αυτός εκφυλίζεται σε απλό DES. Το μεγάλο μέγεθος του κλειδιού, προκαλεί τον TripleDES να απαιτεί μεγαλύτερη υπολογιστική δύναμη να υλοποιηθεί, γιαυτό και στο τέλος αντικαταστάθηκε από τον AES ο οποίος είναι υπολογιστικά «οικονομικότερος». Παρόλα αυτά, ο TripleDES είναι ένας ασφαλής αλγόριθμος.

►Τέλος, ο αλγόριθμος AES, χρησιμοποιεί blocks των 128 bits, με το μέγεθος του κλειδιού να είναι 128, 192 ή 256 bits. Ως πλεονεκτήματα, μπορούμε να θέσουμε το γεγονός ότι είναι υπολογιστικά φθηνότερος από ότι ο TripleDES, υλοποιείται τόσο από hardware όσο και από software, και είναι ο πλέον διαδεδομένος τρόπος κρυπτογραφίας ευαίσθητων δεδομένων καθώς δεν υπάρχει μέχρι σήμερα κάποια πρακτική επίθεση που επιτρέπει σε κάποιον που δεν έχει το κλειδί να αποκτήσει πρόσβαση στα κρυπτογραφημένα δεδομένα, εφόσον ο αλγόριθμος έχει υλοποιηθεί σωστά. Τα μόνα μειονεκτήματα που μπορούμε να εντοπίσουμε είναι ότι κάθε block κρυπτογραφείται με τον ίδιο τρόπο.

3..2

Ο RSA επεξεργάζεται τμήματα, όπου τόσο το αρχικό κείμενο M , όσο και το τελικό C είναι ακέραιοι μεταξύ 0 και $N - 1$. Η κρυπτογράφηση γίνεται μέσω της σχέσης: $C = M^e \bmod N$, ενώ η αποκρυπτογράφηση: $M = C^d \bmod N$. **Αμφότεροι αποστολέας και παραλήπτης γνωρίζουν τα N , e (e =δημόσιο κλειδί παραλήπτη), ενώ μόνο ο παραλήπτης γνωρίζει το d (ιδιωτικό του κλειδί).** Με τη χρήση του RSA key generator block στο cryptool, παράγουμε ζεύγος κλειδιών για τους χρήστες A και B, όπως φαίνεται στην εικόνα 11.

3..3

Στη συνέχεια, εφαρμόζουμε την διάταξη που αναφέρεται στην εκφώνηση, όπου ο χρήστης A κρυπτογραφεί το μήνυμα X με το public key του παραλήπτη B (N, e). Στη συνέχεια ο παραλήπτης χρησιμοποιεί το private key του (N, d) για την αποκρυπτογράφηση του μηνύματος. Σημειώνεται, πως το κλειδί d του χρήστη B χρησιμοποιείται μόνο

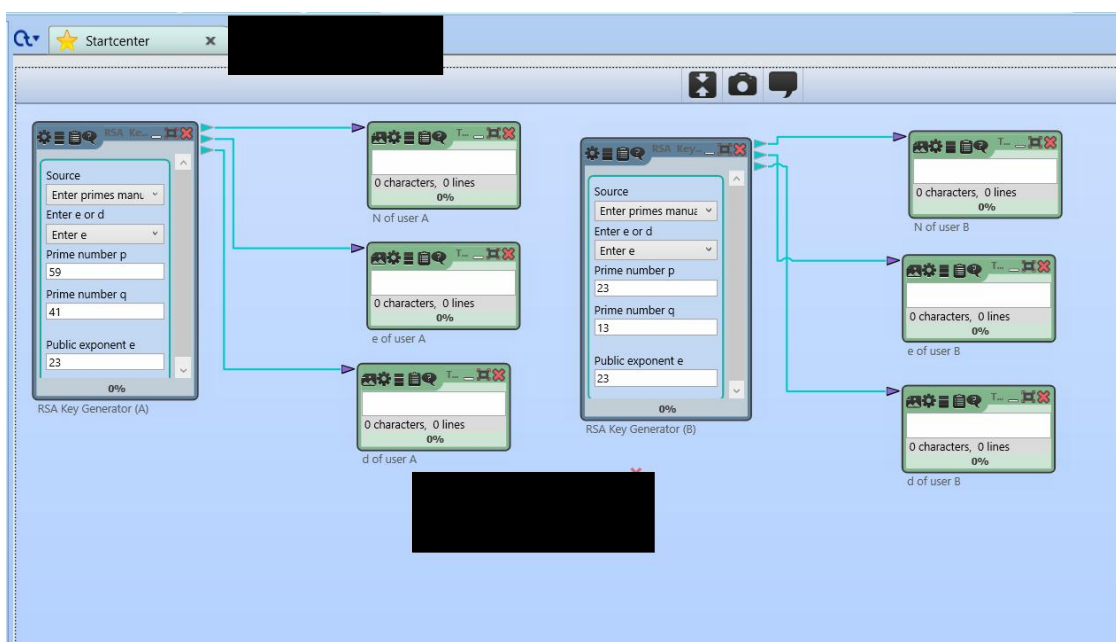


Fig. 11: RSA key generation for users A and B.

στο RSA block που είναι για αποκρυπτογράφηση, καθώς δεν φαίνεται καθαρά στο σχήμα 12 όπου είναι η ζητούμενη διάταξη. Επίσης, το τελικό αποκρυπτογραφημένο κείμενο το περνάμε από ένα string encoder ώστε να είναι σε μορφή κειμένου και όχι σε μορφή Hex.

