Πτυχιακή στον Τομέα της Ασφάλειας

Τίτλος : Στεγανογραφία

Γεωργακόπουλος Νικόλαος
Αριθμός Μητρώου: 184

Πίνακας Περιεχομένων

1.Εισαγωγή :

Σε έναν ιδανικό κόσμο όλοι θα ήμασταν σε θέση να στέλναμε ανοιχτά το κρυπτογραφημένο κείμενό μας μέσο e-mail ή τα αρχεία ο ένας στον άλλο χωρίς το φόβο των αντιποίνων. Παρόλα αυτά υπάρχουν συχνά περιπτώσεις κατά τις οποίες αυτό δεν είναι δυνατό, είτε επειδή εργαζόμαστε σε μια επιχείρηση που δεν επιτρέπει την ανταλλαγή κρυπτογραφημένων μηνυμάτων μέσο e-mail είτε γιατί η τοπική κυβέρνηση δεν εγκρίνει την χρήση της κρυπτογραφημένης επικοινωνίας (μια πραγματικότητα σε μερικά μέρη του κόσμου). Σε αυτό το σημείο η Στεγανογραφία μπαίνει στη ζωή μας.

Η Στεγανογραφία τροποποιεί ένα κομμάτι πληροφορίας και το κρύβει μέσα σε άλλη. Τα αρχεία υπολογιστών (εικόνες, ήχοι, ακόμη και οι αποθηκευτικοί δίσκοι) περιέχουν αχρησιμοποίητους ή ασήμαντους τομείς δεδομένων. Η τέχνη της Στεγανογραφίας εκμεταλλεύεται αυτές τις περιοχές, αντικαθιστώντας τις με χρήσιμες πληροφορίες. Τα αρχεία αυτά μπορούν να ανταλλαχθούν χωρίς κανένας να ξέρει τι βρίσκεται πραγματικά μέσα τους. Μια φωτογραφία μας μπορεί εύκολα να περιέχει μια προσωπική επιστολή σε έναν φίλο. Η καταγραφή μιας σύντομης πρότασης μπορεί να περιέχει τα σχέδια της επιχείρησής μας για ένα νέο μυστικό προϊόν. Η Στεγανογραφία μπορεί επίσης να χρησιμοποιηθεί για να τοποθετήσει ένα κρυμμένο "εμπορικό σήμα" στις εικόνες, τη μουσική, και το λογισμικό, μια τεχνική που θα δούμε παρακάτω με την ονομασία watermarking .

2.Στεγανογραφία :

Η Στεγανογραφία δεν εντάσσεται στην Κρυπτογραφία αλλά θεωρείται σαν ένα άλλο μέσο προστασίας στον τομέα της ασφάλειας. Αντίθετα με την Κρυπτογραφία όπου σκοπός μας είναι να αλλάξουμε την πληροφορία που θέλουμε να ασφαλίσουμε από οποιουδήποτε είδους επίθεση με αποτέλεσμα η επιτιθέμενη οντότητα να μην μπορεί να βγάλει νόημα από τα στοιχεία που κατέχει για επίθεση ενώ στην Στεγανογραφία σκοπός μας είναι να κρύψουμε την ίδια την ύπαρξη της πληροφορίας!
Θα μπορούσαμε να πούμε ότι η Στεγανογραφία είναι συγγενής της Κρυπτογραφίας.

2.1 ΑΝΑΛΥΣΗ ΛΕΞΗΣ :

προέρχεται από τις ελληνικές λέξεις

- *Στεγανό* = καλυμμένο ή αλλιώς σκεπασμένο
- *Γραφή* = γραφή, κείμενο ή ζωγραφιά

που σημαίνει η γραφή "που καλύπτεται" ή αλλιώς " μυστική" γραφή.

2.2 ΟΡΙΣΜΟΣ :

Στεγανογραφία

Είναι τέχνη ή η επιστήμη που σκοπό έχει να κρύψει την πληροφορία έτσι ώστε τα μηνύματα να μπορούν να περάσουν χωρίς να υπάρχει η παραμικρή υποψία.

Στη σημερινή "ψηφιακή εποχή" συχνότερα λαμβάνεται ως μέσο για το κρύψιμο των πληροφοριών στις ψηφιακές εικόνες ή στα αρχεία ήχου.

Τι κάνει πραγματικά;

- Επιτρέπει τη μετάδοση καλυμμένης πληροφορίας με ασφάλεια μεταξύ δύο οντοτήτων.
- Δεν κρυπτογραφεί το μήνυμα, αλλά άντ' αυτού στηρίζεται στο ότι ο εχθρός δεν θα ανακαλύψει ποτέ τη μετάδοση της πληροφορίας σε σχέση με την κρυπτογράφηση που κρύβει το περιεχόμενο του μηνύματος αλλά δεν κρύβει την επικοινωνία μεταξύ δύο οντοτήτων.

Στεγανογραφία	***Ίσον*** **με**	**Κρυπτογραφία**
Μυστική επικοινωνία		**Μυστική πληροφορία**

3 Ιστορική αναδρομή της Στεγανογραφίας : (διάφορες μορφές που εμφανίστηκε)

Καθ' όλη τη διάρκεια της ιστορίας, ένα πλήθος μεθόδων και παραλλαγών έχουν χρησιμοποιηθεί για να κρύψουν πληροφορίες. Τα πρώτα σημάδια εμφάνισης της Στεγανογραφίας είναι από την εποχή του Ηρόδοτου στην αρχαία Ελλάδα, ο οποίος περιγράφει πως οι έλληνες έκρυβαν ένα μήνυμα σε ειδικές ξύλινες πλάκες που χρησιμοποιούσαν για γραφή και μετά το κάλυπταν με κερί. Σε μια ιστορία του ο Δημήρατος θέλησε να ειδοποιήσει τη Σπάρτη ότι ο Ξέρξης σκόπευε να εισβάλει στην Ελλάδα. Για να αποφύγει τη σύλληψη, έξυσε το μήνυμα στην ξύλινη πλάκα και έπειτα κάλυψε την πλάκα με κερί. Οι πλάκες εμφανίστηκαν να είναι κενές και αχρησιμοποίητες ,έτσι πέρασαν τον έλεγχο από τους φρουρούς χωρίς καμία υποψία.

Μια άλλη έξυπνη μέθοδος ήταν να ξυριστεί το κεφάλι ενός αγγελιοφόρου και να του κάνουν τατουάζ ένα μήνυμα ,μια εικόνα ή και ένα χάρτη (π.χ. πειρατές) στο κεφάλι του. Μετά από λίγο χρόνο αφού μακρύνουν τα μαλλιά του , το μήνυμα θα ήταν μη ορατό έως ότου ξυρίσει πάλι το κεφάλι.

Μια άλλη κοινή μορφή αόρατου μηνύματος είναι μέσο της χρήσης των "αόρατων μελανιών". Τέτοια μελάνια χρησιμοποιήθηκαν με πολλή επιτυχία τον δεύτερο παγκόσμιο πόλεμο. Μια αθώα επιστολή μπορεί να περιέχει ένα πολύ διαφορετικό μήνυμα που γράφεται μεταξύ των γραμμών. Νωρίς στον δεύτερο παγκόσμιο πόλεμο η τεχνολογία της Στεγανογραφίας αποτελούταν σχεδόν αποκλειστικά από τα αόρατα

μελάνια. Τα κοινά υλικά για τα αόρατα μελάνια είναι γάλα, ξίδι, χυμοί φρούτων και ούρα. Όλοι αυτά σκουραίνουν όταν θερμαίνονται.
Με τη βελτίωση της τεχνολογίας και την ευκολία ως προς την αποκωδικοποίηση των αόρατων μελανιών, περιπλοκότερα μελάνια αναπτύχθηκαν στο να αντιδρούν μόνο σε συγκεκριμένες χημικές ουσίες. Μερικά μηνύματα έπρεπε επεξεργαστούν κατάλληλα όπως οι φωτογραφίες αναπτύσσονται με διάφορες χημικές ουσίες στα εργαστήρια .

Μηνύματα ως Null Ciphers (όχι κρυπτογράφημα) χρησιμοποιήθηκαν επίσης. Το πραγματικό μήνυμα είναι καμουφλαρισμένο σε ένα αθώο μήνυμα. Λόγω της "φήμης" πολλών ανοικτών κωδικοποιημένων μηνυμάτων, οι ύποπτες επικοινωνίες ανιχνεύθηκαν από διάφορα αντικατασκοπικά φίλτρα . Εν ο λίγης αρκετά "αθώα" μηνύματα μπόρεσαν να διαπεράσουν. Ένα παράδειγμα ενός μηνύματος null cipher περιέχει κρυμμένο ένα μήνυμα τέτοιας μορφής:

Π.χ.: Fi**s**hing fr**e**shwater be**n**ds an**d** sa**l**twater
co**a**sts re**w**ards an**y**one fe**e**ling st**r**essed.
Re**s**ourceful an**g**lers us**u**ally fi**n**d mas**t**erful
le**a**pers fu**n** an**d** ad**m**it sw**o**rdfish ra**n**k
ov**e**rwhelming an**y**day.

Διαβάζοντας το τρίτο γράμμα από κάθε λέξη το μήνυμα που προκύπτει είναι :

Send Lawyers, Guns, and Money.

Το ακόλουθο μήνυμα είναι ένα αληθινό γεγονός που στάλθηκε στον δεύτερο παγκόσμιο πόλεμο :

A**p**parently n**e**utral's p**r**otest i**s** t**h**oroughly d**i**scounted
a**n**d i**g**nored. I**s**man h**a**rd h**i**t. B**l**ockade i**s**sue a**f**fects
p**r**etext f**o**r e**m**bargo o**n** b**y**-products, e**j**ecting s**u**ets a**n**d v**e**getable
o**i**ls.

Διαβάζοντας το δεύτερο γράμμα από κάθε λέξη το μήνυμα που προκύπτει είναι :

Pershing sails from NY June 1.

Η ανίχνευση μηνυμάτων βελτιωνόταν, παράλληλα νέες τεχνολογίες αναπτύχθηκαν που θα μπορούσαν να περάσουν περισσότερες πληροφορίες και να είναι λιγότερο ευδιάκριτες. Οι Γερμανοί ανέπτυξαν τα Microdots (μικροσκοπική τεχνολογία) στην οποία ο κάποτε διευθυντής του FBI J.Edgar Hoover χαρακτήρισε ως " the enemy's masterpiece of espionage." Τα Microdots είναι μικροσκοπικές φωτογραφίες που περιέχουν δεδομένα στο μέγεθος μιας τυπωμένης κουκίδας σε μία δακτυλογραφημένη σελίδα. Τα πρώτα Microdots ανακαλύφθηκαν μεταμφιεσμένα σε έναν δακτυλογραφημένο φάκελο που μεταφέρθηκε από έναν γερμανό πράκτορα το 1941. Το μήνυμα δεν ήταν κρυμμένο, ούτε κρυπτογραφημένο. Ήταν ακριβώς τόσο μικρό ώστε να μην τραβήξει ποτέ τη παραμικρή προσοχή . Παρά το μέγεθος τους, τα Microdots επέτρεψαν τη διαβίβαση μεγάλων σε όγκο πληροφοριών συμπεριλαμβανομένων και διαφορών σχεδίων και φωτογραφιών.

Μετά από πολλές μεθόδους που ανακαλύφθηκαν και που καταδιώχτηκαν , διάφορες κυβερνήσεις έλαβαν ακραία μέτρα που μπορεί να μας φαίνονται πολύ αστεία σήμερα όπως στις ΗΠΑ η απαγόρευση των γρίφων, σταυρολέξων, οδηγίες πλεξίματος, αποκόμματα εφημερίδων, ακόμα και παιδικές ζωγραφιές δεδομένου ότι μπορούν όλα να περιέχουν μυστικά μηνύματα. Οι αρμόδιες αρχές ενέργησαν ακόμη και στην αντικατάσταση των γραμματοσήμων στους φακέλους. Απαγορεύτηκαν επίσης όλες οι διεθνείς παραγγελίες παραδόσεων συγκεκριμένων τύπων λουλουδιών που περιείχαν συγκεκριμένες ημερομηνίες παράδοσης από τις κυβερνήσεις των ΗΠΑ και Βρετανίας. Στην ΕΣΣΔ όλες οι διεθνείς ταχυδρομικές επιστολές απαγορεύτηκαν στην προσπάθεια να εμποδίσουν οποιαδήποτε εχθρική δραστηριότητα.

Με κάθε ανακάλυψη ενός μηνύματος που κρύβεται χρησιμοποιώντας μια υπάρχουσα τεχνολογική εφαρμογή, μια νέα Στεγανογραφική τεχνική επινοείται. Υπάρχουν ακόμη και τάσεις για επαναχρησιμοποίηση παλαιών μεθόδων. Τα σχέδια και οι ζωγραφιές έχουν χρησιμοποιηθεί συχνά για να κρύψουν ή να αποκαλύψουν πληροφορίες. Είναι απλό να κωδικοποιηθεί ένα μήνυμα με την ποικιλία γραμμών, χρωμάτων ή άλλων στοιχείων μίας εικόνας. Οι υπολογιστές φέρνουν μια τέτοια μέθοδο σε νέες διαστάσεις όπως θα δούμε αργότερα.

Ακόμη και το σχεδιάγραμμα ενός εγγράφου μπορεί να παρέχει πληροφορίες για το ίδιο έγγραφο. Ο Brassil και άλλοι σε μια σειρά δημοσιεύσεων εξετάζουν τον προσδιορισμό εγγράφων και το χαρακτηρίζουν σύμφωνα με τη διαμόρφωση της θέσης των γραμμών και των λέξεων. Παρόμοιες τεχνικές μπορούν επίσης να χρησιμοποιηθούν για να παρέχουν κάποιες άλλες "συγκαλυμμένες" πληροφορίες ακριβώς σαν το 0,1 που είναι πηγές πληροφορίας σε έναν υπολογιστή. Όπως σε ένα από τα παραδείγματά τους, η λέξη-μετατόπιση μπορεί να χρησιμοποιηθεί για να βοηθήσει να προσδιοριστεί ένα έγγραφο. Μια παρόμοια μέθοδος μπορεί να εφαρμοστεί για να εμφανίσει ένα εξ ολοκλήρου διαφορετικό μήνυμα. Η ακόλουθη πρόταση:

We explore new steganographic and cryptographic
algorithms and techniques throughout the world to
produce wide variety and security in the electronic web called the Internet.

Αν πριν από κάθε λέξη που θέλουμε να περάσουμε αυξήσουμε το κενό διπλάσια από ότι το κανονικό τότε το αποτέλεσμα του αλγορίθμου είναι :

We explore new steganographic and cryptographic
algorithms and techniques throughout the world to
produce wide variety and security in the electronic web
called the Internet.

Οι προτάσεις που περιέχουν τις κρυμμένες λέξεις εμφανίζονται αβλαβείς καθώς και όλο το μήνυμα φαίνεται αθώο, αλλά ο συνδυασμός αυτός με τον συγκεκριμένο αλγόριθμο παράγει ένα διαφορετικό μήνυμα:

explore the world wide web.

Terror groups hide behind Web encryption and steganography !

By Jack Kelley, USA TODAY

WASHINGTON : ειδικοί εκτιμούν ότι κρυπτογραφημένα σχεδιαγράμματα της επόμενης τρομοκρατικής επίθεσης ενάντια στις Ηνωμένες Πολιτείες και τους συμμάχους της κρύβονται σε ανήλικες φωτογραφίες ανήλικων παιδιών στους διάφορους πορνογραφικούς δικτυακούς χώρους και σε διάφορα σχόλια στα αθλητικά post's, ή ακόμα και σε chat rooms. Ακούγεταί εξεζητημένο, αλλά οι αμερικανοί ανώτεροι υπάλληλοι και οι εμπειρογνώμονες λένε ότι είναι η πιο πρόσφατη μέθοδος επικοινωνίας από τον Οσάμα Μπιν Λάντεν και τους συνεργάτες του για να εξαπατήσει τα συστήματα ασφαλείας. Ο Μπιν Λάντεν, κατηγορείτε ότι με αυτόν τον τρόπο έδωσε οδηγίες για τον βομβαρδισμό το 1998 δύο ΑΜΕΡΙΚΑΝΙΚΩΝ πρεσβειών στην Ανατολική Αφρική, και για άλλες ενέργειες που κρύβουν χάρτες και φωτογραφίες τρομοκρατικών στόχων και σχέδια για τις τρομοκρατικές δραστηριότητες μέσο του διαδικτύου.

U.S. officials say Osama bin Laden is posting instructions for terrorist activities on sports chat rooms, pornographic bulletin boards and other Web sites.

4. Η Στεγανογραφία στην σημερινή "ψηφιακή" εποχή :

Η Στεγανογραφία στους υπολογιστές είναι βασισμένο σε δύο αρχές. Η πρώτη είναι ότι τα αρχεία που περιέχουν εικόνες ή ήχο μπορούν να αλλάξουν κατά μία ορισμένη επέκταση χωρίς καμία αλλοίωση της λειτουργικότητάς τους αντίθετα από άλλους τύπους δεδομένων (π.χ. τα προγράμματα) που πρέπει να είναι ακριβή προκειμένου να λειτουργήσουν . Η άλλη αρχή στηρίζεται στην ανικανότητα του ανθρώπου να διακρίνει τις ελάχιστες αλλαγές στο χρώμα μίας εικόνας ή στην ποιότητα του ήχου, το οποίο είναι ιδιαίτερα εύκολο να χρησιμοποιηθεί και να εφαρμοστεί στα δεδομένα που περιέχουν περιττές πληροφορίες, είτε πρόκειται για ήχο 16-bit , 8-bit ή ακόμα καλύτερα μιας εικόνας 24-bit. Η τροποποίηση που γίνεται στις ψηφιακές εικόνες, αλλάζοντας την τιμή του λιγότερου σημαντικού bit (LSB) του χρώματος του εικονοκυττάρου (Pixel) δεν γίνεται αντιληπτή από το ανθρώπινο μάτι.

