Πανεπιστήμιο Πειραιώς
Τμήμα Πληροφορικής

Κρυπτανάλυση Αλγορίθμων και Εφαρμογές της Κρυπτογραφίας σε Κακόβουλο Λογισμικό

Διδακτορική Διατριβή

Κωνσταντίνου Ε. Πατσάκη

Πειραιάς Σεπτέμβριος 2008

Συμβουλευτική επιτροπή

Επιβλέπων:
Νικόλαος Αλεξανδρής
Καθηγητής Πανεπιστημίου Πειραιώς

Μέλη:
Ευάγγελος Φούντας
Καθηγητής Πανεπιστημίου Πειραιώς

Δέσποινα Πολέμη
Επίκουρος Καθηγήτρια Πανεπιστημίου Πειραιώς

Πανεπιστήμιο Πειραιώς
Τμήμα Πληροφορικής

Διατριβή για την απόκτηση Διδακτορικού Διπλώματος

του Τμήματος Πληροφορικής

«Κρυπτανάλυση Αλγόριθμων και Εφαρμογές της Κρυπτογραφίας σε Κακόβουλο Λογισμικό»

Εξεταστική επιτροπή:

Νικόλαος Αλεξανδρής
Καθηγητής Πανεπιστημίου Πειραιώς

Στέφανος Γκρίτζαλης
Καθηγητής Πανεπιστημίου Αιγαίου

Χρήστος Δουληγέρης
Καθηγητής Πανεπιστημίου Πειραιώς

Γεώργιος Φλέσσας
Καθηγητής Πανεπιστημίου Αιγαίου

Βασίλειος Χρυσικόπουλος
Καθηγητής Ιονίου Πανεπιστημίου

Ευάγγελος Φούντας
Καθηγητής Πανεπιστημίου Πειραιώς

Δέσποινα Πολέμη

Επίκουρος Καθηγήτρια | Πανεπιστημίου Πειραιώς

.....

Κωνσταντίνος Ε. Πατσάκης

Μαθηματικός Ε.Κ.Π.Α, MSc Royal Holloway

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Πανεπιστημίου Πειραιώς.

Πρόλογος

Η κρυπτογραφία αποτελεί ένα σημαντικό χώρο έρευνας και έχει προσελκύσει το ενδιαφέρον πολλών ερευνητών τις τελευταίες δεκαετίες. Το γεγονός ότι μπορεί να δώσει λύση στο πρόβλημα της ιδιωτικότητας των δεδομένων ενός χρήστη, ουσιαστικά την θέτει ως ένα από τους σημαντικότερους άξονες τόσο της ασφάλειας των υπολογιστικών συστημάτων, όσο και των τηλεπικοινωνιών

Η πρόκληση που αντιμετωπίζεται στην παρούσα διατριβή είναι η απάντηση δυο βασικών ερωτημάτων για την κρυπτογραφία. Το πρώτο ερώτημα είναι αν και κατά πόσο κάποιοι από τους γνωστούς αλγόριθμους κρυπτογράφησης είναι ασφαλείς. Στο πλαίσιο αυτής της διατριβής αναπτύχθηκε ένας αλγόριθμος αλγεβρικής επίθεσης σε ένα από τους γνωστότερους αλγόριθμους, τον Data Encrypotion Standard ενώ επιβεβαιώθηκε εκ νέου η ασφάλεια ενός άλλου, του Advanced Encrypotion Standard. Στη συνέχεια, αναζητήθηκαν τρόποι για την ασφαλέστερη παραγωγή δημοσίων κλειδιών.

Το δεύτερο ερώτημα είναι ποια προβλήματα έχουν ή μπορούν να προκληθούν από τη γνώση της κρυπτογραφίας. Αναδιατυπώνοντας το ερώτημα, ποια μπορεί να είναι η κακόβουλη χρήση της κρυπτογραφίας. Στο συγκεκριμένο ερώτημα προσπαθήσαμε να εξετάσουμε τις προεκτάσεις της κρυπτογραφίας στο κακόβουλο λογισμικό. Έτσι ερευνήθηκαν νέες

μορφές ιών υπολογιστών, οι οποίες κάνουν χρήση κρυπτογραφικών τεχνικών, μελετήθηκαν νέα μέτρα προστασίας για το λογισμικό προστασίας.

Ευχαριστίες

Θα ήθελα πρώτα απ' όλα να ευχαριστήσω θερμά τα μέλη της Τριμελούς Επιτροπής που αποτελείται από τον καθηγητή Νικόλαο Αλεξανδρή (επιβλέποντα), τον καθηγητή Ευάγγελο Φούντα και την επίκουρο καθηγήτρια Δέσποινα Πολέμη για τη συνεχή υποστήριξη που μου παρείχαν κατά τη διάρκεια της εκπόνησης της διατριβής στο Πανεπιστήμιο Πειραιως. Οι γνώσεις τους, οι συμβουλές τους και η συνεχής παρουσία τους δίπλα μου ήταν πολύτιμα στοιχεία για την εκπόνηση της διατριβής αυτής. Θα ήθελα να ευχαριστώ ακόμη τους καθηγητές Στέφανο Γκρίτζαλη, Χρήστο Δουληγέρη, Γεώργιο Φλέσσα και Βασίλειο Χρυσικόπουλο που δέχτηκαν να είναι μέλη της εξεταστικής επιτροπής μου.

Επίσης θα ήθελα να ευχαριστήσω τον καθηγητή μου στο Πανεπιστήμιο Αθηνών, Δημήτριο Δεριζιώτη, ο οποίος μου κέντρισε αρχικά το ενδιαφέρον για τον επιστημονικό χώρο τον οποίο πραγματεύεται η διατριβή και μου έδωσε τις αρχικές κατευθύνσεις.

Θα ήθελα τέλος να ευχαριστήσω τους γονείς μου για την υποστήριξη που μου παρείχαν αλλά και την υπομονή που έχουν όλα αυτά τα χρόνια και την οποία είχα πραγματικά ανάγκη.

Δημοσιεύσεις

Σε αυτές τις δημοσιεύσεις που παρουσιάζονται παρακάτω περιλαμβάνονται εργασίες που έχουν δημοσιευτεί σε διεθνή περιοδικά μετά από πλήρη κρίση, σε διεθνή βιβλία με συλλογές άρθρων μετά απο πλήρη κρίση, σε διεθνή συνέδρια μετά από πλήρη κρίση και σε διεθνή συνέδρια μετά από πλήρη κρίση. Οι εργασίες αυτές σχετίζονται με την έρευνα που διεξήχθη στα πλαίσια της παρούσης διατριβής.

Σε διεθνή Περιοδικά μετά από Πλήρη Κρίση

- Constantinos Patsakis, Foundas Evangellos, Christos Lytras, “Improved algorithms for the calculation of Fibonacci numbers”, Journal of Discrete Mathematical Sciences & Cryptography, Volume 11, Number 1, IOS Press, Taru Publishing, February 2008.
- Constantinos Patsakis, Evangellos Foundas, Gregory Chondrocoukis, “Lucas Permutations and some notes on Fibonacci permutations”, Journal of Discrete Mathematical Sciences & Cryptography, Volume 11, Number 2, IOS Press, Taru Publishing,April 2008.
- Patsakis Constantinos, Alexandris Nikolaos, "New malicious agents and SK virii", International Journal On Advances in Security, 2008 , Δεκτό προς δημοσίευση.

Σε Διεθνή Βιβλία με Συλλογές Άρθρων μετά απο Πλήρη Κρίση

- Κωνσταντίνος Πατσάκης, Νικόλαος Αλεξανδρής, “Multimedia Services in Intelligent Environments Advanced Tools and Methodologies” Series: Studies in Computational Intelligence vol 120, κεφάλαιο “Multimedia Information Security”, Editors Tsihrintzis, George A.; Jain, Lakhmi C. 2008, , VIII, pp ISBN: 978-3-540-78491-3
- Constantinos Patsakis, Nikolaos Alexandris “Histographic Steganographic System” New Directions in Intelligent Interactive Multimedia, Studies in Computational Intelligence , Vol. 142 , Studies in Computational Intelligence, Springer-Verlag.
- Constantinos Patsakis, Nikolaos Alexandris “Torrent worms”, Knowledge-Based Software Engineering, Frontiers in Artificial Intelligence and Applications, Vol. 180, IOS Press.

Σε Διεθνή Συνέδρια μετά από Πλήρη Κρίση

- Patsakis Constantinos, Alexandris Nikolaos, “An algebraic attack on DES”, Proceedings of OC'2005, Fourth International Workshop on Optimal Codes and Related Topics (Pamporovo, Bulgaria), 2005.
- Patsakis Constantinos, Alexandris Nikolaos, “Finding factorable spheres in [1,n]”, ACA 2006, 12th International Conference on Applications of Computer Algebra.