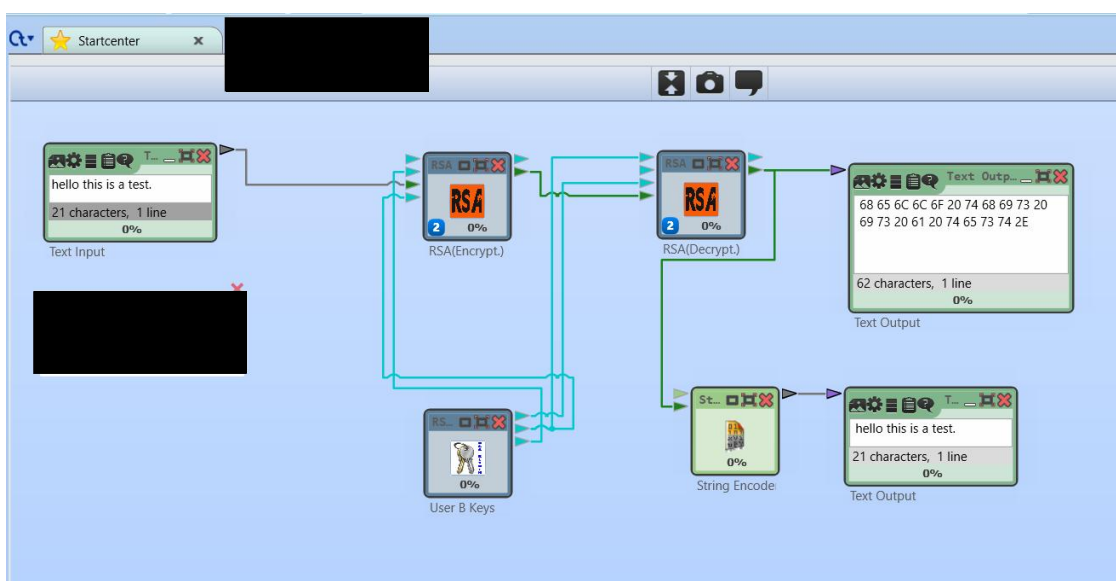


Fig. 12: User A sends a message to user B.

3..4

Συνεχίζουμε με την διάταξη για την πιστοποίηση της ταυτότητας (authentication) κάποιας οντότητας (π.χ., χρήστης) αλλά και για την διασφάλιση ότι το μήνυμα δεν έχει υποστεί αλλαγές (integrity), όπου ο χρήστης A κρυπτογραφεί το μήνυμα X με το private key του (N,d). Στη συνέχεια ο παραλήπτης χρησιμοποιεί το public Key του A (N,e) για την αποκρυπτογράφηση του. Επιτυχημένη αποκρυπτογράφηση πιστοποιεί ότι όντως έχει κρυπτογραφηθεί με το private key που γνωρίζει μόνο ο χρήστης A. Όπως φαίνεται στο σχήμα 13, έχουμε επιτυχής αποκρυπτογράφηση.

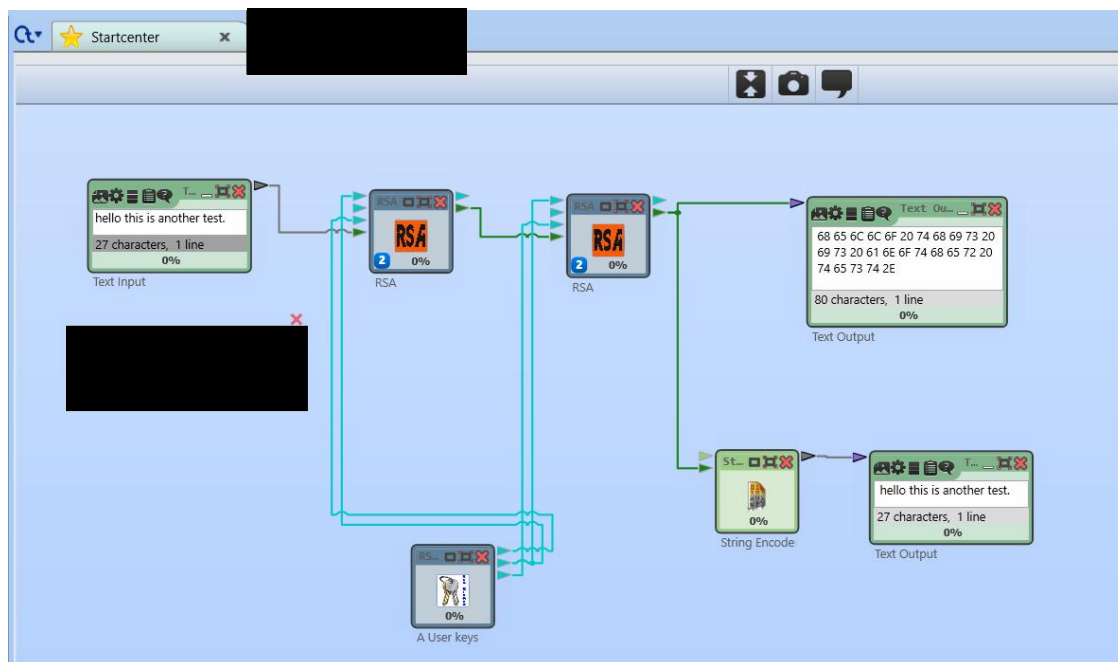


Fig. 13: User A sends a message to user B.

4. Ανίχνευση Εισβολής

Εκκινάμε εγκαθιστώντας το snort και το wireshark στον υπολογιστή μας, σε λειτουργικό περιβάλλον Linux Ubuntu, με τις εντολές:

```
$ sudo apt-get install snort
$ sudo apt-get install wireshark
```

Αφού εξοικειωνόμαστε με τις βασικές εντολές, ξεκινάμε με τα ζητούμενα.

4..1

Με την εντολή `ifconfig` εντοπίζουμε την διεπαφή, η οποία έχει και όνομα `wlx78447699` και `inet 192.168.1.9` όπου είναι η local IP μας. Το όνομα της διεπαφής είναι τέτοιο και όχι π.χ. το `eth0` γιατί είναι USB Wi-Fi adapter.