4.1 Στεγανάλυση :

Ένα από τα μειονεκτήματα της Στεγανογραφίας και άξιο προσοχής είναι το μέγεθος του αρχικού αρχείου και αυτού τού οποίου προκύπτει μετά από κάποια Στεγανογραφική μέθοδο. Κάποιος μπορεί να δει τη διαφορά μεταξύ του αρχικού και του τροποποιημένου αρχείου μόνο συγκρίνοντας τα μεγέθη τους , έτσι μία ύποπτη εχθρική οντότητα μόνο εάν κατέχει το τροποποιημένο αρχείο (και όχι και το αρχικό), μόνο τότε θα φαίνεται το αρχείο αθώο. Η τεχνική κατά την οποία με διάφορες διαδικασίες προσπαθούμε να ανιχνεύσουμε κάποιο κρυμμένο μήνυμα ή δεδομένο μέσα σε μία "αθώα" πληροφορία λέγεται Στεγανάλυση και είναι ουσιαστικά η απάντηση στην Στεγανογραφία όπως η αποκρυπτογράφηση στην κρυπτογράφηση. Για την καλύτερη ασφάλεια συνιστάται κάποιος να χρησιμοποιεί εικόνες με πολλές

διαβαθμίσεις χρωμάτων και κατά προτίμηση να μην είναι συνηθισμένο αρχείο ώστε να μην είναι γνωστό στο κοινό επειδή οι ελάχιστες αλλαγές θεωρούνται έως και αδύνατων να παρατηρηθούν. Η χρησιμοποίηση π.χ. μίας εικόνας ενός διάσημου ζωγράφου που είναι ευρέως γνωστή δεν είναι και μια πολύ καλή ιδέα, επειδή ο καθένας (τουλάχιστον οι ειδικοί στο χώρο) θα ξέρουν πως είναι η εικόνα αρχικά, εκτός από τα σημεία ίδιου χρώματος. Καλύτερα αποτελέσματα θα έχουμε αν εφαρμόσουμε σε μία άσχετη φωτογραφία του σκύλου μας ☺. Γενικά αφού κρύψουμε ένα μήνυμα σε ένα αρχείο προτιμάτε να καταστρέψουμε το πρωτότυπο για καλύτερη ασφάλεια.

Όπως είδαμε η Στεγανογραφία χρησιμοποιείτε για την ασφάλεια της πληροφορίας ώστε να μην μπορεί να εντοπιστεί η ύπαρξή της. Όμως η επιστήμη της Στεγανογραφίας δημιούργησε και άλλους υποκλάδους όπως η τεχνική του Watermarking (υδατογραφήματος) και Fingerprint η οποία είναι μία μέθοδος που κρύβει πληροφορίες ταυτότητας ή άδειες αντικείμένων όπως serial numbers.

4.2 Watermarking:

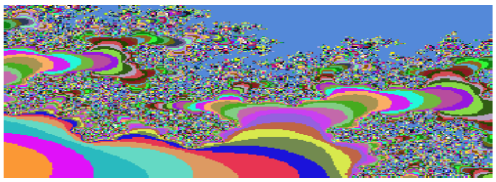
Αν μιλήσουμε για εμπορικές Στεγανογραφικές εφαρμογές που υπάρχουν στο δίκτυο θα πρέπει σίγουρα να αναφέρουμε το ψηφιακό Watermarking που έχει πάρει μια νέα σημασία στην ψηφιακή εποχή. Ακόμα και οι εικόνες, το βίντεο, η μουσική, το κείμενο, και το λογισμικό, όλα αντιγράφονται εύκολα και διανέμονται παράνομα, αναγκάζοντας τους συντάκτες να χάνουν μεγάλο μερίδιο πωλήσεων και σε πολλές περιπτώσεις τα πνευματικά τους δικαιώματα. Το Watermarking είναι μια ειδική τεχνική που εγκαθιστά αόρατα ψηφιακά σημάδια στις εικόνες και στα αρχεία ήχου τα οποία φανερώνουν πληροφορίες πνευματικών δικαιωμάτων. Αυτά τα σημάδια ανιχνεύονται από ειδικά προγράμματα που μπορούν να αντλήσουν πολλές χρήσιμες πληροφορίες από αυτό το ειδικό σήμα (watermark). Όταν το αρχείο δημιουργείται, ταυτόχρονα κρατά τα πνευματικά δικαιώματα , το πώς να έρθει σε επαφή με το συντάκτη κλπ.... Όπως ξέρουμε χιλιάδες γνήσια προϊόντα αναπαράγονται παράνομα και κλέβονται από το δίκτυο κάθε μέρα ώστε να καθίσταται αυτή τη τεχνολογία απαραίτητη και χρήσιμη εάν θέλουμε να προστατέψουμε τα πνευματικά μας δικαιώματα από την πειρατεία.

Υπάρχουν πολλές εταιρίες στο δίκτυο που πωλούν Watermarking προϊόντα. Μία από τις ηγετικές είναι η Digimarc (http://www.digimarc.com) τις οποίας οι πωλήσεις προγραμμάτων ξεπερνούν το ένα εκατομμύριο. Προσφέρει ελεύθερα το πρόγραμμα PictureMarc το οποίο είναι ένα plug-in στο Photoshop και στο CorelDraw, ή το αυτόνομο ReadMarc. Μόλις εγκατασταθεί, μπορούμε αφού ανοίξουμε ένα αρχείο να διαβάσουμε το κρυμμένο watermark που είναι ενσωματωμένο (αν υπάρχει). Για πιο απαιτητικές περιπτώσεις η Digimarc προσφέρει το individual Creator ID (με άδεια ενός έτους) που μας επιτρέπει να ενσωματώνουμε watermark στις εικόνες προτού τις βγάλουμε στον Internet. Υπάρχουν πολλοί πελάτες συμπεριλαμβανομένων σχεδιαστών, φωτογράφων και on-line galleries που το χρησιμοποιούν όπως το γνωστό Playboy (http://www.playboy.com) . Έπειτα εταιρικοί χρήστες μπορούν να χρησιμοποιήσουν το MarcSpider το οποίο ψάχνει όλο το δίκτυο για παράνομες εικόνες και αναφέρει οποιαδήποτε παράνομη αναπαραγωγή τους.
Είναι δυνατόν συντάκτες, σχεδιαστές, δημιουργοί να μην πάσχουν πλέον από κλοπές κλπ ; Η ιστορία μας έχει μάθει ότι σε κάθε πρόβλημα υπάρχει μία λύση αλλά και

αντιθέτως, σε κάθε λύση υπάρχει ένα πρόβλημα! Όπως πολλά άλλα προγράμματα που σπάζουν τους καθιερωμένους μηχανισμούς ασφάλειας, υπάρχουν προγράμματα που προορίζονται να καταδείξουν την αδυναμία των τρέχοντων αλγόριθμων έτσι ώστε οι επιχειρήσεις να παρακινηθούν και να αναπτύξουν ακόμα πιο γερές Watermarking τεχνολογίες.

Παρά τις προσπάθειες των κατασκευαστών το watermarking δεν αποδείχθηκε αρκετά γερό. Το Watremark μπορεί να επιζήσει πολλών πραγμάτων όπως ρυθμίσεις φωτεινότητας και αντίθεσης, που εφαρμόζουν τα ειδικά φίλτρα ακόμα και εκτύπωση η σάρωση, αλλά δεν μπορεί να επιζήσει από ειδικά προγράμματα όπως το StirMark (http://www.cl.cam.ac.uk/users/fapp2/steganography/image_watermarking/stirmark) και UnZign (http://www.altern.org/watermark) τα οποία είναι δύο παραδείγματα λογισμικού που εμφανίστηκαν στο δίκτυο αμέσως μετά την ανακάλυψη κάθε τεχνολογίας watermarking και μπορούν να αφαιρέσουν πληροφορίες πνευματικών δικαιωμάτων από αρχεία. Προφανώς αυτά τα εργαλεία δεν στοχεύουν ενάντια σε κάποιο αλγόριθμο Στεγανογραφίας , αλλά μάλλον σε συγκριτικές μετρήσεις επιδόσεων που μας βοηθούν να ξεχωρίσουμε ώστε να επιλέξουμε το πιο αξιόπιστο Watermarking λογισμικό. Τα συμπεράσματα στα οποία οδηγούμαστε είναι: μερικά Watermarks αφαιρούνται εύκολα ή καταστρέφονται με το χειρισμό των διάφορων χαρακτηριστικών του αρχείου, δυστυχώς σήμερα όλα τα watermarks μπορούν να καταστραφούν χωρίς σημαντική απώλεια στην ποιότητα της εικόνας.

4.3 Εικόνα :



(με την μέθοδο λιγότερου σημαντικού ψηφίου)

Σήμερα, κατά τη μετατροπή μιας εικόνας από αναλογική μορφή σε ψηφιακή ,έχουμε την επιλογή συνήθως μεταξύ τριών διαφορετικών ειδών χρωμάτων :

- 24-bit χρωματισμό: κάθε pixel μπορεί να έχει 2^{24} χρώματα, και αυτά αντιπροσωπεύουν διαφορετικές ποσότητες των τριών βασικών χρωμάτων: κόκκινο (R), πράσινο (G), μπλε (B), που δίνονται από 8-bit (256 τιμές) το κάθε ένα.
- 8-bit χρώμα: κάθε pixel μπορεί να έχει 256 (2^8) χρώματα, που επιλέγονται από μια παλέτα ή αλλιώς από ένα πίνακα χρωμάτων.
- 8-bit κλίμακα του γκρίζου : κάθε pixel μπορεί να έχει 256 (2^8) σκιές της κλίμακας του γκρίζου.

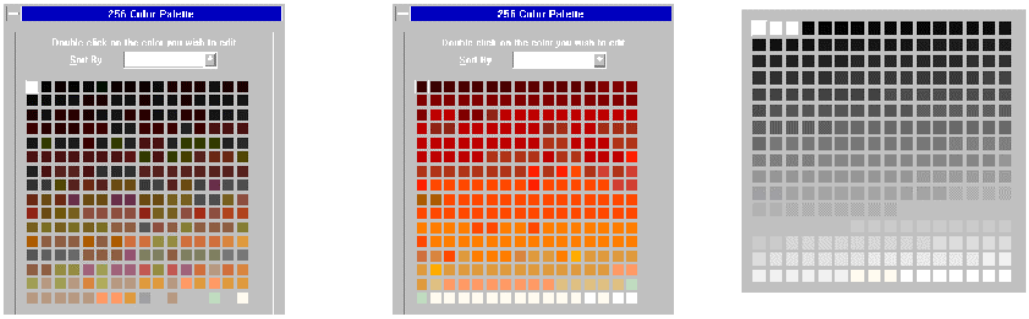


Figure 1: Gray-Scale Palette

Η μέθοδος εισαγωγής LSB τροποποιεί το LSB κάθε χρώματος στις εικόνες 24-bit ή 8-bit.

Παράδειγμα:
Το γράμμα "A" σε κώδικα ASCII είναι το 65 (δεκαδικό), το οποίο είναι το 1000001 στο δυαδικό.
Χρειάζονται τρία διαδοχικά pixel σε μια εικόνα 24-bit για να αποθηκεύσει ένα "A":

Ας θεωρήσουμε ότι τα pixel πριν από την εισαγωγή είναι:

Πρώτο pixel →10000000.10100100.10110101,
Δεύτερο pixel →10110101.11110011.10110111,
Τρίτο pixel →11100111.10110011.00110011

Οι τιμές κάθε pixel μετά από την εισαγωγή ενός "A" θα είναι:

10000001.10100100.10110100,
10110100.11110010.10110110,
11100110.10110010.00110011

(Οι τιμές σε **bold** είναι αυτές που τροποποιήθηκαν από το μετασχηματισμό)

Στο ίδιο παράδειγμα για μια 8-bit εικόνα θα χρειαστούν 8 pixel:

10000000, 10100100, 10110101, 10110101, 11110011,
10110111, 11100111, 10110011

Οι τιμές τους μετά από την εισαγωγή ενός "A" θα ήταν:

10000001, 10100100, 10110100, 10110100, 11110010,
10110110, 11100110, 10110011

Παρατήρηση :Από τα παραπάνω παραδείγματα συμπεράνουμε ότι η εισαγωγή ενός LSB έχει συνήθως πιθανότητα 50% να αλλάξει για κάθε 8-bit, προσθέτοντας πολύ λίγο θόρυβο στην αρχική εικόνα.
Για εικόνες 24-bit η τροποποίηση μπορεί να επεκταθεί μερικές φορές και στο δεύτερο ή ακόμα και τρίτο LSB χωρίς να είναι ορατή η αλλαγή. Οι 8-bit εικόνες έχουν άντ' αυτού ένα περιορισμένο διάστημα πού μπορούν να επιλεγούν τα χρώματα, έτσι είναι συνήθως δυνατό να αλλαχτεί μόνο το LSB σε αυτά χωρίς η τροποποίηση να είναι ανιχνεύσιμη.

Θεωρούμε το παρακάτω κείμενο ένα μήνυμα που θέλουμε να κρύψουμε :

```
          Steganography is the art and science of communicating in a
way which hides the existence of the communication. In contrast to
cryptography, where the "enemy" is allowed to detect, intercept and
modify messages without being able to violate certain security
premises guaranteed by a cryptosystem, the goal of steganography is
to hide messages inside other "harmless" messages in a way that does
not allow any "enemy" to even detect that there is a second secret
message present [Markus Kuhn 1995-07-03].
```

Αριστερά η αρχική εικόνα και δεξιά μετά την ενσωμάτωση του προηγούμενου κειμένου.

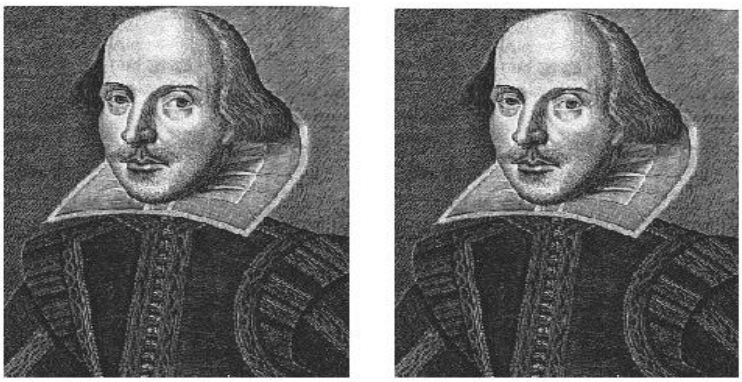


Figure 9: Result of embedding M1 in C2 using *White Noise Storm*.

Εδώ έχουμε την περίπτωση μίας εικόνας φωτογραφία δορυφόρου (δεξιά) κρυμμένη μέσα σε μία εικόνα γνωστού έργου τέχνης (αριστερά) **Renoir's *Le Moulin de la Galette :***

+

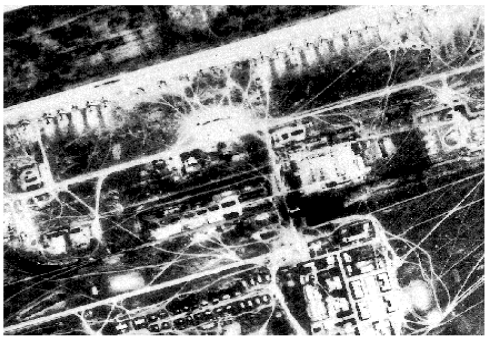


Figure 4: Long-Range Aviation Airfield[5]

Το αποτέλεσμα θα είναι :

4.4 Ήχος :

Στοιχεία που κρύβονται στον ήχο.

Στοιχεία που κρύβονται σε ακουστικά σήματα είναι ιδιαίτερα προκλητικά,
επειδή το ανθρώπινο ακουστικό σύστημα δεν λειτουργεί πέρα από ένα δυναμικό εύρος συχνοτήτων. Μπορεί να αντιλαμβάνεται ένα εύρος της δύναμης πάνω από ένα δισεκατομμύριο προς ένα και ένα εύρος συχνοτήτων μεγαλύτερο από χίλια προς ένα.
Η ευαισθησία σε κάθε πρόσθετο τυχαίο θόρυβο είναι επίσης έντονη. Οι διαταραχές σε ένα αρχείο ήχου μπορεί να εντοπίζονται τόσο χαμηλά όπως ένα bit προς δέκα εκατομμύρια (80 DB κάτω από το επίπεδο περιβάλλοντος).
Εντούτοις, υπάρχουν μερικές "ευαισθησίες" της αντίληψης διαθέσιμες. Το ανθρώπινο ακουστικό σύστημα ενώ έχει μεγάλο δυναμικό εύρος συχνοτήτων , έχει επίσης έναν αρκετά μικρό διαφορικό εύρος. Κατά συνέπεια, οι δυνατοί ήχοι τείνουν να καλύψουν πιο ασθενείς ήχους . Επιπλέον το ανθρώπινο ακουστικό σύστημα είναι ανίκανο να αντιληφθεί την απόλυτη φάση αλλά μόνο τη σχετική φάση. Τέλος, υπάρχουν μερικές περιβαλλοντικές διαστρεβλώσεις τέτοιες ώστε να αγνοούνται από τον ακροατή στις περισσότερες περιπτώσεις.
Εκμεταλλευόμαστε πολλά από αυτά τα γνωρίσματα στη παρακάτω μέθοδο καθώς έχουμε λάβει υπόψη προσεκτικά τις ευαισθησίες του ανθρώπινου ακουστικού συστήματος.
Μία ακουστική ακολουθία σε αναλογική μορφή για να την επεξεργαστούμε σε ψηφιακή μορφή θα πρέπει πρώτα να ορίσουμε την συχνότητα δειγματοληψίας από την οποία όσο πιο πολλά δείγματα έχουμε τόσο μεγαλύτερο το μέγεθος του αρχείου αλλά και καλύτερη η ποιότητα του ήχου. Τα βήματα που ακολουθούμε για να κρύψουμε δεδομένα σε ένα ηχητικό αρχείο δεν διαφέρουν και πολύ από αυτά που κάναμε για την εικόνα. Δηλαδή πάλι σκοπός μας είναι να αλλάξουμε το τελευταίο σημαντικό ψηφίο (LSB).