- Patsakis Constantinos, Alexandris Nikolaos,"New malicious agents and SK virii", International Multi-Conference on Computing in the Global Information Technology, ICCGI 2007, Guadeloupe, France.
- Patsakis Constantinos, Foundas Evangellos V. Karayanni, "On the generation of Fibonacci and Lucas Permutations", ENMA 2007 International Conference on Engineering and Mathematics, July 9-11, Bilbao, Spain.
- Patsakis Constantinos, Alexandris Nikolaos "Histographic Steganographic System", 1st International Symposium on Intelligent Interactive Multimedia Systems and Services, KES IIMSS 2008, University of Piraeus, 9-11 July 2008
- Patsakis Constantinos, Alexandris Nikolaos "Torrent worms" 8th Joint Conference on Knowledge - Based Software Engineering , JCKBSE 2008, University of Piraeus, 25-28 August 2008

Δομή της διατριβής

Η παρούσα διατριβή αποτελείται από επτά κεφάλαια και ένα παράρτημα. Στο πρώτο κεφάλαιο γίνεται μια σύντομη ιστορική αναδρομή των επιτευγμάτων της κρυπτογραφίας, της βασικής θεωρίας και εννοιών που πρόκειται να χρησιμοποιηθούν στη συνέχεια.

Το δεύτερο κεφάλαιο επικεντρώνεται σε ένα νέο είδος κρυπταναλυτικής επίθεσης για τον αλγόριθμο DES. Η επίθεση βασίζεται στην αλγεβρική προσέγγιση του αλγορίθμου και στοχεύει στην εύρεση του κλειδιού σε πάρα πολύ σύντομο χρόνο με όσο το δυνατό λιγότερα μέσα. Στο πλαίσιο της καλύτερης κατανόησης της επίθεσης, αναλύονται τα επιμέρους στοιχεία του αλγορίθμου.

Στο τρίτο κεφάλαιο γίνεται αρχικά η περιγραφή του αλγορίθμου AES και στη συνέχεια, εξετάζεται η ασφάλεια του αλγορίθμου απέναντι στην επίθεση που περιγράφηκε στο προηγούμενο κεφάλαιο και αναλύονται οι λόγοι για τους οποίους η εφαρμογή της στον αλγόριθμο AES δεν έχει κάποιο αποτέλεσμα επιβεβαιώνοντας την ασφάλειά του.

Στο τέταρτο κεφάλαιο ασχολούμαστε με την κρυπτογραφία δημοσίου κλειδιού και πιο συγκεκριμένα με τον αλγόριθμο RSA. Αναλύεται η σημαντικότητα ύπαρξης κάποιων συνθηκών ασφαλείας, αλλά και οι επιπτώσεις από την μη σωστή εφαρμογή τους. Επιπλέον, παρουσιάζονται τρεις αλγόριθμοι για τον υπολογισμό ακολουθιών Fibonacci και Lucas, οι

οποίοι επιταχύνουν την δημιουργία κλειδιών για αλγόριθμους δημοσίου κλειδιού όπως ο RSA,ο Rabin και ο ElGamal. Οι αλγόριθμοι αυτοί είναι οι πιο γρήγοροι που υπάρχουν αυτή τη στιγμή στο είδος τους και με την πιο χαμηλή πολυπλοκότητα.

Στο πέμπτο κεφάλαιο της διατριβής αναλύονται οι χρήσεις της κρυπτογραφίας από κακόβουλο λογισμικό. Εισάγονται νέες έννοιες όπως οι update-oriented κακόβουλοι πράκτορες, οι SK ιοί και τα torrent worms. Αποδεικνύεται η δυνατότητα ύπαρξής τους, οι επιπτώσεις τις οποίες μπορεί να έχουν, αλλά και πιθανές τεχνικές οι οποίες μπορει να χρησιμοποιηθούν από αυτούς τους αλγόριθμους. Παράλληλα παρατίθενται και τα νομικά προβλήματα τα οποία πιθανότατα να προκύψουν από την δημιουργία τους.

Ένας νέος αλγόριθμος, ο στεγανογραφικός αλγόριθμος HSS, περιγράφεται στο έκτο κεφάλαιο της διατριβής. Ο αλγόριθμος αυτός επιτυγχάνει την ενσωματώση μηνυμάτων σε εικόνες, τα οποία δεν γίνονται αντιληπτά από τα σύγχρονα στεγαναλυτικά εργαλεία ούτε από τις μέχρι τώρα γνωστές θεωρητικές επιθέσεις.

Στον επίλογο, που αποτελέι το τελευταίο κεφάλαιο της διατριβής, συνοψίζονται τα συμπεράσματά της και παρατίθενται ορισμένα θέματα για τη περαιτέρω συνέχιση της έρευνας.

Η διατριβή ολοκληρώνεται με το παράρτημα στο οποίο παρατίθεται ο κώδικας σε C++, για την αλγεβρική επίθεση στον DES, ο οποίος περιγράφεται εκτενώς στο δεύτερο κεφάλαιο.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή

Η κρυπτογραφία είναι μια επιστήμη που βασίζεται στα μαθηματικά και αποτελεί τον βασικό κορμό αυτού που σήμερα αποκαλούμε «Ασφάλεια Πληροφοριών». Αρχικός της σκοπός ήταν η μεταφορά δεδομένων με τέτοιο τρόπο που μόνο οι κατάλληλα εξουσιοδοτημένες οντότητες να έχουν πρόσβαση σε αυτά, εξασφαλίζοντας έτσι το απόρρητο στις ψηφιακές επικοινωνίες αλλά και στην αποθήκευση ευαίσθητων πληροφοριών.

Η αρχική της χρήση ήταν για στρατιωτικούς σκοπούς και τη συναντάμε από τα αρχαία χρόνια, είτε για την ασφαλή μεταβίβαση μηνυμάτων από τον βασιλιά προς τα έμπιστά του άτομα, είτε μεταξύ των στρατιωτικών, είτε μεταξύ των στρατιωτικών και του βασιλιά. Σκοπός τότε ήταν προφανώς να μη λάβει ο εχθρός κάποιο μήνυμα.

Το αρχικό μήνυμα ονομάζεται απλό κείμενο (plaintext). Το μήνυμα που προκύπτει από την κρυπτογράφηση του απλού κειμένου ονομάζεται κρυπτογράφημα (ciphertext).

Αποκρυπτογράφηση είναι η διαδικασία ανάκτησης του απλού κειμένου από το κρυπτογράφημα με την εφαρμογή ενός αντίστροφου αλγορίθμου. Η κρυπτογραφημένη επικοινωνία είναι αποτελεσματική, μόνο όταν τα

άτομα που συμμετέχουν σε αυτή μπορούν να ανακτήσουν το περιεχόμενο του αρχικού μηνύματος.

Κρυπτανάλυση, από την άλλη, είναι η επιστήμη που ασχολείται με την ανάλυση των κρυπτογραφημένων κειμένων και έχει ως στόχο την αποκρυπτογράφηση των κρυπτογραφημένων πληροφοριών χωρίς την πλήρη γνώση του αντίστροφου αλγορίθμου κρυπτογράφησης. Πλήρης γνώση θα ήταν να γνωρίζουμε είτε το κλειδί είτε τον αντίστροφο αλγόριθμο. Στην κρυπτανάλυση, για λόγους ασφαλείας δεχόμαστε την αρχή του Kerckhoff [1], "There is no secrecy in the algorithm - It is all in the key.". Αυτό που ήθελε να επισημάνει ο Kerckhoff, είναι πως ο αντίστροφος αλγόριθμος με κάποιον τρόπο, θεμιτό ή αθέμιτο, θα έχει πέσει στα χέρια του κρυπταναλυτή, συνεπώς το όλο βάρος πέφτει στον τρόπο με τον οποίο έχει γίνει η χρήση του κλειδιού από τον αλγόριθμο κρυπτογράφησης.

Ο αλγόριθμος κρυπτογράφησης είναι μια μαθηματική συνάρτηση που χρησιμοποιείται για την κρυπτογράφηση και αποκρυπτογράφηση πληροφοριών. Όσο αυξάνει ο βαθμός πολυπλοκότητας του αλγορίθμου, τόσο μειώνεται η πιθανότητα να τον διαβάλει κάποιος. Ο αλγόριθμος κρυπτογράφησης λειτουργεί σε συνδυασμό με ένα κλειδί (key), για την κρυπτογράφηση του απλού κειμένου. Το ίδιο απλό κείμενο

κρυπτογραφείται σε διαφορετικά κρυπτογραφήματα, όταν χρησιμοποιούνται διαφορετικά κλειδιά.