► Τρέχουμε την εντολή: `sudo snort -v -i wlx78447699`,

► Σε ένα άλλο terminal τρέχουμε την εντολή: `ping twitter.com`.

```
:/var/log/snort$ ping twitter.com
PING twitter.com (104.244.42.129) 56(84) bytes of data:
64 bytes from 104.244.42.129: icmp_seq=1 ttl=54 time=69.8 ms
64 bytes from 104.244.42.129: icmp_seq=2 ttl=54 time=71.7 ms
64 bytes from 104.244.42.129: icmp_seq=3 ttl=54 time=72.2 ms
64 bytes from 104.244.42.129: icmp_seq=4 ttl=54 time=82.6 ms
64 bytes from 104.244.42.129: icmp_seq=5 ttl=54 time=72.0 ms
64 bytes from 104.244.42.129: icmp_seq=6 ttl=54 time=72.3 ms
^C
--- twitter.com ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 14055ms
rtt min/avg/max/mdev = 69.845/73.424/82.595/4.181 ms
```

```
=====
WARNING: No preprocessors configured for policy 0.
01/12-17:35:39.843345 192.168.1.9 -> 104.244.42.129
ICMP TTL:64 TOS:0x0 ID:5736 Iplen:20 DgnLen:84 DF
Type:8 Code:0 ID:12 Seq:3 ECHO
=====
WARNING: No preprocessors configured for policy 0.
01/12-17:35:39.915465 104.244.42.129 -> 192.168.1.9
ICMP TTL:54 TOS:0x0 ID:1932 Iplen:20 DgnLen:84 DF
Type:0 Code:0 ID:12 Seq:3 ECHO REPLY
=====
```

Όπως βλέπουμε, κάνοντας `ping` το `twitter.com` βλέπουμε ότι η IP του είναι η `104.244.42.129`, και δεξιά βλέπουμε ότι το snort όντως εντόπισε το πακέτο που στείλαμε μέσω της εντολής `ping`, γνωρίζοντας την IP μας από πριν, καθώς και το πακέτο το οποίο απάντησε ο server στο twitter. Επειδή το output του snort είναι αρκετά μεγάλο, περιοριστήκαμε σε ένα packet μόνο.

4..2

Για αυτό το ερώτημα, φτιάχνουμε έναν νέο φάκελο με την εντολή `mkdir /logs` ώστε να βάζουμε όλα τα αρχεία `.log` που παράγει το snort.

► Στη συνέχεια τρέχουμε την εντολή `sudo snort -v -i wlx78447699 -l ./logs`, ώστε το log αρχείο να μπει στον φάκελο που δημιουργήσαμε.

► Στη συνέχεια ανοίγουμε έναν browser και κάνουμε κλίκ σε διάφορες σελίδες, όπως το `www.ntua.gr` και το `www.hackthebox.eu`.

► Κλείνουμε την καταγραφή, πάμε στο φάκελο logs και με την εντολή `sudo wireshark snort.log.1610469231` τρέχουμε το αρχείο.

► Στο wireshark, ενεργοποιούμε το

Edit -> Preferences -> Name Resolution -> Resolve network (IP) addresses, για ευκολότερη ανάγνωση των domains που θέλουμε να εντοπίσουμε.

► Στο wireshark ψάχνουμε τα domains των ιστοσελίδων που επισκεφτήκαμε, και ανάμεσα σε 7093 που πρόλαβαν να καταγραφούν εντοπίζουμε τα εξής:

1965 2.112457	www.ntua.gr	nika.local	TLSv1.2	1506 Server Hello	Seq=518	Ack=1421	Win=63104	Len=0	TlsVer=3756	5094 13.413928	nika	speedport.ip	DNS	82 Standard query 0x8082 A run-static.pandora.net				
1966 2.112479	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=1421	Win=63104	Len=0	TlsVer=3756	5095 13.413962	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=67071	Ack=2488	Win=78656	Len=1360	[TCP]	
1967 2.112551	www.ntua.gr	nika.local	TCP	1506 443 -> 33696 [ACK] Seq=1421	Ack=518	Win=65956	Len=1420	TlsVer=1	5096 13.414088	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=68431	Ack=2488	Win=78656	Len=1360	[TCP]	
1968 2.112560	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=2841	Win=61096	Len=0	TlsVer=3756	5097 13.414099	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=69791	Win=72320	Len=9		
1969 2.120620	www.ntua.gr	nika.local	TCP	1342 443 -> 33696 [PSH, ACK] Seq=2841	Ack=518	Win=65956	Len=1256	TS	5098 13.415197	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=69791	Ack=2488	Win=78656	Len=1360	[TCP]	
1970 2.120630	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=4897	Win=61312	Len=0	TlsVer=3756	5099 13.415989	www.hackthebox.eu	nika.local	TLSv1.3	245 Application Data					
1971 2.120658	nika.local	www.ntua.gr	TLSv1.2	1506 Server Hello	Seq=518	Ack=1421	Win=63104	Len=0	TlsVer=3756	5100 13.415999	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=71222	Win=77824	Len=9	
1972 2.120670	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=2841	Win=65956	Len=0	TlsVer=3756	5101 13.416071	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=71222	Ack=2488	Win=78656	Len=1360	[TCP]	
1973 2.120756	www.ntua.gr	nika.local	TCP	1506 443 -> 33696 [ACK] Seq=1421	Ack=518	Win=65956	Len=1420	TlsVer=3	5102 13.416074	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=72582	Ack=2488	Win=78656	Len=1360	[TCP]	
1974 2.120770	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=2841	Win=65956	Len=0	TlsVer=3756	5103 13.416078	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=71222	Win=82368	Len=9		
1975 2.120967	www.ntua.gr	nika.local	TCP	1342 443 -> 33696 [PSH, ACK] Seq=2841	Ack=518	Win=65956	Len=1256	TS	5104 13.416084	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=74042	Ack=2488	Win=78656	Len=1360	[TCP]	
1976 2.120977	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=4897	Win=61312	Len=0	TlsVer=3756	5105 13.416249	www.hackthebox.eu	nika.local	TLSv1.3	1434 Application Data (TCP segment of a reassembled PDU)					
1977 2.127940	www.ntua.gr	nika.local	TCP	1506 443 -> 33696 [ACK] Seq=4897	Ack=518	Win=65956	Len=1420	TlsVer=1	5106 13.416255	nika.local	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=71222	Win=82368	Len=9	
1978 2.127949	www.ntua.gr	nika.local	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=4897	Win=61312	Len=0	TlsVer=3756	5107 13.416264	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=71222	Ack=2488	Win=78656	Len=1360	[TCP]	
1979 2.128950	www.ntua.gr	nika.local	TLSv1.2	1377 Certificate, Server Key Exchange, Server Hello Done					5108 13.416328	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=71222	Ack=2488	Win=78656	Len=1360	[TCP]	
1980 2.128960	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=8808	Win=61312	Len=0	TlsVer=3756	5109 13.416332	nika.local	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=71222	Win=82368	Len=9	
1981 2.129228	www.ntua.gr	nika.local	TCP	1506 443 -> 33696 [ACK] Seq=4897	Ack=518	Win=65956	Len=1420	TlsVer=3	5110 13.417139	www.hackthebox.eu	nika.local	TLSv1.3	416 Application Data					
1982 2.129229	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=8808	Win=61312	Len=0	TlsVer=3756	5111 13.417282	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=72924	Ack=2488	Win=78656	Len=1360	[TCP]	
1983 2.129420	www.ntua.gr	nika.local	TLSv1.2	1377 Certificate, Server Key Exchange, Server Hello Done					5112 13.417287	nika.local	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=81584	Win=99584	Len=9	
1984 2.129425	nika.local	www.ntua.gr	TCP	86 33696 -> 443 [ACK] Seq=518	Ack=8808	Win=61312	Len=0	TlsVer=3756	5113 13.417409	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=81584	Ack=2488	Win=78656	Len=1360	[TCP]	
1985 2.133262	nika.local	www.ntua.gr	TLSv1.2	212 Client Key Exchange, Change Cipher Spec, Encrypted Handshake					5114 13.417615	www.hackthebox.eu	nika.local	TCP	1434 443 -> 46176 [ACK] Seq=82544	Ack=2488	Win=78656	Len=1360	[TCP]	
1986 2.134162	nika.local	www.ntua.gr	TLSv1.2	212 Client Key Exchange, Change Cipher Spec, Encrypted Handshake					5115 13.417621	nika.local	www.hackthebox.eu	nika.local	TCP	74 46176 -> 443 [ACK] Seq=2488	Ack=83904	Win=104960	Len=9	
1987 2.134670	nika.local	www.ntua.gr	TLSv1.2	829 Application Data					5116 13.417672	www.hackthebox.eu	nika.local	TLSv1.3	245 Application Data					