Παράδειγμα :

Έχουμε ένα αρχείο ήχου σε wav μορφή με τα ακόλουθα χαρακτηριστικά :
44100 Hz 16-bit stereo του ενός λεπτού.

Διαστάσεις αρχείου = (16-bit x 44100 Hz x 60sec) x 2 (το stereo είναι δικάναλο)
= 84672000 bit

Έχουμε συνεπώς μέγεθος για να κρύψουμε στο αρχείο (χρησιμοποιώντας τα 2 τελευταία LSB) = 84672000 bit / 16 x 2 = 10584000 bit

προσοχή : Ποτέ δεν κρύβουμε μία πληροφορία σε ένα wav ή bmp αρχείο και μετά το συμπιέζουμε η ακόμα και αν αλλάξουμε τη μορφή του διότι υπάρχει μεγάλος κίνδυνος να χαθούν αυτά που είχαμε κρύψει.

- Τα ηχητικά δείγματα είναι από την φύση τους ανακριβείς εκτιμήσεις της σωστής αξίας τιμών σε μια δεδομένη χρονική στιγμή. Τα ηχητικά δείγματα σε μορφή WAV αποθηκεύονται είτε σαν 8-bit είτε σαν 16-bit που περνούν τελικά από το μετατροπέα της κάρτας ήχου. Για τα δείγματα 8-bit σημαίνει ότι οι τιμές μπορούν να κυμανθούν μεταξύ 0 και 255 δείγματα , για τα 16-bit κυμαίνονται μεταξύ 0 και 65535. Στο συγκεκριμένο παράδειγμα αυτό που θα κάνουμε είναι να διαμοιράσουμε τα bit-pattern που αντιστοιχούν στο αρχείο που θέλουμε να κρύψουμε στα λιγότερα σημαντικά bit του ηχητικού δείγματος.

- παραδείγματος χάριν ας υποθέσουμε ότι ένα ηχητικό δείγμα έχει κάπου τα ακόλουθα οκτώ bytes πληροφορίας:

  132 134 137 141 121 101 74 38

  Σε δυαδικό αυτά θα είναι :

  10000100 10000110 10001001 10001101 01111001 01100101 01001010 00100110

  (το LSB κάθε αριθμού σε κόκκινο)

- Ας υποθέσουμε ότι θέλουμε να κρύψουμε το δυαδικό 11010101 (213) εσωτερικά σε αυτή την ακολουθία. Αντικαθιστούμε απλά το LSB (λιγότερο σημαντικό bit) κάθε byte δείγματος με το αντίστοιχο κομμάτι byte που προσπαθούμε να κρύψουμε. Έτσι η παραπάνω ακολουθία θα αλλάξει ως εξής:

  133 133 137 142 121 100 74 39

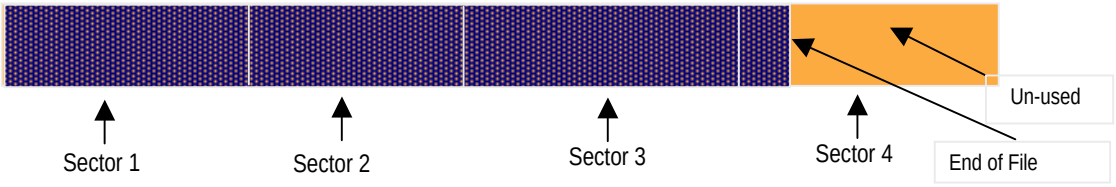
  Στο δυαδικό αυτό είναι :

  10000101 10000101 10001001 10001110 01111001 01100100 01001010 00100111

- Παρατηρούμε ότι η τιμή του ηχητικού δείγματος έχει αλλάξει το πολύ κατά μια μονάδα για το καθένα . Αυτή ή αλλαγή δεν θα είναι αντιληπτή από το ανθρώπινο αυτί και ταυτόχρονα έχουμε κρύψει πάλι 8-bit πληροφορίας μέσα στο δείγμα.

4.5 Κρυμμένα μηνύματα σε αποθηκευτικά μέσα.

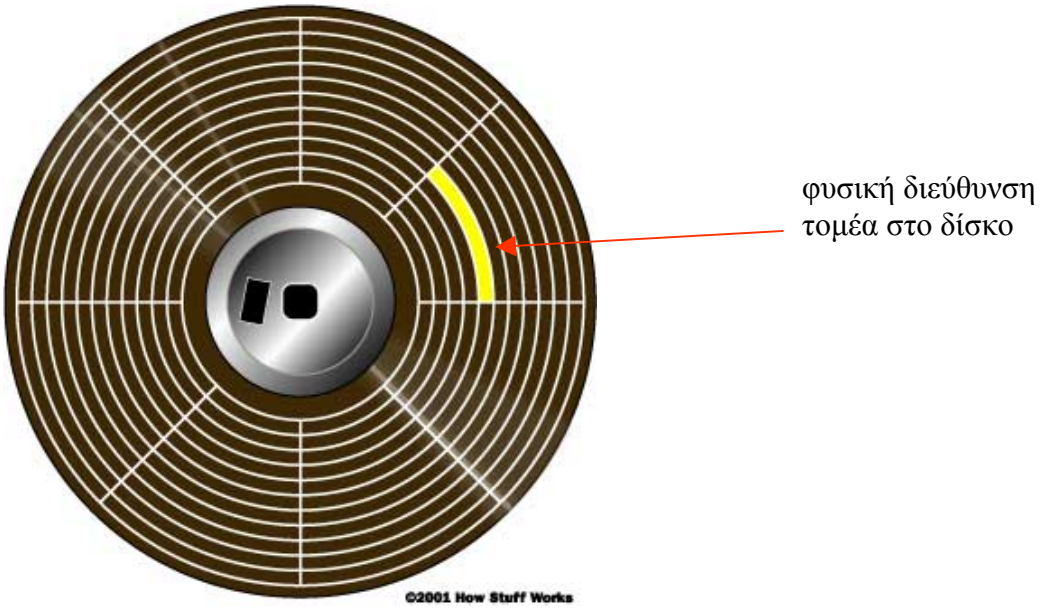
- Σε ένα δίσκο ο χώρος που διατίθεται για ένα αρχείο βρίσκεται στους τομείς ενός προκαθορισμένου μεγέθους (συνήθως 512 bytes). Το πραγματικό μέγεθος του τομέα μπορεί να διαφέρει ανάλογα με το λειτουργικό σύστημα (NTFS,FAT,EXT3) αλλά και από το υλικό που αποτελείτε το αποθηκευτικό μέσο. Πάντα όμως υπάρχουν προκαθορισμένοι φραγμοί για την αποθήκευση δεδομένων.
- Το τέλος του χώρου που καταλαμβάνει ένα αρχείο στη μονάδα αποθήκευσης είναι μαρκαρισμένο με ένα δείκτη όπου δηλώνει το τέλος του (EoF).
- Το σημείο αυτό είναι που δεν χρησιμοποιείτε και μπορούμε να εισάγουμε κρυφά δικά μας δεδομένα.



4.5.1 Κρυμμένα μηνύματα σε συγκεκριμένες θέσεις στο δίσκο:

- είναι δυνατό να γράψουμε άμεσα σε ειδικές περιοχές ενός δίσκου π.χ. σε κάποιο τομέα του δίσκου .
- αυτό σημαίνει ότι ένα μήνυμα οποιουδήποτε μεγέθους μπορεί να γραφτεί στις φυσικές διευθύνσεις του δίσκου.
    - μειονέκτημα

      επειδή το σύστημα κατανομής αρχείων του λειτουργικού δεν γνωρίζει την ύπαρξη των στοιχείων που έχουμε γράψει άμεσα στο δίσκο, θα επικαλύψει τον τομέα όταν χρειαστεί το διάστημα αυτό εκτός αν κάνουμε κάτι για να το αποφύγουμε.



5. Θεωρητικά Συμπεράσματα :

Η Στεγανογραφία είναι μια αρχαία τέχνη που έχει διαδοθεί και αναπτυχθεί με την εμφάνιση του διαδικτύου και γενικά από τα ψηφιακά μέσα. Δεν είναι πλέον μια μέθοδος που περιορίζεται στη μυστική επικοινωνία μεταξύ δύο κατασκόπων ή κάποια άλλη χρήση της σε διάρκεια πολέμου. Τα εργαλεία είναι τώρα προσιτά στους χρήστες και σε αρκετές περιπτώσεις δωρεάν μέσω του διαδικτύου, τα οποία πολλές φορές δεν απαιτούν καμία ειδική γνώση για τη λειτουργία τους.
Αφού ο σκοπός της πτυχιακής είναι η ασφάλεια, πιστεύω ότι η Στεγανογραφία σε συνδυασμό με την Κρυπτογραφία σαν μία υβριδική μορφή θα μπορούσε να πετύχει ένα πολύ καλό υψηλό επίπεδο ασφαλείας και απόρρητου.
Αυτό έχει διάφορα πλεονεκτήματα και μειονεκτήματα. Πλεονέκτημα είναι ότι αυξάνει την δυνατότητα προστασίας της μυστικότητάς μας, ή την επικοινωνία μας όταν το απαιτούν οι συνθήκες. Μειονέκτημα είναι η αυξανόμενη δυνατότητα για τους εγκληματίες και τους τρομοκράτες να επικοινωνούν μεταξύ τους χωρίς να ανιχνεύονται από την δικαιοσύνη. Η απαγόρευση της τεχνολογίας δεν είναι επαρκής για να σταματήσει την εγκληματική χρήση. Η Στεγανάλυση ενώ κατευθύνεται στο να γίνει αποτελεσματική, αντιμετωπίζει πολλά εμπόδια για να γίνει μια αξιόπιστη μέθοδος ανίχνευσης στεγανογραφικής δραστηριότητας.

Η Στεγανογραφία και η Στεγανάλυση είναι ακόμα σε στάδια έρευνας και ανάπτυξης. Δεδομένου ότι οι τεχνικές για το κρύψιμο πληροφορίας βελτιώνονται, το ίδιο ισχύει και για την ανίχνευση. Στην πραγματικότητα, η Στεγανογραφία είναι μέρος του ίδιου κύκλου με την επιβολή του νόμου και της εγκληματικότητας.
Δεδομένου ότι η ικανότητα της επιβολής του νόμου αυξάνεται, έτσι και η ικανότητα της εγκληματικότητας αυξάνεται. Η μόνη επιλογή είναι η συνεχής πρόοδος. Η Συνεχής πρόοδος και η έρευνα είναι ο μόνος τρόπος στο για να μην σταματήσει ο κύκλος υπέρ εκείνων που κάνουν κακή χρήση της τεχνολογίας.

Αξίζει να αναφέρουμε και το δωρεάν πρόγραμμα StegoSoftware λογισμικό με το οποίο μπορούμε να κρύψουμε τα αρχεία οποιουδήποτε τύπου σε αρχεία εικόνας gif και bmp καθώς επίσης και σε wav. Επιπλέον είναι πραγματικά ένα Στεγανογραφικό και Κρυπτογραφικό προϊόν σε ένα, επειδή το αρχείο που κρύβεται το μήνυμα κρυπτογραφείται πρώτα χρησιμοποιώντας έναν από τους συμμετρικούς βασικούς αλγορίθμους: DES , τριπλό DES, και IDEA. Η χρήση του είναι πολύ απλή, απλός "σέρνοντας" το αρχείο στο παράθυρο διαλόγου του προγράμματος, κατόπιν το αρχείο που θέλουμε να κρύψουμε και τέλος επιλέγουμε έναν αλγόριθμο και έναν κωδικό πρόσβασης.

6. Η γλώσσα προγραμματισμού Python :

Λίγα λόγια την Python **..**

Επέλεξα σαν εργαλείο για την υλοποίηση της πτυχιακής μου να χρησιμοποιήσω αν και άγνωστη για την Ελλάδα (ας ελπίζουμε όχι για πολύ) την υψηλού επιπέδου γλώσσα προγραμματισμού Python. Η **Python** είναι μια αλληλοεπιδραστική γλώσσα προγραμματισμού η οποία δημιουργήθηκε από τον Ολλανδό Guido van Rossum το 1990. Αρχικά ήταν scripting γλώσσα για το λειτουργικό σύστημα Amoeba και ικανή για κλήσεις συστήματος. Η γλώσσα χρησιμοποιεί μεταγλωττιστή για την δημιουργία του εκτελέσιμου κώδικα και συσχετίζεται με τις γλώσσες προγραμματισμού Tcl, Perl, Scheme, Java και Ruby. Η Python έχει αναπτυχθεί ως ανοιχτό λογισμικό (open source) και η διαχείρισή της γίνεται από τον μη κερδοσκοπικό οργανισμό Python Software Foundation.

Στο ερώτημα γιατί Python έχω να απαντήσω τα εξής :

- Μία από τις πιο ανεπτυγμένες εταιρίες υψηλής τεχνολογία που πρωτοπορεί σε καινοτομία σήμερα , η πασίγνωστη σε όλους μας Google χρησιμοποιεί την Python από την αρχή της ανάπτυξή της :

  Python has been an important part of Google since the beginning, and remains so as the system grows and evolves. "

  -- Peter Norvig, Google

- Είναι μία γλώσσα που λόγω της απλότητάς της και δυναμικότητας του scripting language ενθαρρύνεται για rapid development .

- Η σύνταξή της είναι τόσο απλή σαν ψευδοκώδικα, όταν ο κώδικας που βλέπουμε δείχνει λογικά σωστός , συνήθως το πρόγραμμα θα τρέχει σωστά.

- Προσφέρει μεγάλη υποστήριξη για ενοποίηση με άλλες γλώσσες προγραμματισμού και εργαλεία.

- Διανέμετε με άδεια OSI-approved που σημαίνει ότι είναι μία γλώσσα 100% open source και ότιδήποτε δημιουργήσουμε με αυτήν μπορούμε να το χρησιμοποιήσουμε ελεύθερα χωρίς κανέναν περιορισμό , ακόμα και ως μία εμπορική εφαρμογή .

- Είναι 100% portable, ο ίδιος κώδικας τρέχει σε Windows, Linux , Mac κτλ. χωρίς μετατροπές.

- Μόλις πρόσφατα η Microsoft έκανε τα πρώτα βήματα για να την ενσωματώσει στην οικογένεια της .NET με την κωδική ονομασία IronPython .

- Την χρησιμοποιεί και η NASA :-P

Για να πάρετε μία ιδέα από τα πλεονεκτήματα της Python , κυκλοφορεί στο διαδίκτυο το δημοφιλές Zen of python από τον Tim Peters:

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!

6.1 Εισαγωγή στα βασικά της Python :

Για αρχή, σκεφτείτε την Python ως ψευδοκώδικα. Είναι σχεδόν αλήθεια. Οι μεταβλητές δεν έχουν τύπους, έτσι δε χρειάζεται να τις δηλώνετε. Εμφανίζονται όταν καταχωρείτε σε αυτές και εξαφανίζονται όταν δεν τις χρησιμοποιείτε άλλο. Η καταχώρηση γίνεται με τον τελεστή =. Η ισότητα ελέγχεται από τον τελεστή ==. Μπορείτε να καταχωρήσετε σε αρκετές μεταβλητές μονομιάς:

```
x, y, z = 1, 2, 3

first, second = second, first

a = b = 123
```

Τα blocks υποδεικνύονται μέσω της εσοχής του κείμενου (indentation) και *μόνο*. (Όχι BEGIN/END ή αγκύλες.) Μερικές συνηθισμένες δομές ελέγχου είναι:

```
if x < 5 or (x > 10 and x < 20):
    print "Η τιμή είναι εντάξει."

if x < 5 or 10 < x < 20:
    print "Η τιμή είναι εντάξει."

for i in [1, 2, 3, 4, 5]:
    print "Αυτός είναι ο αριθμός επανάληψης", i

x = 10
while x >= 0:
    print "ο x εξακολουθεί να είναι μη-αρνητικός."
    x = x - 1
```

Τα πρώτα δύο παραδείγματα είναι ισοδύναμα.

Η μεταβλητή–δείκτης που δίνεται στο βρόχο `for` εκτελεί επαναλήψεις μέσα στα στοιχεία μιας *λίστας* (που γράφεται όπως στο παράδειγμα). Για να φτιάξετε ένα "συνηθισμένο" βρόχο `for` (δηλαδή ένα βρόχο μέτρησης), χρησιμοποιείστε την ενσωματωμένη (built–in) συνάρτηση `range()`.

```
# εκτύπωσε τις τιμές από το 0 μέχρι και το 99
for value in range(100):
    print value
```

(Η γραμμή που ξεκινάει με `#` είναι ένα σχόλιο και αγνοείται από το διερμηνέα.)