Το Διαδίκτυο ήδη χρησιμοποιείται από εκατομμύρια χρήστες, και επεκτείνεται με εκθετικούς ρυθμούς αύξησης. Μπορεί να θεωρηθεί ένας χώρος επικοινωνίας, εκπαίδευσης και οικονομικής δραστηριότητας με διαρκώς αυξανόμενη δύναμη. Η νέα ψηφιακή κοινωνία οφείλει να παρέχει μηχανισμούς προστασίας του απαραβίαστου της προσωπικής ζωής των μελών της, το οποίο αποτελεί θεμελιώδες ανθρώπινο δικαίωμα.

Σε νομικό και κοινωνικό επίπεδο, τίθεται ζήτημα προστασίας του απορρήτου της ηλεκτρονικής αλληλογραφίας, των συναλλαγών, του ιατρικού απορρήτου και γενικότερα το ζήτημα της προστασίας προσωπικών στοιχείων και δεδομένων του κάθε χρήστη του Διαδικτύου, που με διάφορους τρόπους μπορούν να συλλεχθούν από τρίτους και να χρησιμοποιηθούν για οποιονδήποτε σκοπό χωρίς τη συγκατάθεσή του.

Σε ακαδημαϊκό επίπεδο, τίθεται θέμα προστασίας αποτελεσμάτων ακαδημαϊκής έρευνας, ευαίσθητων προσωπικών δεδομένων (βαθμολογία φοιτητών), ακαδημαϊκών μελετών και γενικότερα προστασίας των πνευματικών δικαιωμάτων των μελών της ακαδημαϊκής κοινότητας.

Σε οικονομικό επίπεδο, η ασφάλεια και προστασία των εμπορικών πλέον δεδομένων, όπως η εξασφάλιση της εγκυρότητας των συναλλαγών μέσω της αποδοχής μίας ηλεκτρονικής υπογραφής και η ασφάλεια των συναλλαγών είναι κρίσιμα ζητήματα, που αποτελούν το υπόβαθρο της ψηφιακής παγκόσμιας αγοράς.

Η κρυπτογραφία εξασφαλίζει το απόρρητο των προσωπικών πληροφοριών και είναι η τεχνολογική πλευρά της λύσης στα προαναφερθέντα ζητήματα ασφαλείας.

Λόγω τον παραπάνω η κρυπτογράφηση δεδομένων πρέπει να τηρεί τα παρακάτω:

- Εμπιστευτικότητα (Confidentiality) Πρόκειται για την προστασία των δεδομένων ενάντια σε μη εξουσιοδοτημένη πρόσβαση ή γνωστοποίησή τους. Η υπηρεσία αυτή υλοποιείται μέσω μηχανισμών ελέγχου πρόσβασης στην περίπτωση αποθήκευσης δεδομένων και μέσω κρυπτογράφησης κατά την αποστολή τους
- Ακεραιότητα (Integrity) Είναι η προστασία των δεδομένων ενάντια σε μη εξουσιοδοτημένη τροποποίηση ή αντικατάστασή τους. Παρέχεται από μηχανισμούς κρυπτογραφίας όπως οι ηλεκτρονικές υπογραφές.

- Πιστοποίηση (Authentication) Πρόκειται για την επιβεβαίωση της ταυτότητας ενός ατόμου ή της πηγής αποστολής των πληροφοριών. Κάθε χρήστης που επιθυμεί να επιβεβαιώσει την ταυτότητα ενός άλλου προσώπου ή εξυπηρετητή με τον οποίο επικοινωνεί, βασίζεται στην πιστοποίηση
- Μη-Άρνηση Αποδοχής (Non-Repudiation) Συνδυάζει τις υπηρεσίες της Πιστοποίησης και της Ακεραιότητας. Ο αποστολέας δεδομένων δεν μπορεί να αρνηθεί ότι δημιούργησε και απέστειλε το μήνυμα. Η κρυπτογραφία παρέχει ηλεκτρονικές υπογραφές, κατά συνέπεια μόνο ο αποστολέας του μηνύματος θα μπορούσε να κατέχει τη συγκεκριμένη υπογραφή.

Η χρήση της κρυπτογραφίας ολοένα διευρύνεται, καθιστώντας πλέον αξιόπιστη τη μεταφορά της πληροφορίας για διάφορους λειτουργικούς σκοπούς. Αυτό έχει δώσει στην κρυπτογραφία τη δυνατότητα να χρησιμοποιείται καθημερινά από τον καθένα έμμεσα προκειμένου να ικανοποιήσει τις καθημερινές του ανάγκες. Μερικές από τις πλέον γνωστές της εφαρμογές είναι οι : ασφάλεια συναλλαγών σε τράπεζες δίκτυα – ATM, κινητή τηλεφωνία (Tetra – Tetrapol –GSM [68, 69]), σταθερή τηλεφωνία (cryptophones), διασφάλιση εταιρικών πληροφοριών, στρατιωτικά δίκτυα (τακτικά συστήματα επικοινωνιών μάχης), διπλωματικά δίκτυα (τηλεγραφήματα), ηλεκτρονικές επιχειρήσεις (πιστωτικές κάρτες, πληρωμές), ηλεκτρονική ψηφοφορία, ηλεκτρονική δημοπρασία,

ηλεκτρονικό γραμματοκιβώτιο, συστήματα συναγερμών, συστήματα βιομετρικής αναγνώρισης, έξυπνες κάρτες, ιδιωτικά δίκτυα (VPN), word wide web, δορυφορικές εφαρμογές (δορυφορική τηλεόραση), ασύρματα δίκτυα (hipperlan, bluetooth, 802.11x), συστήματα ιατρικών δεδομένων και άλλων βάσεων δεδομένων, τηλεσυνδιάσκεψη - τηλεφωνία μέσω διαδικτύου (VoIP), διαφύλαξη προσωπικών δεδομένων.

1.1 Ιστορική εξέλιξη της κρυπτογραφίας

Σύμφωνα με τους αλγορίθμους και τα μέσα τα οποία χρησιμοποιούνται στην κρυπτογραφία, θα μπορούσαμε να διακρίνουμε τρεις περιόδους.

1.1.1 **Πρώτη Περίοδος Κρυπτογραφίας** (1900 **π.Χ**. – 1900 **μ.Χ**.)

Κατά την περίοδο αυτή οι περισσότεροι αλγόριθμοι που αναπτύσσονται είναι αλγόριθμοι αντικατάστασης και μάλιστα στην πλειοψηφία τους μονοαλφαβητικοί, όπως επίσης και οι μηχανές, όταν χρησιμοποιούνται, είναι ιδιαίτερα απλές [54]. Όλες αυτές οι τεχνικές κρυπτογράφησης έχουν κρυπταναλυθεί στις μέρες μας, ενώ έχει αποδειχθεί, ότι, εάν μας είναι γνωστό ένα μεγάλο κομμάτι του κρυπτογραφημένου μηνύματος, τότε το αρχικό κείμενο μπορεί να επανακτηθεί με σχετική ευκολία.

Μια επιγραφή από το 1500 π.Χ στην περιοχή της Μεσοποταμίας, η οποία περιγράφει μία μέθοδο κατασκευής σμάλτων για αγγειοπλαστική, θεωρείται ως το αρχαιότερο κρυπτογραφημένο κείμενο με βάση τον Kahn [55]. Το

αρχαιότερο βιβλίο κρυπτοκωδικών στον κόσμο, θεωρείται μία σφηνοειδής επιγραφή στα Σούσα της Περσίας η οποία περιλαμβάνει τους αριθμούς 1 έως 8 και από το 32 έως το 35, τοποθετημένους τον ένα κάτω από τον άλλο, ενώ απέναντι τους βρίσκονται τα αντίστοιχα για τον καθένα σφηνοειδή σύμβολα. Η πρώτη στρατιωτική χρήση της κρυπτογραφίας αποδίδεται στους Σπαρτιάτες. Γύρω στον 5ο π.Χ. αιώνα εφεύραν τη «σκυτάλη», την πρώτη κρυπτογραφική συσκευή, στην οποία, χρησιμοποίησαν για την κρυπτογράφηση, τη μέθοδο της αντικατάστασης. Η «Σπαρτιατική Σκυτάλη» (Σχήμα 1), ήταν μια ξύλινη ράβδος, ορισμένης διαμέτρου, γύρω από την οποία ήταν τυλιγμένη ελικοειδώς μια λωρίδα περγαμηνής. Το κείμενο γραφόταν σε στήλες, ένα γράμμα σε κάθε έλικα, όταν δε ξετύλιγαν τη λωρίδα, το κείμενο λόγω της ανάμειξης των γραμμάτων γινόταν «κρυπτογραφημένο» με «κλειδί» τη διάμετρο της σκυτάλης.