Έτσι, συμπεραίνουμε ότι η καταγραφή έχει γίνει όπως πρέπει, καθώς βλέπουμε στο wireshark τα domains τα οποία επισκεφτήκαμε με τον browser.

4..3

Σε αυτό το υποερώτημα, θα συντάξουμε κανόνα που αφορά κίνηση προς την πόρτα 80 (http) ή 443 (https) προς την ιστοσελίδα του μεταπτυχιακού ΕΔΕΜΜ, δηλαδή: dsml.ece.ntua.gr. Για να εντοπίσουμε την IP, απλώς ανοίγουμε ένα terminal και κάνουμε ping dsml.ece.ntua.gr και αυτό μας γυρνάει την IP την οποία θα χρησιμοποιήσουμε στον κανόνα μας. Διαλέγουμε συγκεκριμένα αυτήν την ιστοσελίδα ώστε να υπάρχει κάποια σταθερότητα, γιατί αν χρησιμοποιήσουμε για παράδειγμα κάποιο διαφορετικό ιστότοπο όπως google, twitter ή αυτά που επισκεφτήκαμε στο προηγούμενο ερώτημα, αυτά κάθε φορά που την επισκεπτόμαστε μπορεί να έχει διαφορετική IP, και όχι αυτή που έχουμε στον κανόνα, καθώς τέτοιες μεγάλες υπηρεσίες πρέπει να είναι πάντα διαθέσιμες, επομένως για να είναι fault tolerant υπάρχουν πολλοί διαφορετικοί hosts σε διαφορετικά data centers διασκορπισμένα σε μια γεωγραφική περιοχή που τρέχουν την ίδια υπηρεσία. Ο κανόνας που συντάσσουμε είναι ο εξής:

```
log tcp 192.168.1.0/24 any -> 147.102.11.50 [80,443]
```

όπου τον αποθηκεύουμε σε ένα αρχείο rules.txt και τον εκτελούμε με την παρακάτω εντολή, ενώ όσο τρέχει επισκεπτόμαστε με τον browser μας την ιστοσελίδα του DSML.

```
sudo snort -v -i wlan78447699a8c6 -l ./logs/ -c rules.txt
```

Συγκεκριμένα, ο κανόνας απομονώνει κίνηση πρωτοκόλλου TCP (καθώς δεν υποστηρίζεται η εντολή any) από οποιαδήποτε IP και Port του τοπικού μας δικτύου προς την IP της ιστοσελίδας του DSML και συγκεκριμένα στις πόρτες 80 (http) και 443 (https). Ο κανόνας log καταγράφει το πακέτο σε αρχείο, το οποίο με την εντολή -l ./logs/ το αποθηκεύει στον κατάλληλο φάκελο. Σημειώνουμε πως η γραφή: 192.168.1.0/24 περιλαμβάνει όλες τις IP στο τοπικό μας δίκτυο, από την 192.168.1.0 ως την 192.168.1.255. Ο υπολογιστής μας συγκεκριμένα είναι η 192.168.1.9.

Στη συνέχεια, στο snort με την παράμετρο -r, διαβάζουμε το log file το οποίο δημιουργήθηκε με τον παραπάνω κανόνα, με την εντολή:

```
sudo snort -v -r ./logs/snort.log.1611333893 -c rules.txt
```

Επειδή το output της εντολής είναι αρκετά μεγάλο, στο screenshot 4.3 δείχνουμε ένα από τα 45 πακέτα τα οποία καταγράφηκαν, το οποίο περιγράφει την μεταφορά πακέτου από τον υπολογιστή μας με IP 192.168.1.9 και port 3886, προς την https port της ιστοσελίδας του DSML. Επίσης, στο παρακάτω screenshot ??, βλέπουμε

```

=====
WARNING: No preprocessors configured for policy 0.
01/22-18:45:00.274678 192.168.1.9:38846 -> 147.102.11.50:443
TCP TTL:64 TOS:0x0 ID:24741 IpLen:20 DgmLen:52 DF
***A*** Seq: 0x36D2A74C Ack: 0x44B2A3A5 Win: 0x1E0 TcpLen: 32
TCP Options (3) => NOP NOP TS: 2915570035 3327923663
=====