Ωραία· τώρα γνωρίζετε αρκετά για να υλοποιήσετε (θεωρητικά) οποιονδήποτε αλγόριθμο στην Python. Ας προσθέσουμε μερική *βασική* αλληλεπίδραση με τον χρήστη. Για να πάρετε είσοδο από τον χρήστη (από ένα νεύμα/prompt κειμένου), χρησιμοποιήστε την ενσωματωμένη συνάρτηση `input`.

```
x = input("Παρακαλώ δώστε έναν αριθμό: ")
print "Το τετράγωνο αυτού του αριθμού είναι", x * x
```

Η συνάρτηση `input` εμφανίζει το νεύμα που δώθηκε (το οποίο μπορεί να είναι άδειο) και αφήνει το χρήστη να εισάγει οποιαδήποτε έγκυρη τιμή Python. Σε αυτή την περίπτωση αναμένουμε έναν αριθμό — αν εισαχθεί κάτι άλλο (όπως ένα αλφαριθμητικό), το πρόγραμμα θα "χτυπήσει" (crash). Για να αποφθεχθεί αυτό θα χρειαστούμε μερικό έλεγχο σφαλμάτων. Δε θα αναφερθώ σε αυτό εδώ· θα αρκεστώ στο να πω ότι αν θέλετε την είσοδο του χρήστη αποθηκευμένη *αυτολεξεί* ως ένα αλφαριθμητικό (έτσι ώστε να μπορεί να εισαχθεί *οτιδήποτε*), χρησιμοποιήστε τη συνάρτηση `raw_input`. Αν θέλετε να μετατρέψετε ένα αλφαριθμητικό εισόδου `s` σε ακέραιο, μπορείτε να χρησιμοποιήσετε `int(s)`.

Σημείωση: Αν θέλετε να εισάγετε ένα αλφαριθμητικό με την `input`, ο χρήστης θα πρέπει να γράψει τα εισαγωγικά. Στην Python, τα αλφαριθμητικά μπορούν να εσωκλείονται σε μονά ή διπλά εισαγωγικά.

Λοιπόν, έχουμε καλύψει τις δομές ελέγχου, την είσοδο και την έξοδο — τώρα χρειαζόμαστε μερικές κομψές δομές δεδομένων. Οι πιο σημαντικές είναι οι *λίστες* και τα *λεξικά*. Οι λίστες γράφονται με αγκύλες, και μπορούν (φυσικά) να είναι εμφωλευμένες:

```
name = ["Παπαδόπουλος", "Γιάννης"]

x = [[1, 2, 3], [y, z], [[[]]]]
```

Ένα από τα ωραία πράγματα με τις λίστες είναι ότι μπορείτε να έχετε πρόσβαση στα στοιχεία τους ξεχωριστά ή σε ομάδες, μέσω της *δεικτοδότησης (indexing)* και της *διαμέρισης (slicing)*. Η δεικτοδότηση γίνεται (όπως σε πολλές άλλες γλώσσες) προσθέτοντας στο τέλος της λίστας το δείκτη σε αγκύλες. (Σημειώστε ότι το πρώτο στοιχείο έχει δείκτη 0.)

```
print name[1], name[0] # εκτυπώνει "Γιάννης Παπαδόπουλος"

name[0] = "Παπανικολάου"
```

Η διαμέριση είναι σχεδόν όπως η δεικτοδότηση, εκτός από το ότι υποδεικνύετε τόσο τον αρχικό όσο καί τον τελικό δείκτη του αποτελέσματος, με μια άνω–κάτω τελεία (`:`) να τους χωρίζει:

```
x = ["spam", "spam", "spam", "spam", "spam", "αυγά", "και", "spam"]

print x[5:7] # εκτυπώνει τη λίστα ["αυγά", "και"]
```

Προσέξτε ότι το τέλος δεν περιλαμβάνεται. Αν κάποιος από τους δείκτες παραλειφθεί, γίνεται η υπόθεση ότι θέλετε τα πάντα προς αυτή την κατεύθυνση. Π.χ. `list[:3]` σημαίνει "κάθε στοιχείο από την αρχή της `list` μέχρι το στοιχείο 3, χωρίς αυτό." (Θα μπορούσε να υποστηριχθεί ότι στην πραγματικότητα θα σήμαινε 4, μιας και το μέτρημα αρχίζει από το 0.) `list[3:]`, από την άλλη μεριά, θα σήμαινε "κάθε στοιχείο από την `list`, αρχίζοντας από το στοιχείο 3 (περιλαμβάνοντάς το) έως και το τελευταίο". Για πραγματικά ενδιαφέροντα αποτελέσματα, μπορείτε να χρησιμοποιήσετε αρνητικούς αριθμούς επίσης: `list[-3]` είναι το τρίτο στοιχείο από το τέλος της `list`...

Σχετικά με το θέμα της δεικτοδότησης, ενδέχεται να ενδιαφέρεστε να γνωρίζετε ότι η ενσωματωμένη συνάρτηση `len` σας δίνει το μήκος μιας λίστας.

Τώρα, λοιπόν — τι γίνεται με τα λεξικά; Για να το θέσω απλά, είναι σαν λίστες, μόνο που τα περιεχόμενά τους δεν είναι ταξινομημένα. Πώς τα δεικτοδοτείς λοιπόν; Κάθε στοιχείο έχει ένα *κλειδί*, ή ένα "όνομα" το οποίο χρησιμοποιείται για να βρεθεί το στοιχείο όπως ακριβώς σε ένα πραγματικό λεξικό. Μερικά παραδείγματα λεξικών:

```
{"Αγγελική": 23452532, "Βαγγέλης": 252336, "Γεωργία": 2352525,
"Δώρα": 23624643}

person = {'όνομα': "Ρομπέν", 'επίθετο': "των Δασών", 'απασχόληση':
"άνεργος"}
```

Τώρα, για να πάρουμε το επάγγελμα του `person`, χρησιμοποιούμε την έκφραση `person["απασχόληση"]`. Αν θέλαμε να αλλάξουμε το επίθετό του, θα μπορούσαμε να γράψουμε:

```
person['επίθετο'] = "του Locksley"
```

Απλό δεν είναι; Όπως οι λίστες, τα λεξικά μπορούν να περιέχουν άλλα λεξικά ή και λίστες. Και φυσικά οι λίστες μπορούν να περιέχουν λεξικά επίσης. Με αυτό τον τρόπο, μπορείτε εύκολα να φτιάξετε μερικές αρκετά εξελιγμένες δομές δεδομένων.

Συναρτήσεις

Επόμενο βήμα: Αφαίρεση. Θέλουμε να δώσουμε ένα όνομα σε ένα κομμάτι κώδικα, και να το καλέσουμε με μερικές παραμέτρους. Με άλλα λόγια — θέλουμε να ορίσουμε μια συνάρτηση (ή "διαδικασία"). Αυτό είναι εύκολο. Χρησιμοποιήστε τη λέξη–κλειδί `def` ως εξής:

```
def square(x):
    return x * x

print square(2) # εκτυπώνει 4
```

Για εκείνους από σας που το καταλαβαίνετε: Όταν περνάτε μια παράμετρο σε μια συνάρτηση, "δένετε" (bind) την παράμετρο στην τιμή, δημιουργώντας έτσι μια νέα αναφορά (reference). Αν αλλάξετε τα "περιεχόμενα" αυτού του ονόματος παραμέτρου (δηλαδή το "επαναδέσετε") αυτό δε θα επηρεάσει την αρχική. Αυτό δουλεύει ακριβώς όπως στη Java, για παράδειγμα. Ας ρίξουμε μια ματιά σε ένα παράδειγμα:

```
def change(some_list):
    some_list[1] = 4

x = [1, 2, 3]
change(x)
print x # εκτυπώνει [1, 4, 3]
```

Όπως μπορείτε να δείτε, περνιέται η αρχική λίστα, και αν η συνάρτηση την αλλάξει, αυτές οι αλλαγές "κουβαλιόνται" στο μέρος όπου η συνάρτηση κλήθηκε. Σημειώστε, όμως, τη συμπεριφορά στο ακόλουθο παράδειγμα:

```
def nochange(x):
    x = 0

y = 1
nochange(y)
print y # εκτυπώνει 1
```

Γιατί δεν υπάρχει αλλαγή τώρα; Διότι *δεν αλλάζουμε την τιμή*! Η τιμή που περνιέται είναι ο αριθμός 1 — δε μπορούμε να αλλάξουμε έναν αριθμό με τον ίδιο τρόπο που αλλάζουμε μια λίστα. Ο αριθμός 1 είναι (και θα είναι πάντα) ο αριθμός 1. Αυτό που *όντως* κάναμε είναι η αλλαγή των περιεχομένων της τοπικής μεταβλητής (παράμετρου) `x`, και αυτό *δεν* "κουβαλιέται" στο περιβάλλον.

Για εκείνους από σας που δεν το κατάλαβαν: Μην ανησυχείτε — είναι αρκετά εύκολο αν δεν το σκέφτεστε πολύ. :)

Η Python έχει όλων των ειδών λογής πρακτικά πράγματα όπως *ονομαζόμενα ορίσματα* και *προκαθορισμένα ορίσματα* και μπορεί να διαχειριστεί ένα μεταβλητό αριθμό από ορίσματα σε μια μόνο συνάρτηση. Για περισσότερες πληροφορίες πάνω σε αυτό, δείτε τον τομέα 4.7 του Python tutorial.

Αν γνωρίζετε πώς να χρησιμοποιείτε συναρτήσεις γενικά, αυτό είναι βασικά αυτό που χρειάζεται να ξέρετε για αυτές στην Python. (Ω, ναι… Η λέξη–κλειδί `return` σταματάει την εκτέλεση της συνάρτησης και επιστρέφει τη δοσμένη τιμή.)

Ένα πράγμα που θα ήταν χρήσιμο να γνωρίζετε, παρόλα αυτά, είναι ότι οι συναρτήσεις είναι *τιμές* στην Python. Έτσι, αν έχετε μια συνάρτηση όπως η `square`, μπορείτε να κάνετε κάτι σαν:

```
queeble = square
print queeble(2) # εκτυπώνει 4
```

Για να καλέσετε μια συνάρτηση χωρίς ορίσματα πρέπει να θυμάστε να γράφετε `doit()` και όχι `doit`. Το τελευταίο, όπως δείχθηκε, επιστρέφει μόνο την ίδια τη συνάρτηση, ως τιμή. (Αυτό ισχύει και για τις μεθόδους σε αντικείμενα επίσης… Δείτε παρακάτω.)

Αντικείμενα

Υποθέτω ότι γνωρίζετε πώς δουλεύει ο αντικειμενοστρεφής προγραμματισμός. (Διαφορετικά, αυτός ο τομέας ενδέχεται να μην έχει και πολύ νόημα για σας. Κανένα πρόβλημα… Αρχίστε να παίζετε χωρίς τα αντικείμενα. :)) Στην Python καθορίζετε κλάσεις με τη λέξη–κλειδί (έκπληξη!) `class`, έτσι:

```
class Basket:

    # πάντα να θυμάστε το όρισμα self
    def __init__(self, contents=None):
        self.contents = contents or []

    def add(self, element):
        self.contents.append(element)

    def print_me(self):
        result = ""
```

```
        for element in self.contents:
            result = result + " " + `element`
        print "Περιέχει: " + result
```

Νέα πράγματα εδώ:

1. Όλες οι μέθοδοι (συναρτήσεις σε ένα αντικείμενο) δέχονται ένα επιπλέον όρισμα στην αρχή της λίστας ορισμάτων, το οποίο περιέχει το ίδιο το αντικείμενο. (Λέγεται `self` σε αυτό το παράδειγμα και είναι θέμα επιλογής.)

2. Οι μέθοδοι καλούνται ως εξής: `object.method(arg1, arg2)`.

3. Μερικά ονόματα μεθόδων, όπως `__init__` (με δύο underscore πριν και μετά), είναι προκαθορισμένες, και σημαίνουν ξεχωριστά πράγματα. `__init__` είναι το όνομα του *constructor* της κλάσης, δηλαδή είναι η συνάρτηση η οποία καλείται όταν δημιουργείτε ένα στιγμιότυπο.

4. Μερικά ορίσματα μπορεί να είναι *προαιρετικά* και να τους δωθεί προκαθορισμένη τιμή (όπως αναφέρθηκε πριν, στον τομέα των συναρτήσεων). Αυτό γίνεται γράφοντας τον ορισμό σαν:

   `def spam(age=32): ...`
   Εδώ, η `spam` μπορεί να κληθεί με μία ή μηδέν παραμέτρους. Αν καμία δε χρησιμοποιηθεί, τότε η παράμετρος `age` θα έχει την τιμή 32.

5. "Λογική βραχυκυκλώματος." Αυτό είναι έξυπνο… Δείτε παρακάτω.

6. Τα ανάποδα μονά εισαγωγικά μετατρέπουν ένα αντικείμενο στην αλφαριθμητική του αναπαράσταση. (Έτσι αν το `element` περιέχει τον αριθμό 1, τότε το `` `element` `` είναι το ίδιο με `"1"` ενώ το `'element'` είναι ένα ακριβές/literal αλφαριθμητικό.)

7. Το σύμβολο της πρόσθεσης + χρησιμοποιείται επίσης για την ένωση λιστών, και τα αλφαριθμητικά στην πραγματικότητα είναι απλά λίστες χαρακτήρων (που σημαίνει ότι μπορείτε να χρησιμοποιήσετε δεικτοδότηση και διαμέριση και τη συνάρτηση `len` σε αυτά. Καλό, ε;)

Καμία μέθοδος ή μεταβλητή μεθόδου δεν είναι προστατευμένη (ή ιδιωτική ή κάτι παρόμοιο) στην Python. Η ενθυλάκωση (encapsulation) είναι αρκετά θέμα προγραμματιστικού στυλ. (Αν το χρειάζεστε *πραγματικά*, υπάρχουν ονομαστικές συμβάσεις που θα επιτρέψουν λίγη "απομόνωση".)

Τώρα, σχετικά με αυτή τη λογική βραχυκυκλώματος…

Όλες οι τιμές στην Python μπορούν να χρησιμοποιηθούν ως λογικές τιμές. Μερικές από τις πιο "άδειες", όπως `[]`, `0`, `""` και `None` αναπαριστούν λογική αναλήθεια (ψεύδος), ενώ οι περισσότερες άλλες τιμές (όπως `[0]`, `1` ή `"Γεια σου, κόσμε!"`) αναπαριστούν λογική αλήθεια.

Τώρα, λογικές εκφράσεις όπως `a and b` εκτιμώνται ως εξής: Πρώτα, έλεγξε αν η `a` είναι αληθής. Αν *δεν* είναι, τότε απλά επέστρεψέ τη. Αν *είναι*, τότε απλά επέστρεψε τη `b` (η οποία θα αναπαριστά την τιμή αληθείας της έκφρασης). Η αντίστοιχη λογική για `a or b` είναι: Αν η `a` είναι αληθής, τότε επέστρεψέ τη. Αν δεν είναι, τότε επέστρεψε τη `b`.

Αυτός ο μηχανισμός κάνει τα `and` και `or` να συμπεριφέρονται σαν τους δυαδικούς τελεστές που αναμένεται ότι υλοποιούν, αλλά σας επιτρέπουν επίσης να γράψετε σύντομες και γλυκές εκφράσεις συνθήκης. Για παράδειγμα, η πρόταση

```
if a:
    print a
```

```
else:
    print b
```

Θα μπορούσε να γραφεί διαφορετικά:

```
print a or b
```

Στην πραγματικότητα, αυτό είναι κάτι σαν ιδίωμα της Python, έτσι ενδέχεται κάλλιστα να το συνηθίσετε. Αυτό είναι εκείνο που κάνουμε στη μέθοδο `Basket.__init__`. Το όρισμα `contents` έχει μια προκαθορισμένη τιμή `None` (που είναι, πέρα από άλλα, ψευδής). Έτσι, για να ελέγχαμε αν είχε μια τιμή, θα μπορούσαμε να γράψουμε:

```
if contents:
    self.contents = contents
else:
    self.contents = []
```

Φυσικά, τώρα γνωρίζετε ότι υπάρχει ένας καλύτερος τρόπος. Και γιατί δεν του δίνουμε την προκαθορισμένη τιμή `[]` εξ αρχής; Λόγω του τρόπου με τον οποίο δουλεύει η Python, αυτό θα έδινε σε όλα τα Basket την ίδια άδεια λίστα ως προκαθορισμένο περιεχόμενο. Μόλις κάποιο από αυτά άρχιζε να γεμίζει, όλα θα περιείχαν τα ίδια στοιχεία, και το προκαθορισμένο δε θα ήταν πλέον άδειο… Για να μάθετε περισσότερα σχετικά με αυτό, πρέπει να διαβάσετε την τεκμηρίωση και να ψάξετε για τη διαφορά μεταξύ *ταυτότητας* και *ισότητας*.

Ένας άλλος τρόπος για να γίνουν τα παραπάνω είναι:

```
def __init__(self, contents=[]):
    self.contents = contents[:]
```

Μπορείτε να μαντέψετε πώς δουλεύει αυτό; Αντί να χρησιμοποιούμε την ίδια άδεια λίστα παντού, χρησιμοποιούμε την έκφραση `contents[:]` για να φτιάξουμε ένα αντίγραφο. (Απλά διαμερίζουμε το όλο πράγμα.)