Σχήμα 1 **Σπαρτιατική Σκυτάλη**

Ο Ιούλιος Καίσαρας χρησιμοποιούσε στην αλληλογραφία του με τον Κικέρωνα και με άλλους φίλους του μια μέθοδο κρυπτογράφησης αντικαθιστώντας τα γράμματα του κειμένου, με γράμματα, που βρίσκονται 3 θέσεις μετά, στο Λατινικό Αλφάβητο [54, 55]. Το σύστημα κρυπτογράφησης που στηρίζεται στην αντικατάσταση των γραμμάτων του αλφαβήτου με άλλα που βρίσκονται σε καθορισμένο αριθμό θέσης πριν ή μετά λέγεται πλέον κρυπτοσύστημα αντικατάστασης του Καίσαρα.

Κατά τη διάρκεια του Μεσαίωνα, η κρυπτολογία ήταν η εξέλιξη της κρυπτογραφίας και μεταφέρεται στον Αραβικό κόσμο. Εμφανίστηκαν βιβλία που περιείχαν κρυπταλφάβητα, όπως το αλφάβητο «Dawoudi» που πήρε το όνομα του από τον βασιλιά Δαυίδ. Οι Άραβες είναι οι πρώτοι που ανακάλυψαν αλλά και χρησιμοποίησαν μεθόδους κρυπτανάλυσης. Το κυριότερο εργαλείο στην κρυπτανάλυση, η χρησιμοποίηση των συχνοτήτων των γραμμάτων κειμένου, σε συνδυασμό με τις συχνότητες εμφάνισης στα κείμενα των γραμμάτων της γλώσσας, ανακαλύφθηκε από αυτούς γύρω στον 14ο αιώνα.

Επιστρέφοντας στην Ευρώπη στη συνέχεια αξίζει να αναφερθούμε στον Γάλλο Vigenere, του οποίου ο πίνακας πολυαλφαβητικής αντικατάστασης, χρησιμοποιείται ακόμη και σήμερα. Ο C.Wheatstone, γνωστός από τις μελέτες του στον ηλεκτρισμό, παρουσίασε την πρώτη μηχανική

κρυπτοσυσκευή, η οποία απετέλεσε τη βάση για την ανάπτυξη των κρυπτομηχανών της δεύτερης ιστορικής περιόδου της κρυπτογραφίας.

1.1.2 **Δεύτερη Περίοδος Κρυπτογραφίας** (1900 **μ.Χ**. – 1950 **μ.Χ**.)

Η δεύτερη περίοδος ξεκινά στις αρχές του 20ου αιώνα και φτάνει μέχρι το 1950. Στο διάστημα αυτό μεσολαβούν οι δύο παγκόσμιοι πόλεμοι. Η ανάγκη για ασφαλή μετάδοση δεδομένων μεταξύ των εμπόλεμων ζωνών οδήγησε σε μια νέα ανάπτυξη της κρυπτογραφίας. Μάλιστα λόγω της προόδου τόσο στη μηχανική όσο και στην ηλεκτρονική, η πρόοδος ήταν τεράστια. Τα κρυπτοσυστήματα αυτής της περιόδου γίνονται πολύπλοκα και αποτελούνται από μηχανικές και ηλεκτρομηχανικές κατασκευές. Η κρυπτανάλυσή τους, απαιτεί μεγάλο αριθμό προσωπικού, το οποίο εργαζόταν επί μεγάλο χρονικό διάστημα ενώ ταυτόχρονα γίνεται αισθητή η ανάγκη για μεγάλη υπολογιστική ισχύ. Παρά την πολυπλοκότητα που αποκτούν τα συστήματα κρυπτογράφησης κατά τη διάρκεια αυτής της περιόδου η κρυπτανάλυση τους είναι συνήθως επιτυχημένη. Πιο γνωστή «κρυπτομηχανή», όπως ονομάζονταν οι συσκευές κρυπτογράφησης, ήταν οι Enigma τις οποίες χρησιμοποιούσαν οι Γερμανοί κατά την διάρκεια του Β' παγκοσμίου πολέμου.

Η πρώτη παραβίαση ασφάλειας των μηχανών γίνεται από τον M. Rejewski, στην Πολωνία, χρησιμοποιώντας θεωρητικά μαθηματικά το 1932 [56, 57]. Το γεγονός αυτό θεωρείται μια από τις σημαντικότερες ανακαλύψεις στην κρυπτανάλυση της χιλιετίας που πέρασε. Το 1939 ο γερμανικός στρατός κάνει κάποιες αλλαγές και οι Πολωνοί δεν μπόρεσαν πλεόν να συνεχίσουν την κρυπτανάλυση των μηχανών, καθώς κάτι τέτοιο θα απαιτούσε περισσότερους υπολογιστικούς πόρους από όσους είχαν στη διάθεσή τους.

Η κρυπτανάλυση των Enigma θα συνεχιστεί από τους Βρετανούς και τους Γάλλους στους οποίους ενσωματώνονται τόσο ο Rejewski όσο και ο Szyfrow. Συνεχίζονται έτσι οι προσπάθειες από τον Alan Turing, τον Gordon Welchman [58], και από πολλούς άλλους στο Bletchley Park και οδηγεί σε συνεχείς παραβιάσεις των διαφόρων νέων παραλλαγών των Enigma που επιτυγχάνουν οι Γερμανοί.

Στην άλλη όχθη του Ατλαντικού η συνεργασία κρυπτογράφων του αμερικανικού ναυτικού με Βρετανούς και Ολλανδούς κρυπτογράφους μετά από το 1940 οδήγησε στο σπάσιμο αρκετών κρυπτόσυστημάτων του Ιαπωνικού ναυτικού όπως το JN-25, τις μηχανές Purple, τη "Μηχανή-Μ" και την «Red».

Οι σύμμαχοι εκείνο τον καιρό, χρησιμοποιούν το βρετανικό TypeX και το αμερικανικό SIGABA, τα οποία ήταν ηλεκτρομηχανικά σχέδια παρόμοια στο

πνεύμα με το Enigma, με σημαντικές βελτιώσεις. Κανένα δεν έγινε γνωστό ότι παραβιάστηκε κατά τη διάρκεια του πολέμου.

1.1.3 **Τρίτη Περίοδος Κρυπτογραφίας** (1950 **μ.Χ**. - **Σήμερα**)

Η τρίτη περίοδος της κρυπτογραφίας αρχίζει ουσιαστικά με τον Claude Shannon, τον πατέρα των μαθηματικών συστημάτων κρυπτογραφίας. Η δημοσίευση του «Communication Theory of Secrecy Systems» το 1949 και αργότερα το βιβλίο του, «Mathematical Theory of Communication» με τον Warren Weaver αποτελούν μέχρι σήμερα τα θεμέλια τόσο για την κρυπτογραφία και την κρυπτανάλυση, όσο και για τη θεωρία πληροφοριών. Από εκείνο το σημείο η κρυπτογραφία ουσιαστικά εξαφανίζεται και φυλάσσεται από τις μυστικές υπηρεσίες κυβερνητικών επικοινωνιών όπως η NSA. Ελάχιστα θα γίνουν γνωστά μέχρι το ’70 στο θέμα αυτό.

Η δεκαετία του 70, σηματοδοτείται από δύο μεγάλα γεγονότα για την κρυπτογραφία, την δημοσίευση του σχεδίου προτύπου κρυπτογράφησης DES [3] (Data Encryption Standard) στον ομοσπονδιακό κατάλογο της Αμερικής στις 17 Μαρτίου 1975 και τη δημιουργία κρυπτογραφίας δημοσίου κλειδιού από τους Rivest, Shamir και Adleman [12].

Το πρότυπο DES θα δώσει μια νέα διάσταση στην κρυπτανάλυση, θα οδηγήσει στην ανάπτυξη νέων εργαλείων, ενώ παράλληλα θα καθιερώσει νέους αλγόριθμους κρυπτογράφησης, πολύ πιο ισχυρούς και με πολύ διαφορετικό σχεδιασμό από τους προκατόχους του. Χαρακτηριστικό είναι ότι κάποιες παραλλαγές του χρησιμοποιούνται μέχρι σήμερα.

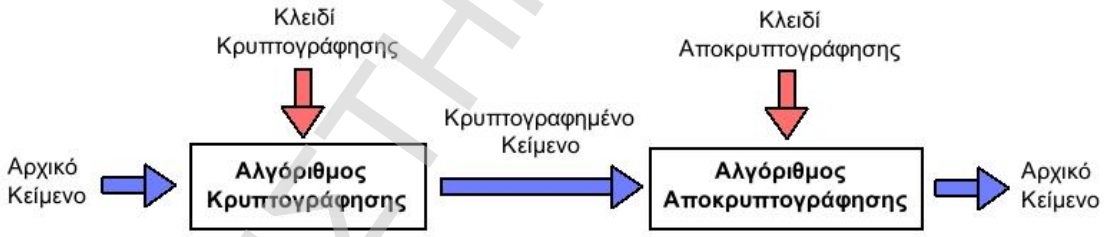
Η κρυπτογραφία δημοσίου κλειδιού θα ανοίξει ουσιαστικά την πόρτα για νέες εφαρμογές της κρυπτογραφίας. Η κρυπτογραφία θα χρησιμοποιηθεί από εταιρίες και ιδιώτες, θα δημιουργήσει και θα ανθίσει το ηλεκτρονικό εμπόριο, το ηλεκτρονικό επιχειρείν και οι ηλεκτρονικές τραπεζικές συναλλαγές.