```

πως από τα 45 πακέτα που καταγράφηκαν, και τα 45 ήταν μορφής TCP, το οποίο είναι αναμενόμενο καθώς αυτό ορίσαμε στον κανόνα μας, όμως είναι καλό να το επιβεβαιώνουμε. Είναι σημαντικό να σημειώσουμε πως οποιαδήποτε κίνηση συνέβη σε αυτήν την επικοινωνία που το πρωτόκολλο ήταν μορφής UDP, δεν έχει καταγραφεί επειδή στον κανόνα μας είχαμε βάλει μόνο TCP, αν και οι δύο ξεχωριστές ports που έχουμε, υποστηρίζουν και τα δύο πρωτόκολλα¹ (TCP, UDP).

```

=====
Packet I/O Totals:
  Received:      45
  Analyzed:      45 (100.000%)
  Dropped:       0 ( 0.000%)
  Filtered:      0 ( 0.000%)
  Outstanding:   0 ( 0.000%)
  Injected:      0
=====
Breakdown by protocol (includes rebuilt packets):
  Eth:           45 (100.000%)
  VLAN:          0 ( 0.000%)
  IP4:           45 (100.000%)
  Frag:          0 ( 0.000%)
  ICMP:          0 ( 0.000%)
  UDP:           0 ( 0.000%)
  TCP:           45 (100.000%)
=====

```

¹ source:[Wikipedia](#)

4..4

Προκειμένου να απομονώσουμε κίνηση http, SMTP, ftp και telnet, πρέπει να μάθουμε για το καθένα σε ποια port ακούει, όπου και βρίσκουμε τις πληροφορίες που χρειαζόμαστε από την [πηγή](#) που χρησιμοποιήσαμε και πριν. Επομένως, έχουμε:

► Για (αμφίδρομη) κίνηση **FTP**:

```
log tcp any any <> any [20,21]
```

► Για κίνηση **http**:

```
log tcp any any <> any 80
```

► Για κίνηση **SMTP**:

```
log tcp any any <> any [25,465,587]
```

► Για κίνηση **telnet**:

```
log tcp any any <> any 23
```

4..5

Αναζητούμε στο διαδίκτυο malware trace files, δηλαδή αρχεία pcap τα οποία μπορούμε να διαβάσουμε με το wireshark, και να εντοπίσουμε τα πακέτα τα οποία μεταφέρουν κάποιο malware στον επισκέπτη κάποιου ιστοτόπου. Σκοπός μας μετά, είναι να γράψουμε έναν κανόνα, ώστε όταν διαβάζουμε με το snort το pcap αρχείο, αυτό να μας κάνει alert ότι ανιχνεύτηκε κάποιο malware.

Διαλέγουμε ένα pcap αρχείο από το <https://www.malware-traffic-analysis.net/>, συγκεκριμένα το <https://www.malware-traffic-analysis.net/2014/11/06/index.html>. Κατεβάζουμε το αρχείο και το ανοίγουμε με το wireshark ώστε να το επεξεργαστούμε.

Τα περισσότερα malwares, είναι στην μορφή κάποιου executable, δηλαδή κατεβαίνουν στον υπολογιστή μας όταν επισκεφτόμαστε κάποια ιστοσελίδα, και εκτελούν κακόβουλο λογισμικό σε αυτό, χωρίς εμείς να το καταλάβουμε. Γι'αυτό, για να εντοπίσουμε πιθανά τέτοια αρχεία, στο wireshark επιθεωρούμε τα http objects τα οποία μεταφέρονται σε όλα τα πακέτα του pcap file. Στο wireshark κάνουμε τα εξής: File - Export Objects - HTTP... και βλέπουμε αυτά που φαίνονται στην εικόνα 14.

Το μάτι μας πέφτει σε αυτά με content type: Application. Επομένως, πρέπει κάπως να ελέγξουμε ποια από αυτά είναι κακόβουλα. Επομένως, τα κάνουμε save, και στην πορεία, με την βοήθεια της εντολής md5sum <filename> στην γραμμή εντολών, παίρνουμε το md5 hash του αρχείου, και το κάνουμε αναζήτηση στην βάση δεδομένων της ιστοσελίδας [Virus Total](#), το αποτέλεσμα της αναζήτησης για το αρχείο με file-name: 1415286120 φαίνεται στην εικόνα 15.

Όμως, δεν είναι μόνο αυτό κακόβουλο, είναι και τα υπόλοιπα αρχεία όπου προέρχονται από τον host grannityrektonaver.co.vu. Παρατηρούμε όμως, ότι και τα