Έτσι, για να φτιάχναμε πραγματικά ένα `Basket` και να το χρησιμοποιούσαμε (δηλαδή να καλούσαμε μερικές μεθόδους του) θα κάναμε κάτι σαν αυτό:

```
b = Basket(['μήλο', 'πορτοκάλι'])
b.add("λεμόνι")
b.print_me()
```

Υπάρχουν και άλλες μαγικές μέθοδοι εκτός από την `__init__`. Μία τέτοια μέθοδος είναι η `__str__` η οποία καθορίζει πώς το αντικείμενο θέλει να φαίνεται αν αντιμετωπιστεί σαν ένα αλφαριθμητικό. Θα μπορούσαμε να χρησιμοποιήσουμε αυτό στο καλάθι μας αντί για την `print_me`:

```
def __str__(self):
    result = ""
    for element in self.contents:
        result = result + " " + `element`
    return "Contains:" + result
```

Τώρα, αν θέλαμε να εκτυπώσουμε το καλάθι `b`, θα μπορούσαμε απλά να χρησιμοποιήσουμε:

```
print b
```

Καλό, ε;

Το subclassing γίνεται ως εξής:

```
class SpamBasket(Basket):
```

```
    # ...
```

Η Python επιτρέπει πολλαπλή κληρονομικότητα, έτσι μπορείτε να έχετε αρκετές υπερκλάσεις στις παρανθέσεις, χωρισμένες με κόμματα. Οι κλάσεις αρχικοποιούνται ως εξής: `x = Basket()`. Οι constructors, όπως είπα, φτιάχνονται ορίζοντας την ειδική συνάρτηση μέλους `__init__`. Ας πούμε ότι το `SpamBasket` είχε έναν `__init__(self, type)`. Τότε θα μπορούσατε να φτιάξετε ένα καλάθι spam ως εξής: `y = SpamBasket("μήλα")`.

Αν, στον constructor της `SpamBasket`, χρειαζόσασταν να καλέσετε τον constructor μιας ή περισσότερων υπερκλάσεων, θα μπορούσατε να τον καλέσετε ως εξής: `Basket.__init__(self)`. Σημειώστε ότι, παράλληλα με την παροχή των συνηθισμένων παραμέτρων, πρέπει να παρέχετε ρητά τη `self`, μιας και η υπερκλάση `__init__` δε γνωρίζει με ποιό στιγμιότυπο έχει να κάνει.

Για περισσότερα σχετικά με τα θαύματα του αντικειμενοστρεφή προγραμματισμού στην Python, δείτε τον τομέα 9 του tutorial.

Ένα "Jedi Mind" Κόλπο

(Αυτός ο τομέας είναι εδώ μόνο επειδή νομίζω ότι είναι ωραίος. *Δεν* είναι απαραίτητο να τον διαβάσετε για να αρχίσετε να μαθαίνετε την Python. Δείτε το τέλος του τομέα για μια σημείωση σχετικά με αλλαγές για την Python 2.1.)

Σας αρέσουν οι έννοιες που κάνουν το μυαλό να "σαστίζει"; Τότε, αν είστε πραγματικά τολμηροί, ίσως να θελήσετε να ελέγξετε την έκθεση του Guido van Rossum στις μετακλάσεις. Αν, όμως, προτιμάτε να *μην* εκραγεί το μυαλό σας, ίσως να ικανοποιηθείτε με αυτό το μικρό κόλπο.

Η Python χρησιμοποιεί δυναμικούς αντί για λεκτικούς χώρους ονομάτων (namespaces). Αυτό σημαίνει ότι αν έχετε μια συνάρτηση σαν αυτή:

```
def orange_juice():
    return x * 2
```

… όπου μια μεταβλητή (σε αυτή την περίπτωση η `x`) δεν είναι "δεμένη" σε ένα όρισμα και δεν της δίνεται μια τιμή μέσα στη συνάρτηση, η Python θα χρησιμοποιήσει την τιμή που έχει όπου και όταν η συνάρτηση κληθεί. Σε αυτή την περίπτωση:

```
x = 3
y = orange_juice()
# η y είναι τώρα 6
x = 1
y = orange_juice()
# η y είναι τώρα 2
```

Συνήθως, αυτό είναι το είδος της συμπεριφοράς που θέλετε (αν και το πάνω παράδειγμα είναι λίγο contrived — σπάνια η πρόσβασή σας σε μεταβλητές είναι έτσι.) *Όμως*, μερικές φορές μπορεί να είναι ωραίο να έχετε κάτι σαν ένα στατικό χώρο ονομάτων, δηλαδή, την αποθήκευση μερικών τιμών από το περιβάλλον στη συνάρτηση όταν αυτή δημιουργείται. Ο τρόπος για να το κάνετε αυτό στην Python είναι μέσω των προκαθορισμένων ορισμάτων.

```
x = 4
def apple_juice(x=x):
    return x * 2
```

Εδώ, στο όρισμα `x` δίνεται μια προκαθορισμένη τιμή η οποία είναι ίδια με την *τιμή* της μεταβλητής `x` τη στιγμή που η συνάρτηση ορίζεται. Έτσι, όσο κανείς δεν παρέχει ένα όρισμα για τη συνάρτηση, θα δουλέψει ως εξής:

```
x = 3
y = apple_juice()
# η y είναι τώρα 8
x = 1
y = apple_juice()
# η y είναι τώρα 8
```

Έτσι — η τιμή της `x` δεν αλλάζει. Αν αυτό ήταν όλο κι όλο αυτό που θέλαμε, θα μπορούσαμε να είχαμε γράψει επίσης

```
def tomato_juice():
    x = 4
    return x * 2
```

ή ακόμα και

```
def carrot_juice():
    return 8
```

Όμως, το *νόημα* είναι ότι η τιμή της `x` λαμβάνεται από το *περιβάλλον* τη στιγμή που η συνάρτηση ορίζεται. Πώς είναι αυτό χρήσιμο; Ας πάρουμε ένα παράδειγμα — μια συνάρτηση η οποία συνθέτει δύο άλλες συναρτήσεις.

Θέλουμε μια συνάρτηση που να δουλεύει ως εξής:

```
from math import sin, cos

sincos = compose(sin, cos)

x = sincos(3)
```

Όπου `compose` είναι η συνάρτηση που θέλουμε να φτιάξουμε, και η `x` έχει την τιμή `-0.836021861538`, που είναι το ίδιο με `sin(cos(3))`. Τώρα, πώς το κάνουμε αυτό;

(Σημειώστε ότι χρησιμοποιούμε συναρτήσεις ως ορίσματα εδώ… Αυτό είναι ένα αρκετά καλό κόλπο από μόνο του.)

Φανερά, η `compose` παίρνει δύο συναρτήσεις ως παράμετρους, και επιστρέφει μια συνάρτηση η οποία ξαναπαίρνει μια παράμετρο. Έτσι, μια λύση–σκελετός θα μπορούσε να είναι:

```
def compose(fun1, fun2):
    def inner(x):
        pass # ...
    return inner
```

Θα μπορούσαμε να μπούμε στον πειρασμό να βάλουμε `return fun1(fun2(x))` μέσα στη συνάρτηση `inner` και να το αφήσουμε έτσι. Όχι, όχι, όχι. Αυτό θα συμπεριφερόταν πολύ παράξενα. Φανταστείτε το ακόλουθο σενάριο:

```
from math import sin, cos

# λανθασμένη εκδοχή
def compose(fun1, fun2):
    def inner(x):
        return fun1(fun2(x))
    return inner

def fun1(x):
```

```
    return x + " κόσμε!"

def fun2(x):
    return "Γεια σου,"

sincos = compose(sin, cos) # χρησιμοποιώντας τη λανθασμένη εκδοχή

x = sincos(3)
```

Τώρα, τι τιμή θα είχε η `x`; Σωστά: `"Γεια σου, κόσμε!"`. Γιατί συμβαίνει αυτό; Διότι όταν καλείται, λαμβάνει τις τιμές της `fun1` και `fun2` από το περιβάλλον, όχι αυτές που βρίσκονταν τριγύρω όταν δημιουργήθηκε. Για να πάρουμε μια λύση που να δουλεύει, το μόνο που έχουμε να κάνουμε είναι να χρησιμοποιήσουμε την τεχνική που περιέγραψα νωρίτερα:

```
def compose(fun1, fun2):
    def inner(x, fun1=fun1, fun2=fun2):
        return fun1(fun2(x))
    return inner
```

Τώρα το μόνο που έχουμε να κάνουμε είναι να ελπίζουμε να μην παράσχει κάποιος στη συνάρτηση που προκύπτει περισσότερα από ένα γνωρίσματα, μιας και αυτό θα την κατέστρεφε. :) Και επί τη ευκαιρία, από τη στιγμή που δε χρειαζόμαστε το όνομα `inner`, και περιέχει μόνο μια έκφραση, μπορούμε επίσης να χρησιμοποιήσουμε μια *ανώνυμη* συνάρτηση, χρησιμοποιώντας τη λέξη–κλειδί `lambda`:

```
def compose(f1, f2):
    return lambda x, f1=f1, f2=f2: f1(f2(x))
```

Λακωνικό, αλλά καθαρό. Θα το λατρέψετε. :)

(Και αν δεν καταλάβατε τίποτα από αυτό, μην ανησυχείτε. Τουλάχιστον ελπίζω ότι σας έπεισε ότι η Python είναι περισσότερο από "απλά μια γλώσσα scripting"… :))

Σημείωση για την Python 2.1 και τα εμφωλευμένα πεδία

Με τον ερχομό της Python 2.1, η γλώσσα τώρα έχει (προαιρετικά) στατικά εμφωλευμένα πεδία ή χώρους ονομάτων. Αυτό σημαίνει ότι μπορείτε να κάνετε τα πράγματα που περιγράφονται στον τομέα αυτό χωρίς μερικά από τα κόλπα. Πλέον μπορείτε απλά να γράφετε:

```
# αυτό το "μαγικό" δε θα είναι απαραίτητο από την Python 2.2
from __future__ import nested_scopes

def compose(fun1, fun2):
    def inner(x):
        return fun1(fun2(x))
    return inner
```

… και θα δουλέψει όπως πρέπει.

Και τώρα…

Λίγα πράγματα κοντά στο τέλος. Οι πιο χρήσιμες συναρτήσεις και κλάσεις τοποθετούνται σε *μονάδες (modules)*, που στην πραγματικότητα είναι αρχεία κειμένου που περιέχουν κώδικα. Μπορείτε να τις εισάγετε και να τις χρησιμοποιήσετε στα ίδια σας τα προγράμματα. Για παράδειγμα, για να χρησιμοποιήσετε τη μέθοδο `split` από την πρότυπη (standard) μονάδα `string`, μπορείτε να κάνετε:

```
import string
```

```
x = string.split(y)
```

Ή…

```
from string import split

x = split(y)
```

Σημείωση: Η μονάδα `string` δεν χρησιμοποιείται και πολύ πλέον· αντί για τον κώδικα παραπάνω, *πρέπει* να χρησιμοποιείτε `x = y.split()`.

Για περισσότερες πληροφορίες σχετικά με τις πρότυπες μονάδες βιβλιοθήκης, ρίξτε μια ματιά στο http://www.python.org/doc/lib/. Περιέχουν πολλά χρήσιμα πράγματα.

Όλος ο κώδικας στη μονάδα/script εκτελείται όταν εισάγεται. Αν θέλετε το πρόγραμμά σας να μπορεί τόσο να εισαχθεί ως μονάδα όσο και να τρέξει ως πρόγραμμα, ενδέχεται να θέλετε να προσθέσετε κάτι σαν αυτό στο τέλος του:

```
if __name__ == "__main__": go()
```

Αυτός είναι ένας μαγικός τρόπος για να λέτε ότι αν η μονάδα αυτή εκτελεστεί ως script (δηλαδή, δεν εισάγεται μέσα σε κάποιο άλλο script), τότε η συνάρτηση `go` πρέπει να κληθεί. Φυσικά, Θα μπορούσατε να κάνετε οτιδήποτε μετά την άνω–κάτω τελεία εκεί… :)

Και για όσους από σας που θέλετε να κάνετε ένα εκτελέσιμο script σε UN*X, χρησιμοποιήστε την ακόλουθη πρώτη γραμμή για να το κάνετε να τρέχει από μόνο του:

```
#!/usr/bin/env python
```

Τέλος, μια σύντομη αναφορά σε μια σημαντική έννοια: Εξαιρέσεις. Μερικές λειτουργίες (όπως η διαίρεση με το μηδέν ή η ανάγνωση από ένα αρχείο που δεν υπάρχει) παράγουν μια συνθήκη σφάλματος ή *εξαίρεση*. Μπορείτε ακόμα και να φτιάξετε δικές σας και να τις ενεργοποιήσετε (raise) στις σωστές χρονικές στιγμές.

Αν δε γίνει τίποτα για την εξαίρεση, το πρόγραμμά σας τελειώνει και εκτυπώνει ένα μήνυμα σφάλματος. Μπορείτε να το αποφύγετε αυτό με μια `try`/`except`–πρόταση. Για παράδειγμα:

```
def safe_division(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return None
```

Η `ZeroDivisionError` είναι μια πρότυπη εξαίρεση. Στην περίπτωση αυτή, θα *μπορούσατε* να είχατε ελέγξει αν η `b` ήταν μηδέν, αλλά σε πολλές περιπτώσεις, αυτή η στρατηγική δεν είναι εφικτή. Και εξάλλου, αν δεν είχαμε την ενότητα (clause) `try` στη `safe_division`, κάνοντάς την έτσι μια ριψοκίνδυνη συνάρτηση προς κλήση, θα μπορούσαμε ωστόσο να κάνουμε κάτι σαν:

```
try:
    unsafe_division(a, b)
except ZeroDivisionError:
    print "Κάτι διαιρέθηκε με μηδέν στην unsafe_division"
```

Σε περιπτώσεις όπου *κανονικά* δε θα είχατε κάποιο συγκεκριμένο πρόβλημα, αλλά θα ήταν *πιθανό* να συμβεί, η χρήση εξαιρέσεων σας επιτρέπει να αποφύγετε ελέγχους που κοστίζουν κ.τ.λ..

Λοιπόν — αυτό είναι. Ελπίζω ότι μάθατε κάτι. Τώρα πηγαίντε και παίξτε. Και θυμηθείτε το μότο εκμάθησης της Python: "Use the source, Luke." (Μετάφραση: Διαβάστε ό,τι κώδικα πέσει στα χέρια σας :)). Για αρχή, ορίστε ένα παράδειγμα. Είναι ο γνωστός αλγόριθμος *QuickSort* του Hoare.

Ένα πράγμα που ίσως αξίζει να αναφερθεί σχετικά με αυτό το παράδειγμα. Η μεταβλητή `done` ελέγχει αν η `partition` έχει τελειώσει την κίνηση στα στοιχεία ή όχι. Έτσι όταν μια από τους δύο εσωτερικούς βρόχους θέλει να τελειώσει ολόκληρη την ακολουθία ανταλλαγής, θέτει τη `done` σε `1` και μετά φεύγει με `break`. Γιατί οι εσωτερικοί βρόχοι χρησιμοποιούν τη `done`; Γιατί, όταν ο πρώτος εσωτερικός βρόχος τελειώνει με μια `break`, το αν ο επόμενος βρόχος θα αρχίσει εξαρτάται από το αν ο κύριος βρόχος έχει τελειώσει, δηλαδή, από το αν η `done` έχει τεθεί σε `1`:

```
while not done:
    while not done:
        # επαναλαμβάνει μέχρι ένα break

    while not done:
        # εκτελείται μόνο αν ο πρώτος δεν όρισε την done σε 1
```

Μια ισοδύναμη, ίσως καθαρότερη, αλλά κατά τη γνώμη μου λιγότερο ωραία έκδοση θα ήταν:

```
while not done:
    while 1:
        # επαναλαμβάνει μέχρι ένα break

    if not done:
        while 1:
            # εκτελείται μόνο αν ο πρώτος δεν όρισε την done σε 1
```

Ο μόνος λόγος που χρησιμοποίησα τη μεταβλητή `done` στον πρώτο βρόχο ήταν ότι μου άρεσε να διατηρήσω τη συμμετρία μεταξύ των δύο. Μ' αυτό τον τρόπο κάποιος θα μπορούσε να αντιστρέψει τη σειρά τους και ο αλγόριθμος θα εξακολουθούσε να δουλεύει.

7. Υλοποίηση εφαρμογής στεγανογραφίας “Kryfto” :

Το Kryfto είναι μία εφαρμογή που έχει σκοπό να παρέχει ασφάλεια προσφέροντας υπηρεσίες μέσο δικτύου χωρίς να είναι απαραίτητη η οποιαδήποτε εγκατάσταση κάποιου λογισμικού από την πλευρά του Client αρκεί μόνο να έχει πρόσβαση στον Server που τρέχει το Kryfto και την χρήση κάποιου δημοφιλές Browser.
Η ασφάλεια που προσφέρει το Kryfto είναι κάνοντας χρήση Στεγανογραφίας και με την επιλογή να προσθέσουμε Κρυπτογραφία αν το επιθυμεί ο χρήστης. Προς το παρόν η χρήση του είναι περιορισμένη στο να «κρύβει» πληροφορία μόνο σε αρχεία εικόνας τύπου BMP 24bit , μπορεί παρόλα αυτά να κρύβει οποιοδήποτε είδους αρχείου μέσα στην εικόνα (κείμενο, video, mp3 ακόμα και μία άλλη εικόνα οποιουδήποτε είδους).