Θα πρέπει να δούμε επιπλέον την αλλαγή στην αντιμετώπιση της κρυπτογραφίας, η οποία μέχρι τις αρχές του 2000 ήταν απαγορευμένη στο μεγαλύτερο μέρος του ανεπτυγμένου κόσμου. Χαρακτηριστικό παράδειγμα ήταν η αντικατάσταση του DES από τον AES το 2001, όταν ανήγγειλε το National Institute of Standards and Technology (NIST) το FIPS 197. Μετά από έναν ανοικτό διεθνή διαγωνισμό, το NIST επέλεξε τον αλγόριθμο Rijndael [29], που υποβλήθηκε από δύο Βέλγους κρυπτογράφους, για να είναι το πρότυπο AES (Advanced Encryption Standard). Όλοι οι διαγωνιζόμενοι θα έπρεπε να υποβάλουν τους αλγορίθμους τους με πλήρη στοιχεία σχεδιασμού και να προβάλουν πέρα από στοιχεία απόδοσης σε επεξεργαστές και αποδείξεις ασφάλειας απέναντι σε συγκεκριμένες επιθέσεις.

1.2 Μαθηματική μοντελοποίηση

Ο αντικειμενικός στόχος της κρυπτογραφίας είναι να δώσει τη δυνατότητα σε δύο πρόσωπα, τα οποία έχει καθιερωθεί να αναφέρονται πλέον ως Alice και Bob, να επικοινωνήσουν μέσα από ένα μη ασφαλές κανάλι με τέτοιο τρόπο ώστε ένα τρίτο πρόσωπο, μη εξουσιοδοτημένο (ένας αντίπαλος), να μην μπορεί να παρεμβληθεί στην επικοινωνία ή να κατανοήσει το περιεχόμενο των μηνυμάτων [5].

Η διαδικασία της κρυπτογράφησης και της αποκρυπτογράφησης φαίνεται στο παρακάτω σχήμα (Σχήμα 2)



Σχήμα 2 Σύστημα κρυπτογράφησης – αποκρυπτογράφησης

Η κρυπτογράφηση και αποκρυπτογράφηση ενός μηνύματος γίνεται με τη βοήθεια ενός αλγόριθμου κρυπτογράφησης (cipher) και ενός κλειδιού κρυπτογράφησης (key). Συνήθως ο αλγόριθμος κρυπτογράφησης είναι γνωστός, οπότε η εμπιστευτικότητα του κρυπτογραφημένου μηνύματος που μεταδίδεται βασίζεται ως επί το πλείστον στη μυστικότητα του κλειδιού κρυπτογράφησης. Το μέγεθος του κλειδιού κρυπτογράφησης μετριέται σε

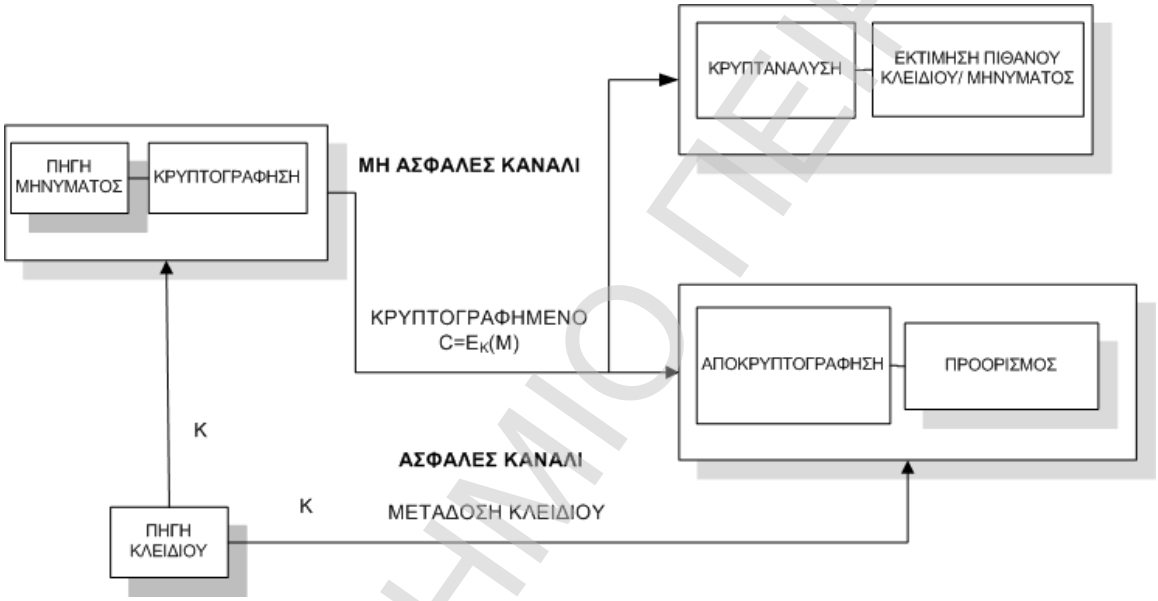
αριθμό bits. Γενικά ισχύει ο εξής κανόνας: όσο μεγαλύτερο είναι το κλειδί κρυπτογράφησης, τόσο δυσκολότερα μπορεί να αποκρυπτογραφηθεί το κρυπτογραφημένο μήνυμα από επίδοξους εισβολείς. Διαφορετικοί αλγόριθμοι κρυπτογράφησης απαιτούν διαφορετικά μήκη κλειδιών για να πετύχουν το ίδιο επίπεδο ανθεκτικότητας σε κρυπταναλυτικές επιθέσεις.

Ένα κρυπτοσύστημα (σύνολο διαδικασιών κρυπτογράφησης - αποκρυπτογράφησης) χαρακτηρίζεται από την πεντάδα (P, C, k, E , D), όπου:

- Το P είναι ο χώρος όλων των δυνατών μηνυμάτων.
- Το C είναι ο χώρος όλων των δυνατών κρυπτογραφημένων μηνυμάτων.
- Το k είναι ο χώρος όλων των δυνατών κλειδιών.
- Η E είναι η συνάρτηση κρυπτογράφησης.
- Η D είναι η συνάρτηση αποκρυπτογράφησης

Η συνάρτηση κρυπτογράφησης E δέχεται δύο μεταβλητές, το κείμενο προς κρυπτογράφηση, μέσα από τον χώρο P, και το κλειδί, από τον χώρο k και παράγει μία ακολουθία χαρακτήρων που ανήκει στον χώρο C. Η συνάρτηση αποκρυπτογράφησης D με τη σειρά της, δέχεται δύο μεταβλητές, μια το κρυπτοκείμενο από τον χώρο C και μια το κλειδί από τον χώρο k και παράγει μια ακολουθία χαρακτήρων που ανήκει στον χώρο P.

Ο αντίπαλος παρακολουθεί την επικοινωνία, ενημερώνεται για την κρυπτοακολουθία αλλά δεν έχει γνώση για το κλειδί που χρησιμοποιήθηκε και δεν μπορεί να αναδημιουργήσει το μήνυμα



Σχήμα 3 **Μοντέλο τυπικού κρυπτοσυστήματος ιδιωτικού κλειδιού**

1.3 Είδη κρυπτογράφησης

1.3.1 **Κρυπτογραφία δημοσίου κλειδιού**

Βασικός σκοπός της κρυπτογραφίας δημοσίου κλειδιού ή αλλιώς και ασύμμετρη κρυπτογραφία, είναι να επιτρέψει σε δύο ενδιαφερόμενους Α και Β να επικοινωνήσουν χωρίς να έχουν κάποιο κοινό κλειδί εκ των προτέρων. Συμφωνούν μόνο στη χρήση δύο αλγόριθμων Ε και D,

κρυπτογράφησης και αποκρυπτογράφησης, αντίστοιχα. Στην αρχή ο Α διαλέγει δυο τυχαία "κλειδιά" το ένα ονομάζεται δημόσιο **e** και το άλλο ιδιωτικό **d**. Ο Α μετά τη δημιουργία των δύο κλειδιών δημοσιεύει το κλειδί **e** και κρατάει μυστικό το κλειδί **d**.