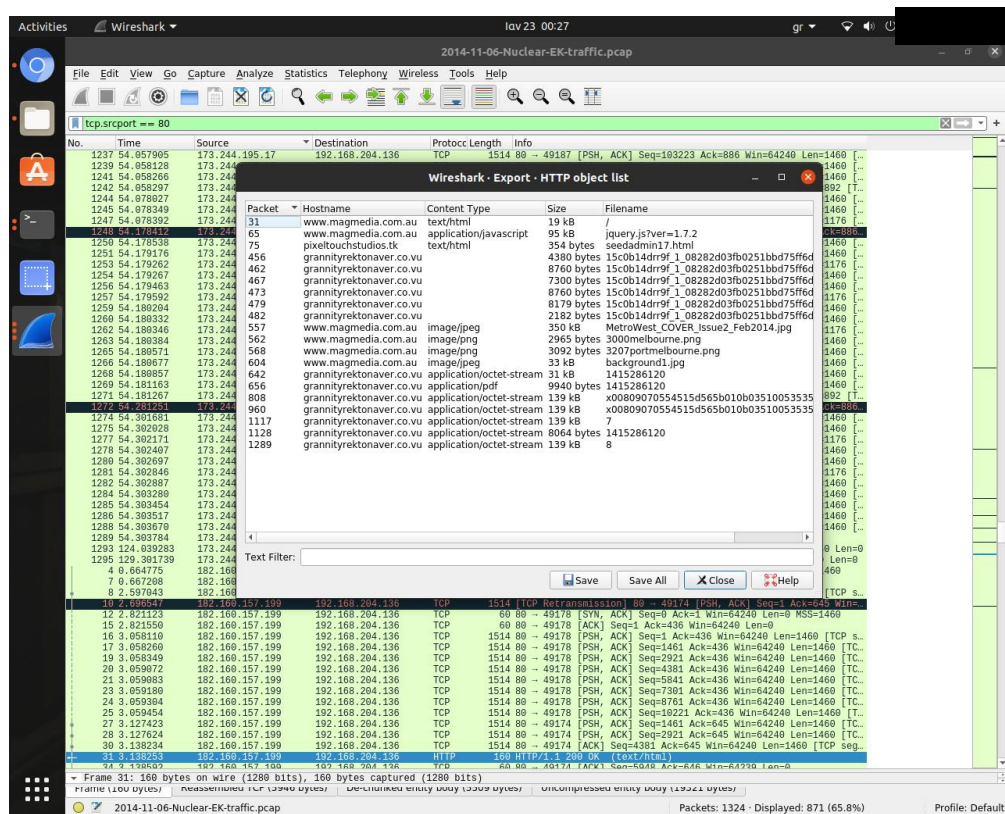


Fig. 14: Wireshark HTTP Objects of pcap file

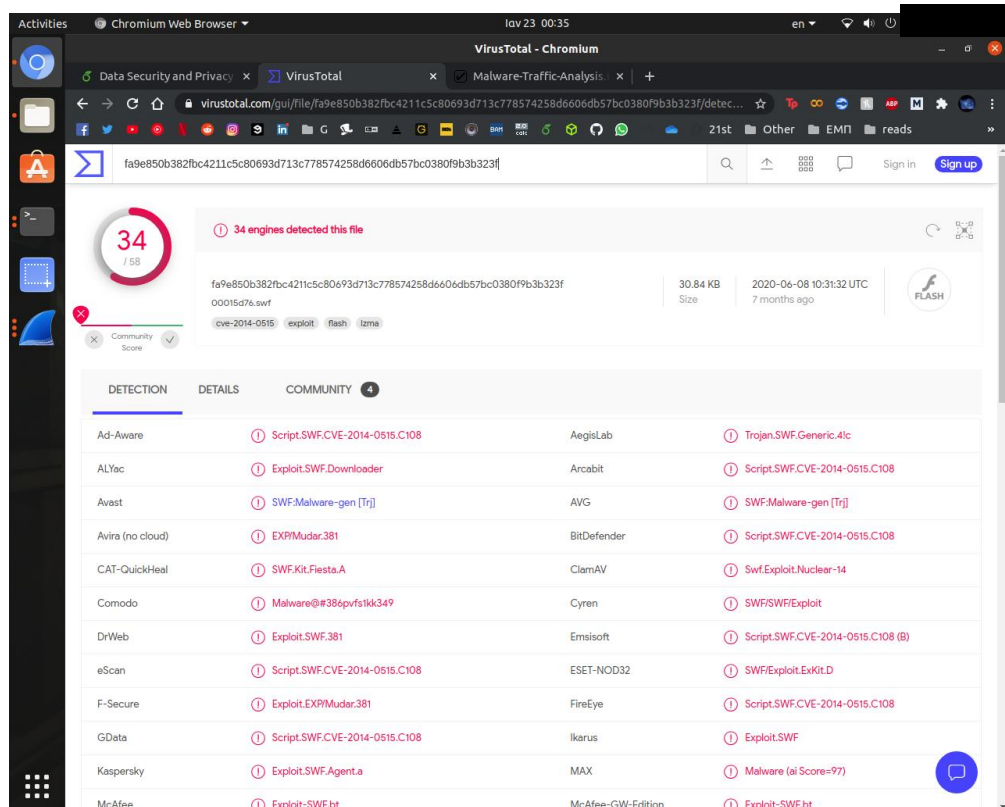


Fig. 15: Search result of file hash on virus total.

7 πακέτα, λόγω του ότι προέρχονται από τον ίδιο host, έχουν ίδιο sourceIP + sourcePORT και ίδιο destinationIP + destinationPORT, εκ των οποίων δύο από αυτά τα πακέτα φαίνονται στην εικόνα 16. Τα Ports τα βρίσκουμε κάνοντας follow TCP stream στο κάθε πακέτο.