Όπως είδαμε για την υλοποίηση του Kryfto χρησιμοποιήσαμε Python, αυτό όχι μόνο δεν μας περιορίζει αλλά επιπλέον έχουμε την δυνατότητα να επωφεληθούμε από μία σειρά διαθέσιμων εργαλείων που μας παρέχει και τα οποία στην πλειοψηφία τους είναι γραμμένα σε .... Python ;)

Τα εργαλεία που χρησιμοποιήσαμε είναι :

Python 2.5 -> Η πιο πρόσφατη έκδωσε αυτής της σύχρονής γλώσσας (www.python.org)

Karrigell 2.3.1-> Ένα δυνατό αλλά και συγχρόνως πολύ απλό open source Web Framework με full built in web server χωρίς καμία εγκατάσταση (αρκεί να τρέχει η Python) το οποίο είναι γραμμένο 100% σε Python. (www.karrigell.com)

PyCrypto -> Βιβλιοθήκη με αλγορίθμους κρυπτογράφησης , Hash συναρτήσεις, χρήση δημοσίων κλειδιών. (http://www.amk.ca/python/code/crypto)

PIL 1.1.5 (Python Image Library) -> Όπως μας λέει και το όνομα, μια ολοκληρωμένη βιβλιοθήκη με πολύτιμα εργαλεία για επεξεργασία εικόνας . (http://www.pythonware.com/products/pil/)

7.1 Kryfto setup:

Αφού έχουμε κατεβάσει και εγκαταστήσει με την ακόλουθη σειρά Python, PyCrypto και PIL , ήμαστε έτοιμοι να χρησιμοποιήσουμε το Kryfto. Για καλή μας τύχη δεν χειριάζετε καμία εγκατάσταση, απλά κάνουμε unzip το αρχείο KryftoServer.zip οπουδήποτε θέλουμε . Μέσα στο φάκελο αυτό υπάρχει το Karrigell.py. Σε περιβάλλον Windows κάνουμε διπλό κλικ αυτό το αρχείο και ..... ο Server μας τρέχει . Για να το δοκιμάσουμε στην πράξη ανοίγουμε οποιοδήποτε Browser διαθέτουμε και πληκτρολογούμε για διεύθυνση : http://localhost:8888/Kryftosite/

7.2 Κώδικας Steganography πίσω από το Kryfto :

Οι δύο κύριες λειτουργίας στην εφαρμογή μας είναι η στεγανογράφηση και η αποστεγανογράφηση . Για χάρη της υλοποίησής του Kryfto αυτές τις δύο διαδικασίες τις έχουμε ονομάσει enCryft και deCryft αντίστοιχα. Στην συνέχεια αναλύουμε με λίγα λόγια τον κώδικα αυτόν τον 2 μεθόδων.

7.2.A enCryft() :

```
def enCryft(CoverFile='Original.bmp', file2hide='SecretFile.txt',
CarrierFile='Steganographed.bmp', BIT_DEPTH =
1,cipherAlgorithm=None,passPhrase=None):

    '''enCryft is used to hide any type of file inside an image file.
    The current version support only BMP type.

    Argument list:

    CoverFile -> The file-name of the image that will be used as a cover
to hide the secret data inside.
    NOTE: The original image will not change .

    file2hide -> The file-name that you want to hide inside the
'CoverFile' .

    CarrierFile -> The file-name of the new image file that will be
created after the
    steganography.
    Default value is Steganographed.bmp

    BIT_DEPTH -> The compact level of the LSB encoding (1 - 8). The higher
the level is ,
    the more data can be hided inside the cover reducing the image quality
.
    Default value is 1.

    cipherAlgorithm -> The encryption algortihm that will be used to
ecrypt the data. (must be one of the existing list)

    passPhrase -> Tha passphrase used as a key by the choosed crypto
algorithm to encrypt the data.
    '''
```

Η μέθοδος enCryft δέχετε 6 παραμέτρους ;

CoverFile -> Την εικόνα που θα χρησιμοποιήσουμε σαν "κάλυμμα" για να κρύψουμε μέσα της πληροφορία (πρέπει να είναι BMP 24 bit)
File2hide -> Το αρχείο με τα δεδομένα που θέλουμε να κρύψουμε μέσα στην εικόνα. Μπορεί να είναι οποιαδήποτε μορφής αρκεί να χωράει στην εικόνα.

CarrierFile -> Το όνομα που θα αποθηκευτεί η καινούρια εικόνα μετά την κωδικοποίηση με κρυμμένη μέσα της την πληροφορία.

BIT_DEPTH -> Τον βαθμό της κωδικοποίησης (μέχρι ποιο LSB θα αλλάξουμε την τιμή για να κρύψουμε την πληροφορία)

cipherAlgorithm -> Τον αλγόριθμο κρυπτογράφησης για να κρυπτογραφήσουμε τα δεδομένα που θα κρύψουμε. Πρέπει να υπάρχει στην λίστα του προγράμματος αλλιώς δεν υποστηρίζεται...

passPhrase -> Το συνθηματικό που θα χρησιμοποιήσουμε για την κρυπτογράφηση. Χρησιμοποιείτε σαν κλειδί για την κωδικοποίηση.(Υπάρχουμε μερικοί τεχνικοί περιορισμοί για το μήκος του paasword. Δες το encryption.py για περισσότερες λεπτομέρειες)
Σημείωση: Σε περίπτωση που δεν οριστεί κάποιο αλγόριθμο κρυπτογράφησης, η εισαγωγή συνθηματικού αγνοείτε.

Στην συνέχεια βλέπετε την "καρδιά" του πηγαίου, εκεί που γίνονται όλες οι πράξεις με τα pixel manipulation και τα binary values που διαβάζουμε από την πηγή και κρύβουμε στην εικόνα :

```
for x in range(0,width):
        if END_OF_ENCODING_FLAG == True :
            break
        for y in range(0,height):

            pixelValue=imTmp.getpixel((x,y)) # RGB in decimal form  exp:
(150,33,240)

            R=pixelValue[0]
            G=pixelValue[1]
            B=pixelValue[2]

            # --- Header Insert Start Section ---

            if headerFlag == False:
                # considering if value the chage the bit else pass $$$
                R = ReplaceBit(R,0,ExtractBit(header,r))
                r+=1
                G = ReplaceBit(G,0,ExtractBit(header,r))
                r+=1
                B = ReplaceBit(B,0,ExtractBit(header,r))
                r+=1

                #length of header + BIT_DEPTH
                if r >= 29 :
                    headerFlag=True # Header have been inserted

                msgStatus = msgStatus + "R=%i G=%i B=%i \n"%(R,G,B)
                msgStatus = msgStatus + 'headerFlag is --> %s and whe are
in the cordinations x=%i y=%i r=%i \n'%(headerFlag,x,y,r)
```

```
                imTmp.putpixel((x,y),(R,G,B))
                continue
            # --- Header End ---
            # ---- Message/Data Start ----
            iSB=0

            for p in range(BIT_DEPTH):

                if index >= msgSize:    # must be checked before the main
process start ... $$$
                    END_OF_ENCODING_FLAG = True
                    break

                # Check if the current character (or binary symbol) have
been encoded in the pixel so it can proceed with the next one
                if nextChar == True :

                    # character (or binary symbol) to ASCII value to .
                    # char val must not be greater than 8 byte (255) $$$
                    charVal=ord(text2code[index])

                    index=index+1
                    r=0

                    nextChar=False

                R = ReplaceBit(R,iSB,ExtractBit(charVal,r))
                r+=1
                G = ReplaceBit(G,iSB,ExtractBit(charVal,r))
                r+=1
                B = ReplaceBit(B,iSB,ExtractBit(charVal,r))
                r+=1

                iSB+=1

                # The current character (or binary symbol) have been
inserted , signal for the next one
                if r >= 9:
                    nextChar = True

            imTmp.putpixel((x,y),(R,G,B))


    msgStatus = msgStatus + '------------ ############### ---- \n'
```

7.2.B deCryft() :

```
def
deCryft(CarrierFile='Steganographed.bmp',ExtractedFile='extracted',passPhrase=
None):
    '''deCryft is used to extract secret data from inside an image file.

    Argument list:

    CarrierFile -> The name of the image file that has the secret data.

    ExtractedFile -> The file-name used to save the extracted file.

    passPhrase -> passPhrase -> The passphrase used as a key by the choosed
crypto algorithm (when the data was encrypted) so it can be decrypted
successfully.
    '''
```

Η αντίστροφή διαδικασία deCryft είναι πιο απλή γιατί δέχεται μόνο 3 ορίσματα αφού μερικές πληροφορίες όπως το extension του αρχείου, το BIT_DEPTH κωδικοποίησης LSB και τον αλγόριθμο κρυπτογράφησης είναι ενσωματωμένα σε ένα header που καταχωρήθηκε κατά την κωδικοποίηση του προγράμματος. Οπότε το μόνο που μας μένει να ορίσουμε είναι :

CarrierFile -> Η στεγανογραφημένη εικόνα που περιέχει την μυστική πληροφορία.

ExtractedFile -> Το όνομα που θα σωθεί το αρχείο με την πληροφορία που εξάγαμε.

passPhrase -> Το συνθηματικό για να χρησιμοποιηθεί σαν κλειδί αποκρυπτογράφησης για να είναι επιτυχής η αντίστροφη διαδικασία. Αν το συνθηματικό δεν είναι σωστό τα δεδομένα που θα εξάγουμε δεν θα είναι σε λογική μορφή.
Σημείωση: Σε περίπτωση που δεν οριστεί κάποιο αλγόριθμο κρυπτογράφησης, η εισαγωγή συνθηματικού αγνοείτε.

Αναλόγως έχουμε το πηγαίο κώδικα της αντίστροφης διαδικασία αποκωδικοποίησης. Εδώ επίσης γίνονται όλες οι πράξεις με τα pixel manipulation και τα binary values που διαβάζουμε αυτήν την φορά από την εικόνα (pixels values) και τα αποθηκεύουμε σε ανάλογα bit's για να σχηματιστεί το κρυμμένο αρχείο με τα δεδομένα που ήταν κρυμμένα στην εικόνα. :

```
  for x in range(0,width):
           for y in range(0,height):

               pixelValue=imTmp2.getpixel((x,y))

               R=pixelValue[0]
               G=pixelValue[1]
               B=pixelValue[2]

               # ---- Header Retrieving Section ---
```

```
            if headerFlag == False:

                header = ReplaceBit(header,r,ExtractBit(R,0))
                r+=1

                header = ReplaceBit(header,r,ExtractBit(G,0))
                r+=1

                header = ReplaceBit(header,r,ExtractBit(B,0))
                r+=1

                if r >= 29 : # Header extract completed
                    msgStatus = msgStatus + ' header (before >> 3) = %d
\n' %(header)
                    for i in range(3):
                        BIT_DEPTH =
ReplaceBit(BIT_DEPTH,i,ExtractBit(header,27 + i)) # extract the BIT_DEPTH
information

                    for i in range(3):
                        header = ReplaceBit(header,27 + i,0) # remove the
BIT_DEPTH info from the header

                    msgStatus = msgStatus + 'Header after !!!! = %d \n'
%(header)

                    BIT_DEPTH = BIT_DEPTH + 1 # (state 1 in bit's are 000)

                    length = header
                    msgStatus = msgStatus + '\n BIT_DEPTH = %d \n r = %d
\n header = %d' %(BIT_DEPTH,r,header)

                    headerFlag= True

                    msgStatus = msgStatus + 'Header retrieved successful !
message size is %i and BIT_DEPTH = %i \n'%(length,BIT_DEPTH)
                    msgStatus = msgStatus + 'Pocessing with the data...\n'
                continue

               # ---- Header End ---

            # ---- Message extracted Section ----
             iSB=0
             for p in range(BIT_DEPTH):

                if j < length:

                    if nextChar == True:
                        rr = 0  #$$$
                        charVal = 0
                        nextChar = False
```

```
                    charVal = ReplaceBit(charVal,rr,ExtractBit(R,iSB))
                    rr+=1

                    charVal = ReplaceBit(charVal,rr,ExtractBit(G,iSB))
                    rr+=1

                    charVal = ReplaceBit(charVal,rr,ExtractBit(B,iSB))
                    rr+=1

                    iSB+=1

                    # the 9th bit must always be 0 because if it is 1 is
greater than the max value allowed (255), improvement needed ... $$$
                    if charVal > 255:
                        msgStatus = msgStatus +  '\n Warning !!! \ncharVal
= %d || rr=%d nextChar = %s || j=%d ' %(charVal,rr,nextChar,j)
                        charVal = ReplaceBit(charVal,8,0)
                        msgStatus = msgStatus +  '\ncharVal now is = %d'
%(charVal)

                    # bits are ready to format a value and continue with
the next one $$
                    if rr >= 9 :
                        msgText=msgText + chr(charVal)
                        nextChar = True
                        j+=1 ###$$$len(msgText > length gia end afths ths
if)

            # ---- Message End ----
```

7.2.C Η πληροφορία HEADER στην κωδικοποίηση :

Όπως προσέξατε η διαδικασία για deCryft είναι πολύ πιο απλή αφού χρειάζεται μόνο 3 παραμέτρους (από τις οποίες μόνο η πρώτη είναι υποχρεωτική) . Αυτό γίνετε γιατί όπως αναφέραμε το "βάθος τις κωδικοποίησης LSB" όπως και με ποιόν αλγόριθμο έγινε η κρυπτογράφηση (αν χρησιμοποιήθηκε κρυπτογράφηση) είναι ενσωματωμένη στην εικόνα που κρύψαμε τα δεδομένα. Για το λόγο αυτό φτιάξαμε ένα δικό μας πρωτόκολλο και δημιουργήσαμε ένα HEADER το οποίο περιέχει πληροφορίες όπως αυτές που αναφέραμε προηγουμένως, το μέγεθος του αρχείου καθώς και την κατάληξη (extension) που είχε πριν το κρύψουμε μέσα στην εικόνα.
Σημείωση : Σε περίπτωση που χρησιμοποιήσαμε τον web text editor , θεωρούμε ότι το αρχείο που θα ανακτηθεί θα έχει κατάληξη .txt

Στην εικόνα αυτό που κρύβετε (και ανακτάτε) είναι το HEADER + ΔΕΔΟΜΕΝΑ.

Για το HEADER πρέπει να ξέρουμε ότι δημιουργείτε και ορίζετε με την σειρά ως εξής:

27 bits για το μέγεθος του αρχείου που θα κρύψουμε / ανακτήσουμε

3 bits για να υποδείξουμε σε ποιό βαθμό έγινε η κωδικοποίηση LSB

24 bits για την περιγραφή της κατάληξης του αρχείου

8 bits για τον αλγόριθμο κρυπτογράφησης που χρησιμοποιήθηκε (αν υπάρχει)

Οπότε το HEADER = 62 bits

Η διαδικασία αποκωδικοποίησης του HEADER γίνετε με ανάλογο τρόπο όπως στην κωδικοποίηση, παρακάτω έχουμε το κύριο μέρος του κώδικα από το enCryft για την δημιουργία του HEADER :

```
header = msgSize

    if header > 134217727 : # 134217727 in binary is 27 bit
(HeaderBitLength)
        print 'Fatal error !!! header size capacity more than the maximum
of the HeaderBitLength allowed... \n'

# Must be position 27 ( 0 - 27 equal to 28 different values ) , 3 bit to
express the BIT_DEPTH (3 bit = 8 different values)
    for i in range(3):
        header = ReplaceBit(header, 27 + i , ExtractBit(BIT_DEPTH-1,i))

    text2code=file2hide[-3:] + text2code # Insert the extension file

algoDict={None:'0','DES3':'1','Blowfish':'2','IDEA':'3','RC5':'4','ARC2':'
5','CAST':'6'}

text2code = algoDict[cipherAlgorithm] + text2code #Insert the Crypto
```

7.2.D ExtractBit και ReplaceBit :

Όλη η δύναμη της κωδικοποίησης γίνετε βεβαίως από τις συναρτήσεις getPixel setPixel για την επεξεργασία των τιμών κάθε pixel αλλά και για να αλλάξουμε τις τιμές αυτές επεμβαίνουμε χαμηλά στην binary τιμή τους . Επειδή έχουμε να κάνουμε με εικόνα και κάθε εικόνα είναι ένας μεγάλος πίνακας από pixels , αυτό σημαίνει ότι επεξεργασία κάνουμε για ένα pixel , αυτή η πράξη θα γίνει εκατοντάδες ή και χιλιάδες φορές. Οπότε σε αυτό το σημείο καλό είναι η πράξεις μας να είναι μετρημένες και όσο πιο γρήγορες γιατί μία μικρή καθυστέρηση επί πολλά pixels που έχουμε να επεξεργαστούμε θα έχει σαν αποτέλεσμα μία μεγάλη καθυστέρηση στην όλη κωδικοποίηση. Όλο η δύναμη της binary manipulation είναι γραμμένες σε δύο συναρτήσεις , την ExtractBit kai ReplaceBit που αν και πολύ μικρές σε μέγεθος (ελάχιστες γραμμές κώδικα) είναι αυτές που κάνουν την σημαντικότερη δουλειά.