Όταν ο Α θέλει να στείλει ένα μήνυμα **m** στον Β, το κρυπτογραφεί χρησιμοποιώντας το δημόσιο κλειδί **e**, στέλνει δηλαδή το: **m'=E(m,e)**. Ο Β λαμβάνοντας το μήνυμα **m'** υπολογίζει το αρχικό μήνυμα, με τη χρήση του ιδιωτικού του κλειδιού **d** με την ακόλουθη διαδικασία: **m=D(m', d)=D(E(m, e), d)**.

Θα πρέπει να τονίσουμε ότι προκειμένου το πρωτόκολλο αυτό να είναι ασφαλές, θα πρέπει να είναι υπολογιστικά αδύνατο για όποιον δεν έχει γνώση του **d** να βρει το **m**.

Τα βασικά πλεονεκτήματα της κρυπτογραφίας δημόσιου κλειδιού είναι:

- Δεν απαιτείται προηγούμενη συμφωνία για τα κλειδιά
- Δεν χρειάζεται μεταφορά κλειδιών.
- Νέες οντότητες μπορούν να αρχίσουν να επικοινωνούν αμέσως.
- Έχει δοκιμαστεί επαρκώς στην πράξη με ικανοποιητικά αποτελέσματα.
- Βασίζεται στη σοβαρή και αυστηρή θεωρία της Υπολογιστικής Πολυπλοκότητας.

Η κρυπτογραφία δημόσιου κλειδιού χρησιμοποιείται ευρύτατα σήμερα. Η τεράστια άνθηση των ηλεκτρονικών εφαρμογών που απαιτούν ασφάλεια

δεν θα ήταν εφικτή εάν υπήρχε μόνο κρυπτογραφία ιδιωτικού κλειδιού κυρίως λόγω της δυσκολίας διανομής και συντήρησης κλειδιών.

1.3.2 **Κρυπτογραφία συμμετρικού κλειδιού**

Αντίθετα με την κρυπτογραφία δημοσίου κλειδιού, στην κρυπτογραφία συμμετρικού κλειδιού ή αλλιώς και κρυπτογραφία ιδιωτικού κλειδιού, το κλειδί είναι ένα, τόσο για κρυπτογράφηση όσο και για αποκρυπτογράφηση και πρέπει να φυλάσσεται μυστικό. Οι δύο οντότητες Α και Β έχουν εκ των προτέρων ανταλλάξει ένα μυστικό κλειδί μέσω ενός ασφαλούς διαύλου επικοινωνίας. Στο Σχήμα 3 παρουσιάζεται το μοντέλο ενός τυπικού κρυπτοσυστήματος ιδιωτικού κλειδιού.

Ανάλογα με το πώς γίνεται η χρήση του κλειδιού αλλά και με το πώς ο κάθε αλγόριθμος κρυπτογραφεί τα γράμματα του κειμένου, έχουμε την παρακάτω κατηγοριοποίηση [9]:

- Αλγόριθμους κωδικοποίησης με τμήματα n-bit (block cipher), όπου το μήνυμα διαιρείται σε τμήματα των n-bit, και το κάθε τμήμα κρυπτογραφείται ανεξάρτητα από το άλλο.
- Αλγόριθμους ροής (stream cipher), όπου η κρυπτογράφηση γίνεται διαδοχικά, 1-bit κάθε φορά. Χρησιμοποιείται γεννήτρια «τυχαίων» bit και εφαρμόζονται κάποιες μαθηματικές πράξεις πάνω στα bit που

θέλουμε να κρυπτογραφήσουμε. Το μεγάλο πρόβλημα σε αυτήν την περίπτωση είναι ο συγχρονισμός πομπού και δέκτη.

1.4 Κρυπταναλυτικές επιθέσεις σε αλγορίθμους

Υπάρχουν πέντε βασικές κρυπταναλυτικές επιθέσεις κατηγοριοποιημένες ανάλογα με την ικανότητα του αντιπάλου τόσο σε πόρους όσο και υπολογιστική ισχύ και το επίπεδο πρόσβασης που μπορεί να έχει στην μεταδιδόμενη πληροφορία.

- **Επίθεση βασισμένη στο κρυπτοκείμενο**

  Ο επιτιθέμενος έχει στη διάθεση του Ν κρυπτογραφημένα μηνύματα με δεδομένη τη γνώση του αλγορίθμου. Σκοπός είναι να ανακαλύψει τα μηνύματα που περικλείουν τα κρυπτογραφημένα μηνύματα ή να εξαγάγει το κλειδί ή μέρος του, που χρησιμοποιήθηκε.

- **Επίθεση βασισμένη στη γνώση ζευγών μηνυμάτων και κρυπτογραφημένων μηνυμάτων**

  Ο επιτιθέμενος έχει στην κατοχή του μερικά ζευγάρια μηνυμάτων και κρυπτογραφημένων μηνυμάτων. Ο στόχος του είναι η εξαγωγή κλειδιού ή μέρους του, ή η εύρεση ενός αλγορίθμου για την αποκρυπτογράφηση νέων μηνυμάτων (προσεγγιστικός αλγόριθμος) με το ίδιο κλειδί.

- **Επίθεση βασισμένη στην επιλογή μηνυμάτων**

Ο επιτιθέμενος έχει καταφέρει να αποκτήσει πρόσβαση στη επιλογή του μηνύματος που θα κρυπτογραφηθεί. Στόχος είναι η εξαγωγή του κλειδιού ή μέρους του ή ενός προσεγγιστικού αλγορίθμου.

- **Επίθεση βασισμένη στην επιλογή κρυπτογραφημένων μηνυμάτων**

  Ο επιτιθέμενος μπορεί να επιλέξει κρυπτογραφημένα μηνύματα για αποκρυπτογράφηση και έχει πρόσβαση στα αποκρυπτογραφημένα κείμενα.

- **Προσαρμοσμένη επίθεση βασισμένη στην επιλογή μηνυμάτων – κλειδιών**

  Ο επιτιθέμενος επιλέγει μια σχέση μεταξύ του άγνωστου κλειδιού και του ενός αρχικού δικού του κλειδιού. Στη συνέχεια τροφοδοτεί με δεδομένα δύο συστήματα, ένα το δικό του που λειτουργεί με το δικό του κλειδί και ένα το οποίο λειτουργεί με το άγνωστο κλειδί. Με βάση τα συμπεράσματα από την ανάλυση της εισόδου και της εξόδου των δύο συστημάτων, ο επιτιθέμενος προσεγγίζει μετά από κάποιες δοκιμές το σωστό κλειδί.

1.5 Επιθέσεις στο κανάλι επικοινωνίας

Υπάρχουν τέσσερις βασικές επιθέσεις στο κανάλι επικοινωνίας (Σχήμα 4), αν τις κατηγοριοποιήσουμε με βάση τη συμπεριφορά του αντιπάλου. Αυτές είναι:

- **Διακοπή γραμμής**

  Ο επιτιθέμενος έχει τη δυνατότητα να διακόψει τη ροή της πληροφορίας από τον αποστολέα στον παραλήπτη.

- **Υποκλοπή πληροφορίας από το κανάλι**

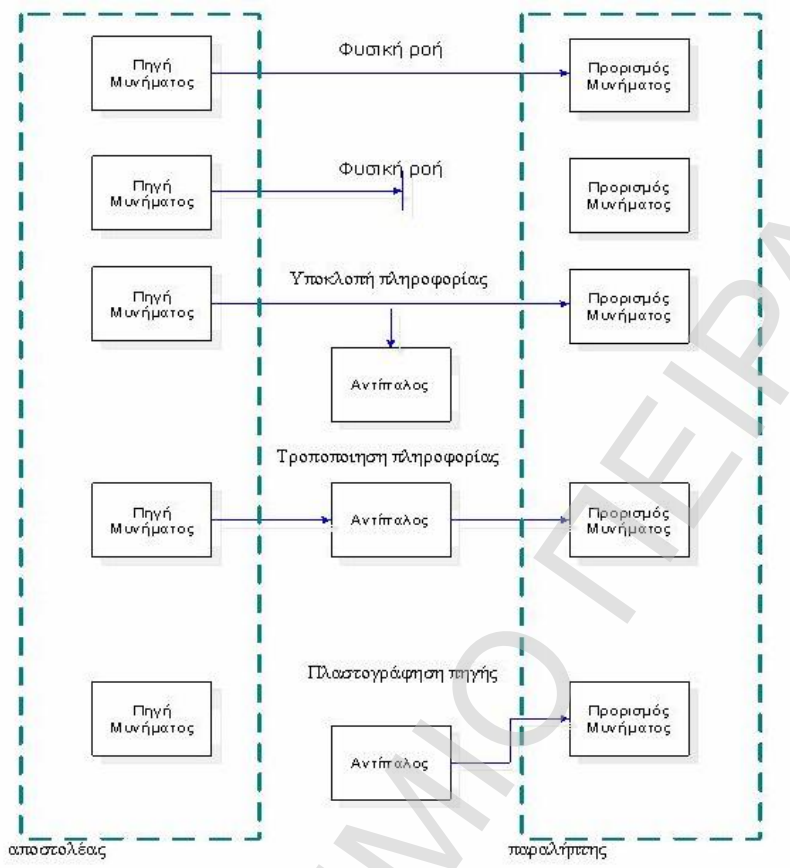
  Ο επιτιθέμενος αντιγράφει τις πληροφορίες που διαβιβάζονται στο κανάλι επικοινωνίας

- **Τροποποίηση πληροφορίας στο κανάλι**

  Ο επιτιθέμενος τροποποιεί τις πληροφορίες που διαβιβάζονται στο κανάλι με τέτοιο τρόπο ώστε να αλλάξει το περιεχόμενο ή να αναγεννά δική του πληροφορία.