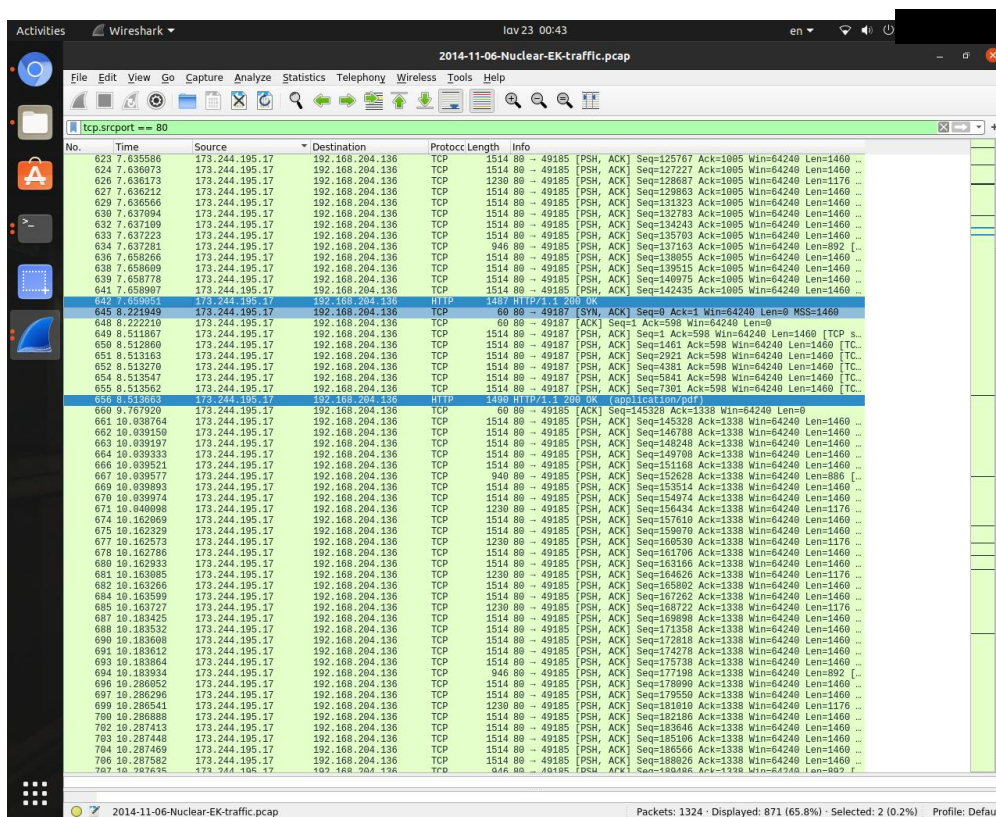


Fig. 16: Screenshot of two of the suspect packages on Wireshark.

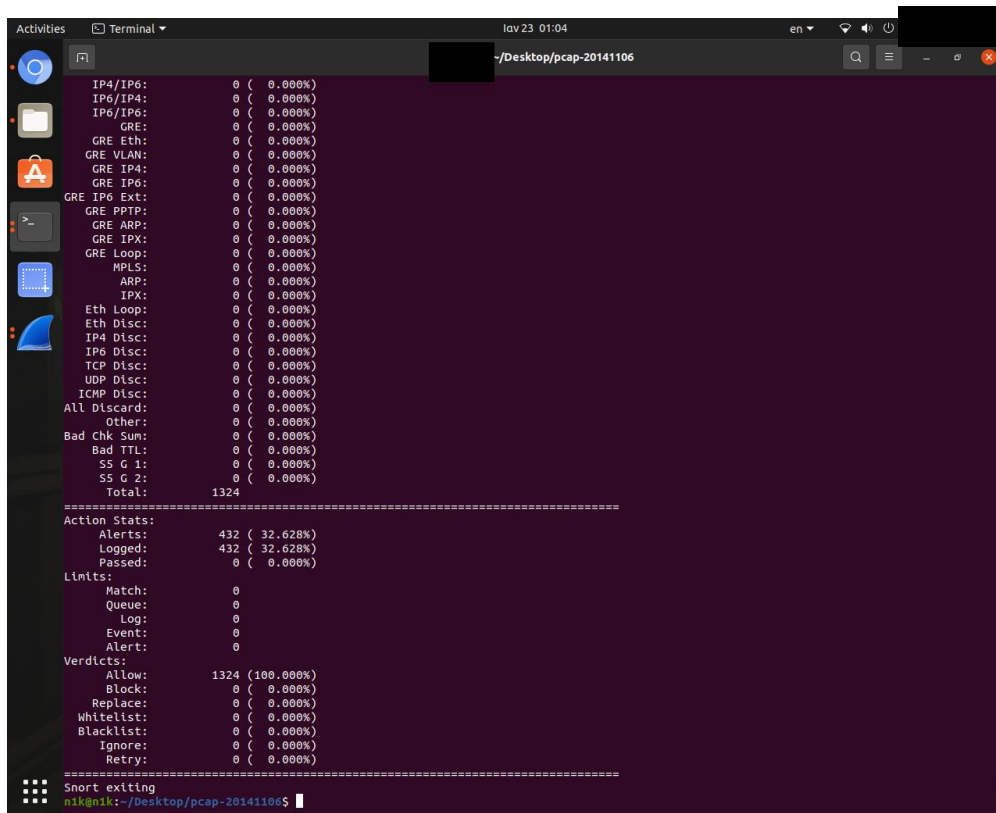
Επομένως, αυτό που πρέπει να περιέχει ο κανόνας που θα γράψουμε, είναι την επικοινωνία μεταξύ source destination. Ο κανόνας για το snort είναι:

```
alert tcp 173.244.195.17 80 -> 192.168.204.136 49185
(msg:"Malware detected by [REDACTED]!!!";sid:1000001; )
```

Αυτό που πρέπει να κάνουμε στη συνέχεια, είναι να τρέξουμε το pcap αρχείο σε playback mode (-r), χρησιμοποιώντας τον κανόνα τον οποίο συντάξαμε. Για να το κάνουμε αυτό, κάνουμε το εξής:

```
sudo snort -v -r 2014-11-06-Nuclear-EK-traffic.pcap -c rule.txt -l .
```

Το αποτέλεσμα αυτού, φαίνεται στην εικόνα 17. Βλέπουμε ότι ο κανόνας μας είναι αρκετά γενικός καθώς φιλτράρει όλη την κίνηση μεταξύ των source destination, γ'αυτό και παίρνει 432 alerts που αποτελούν το 32.6% όλων. Παρατηρούμε πως στον φάκελο που βρισκόμαστε, έχει δημιουργηθεί ένα log file (εξαιτίας του -l) καθώς και ένα αρχείο alert. Για να το διαβάσουμε αυτό, τρέχουμε στην γραμμή εντολών το cat alert. Το output φαίνεται στην εικόνα 18.