Η πρώτη δέχεται δύο ορίσματα, πρώτο όρισμα την Byte τιμή και δεύτερο την θέση του bit που θέλουμε να εξάγουμε (από το πρώτο όρισμα) για να συγκρίνουμε.
Πχ. Έχουμε τον αριθμό 5 ο οποίος στο δυαδικό έχει την τιμή 101
Αν δώσουμε number = ExtractBit(5,0) , η θέση 0 θα μας επιστρέψει το πρώτο bit το οποίο είναι 1 , άρα number = 1
Αν δώσουμε number = ExtractBit(5,1) , η θέση 1 θα μας επιστρέψει το δεύτερο bit το οποίο είναι 0 , άρα number =0

Η ReplaceBit παρόμοια με την ExtractBit δέχετε 3 ορίσματα, τα δύο πρώτα έχουν την ίδια λογική με την ExtractBit απλά στο τρίτο όρισμα δίνουμε την τιμή που θα αντικαταστήσει το συγκεκριμένο bit.
Πχ. Έχουμε πάλι τον αριθμο 5 ο οποίος στο δυαδικό έχει την τιμή 101
Αν δώσουμε number=ReplaceBit(5,0,0), η θέση 0 της δυαδικής τιμής 5 είναι 1 και με το τρίτο όρισμα το κάνουμε 0, άρα το number θα έχει νέα τιμή 4 (σε δυαδικό 100)
Αν δώσουμε number=ReplaceBit(5,0,1), η θέση 0 της δυαδικής τιμής 5 είναι 1 και με το τρίτο όρισμα το κάνουμε 1, άρα το number θα παραμένει 5 (σε δυαδικό 101)

```
def ExtractBit(byteRead , postition):
    '''Extract the bit at the 'position' from the
'byteRead'.'''
    value=(byteRead & (1 << postition)) >> postition
    return value

def ReplaceBit(byteWrite , postition, value) :
    '''Replace the 'byteWrite' bit in the 'position' with
the bit 'value' . '''
    if value == 1 :
        byteWrite = byteWrite | (1 << postition)
    else :
        byteWrite = byteWrite & ~ (1 << postition)

    return byteWrite
```

7.2.E Encryption.py :

Όπως αναφέραμε πριν, για το κομμάτι της κρυπτογραφία χρησιμοποιήσαμε το PyCrypto library . Ο κάθε αλγόριθμος όμως έχει κάποιες ιδιαιτερότητες όπως ένα καθορισμένο μήκος κλειδιού (ανάλογα τον καθένα) και ότι όλα τα δεδομένα εισαγωγής για κρυπτογράφηση πρέπει να είναι τμήματα του 8 (blocks). Για την απλούστευση της χρησιμοποίησης από τον χρήστη έχουμε κάνει αυτόματες μετατροπές και στο μέγεθος του μηνύματος για εισαγωγή κρυπτογράφησης αλλά και στο μέγεθος του κλειδιού (ποτέ κενό string για κλειδί) που απαιτεί κάθε αλγόριθμος χωρίς να επηρεάζει την χρήση τους όπως βλέπετε παρακάτω.

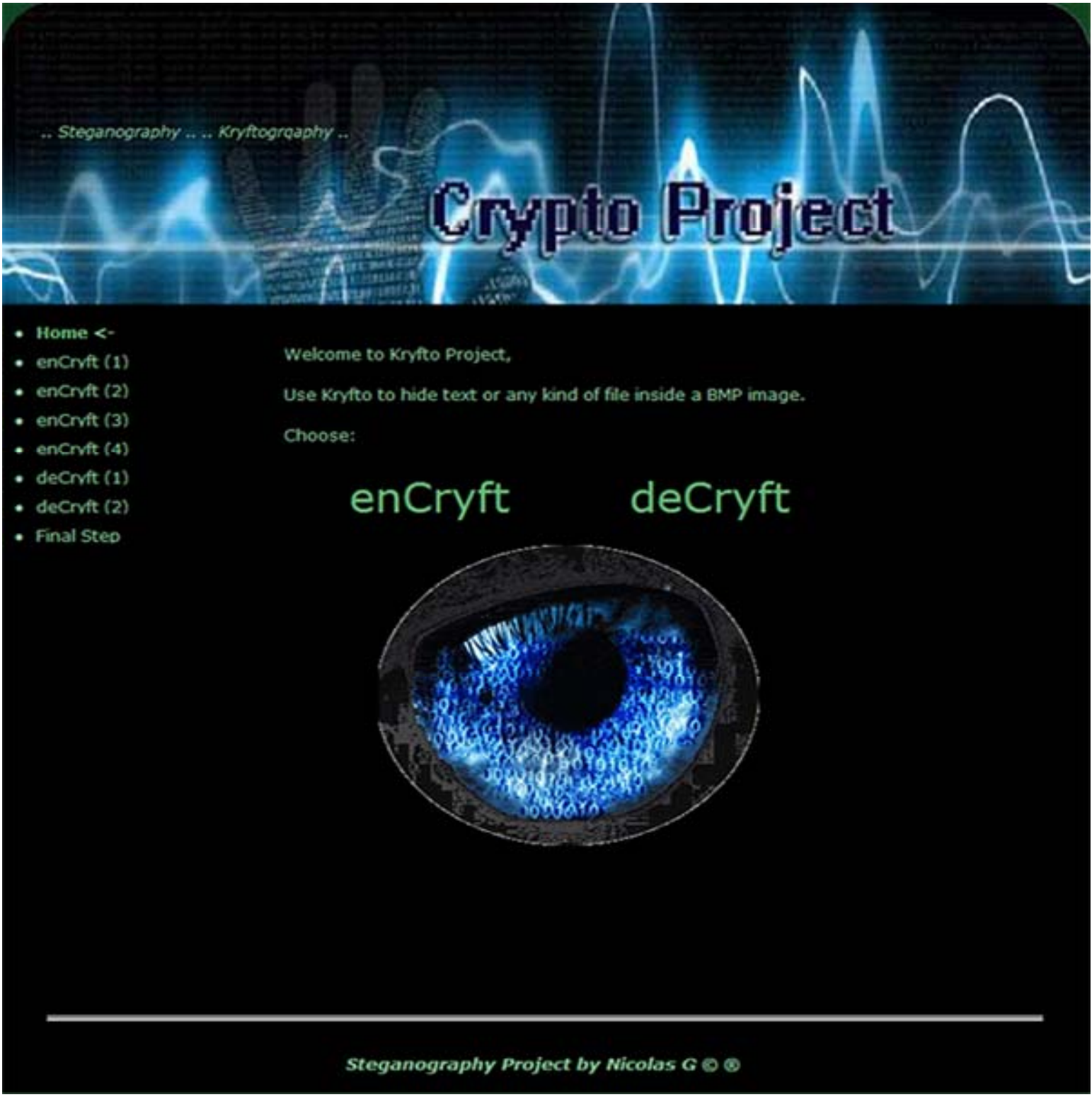
```
list=['DES3','Blowfish','IDEA','RC5','ARC2','CAST',’AES’]

def
encryption(_cryptoAlgorithm='Blowfish',plainText='Plain
info here..',key='Key or Passphrase'):

    #Key cannot be the null string
    if key == None:
        key = ' '
    if len(key) == 0 :
        key = ' '

    import ezPyCrypto

    if _cryptoAlgorithm == 'DES3': # Key must be either 16
or 24 bytes long...

        if len(key) < 16 :
            key = key + (16 - len(key) ) * ' '
        elif len(key)> 16 and len(key) < 24:
            key = key + ( 24 - len(key)) * ' '
        elif len(key) > 24:
            key = key[:24]

        obj = ezPyCrypto.DES3.new(key)

    if _cryptoAlgorithm == 'Blowfish':
        obj = ezPyCrypto.Blowfish.new(key)

    if _cryptoAlgorithm == 'IDEA': # Key must be exactly
16 bytes long

        if len(key) < 16 :
            key = key + ( 16 - len(key) ) * ' '
        elif len(key) > 16 :
            key = key[:16]

        obj = ezPyCrypto.IDEA.new(key)

    if _cryptoAlgorithm == 'RC5':
```

```
        obj = ezPyCrypto.RC5.new(key)

    if _cryptoAlgorithm == 'ARC2':
        obj = ezPyCrypto.ARC2.new(key)

    if _cryptoAlgorithm == 'CAST': # Key must be at least
5 bytes long and maximum 16

        if len(key) < 5 :
            key = key + ( 5 - len(key) ) * ' '
        elif len(key) > 16 :
            key = key[:16]

        obj = ezPyCrypto.CAST.new(key)

    #Plain text is always same lenght with cipher text

    plainText = plainText + ( ((len(plainText)/8)+1)*8 -
len(plainText) )*' '
    cipher=obj.encrypt(plainText)

    return cipher

def decryption(_cryptoAlgorithm,cipher,key):

    import ezPyCrypto

    if _cryptoAlgorithm == 'DES3': # Key must be either 16
or 24 bytes long...

        if len(key) < 16 :
            key = key + (16 - len(key) ) * ' '
        elif len(key)> 16 and len(key) < 24:
            key = key + ( 24 - len(key)) * ' '
        elif len(key) > 24:
            key = key[:24]

        obj = ezPyCrypto.DES3.new(key)

    if _cryptoAlgorithm == 'Blowfish':
        obj = ezPyCrypto.Blowfish.new(key)

    if _cryptoAlgorithm == 'IDEA': # Key must be exactly
16 bytes long

        if len(key) < 16 :
            key = key + ( 16 - len(key) ) * ' '
        elif len(key) > 16 :
            key = key[:16]

        obj = ezPyCrypto.IDEA.new(key)
```

```
    if _cryptoAlgorithm == 'RC5':
        obj = ezPyCrypto.RC5.new(key)

    if _cryptoAlgorithm == 'ARC2':
        obj = ezPyCrypto.ARC2.new(key)

    if _cryptoAlgorithm == 'CAST': # Key must be at least
5 bytes long and maximum 16

        if len(key) < 5 :
            key = key + ( 5 - len(key) ) * ' '
        elif len(key) > 16 :
            key = key[:16]

        obj = ezPyCrypto.CAST.new(key)

    plainText=obj.decrypt(cipher)

    return plainText
```

7.3 Το Kryfto στην πράξη:

Ξεκινώντας ο επισκέπτης επιλέγει την διαδικασία την οποία θέλει να ακολουθήσει,να στεγανογραφήσει ή να αποστεγανογραφήσει κάποια πληροφορία. Αυτές οι 2 διαδικασίες έχουν μετονομαστεί σε enCryft και deCrfyt (παρομοίως από την κρυπτολογία encrypt και decrypt αντίστοιχα) :



enCryft step 1:

Επιλέγοντας να στεγανογραφήσουμε (enCryft) το πρώτο μας βήμα είναι να κάνουμε upload στον server το αρχείο που θα χρησιμοποιήσουμε σαν κάλυμμα η αλλιώς Cover file , για να κρύψουμε την πληροφορία που θέλουμε μέσα του. (Υπενθυμίζουμε ότι το αρχείο αυτό έχει περιορισμό στο ότι πρέπει να είναι BMP 24 bit)

enCryft step 2:

Στο δεύτερο βήμα στεγανογραφίας επιλέγουμε αν θα κάνουμε upload κάποιο αρχείο στον server για να κρύψουμε μέσα στο Cover ή μπορούμε να χρησιμοποιήσουμε τον Web editor online για να γράψουμε το κείμενο που θα κρύψουμε.

enCryft step 3:

Εφόσον έχουμε επιλέξει να κάνουμε upload το μυστικό αρχείο που θέλουμε να κρύψουμε, η διεπαφή μας καθοδηγεί στο να επιλέξουμε το αρχείο που θα φορτώσομε για να κρύψουμε. Αυτό το αρχείο μπορεί να είναι σε οποιαδήποτε μορφής αρκεί να μπορεί να χωράει στο Cover που θα το έχει ενσωματωμένο.

enCryft step 3β:

Στην άλλη περίπτωση που επιλέξαμε να γράψουμε το κείμενο on-line μέσω web θα βρεθούμε σε μία σελίδα με φορτωμένο τον text editor FCKEditor , έναν πολύ δυνατού επεξεργαστή κειμένου ανάλογο τον επαγγελματικών που χρησιμοποιούμε σε εγκατεστημένους υπολογιστές όπως το Word, OpenOffice κτλ.

enCryft step 4:

Στο τελευταίο βήμα επιλέγουμε σε ποιο βαθμό θα γίνει η κωδικοποίηση , δηλαδή μέχρι ποιο **LSB** θα επέμβουμε για να αλλάξουμε την τιμή του και αν θα χρησιμοποιηθεί κάποιος αλγόριθμος κρυπτογραφίας για περισσότερη ασφάλεια. Αν θέλουμε να κρυπτογραφήσουμε στα δεδομένα πριν την στεγανογράφηση , επιλέγουμε έναν από τους αλγόριθμους που είναι διαθέσιμοι παρακάτω. Αφού έχουμε κάνει τις επιλογές μας μπορούμε να καθορίσουμε και το password που θα χρησιμοποιήσουμε για την κωδικοποίηση ώστε να έχουμε περισσότερη ασφάλεια. Το password αυτό πρέπει να μεριμνήσουμε να το γνωρίζει μόνο αυτός που θα κληθεί αργότερα να κάνει την αντιστροφή διαδικασία της αποστεγανογράφησης αλλιώς δεν θα είναι δυνατόν να εξάγει την κρυμμένη πληροφορία με επιτυχία .

enCryft Final step :

Τέλος όταν έχουμε ακολουθήσουμε σωστά όλα τα βήματα , περιμένουμε κάποιο χρόνο για την κωδικοποίηση . Αφού ολοκληρωθεί η διαδικασία με επιτυχία , μας εμφανίζει ανάλογο μήνυμα με ένα link που δείχνει στο καινούριο (Στεγανογραφημένο) αρχείο που δημιουργήθηκε με την default ονομασία Steganographed.bmp. Εφόσον το κάνουμε download επιλέγουμε την ονομασία που θέλουμε να το σώσουμε και έχουμε μία "αθώα"εικόνα που έχει κρυμμένα τα δεδομένα επιλέξαμε .

deCryft step 1:

Η αντίστροφη διαδικασία της αποστεγανογράφησης (deCryft) είναι πιο απλή λόγου ότι φορτώνομαι μόνο ένα αρχείο που έχει την κρυμμένη πληροφορία (Cover file) και διότι δεν χεριάζετε να ορίσουμε πολλές παραμέτρους παρά μόνον τον κωδικό κρυπτογράφησης αν αυτό δόθηκε στην διαδικασία κρυπτογραφήσεις. Η κρυμμένη πληροφορία , ο βαθμός κωδικοποίησης (LSB) καθώς και ο αλγόριθμος κρυπτογράφησης (αν χρησιμοποιήθηκε) έχει αποθηκευτεί μαζί με τα δεδομένα που κρύφτηκαν σαν Header για την απλούστευση της αντίστροφης διαδικασίας.

Πρώτα επιλέγουμε το αρχείο (Cover) που θα κάνουμε upload στον server και περιέχει τα κρυμμένα δεδομένα :

deCryft step 2:

Πληκτρολογούμε το συνθηματικό (αν χρησιμοποιήθηκε κάποιο) που πρέπει να το ξέρουμε από το άτομο που μας έστειλε την εικόνα (Steganographed file), σε περίπτωση που δεν χρησιμοποιήθηκε συνθηματικό αφήνουμε το πεδίο Password κενό:

deCryft Final step :

Τέλος αφού έχουμε ακολουθήσουμε σωστά τα βήματα , περιμένουμε κάποιο χρόνο για την αποκωδικοποίηση . Αφού ολοκληρωθεί η διαδικασία με επιτυχία , μας εμφανίζει ανάλογο μήνυμα με ένα link που δείχνει στο κρυμμένο αρχείο που ανακτήσαμε από τα στεγανογραφημένα δεδομένα που υπήρχαν μέσα στην εικόνα. Εφόσον το κάνουμε download επιλέγουμε την ονομασία που θέλουμε να το σώσουμε :

8. Παράρτημα Κρυπτογραφία:

Όπως είδαμε πριν την κωδικοποίηση έχουμε επιλογή να κρυπτογραφήσουμε τα δεδομένα για περισσότερη ασφάλεια. Από την λίστα με τους διαθέσιμους αλγορίθμους που υποστηρίζει το Kryfto , θα αναλύσουμε μερικές τεχνικές λεπτομέρειες για τους 3 πιο διαδεδομένους, AES – DES3 – Blowfish .

Block Ciphers

Cipher	Key Size/Block Size
ARC2	Variable/8 bytes
Blowfish	Variable/8 bytes
CAST	Variable/8 bytes
DES	8 bytes/8 bytes
DES3	(Triple DES) 16 bytes/8 bytes
IDEA	16 bytes/8 bytes
RC5	Variable/8 bytes

Οι Block ciphers κρυπτογραφούν multibyte δεδομένα με συγκεκριμένο σταθερό μέγεθος (συνήθως 8 ή 16 bytes μέγεθος) , και μπορούν να εφαρμοστούν με διάφορες μορφές. Η μορφές που υποστηρίζονται με τις συγκεκριμένες προδιαγραφές είναι :

```
Number    Constant      Description

1         MODE_ECB      Electronic Code Book
2         MODE_CBC      Cipher Block Chaining
3         MODE_CFB      Cipher Feedback
5         MODE_OFB      Output Feedback
6         MODE_CTR      Counter
```

Όλοι οι αλγόριθμοι έχουν κοινό στοιχείο το ότι τα δεδομένα εισόδου για κρυπτογράφηση πρέπει να είναι τμήματα τον 8 blocks και το κάθε ένα έχει δικό του περιορισμό για το μέγεθος του κλειδιού. Όλες οι συναρτήσεις κρυπτογράφησης έχουν ρυθμιστεί να τρέχουν σε CBC mode αφού είναι πιο σύγχρονος και παράγει ασφαλέστερα αποτελέσματα. Για την απλούστευση στην χρήση τους δημιουργήθηκαν ειδικές διεπαφές που χειρίζονται την κάθε περίπτωση ανάλογα, χωρίς να μπλέκει τον απλό χρήστη με πολύπλοκες διαδικασίες. Όταν το κλειδί απαιτεί κάποιες προδιαγραφές, φροντίσαμε ώστε αν το κλειδί που δόθηκε είναι μικρότερο από το απαιτούμενο, να προσθέτουμε κενά διαστήματα μέσα στην συμβολοσειρά. Στην διαδικασία της αποκρυπτογράφησης ακολουθούνται οι ίδιες προδιεργασίες ώστε να έχουμε ίδιο κλειδί με αυτό που χρησιμοποιήθηκε στην κρυπτογράφηση.