- **Πλαστογράφηση πηγής**

  Ο επιτιθέμενος προσποιείται ότι είναι ένα από τα μέλη.



Σχήμα 4 **Επιθέσεις στο κανάλι επικοινωνίας**

1.6 Ταξινόμηση Μοντέλων αξιολόγησης ασφάλειας

Υπάρχουν τέσσερα βασικά μοντέλα για την αξιολόγηση των αλγορίθμων: η ασφάλεια άνευ όρων, η υπολογιστική ασφάλεια, η θεωρία πολυπλοκότητας και η αποδείξιμη ασφάλεια.

1.6.1 **Ασφάλεια άνευ όρων**

Η ασφάλεια άνευ όρων ή αλλιώς και γνωστή και ως τέλεια ασφάλεια. Αυτό το μοντέλο εστιάζεται στη δυνατότητα προσφοράς από ένα

κρυπτοσύστημα άνευ όρων ασφάλειας. Η βασική υπόθεση είναι ότι όσο και κρυπτοκείμενο και αν έχει στην κατοχή του ο αντίπαλος δεν υπάρχει αρκετή πληροφορία για να ανακτήσει το αποκρυπτογραφημένο κείμενο όση υπολογιστική ισχύ και αν έχει στην διάθεσή του. Χαρακτηριστικό παράδειγμα είναι το λεγόμενο one time pad, όπου διαρκώς παράγονται νέοι πραγματικά τυχαίοι αριθμοί και κάθε πιθανό αρχικό κείμενο είναι το ίδιο πιθανό να έχει κρυπτογραφηθεί σε αυτό το οποίο έχει στην κατοχή του ο επιτιθέμενος.

1.6.2 **Υπολογιστική ασφάλεια**

Στην περίπτωση της υπολογιστικής ασφάλειας ή αλλιώς και πρακτικής ασφάλειας, η μέτρηση εστιάζεται στην υπολογιστική προσπάθεια που χρειάζεται ο επιτιθέμενος προκειμένου να διασπαστεί ένα κρυπτοσύστημα. Στόχος των συγχρόνων συστημάτων να έχουν μεγάλο παράγοντα δυσκολίας ώστε να μην είναι χρονικά δυνατό να διασπαστούν με τα διαθέσιμα ή τα «μελλοντικά» μέσα.

1.6.3 **Ασφάλεια – θεωρία πολυπλοκότητας**

Αυτό το μοντέλο εστιάζεται στην ταξινόμηση της υπολογιστικής ικανότητας του αντιπάλου στην επίλυση υπολογιστικών προβλημάτων ανάλογα με τους πόρους που απαιτούνται για την επίλυσή τους. Οι πόροι αυτοί είναι:

- Το μέγεθος δεδομένων που χρειάζονται ως είσοδο στην επίθεση
- Ο υπολογιστικός χρόνος που χρειάζεται για να εκτελεστεί η επίθεση

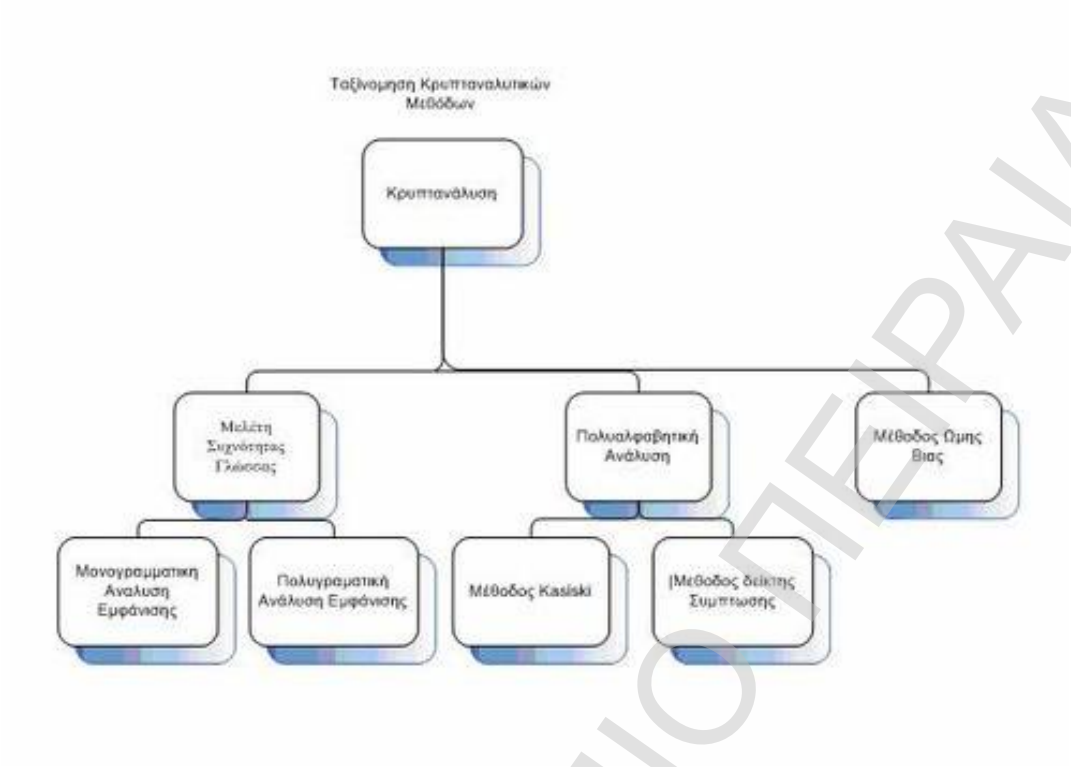
- Το μέγεθος του χώρου αποθήκευσης που χρειάζεται για την επίθεση
- Το πλήθος των επεξεργαστών

1.6.4 **Αποδείξιμη ασφάλεια**

Στο μοντέλο αυτό, το ενδιαφέρον εστιάζεται στην απόδειξη ισοδυναμίας του μαθηματικού μοντέλου του κρυπτοσυστήματος με κάποιο πολύ γνωστό δύσκολο στην επίλυσή του πρόβλημα (θεωρίας αριθμών). Χαρακτηριστικό παράδειγμα η ανάλυση μεγάλων ακεραίων σε γινόμενο πρώτων.

1.7 Κρυπτανάλυση Κλασικών Κρυπτοσυστημάτων

Υπάρχουν διάφορα είδη κρυπταναλυτικών επιθέσεων (Σχήμα 5) για τα κλασικά κρυπτοσυστήματα, τα περισσότερα από τα οποία βασίστηκαν πάνω στην γλωσσική δομή του μηνύματος. Στις νεότερες μορφές κρυπτανάλυσης κλασικών κρυπτοσυστημάτων παρατηρείται ή είσοδος της στατιστικής στην ανάλυση.



Σχήμα 5 **Ταξινόμηση Κρυπταναλυτικών Μεθόδων**

1.7.1 **Διαφορική Κρυπτανάλυση**

Η διαφορική κρυπτανάλυση παρουσιάστηκε αρχικά από τους **Biham** και **Shamir [6]** στο **CRYPTO 90** ως επίθεση στον **DES.** Αν και ο αρχικός στόχος της επίθεσης ήταν ο **DES**, η ευρεία εφαρμογή του σε πολυάριθμους κρυπτογραφικούς αλγορίθμους είχε ως αποτέλεσμα την παγίωση αυτής της μεθόδου κρυπτανάλυσης στην ασφάλεια των κρυπτογραφικών αλγορίθμων.

Για παράδειγμα στον αλγόριθμο κρυπτογράφησης που θεωρείται το προηγμένο πρότυπο κρυπτογράφησης, του **Rijndael**, είναι εμφανής η χρήση τεχνικών που αποβλέπουν στην αποφυγή γραμμικής και διαφορικής κρυπτανάλυσης.