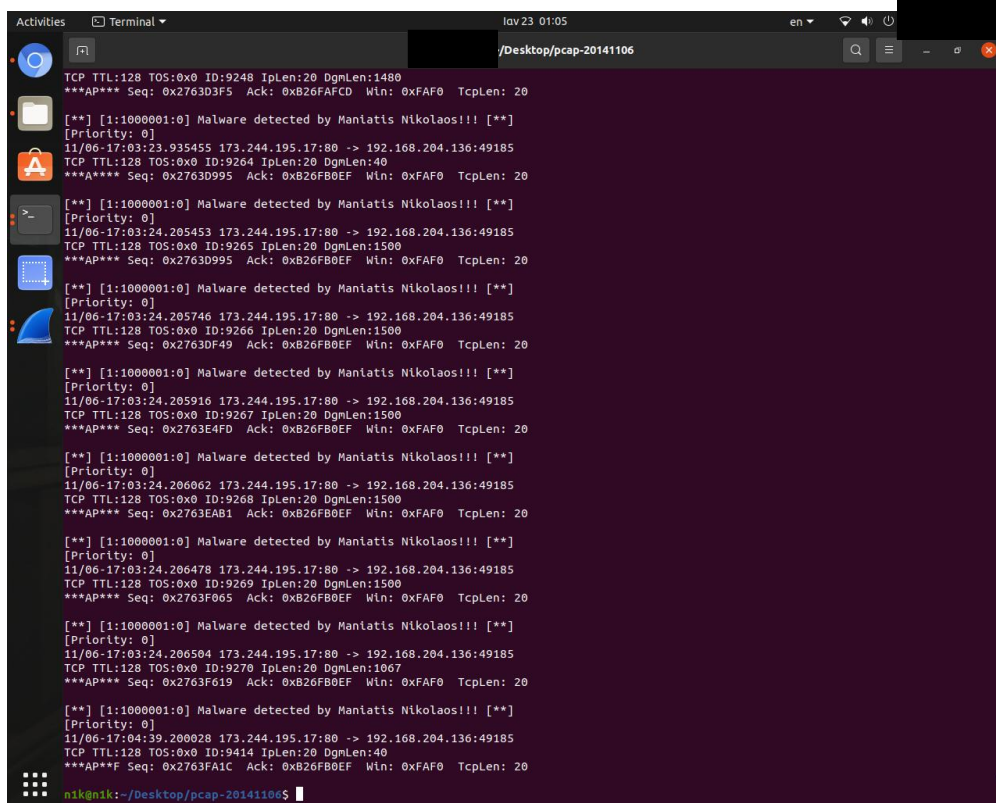
```

Activities Terminal lav 23 01:04 en /Desktop/pcap-20141106

IP4/IP6: 0 ( 0.000%)
IP6/IP4: 0 ( 0.000%)
IP6/IP6: 0 ( 0.000%)
GRE: 0 ( 0.000%)
GRE Eth: 0 ( 0.000%)
GRE VLAN: 0 ( 0.000%)
GRE IP4: 0 ( 0.000%)
GRE IP6: 0 ( 0.000%)
GRE IP6 Ext: 0 ( 0.000%)
GRE PPTP: 0 ( 0.000%)
GRE ARP: 0 ( 0.000%)
GRE IPX: 0 ( 0.000%)
GRE Loop: 0 ( 0.000%)
HPLS: 0 ( 0.000%)
ARP: 0 ( 0.000%)
IPX: 0 ( 0.000%)
Eth Loop: 0 ( 0.000%)
Eth Disc: 0 ( 0.000%)
IP4 Disc: 0 ( 0.000%)
IP6 Disc: 0 ( 0.000%)
TCP Disc: 0 ( 0.000%)
UDP Disc: 0 ( 0.000%)
ICMP Disc: 0 ( 0.000%)
All Discard: 0 ( 0.000%)
Other: 0 ( 0.000%)
Bad Chk Sum: 0 ( 0.000%)
Bad TTL: 0 ( 0.000%)
SS G 1: 0 ( 0.000%)
SS G 2: 0 ( 0.000%)
Total: 1324
=====
Action Stats:
Alerts: 432 ( 32.628%)
Logged: 432 ( 32.628%)
Passed: 0 ( 0.000%)
Limits:
Match: 0
Queue: 0
Log: 0
Event: 0
Alert: 0
Verdicts:
Allow: 1324 (100.000%)
Block: 0 ( 0.000%)
Replace: 0 ( 0.000%)
Whitelist: 0 ( 0.000%)
Blacklist: 0 ( 0.000%)
Ignore: 0 ( 0.000%)
Retry: 0 ( 0.000%)
=====
Snort exiting
n1k@n1k:~/Desktop/pcap-20141106$

```

Fig. 17: Output of snort playback mode on pcap file.



```

Activities Terminal lav 23 01:05 en /Desktop/pcap-20141106

TCP TTL:128 TOS:0x0 ID:9248 Iplen:20 DgmLen:1480
***A*** Seq: 0x2763D3F5 Ack: 0xB26FAFCD Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:23.935455 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9264 Iplen:20 DgmLen:40
***A*** Seq: 0x2763D995 Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:24.205453 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9265 Iplen:20 DgmLen:1500
***A*** Seq: 0x2763D995 Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:24.205746 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9266 Iplen:20 DgmLen:1500
***A*** Seq: 0x2763DF49 Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:24.205916 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9267 Iplen:20 DgmLen:1500
***A*** Seq: 0x2763E4FD Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:24.206062 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9268 Iplen:20 DgmLen:1500
***A*** Seq: 0x2763EAB1 Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:24.206478 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9269 Iplen:20 DgmLen:1500
***A*** Seq: 0x2763F065 Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:03:24.206504 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9270 Iplen:20 DgmLen:1067
***A*** Seq: 0x2763F619 Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

[**] [1:1000001:0] Malware detected by Maniatis Nikolaos!!! [**]
[Priority: 0]
11/06-17:04:39.200028 173.244.195.17:80 -> 192.168.204.136:49185
TCP TTL:128 TOS:0x0 ID:9414 Iplen:20 DgmLen:40
***A*** Seq: 0x2763FA1C Ack: 0xB26FB0EF Win: 0xFAF0 TcpLen: 20

n1k@n1k:~/Desktop/pcap-20141106$

```

Fig. 18: Malware Detection message using our snort custom rule.

Το συγκεκριμένο malware, κατεβαίνει στον υπολογιστή μας με το extension .swf που σημαίνει ότι είναι ένα swf file. Αυτά τα αρχεία έχουν συνήθως interactive περιεχόμενο και χρησιμοποιούνται από το Adobe flash, π.χ. flash player στους browsers. Όμως, στην πραγματικότητα, αυτό το αρχείο περιέχει μέσα ένα silverlight exploit. Το silverlight είναι ένα αντίστοιχο με το flash, πρόγραμμα, του οποίου κάποια ευπάθεια π.χ. [αυτή](#) εκμεταλλεύεται το malware προκειμένου να καταφέρει να εκτελέσει κάποιο payload στον υπολογιστή του θύματος.