AES (Advanced Encryption Standard)

Το ακρωνύμιο AES προέρχεται από την φράση *Advanced Encryption Standard*. Είναι ένας block cipher που προορίζεται να γίνει τυποποίηση του FIPS και να αντικαταστήσει τον DES. Ο DES βρίσκεται ήδη πολλά χρόνια σε χρήση και από το 1998 το NIST δεν τον ανανεώνει.

Ο αλγόριθμος αυτός μας περιορίζει στο ότι το κλειδί πρέπει να είναι 16, 24 ή 32 bytes. Για τον λόγο αυτό προσθέτουμε διαστήματα ανάλογα το τρέχον μήκος. Δίνεται ο κώδικας για την συγκεκριμένη ενέργεια παρακάτω :

```
if _cryptoAlgorithm == 'DES3': # Key must be either 16 or 24
bytes long...

        if len(key) < 16 :
            key = key + (16 - len(key) ) * ' '
        elif len(key)> 16 and len(key) < 24:
            key = key + ( 24 - len(key)) * ' '
        elif len(key) > 24:
            key = key[:24]

        from Crypto.Cipher import DES3
        obj = DES3.new(key , DES3.MODE_CBC)
```

DES3 η αλλιώς Triple-DES

Είναι μια παραλλαγή του DES όπου το μήνυμα κρυπτογραφείται και αποκρυπτογραφείται διαδοχικά με διαφορετικά κλειδιά για την ενίσχυση του βασικού αλγόριθμου.

Παρομοίως με τον AES , ο DES3 μας περιορίζει στο ότι το κλειδί πρέπει να είναι 16 ή 24. Για τον λόγο αυτό προσθέτουμε διαστήματα ανάλογα το τρέχον μήκος. Δίνεται ο κώδικας για την συγκεκριμένη ενέργεια παρακάτω :

```
if _cryptoAlgorithm == 'DES3': # Key must be either 16 or
24 bytes long...

        if len(key) < 16 :
            key = key + (16 - len(key) ) * ' '
        elif len(key)> 16 and len(key) < 24:
            key = key + ( 24 - len(key)) * ' '
        elif len(key) > 24:
            key = key[:24]

        from Crypto.Cipher import DES3
        obj = DES3.new(key , DES3.MODE_CBC)
```

Blowfish

Ο Blowfish είναι ένας block cipher που κατασκευάστηκε από τον Schneier. Είναι ένας Feistel cipher με μέγεθος block 64 bits και μεταβλητό μήκος κλειδιού, με μέγιστο μήκος 448 bits. Όλες οι διεργασίες βασίζονται σε X-OR πράξεις και προσθέσεις λέξεων των 32 bits. Από το κλειδί παράγεται πίνακας με τα subkeys που χρησιμοποιούνται σε κάθε γύρο επανάληψης της κρυπτογράφησης. Έχει σχεδιασθεί για 32-bit μηχανές και είναι σημαντικά ταχύτερος από τον DES. Παρ' όλες τις αδυναμίες που έχουν ανακαλυφθεί καθ' όλη την διάρκεια της ύπαρξης του, θεωρείται ακόμα ασφαλής αλγόριθμος.

Ο Blowfish από την άλλη δεν έχουμε κανέναν περιορισμό στο ελάχιστο μέγεθος κλειδιού και απλός ρυθμίζουμε να τρέχει σε CBC mode :

```
if _cryptoAlgorithm == 'Blowfish':
        from Crypto.Cipher import Blowfish
        obj = Blowfish.new(key , Blowfish.MODE_CBC)
```

DES3, Blowfish , AES … κληρονομιά του DES

Το πρότυπο κρυπτογράφησης δεδομένων Data Encryption Standard ή DES ήταν η εφεύρεση της Εθνικής Υπηρεσίας Προτύπων των ΗΠΑ (NIST) στα μέσα της δεκαετίας του 1970. Ο πρώτος σύγχρονος, δημόσιος, ελεύθερα διαθέσιμος αλγόριθμος κρυπτογράφησης. Για περισσότερο από δύο δεκαετίες, το DES αποτελούσε την κινητήριο δύναμη της εμπορικής κρυπτογραφίας.

Με την πάροδο των δεκαετιών, το DES έχει χρησιμοποιηθεί για να προστατεύει τα πάντα· από βάσεις δεδομένων σε Η/Υ μέχρι συνδέσεις επικοινωνιών μεταξύ των μηχανημάτων ανάληψης χρημάτων και των τραπεζών, μεταδόσεις πληροφοριών μεταξύ οχημάτων και τμημάτων της αστυνομίας. Όποιος κι αν είσαι, είμαι σίγουρος ότι πολλές φορές στη ζωή σου, η ασφάλεια των δεδομένων σου προστατευόταν από το DES.

Τον Οκτώβρη του 2004, η πρώην Εθνική Υπηρεσία Προτύπων – η υπηρεσία τώρα ονομάζεται Εθνικό Ιστιτούτο Προτύπων και Τεχνολογίας ή NIST – πρότεινε την απόσυρση του DES ως προτύπου κρυπτογράφησης, σηματοδοτώντας το πέρας του πιο σημαντικού τεχνολογικού προτύπου της ομοσπονδιακής κυβέρνησης.

Σήμερα, η κρυπτογραφία αποτελεί ένα από τα πλέον βασικά εργαλεία της ασφάλειας των Η/Υ, ενώ πριν από 30 χρόνια σχεδόν δεν υπήρχε ως ακαδημαϊκή επιστήμη. Την εποχή που το Διαδίκτυο ήταν σε στάδιο εξερεύνησης, η κρυπτογραφία δεν αποτελούσε ακόμη αναγνωρισμένο κλάδο των μαθηματικών. Οι μυστικοί κώδικες ήταν πάντα αξιοπερίεργοι, αλλά ήταν περιορισμένοι σε απλούς κώδικες βασισμένοι στην αλφάβητο. Κατά τη διάρκεια του Β' Παγκοσμίου Πολέμου, στα μυστικά κυβερνητικά εργαστήρια, η κρυπτογραφία εισήχθη στην εποχή των Η/Υ και αποτέλεσε τμήμα των μαθηματικών. Εφόσον, όμως, δεν υπήρχαν καθηγητές για να την διδάξουν και συνέδρια για να συζητηθεί, η όλη έρευνα γύρω από την

κρυπτογραφία στις ΗΠΑ διεξήχθη από την Υπηρεσία Εθνικής Ασφάλειας των ΗΠΑ (NSA).

Και μετά εμφανίστηκε το DES.

Στις αρχές της δεκαετίας του 1970, ήταν μία καινοτομική ιδέα. Η Εθνική Υπηρεσία Προτύπων αποφάσισε ότι θα πρέπει να υπάρχει ένα ελεύθερο πρότυπο κρυπτογράφησης. Επειδή η υπηρεσία δεν ήθελε να είναι στρατιωτικό, ζήτησε αλγόριθμους κρυπτογράφησης από το κοινό. Έλαβε μόνο μία σοβαρή απάντηση – το πρότυπο κρυπτογράφησης δεδομένων – από τα εργαστήρια της εταιρείας IBM. Το 1976, το DES αποτέλεσε τον πρότυπο αλγόριθμο κρυπτογράφησης της κυβέρνησης για «ευαίσθητες αλλά μη απόρρητες» πληροφορίες. Σε αυτές περιλαμβάνονται οι προσωπικές, οικονομικές και λογιστικές πληροφορίες. Και βέβαια, επειδή δεν υπήρχε τίποτε άλλο, οι εταιρείες άρχισαν να χρησιμοποιούν το DES όποτε χρειάζονταν έναν αλγόριθμο κρυπτογράφησης. Βέβαια, υπήρχαν άτομα που δεν πίστευαν ότι το DES ήταν ασφαλής.

Όταν η εταιρεία IBM κατέθεσε το DES ως πρότυπο, κανείς εκτός της NSA δεν διέθετε τις ειδικές γνώσεις για να τον αναλύσει. Η NSA έκανε δύο αλλαγές στο DES · τροποποίησε τον αλγόριθμο και μείωσε το μέγεθος του κλειδιού περισσότερο από το μισό.

Η δύναμη ενός αλγόριθμου βασίζεται σε δύο πράγματα: πόσο αναπτυγμένα είναι τα μαθηματικά και το μέγεθος του κλειδιού. Ένας σίγουρος τρόπος για να σπάσει ένας αλγόριθμος είναι να δοκιμαστεί κάθε δυνατό κλειδί. Οι σύγχρονοι αλγόριθμοι διαθέτουν μεγάλα κλειδιά, οπότε κάτι τέτοιο είναι αδύνατον · ακόμη κι αν κατασκευάσουμε έναν Η/Υ από όλα τα άτομα του πλανήτη και τον αφήσουμε να λειτουργήσει για εκατομμύρια χρόνια, δεν θα τα καταφέρουμε. Οπότε, οι ειδικοί της κρυπτογραφίας αναζητούν συντομεύσεις. Αν τα μαθηματικά είναι αδύναμα, υπάρχει περίπτωση να βρεθεί πιο γρήγορα το κλειδί: «σπάζοντας» τον αλγόριθμο.

Οι αλλαγές της NSA προκάλεσαν τη διαμαρτυρία των ελάχιστων ατόμων που παρακολουθούσαν τις εξελίξεις, τόσο σχετικά με το «αόρατο χέρι» της NSA – οι τροποποιήσεις δεν κοινοποιήθηκαν και δεν δόθηκε λογική εξήγηση για το τελικό σχέδιο – όσο και για το μικρό μήκος του κλειδιού.

Η διαμαρτυρία έδωσε το έναυσμα για έρευνα. Δεν πρόκειται για υπερβολή όταν λέμε ότι η δημοσίευση του DES δημιούργησε τη σύγχρονη ακαδημαϊκή επιστήμη της κρυπτογραφίας. Οι πρώτοι ακαδημαϊκοί ειδικοί της κρυπτογραφίας ξεκίνησαν τη σταδιοδρομία τους προσπαθώντας να σπάσουν το DES, ή τουλάχιστον προσπαθώντας να καταλάβουν την τροποποίηση του NSA. Και σχεδόν όλοι οι αλγόριθμοι της κρυπτογραφίας – ιδιαίτερα η κρυπτογραφία με δημόσιο κλειδί – έχουν την απαρχή τους στο DES. Ανακοινώσεις που αναλύουν διάφορες πλευρές του DES δημοσιεύονται μέχρι και σήμερα.

Γύρω στα μέσα της δεκαετίας του 1990, άρχισε να αποτελεί πραγματικότητα η πεποίθηση ότι το NSA μπορεί να σπάσει το DES αν δοκιμάσει κάθε δυνατό κλειδί. Έγινε παρουσίαση της δυνατότητας αυτής το 1998, όταν κατασκευάστηκε ένα μηχάνημα αξίας $220,000 που θα μπορούσε να βρει, εξαντλώντας όλους τους πιθανούς συνδυασμούς, ένα κλειδί DES μέσα σε μερικές μέρες. Το 1985, η ακαδημαϊκή κοινότητα πρότεινε μία μεταβλητή DES χρησιμοποιώντας τα ίδια μαθηματικά, αλλά μεγαλύτερο κλειδί, που ονομάστηκε triple-DES. Η μεταβλητή

αυτή χρησιμοποιείται σε πολύ περισσότερες ασφαλείς εφαρμογές αντί του DES εδώ και χρόνια, αλλά είχε έρθει η ώρα για ένα νέο πρότυπο. Το 1997, το NIST ζήτησε έναν αλγόριθμο που θα αντικαταστούσε το DES.

Η διαδικασία παρουσιάζει την απόλυτη μεταμόρφωση της κρυπτογραφίας από μία μυστικοπαθή τεχνολογία του NSA σε μία παγκόσμια δημόσια τεχνολογία. Το NIST για ακόμη μία φορά ζήτησε αλγόριθμους από το κοινό, αλλά αυτή τη φορά η υπηρεσία έλαβε 15 αιτήσεις από 10 χώρες. Ο αλγόριθμος, Blowfish, ήταν ένας από αυτούς. Και μετά από δύο χρόνια ανάλυσης και συζήτησης, το NIST επέλεξε έναν βελγικό αλγόριθμο, το Rijndael, για να αποτελέσει το πρότυπο κρυπτογράφησης δεδομένων AES (Advanced Encryption Standard) στο τέλος μιας πολύ μακροχρόνιας και σύνθετης διαδικασίας αξιολόγησης. Κατά την διάρκεια της αξιολόγησης, ο NIST θεώρησε το Rijndael ως τον καλύτερο γενικό αλγόριθμο για το AES. Το Rijndael εμφανίζει να είναι με συνέπεια πολύ καλός εκτελεστής και στο υλικό και στο λογισμικό. Ο βασικός χρόνος οργάνωσης του είναι άριστος, και η βασική ευκινησία του είναι καλή. Οι απαιτήσεις μνήμης του Rijndael, που είναι πολύ χαμηλές , τον καθιστούν ακριβώς κατάλληλο για μη εξουσιοδοτημένη πρόσβαση σε πληροφορίες.

Ο κόσμος της κρυπτογραφίας ήταν εντελώς διαφορετικός πριν από 30 χρόνια απ’ ότι είναι σήμερα. Γνωρίζουμε περισσότερα πράγματα για την κρυπτογραφία και διαθέτουμε περισσότερους αλγόριθμους μεταξύ των οποίων μπορούμε να επιλέξουμε. Το AES δεν θα αποτελέσει ένα πανταχού παρόν πρότυπο όπως ήταν το DES. Ωστόσο, αρχίζει να διεισδύει στους χώρους των προϊόντων ασφαλείας των τραπεζών, στα πρωτόκολλα ασφαλείας του Διαδικτύου, ακόμη και στα αυτόματα συστήματα ψηφοφορίας. Ένα πρότυπο του NIST αποτελεί την επίσημη έγκριση της ποιότητας και ασφάλειας και οι πωλητές το αναγνωρίζουν.

Οπότε, πόσο καλή είναι η NSA στην κρυπτογραφία; Πάντως είναι καλύτερη από τον ακαδημαϊκό κόσμο. Διαθέτει περισσότερους μαθηματικούς που ασχολούνται με τα προβλήματα και για περισσότερο χρονικό διάστημα και έχουν πρόσβαση σε οτιδήποτε έχει εκδοθεί από τον ακαδημαϊκό κόσμο, ενώ εκείνοι δεν χρειάζεται να δημοσιεύουν τα αποτελέσματά τους στο κοινό. Είναι ένα έτος μπροστά από την υψηλή τεχνολογία; Πέντε χρόνια; Μία δεκαετία; Κανείς δεν γνωρίζει.

Η ακαδημαϊκή κοινότητα αφιέρωσε είκοσι χρόνια για να ανακαλύψει ότι οι τροποποιήσεις της NSA ουσιαστικά βελτίωσαν την ασφάλεια του DES. Αυτό σημαίνει ότι τη δεκαετία του 1970, το NSA ήταν κατά δύο δεκαετίες μπροστά από την υψηλή τεχνολογία.

Σήμερα, η NSA συνεχίζει να είναι η πιο έξυπνη, αλλά και οι περισσότεροι από εμάς έχουμε αρχίσει και κερδίζουμε έδαφος. Το 1999, η ακαδημαϊκή κοινότητα ανακάλυψε μία αδυναμία σε κάποιον άλλο αλγόριθμο της NSA τον SHA-1 που παρουσίασε αδυναμίες, για τις οποίες πιστεύεται ότι η NSA δεν γνώριζε τίποτε.

Ενδεχομένως τώρα να είμαστε κατά δύο χρόνια πίσω.

9. Αναφορές – Παραπομπές (references)

[A01] Al-Qaeda Said to be Using Stegged Porn T. C. Greene
12 05 2003 http://www.theregister.co.uk/content/6/30654.html

[A02] 9/11 Plot Hidden in E-Porn N. Lathem
09 05 2003 http://www.nypost.com/news/worldnews/57502.htm

[A03] Coded Pornography, WTC Pictures Found
on Terror Cell Computers A. Salomon
08 05 2003 http://more.abcnews.go.com/sections/us/dailynews/I
TeamInsider_030508.html

[A04] Al-Qaeda Poised to Strike Hard Via the
Internet T. C. Greene
07 10 2002 http://www.theregister.co.uk/content/archive/26134.
Html

[A05] Spy Games: Decoding Osama's Secrets at 16 M. Kumar
04 10 2002 http://www.hindustantimes.com/2002/Oct/04/674_7
7069,00310001.htm

[A06] Web Site With Area Ties Stirs Terrorism
Concern J. Lynott
20 07 2002 http://www.timesleader.com/mld/timesleader/news/3698733.htm

[A07] Hunt for Hidden Web Messages Goes On W. Knight
12 07 2002 http://www.newscientist.com/news/news.jsp?id=ns9
9992543

[A08] Hidden Messages: Any There There?" F. Manjoo
08 11 2001 http://www.wired.com/news/technology/0,1282,482
35,00.html

[A09] Watching the Web for Wicked Messages P. Eng
11 10 2001 http://more.abcnews.go.com/sections/scitech/DailyN
ews/webwatch011011.html

[A10] Internet Link in Terror Probe (not provided)
10 10 2001 http://news.bbc.co.uk/2/hi/uk_news/scotland/159090
8.stm

[A11] France Terror Code 'Breakthrough' (not provided)
05 10 2001 http://news.bbc.co.uk/2/hi/europe/1580593.stm

[A12] A Secret Language B. Ross
04 10 2001 http://more.abcnews.go.com/sections/primetime/dail
ynews/primetime_011004_steganography.html