Η διαφορική κρυπτανάλυση εκμεταλλεύεται την υψηλή πιθανότητα ορισμένων γεγονότων στις διαφορές των αρχικών κειμένων καθώς και στις διαφορές στον τελευταίο κύκλο του αλγόριθμου. Για παράδειγμα έστω ένα σύστημα με είσοδο

$$X= [X_1X_2...X_n]$$

και έξοδο

$$Y= [Y_1Y_2...Y_n]$$

Έστω δύο είσοδοι του συστήματος X' και X'' με τις αντίστοιχες εξόδους Y' και Y''. Η διαφορά της εισόδου δίνεται από τη σχέση

$$\Delta X = X' \oplus X''$$

όπου «$\oplus$» συμβολίζει την λογική διάζευξη των n-bit διανυσμάτων και συνεπώς

$$\Delta X = [\Delta X_1\Delta X_2...\Delta X_n].$$

όπου $\Delta X_i = X'_i \oplus X''_i$ και X'_i και X''_i συμβολίζουν το i bit των X'_i και X''_i.
Ομοίως για τη διαφορά της εξόδου έχουμε:

$$\Delta Y = Y' \oplus Y''$$

και:

$$\Delta Y =[\Delta Y_1 \Delta Y_2...\Delta Y_n]$$

όπου $\Delta Y_i = Y'_i \oplus Y''_i$

Η διαφορική κρυπτανάλυση ανήκει στις επιθέσεις επιλεγμένου αρχικού κειμένου, δηλαδή αυτός που πραγματοποιεί την επίθεση είναι ικανός να συλλέξει εισόδους καθώς και να εξετάσει εξόδους σε μια προσπάθεια να αποκτήσει το κλειδί. Στη διαφορική κρυπτανάλυση αυτός που πραγματοποιεί την επίθεση θα διαλέξει ζεύγη εισόδων ,X' και X'' που θα ικανοποιούν μια συγκεκριμένη τιμή της ΔX, γνωρίζοντας ότι για τη συγκεκριμένη τιμή της ΔX, μια συγκεκριμένη τιμή της ΔY θα εμφανιστεί με υψηλή πιθανότητα.

Σκοπός της επίθεσης είναι η κατασκευή του διαφορικού (ΔΧ,ΔΥ) που περικλείει bits αρχικών κειμένων όπως αναπαρίστανται από τη Χ και την είσοδο του τελευταίου κύκλου του κώδικα όπως αυτό αναπαρίσταται από τη Υ. Η κατασκευή αυτή θα πραγματοποιηθεί με την εξέταση υψηλά πιθανών διαφορικών χαρακτηριστικών , όπου ένα διαφορικό χαρακτηριστικό είναι η ακολουθία των διαφορών των εισόδων και εξόδων στους κύκλους, έτσι ώστε η διαφορά της εξόδου ενός κύκλου να αντιστοιχεί στη διαφορά της εισόδου στον αμέσως επόμενο γύρο. Χρησιμοποιώντας το υψηλά πιθανό χαρακτηριστικό μας δίνεται η δυνατότητα να εξαγάγουμε και να αξιοποιήσουμε πληροφορίες από τον τελευταίο κύκλο του κώδικα ώστε να πάρουμε bits από τον τελευταίο επίπεδο των υποκλειδιών.

Όπως και με τη γραμμική κρυπτανάλυση που θα δούμε στη συνέχεια, για να κατασκευάσουμε υψηλά πιθανά διαφορικά χαρακτηριστικά, εξετάζουμε τις ιδιότητες ανεξάρτητων S-boxes και χρησιμοποιούμε αυτές τις ιδιότητες για να καθορίσουμε το συνολικό διαφορικό χαρακτηριστικό. Συγκεκριμένα, θεωρούμε τις διαφορές εισόδου εξόδου των S-boxes προκειμένου να καθορίσουμε, ένα υψηλής πιθανότητας, ζεύγος διαφοράς. Ο συνδυασμός των ζευγών διαφοράς των S-boxes από κύκλο σε κύκλο έτσι ώστε η μη μηδενική διαφορά των bits εξόδου από ένα γύρο να αντιστοιχεί στη μη μηδενική διαφορά των bits εισόδου του επόμενου γύρου επιτρέπει την εύρεση διαφορικού υψηλής πιθανότητας που θα αποτελείται από τη διαφορά αρχικού κειμένου και τη διαφορά της εισόδου στον τελευταίο κύκλο. Τα bits υποκλειδιού του κώδικα καταλήγουν να μην εμφανίζονται στην έκφραση διαφοράς επειδή περιλαμβάνονται ταυτόχρονα και στα δύο

ζεύγη δεδομένων και συνεπώς λαμβάνοντας υπόψη την επιρροή της διαφοράς των exclusive-Or bits υποκλειδιού με τον εαυτό τους , το αποτέλεσμα είναι μηδέν.

1.7.2 **Γραμμική Κρυπτανάλυση**

Η γραμμική κρυπτανάλυση παρουσιάστηκε από τον Matsui στο EUROCRUPT 93 ως η θεωρητική απειλή απέναντι στον DES [4], ενώ αργότερα χρησιμοποιήθηκε επιτυχώς και πρακτικά στην κρυπτανάλυση του DES

Η γραμμική κρυπτανάλυση προσπαθεί να εκμεταλλευτεί το υψηλό ποσοστό επανάληψης γραμμικών εκφράσεων που αφορούν στα bits του αρχικού κειμένου και στα bits του κρυπτογραφημένου κειμένου και σε υποκλειδιά. Τα bits που χρησιμοποιούνται προκύπτουν από το δεύτερο γύρο του κρυπταλγορίθμου και μετά.

Το είδος αυτό της επίθεσης βασίζεται στο γεγονός ότι αυτός που πραγματοποιεί την επίθεση, κατέχει πληροφορίες σχετικά με μια ομάδα αρχικών κειμένων και συσχετιζόμενων κρυπτογραφημάτων. Ωστόσο το άτομο αυτό δε γνωρίζει ποια αρχικά κείμενα και ποια κρυπτογραφήματα

είναι διαθέσιμα, για αυτό θεωρούμε ότι αυτός που επιτίθεται έχει γνώση για τυχαίο αριθμό αρχικών κειμένων και κρυπτογραφημάτων.

Η βασική ιδέα είναι να προσεγγίσουμε τη λειτουργία ενός μέρους του κρυπτολγαρίθμου με μια γραμμική μορφή της οποίας η γραμμικότητα αναφέρεται σε **mod-2 bit** (πχ **exclusive OR** ,το οποίο συμβολίζεται με $\oplus$). Μια τέτοια έκφραση είναι η μορφή

$$X_{i_1} \oplus X_{i_2} \oplus ... \oplus X_{i_u} \oplus Y_{j_1} \oplus Y_{j_2} \oplus ... \oplus Y_{j_v} = 0 \qquad (1)$$

όπου X_i το i-οστό **bit** της εισόδου X[X1 ,X2 ,...] και το Y_j το j-οστό **bit** της εξόδου Y=[Y1 ,Y2 ,...]. Η εξίσωση αυτή παριστάνει το άθροισμα **exclusive-OR** των **u bits** εισόδου και **v bits** εξόδου.

Η προσέγγιση με τη γραμμική κρυπτανάλυση έχει ως στόχο να ορίσουμε γραμμικές εκφράσεις οι οποίες έχουν υψηλό ή χαμηλό ποσοστό επανάληψης. Εάν ο κρυπταλγόριθμος έχει υψηλή ή χαμηλή απόκλιση, αυτό είναι ένδειξη έλλειψης ικανότητας του κρυπταλγορίθμου για τυχαιότητα, δηλαδή παράγει επαναλαμβανόμενα αποτελέσματα, είναι επομένως ευάλωτος.

Έστω ότι επιλέγουμε τυχαίες τιμές για u+v bits και τις τοποθετούμε στην εξίσωση 1, τότε η πιθανότητα η γραμμική έκφραση να στηριχθεί είναι ακριβώς ½ και αυτό αποτελεί μέτρο για την εφαρμογή της κρυπτανάλυσης. Όσο πιο μακριά είναι η τιμή της πιθανότητας από την τιμή ½, τόσο πιο εύκολο θα είναι ο κρυπταναλυτής να εφαρμόσει τη γραμμική κρυπτανάλυση. Έστω λοιπόν ότι η πιθανότητα έχει τιμή P_L τότε η απόκλιση είναι P_L-1/2. Όσο μεγαλύτερη είναι η απόσταση της πιθανότητας εμφάνισης $|P_L - 1/2|$, τόσο καλύτερα εφαρμόζεται η γραμμική κρυπτανάλυση, καθώς προκειμένου να πραγματοποιηθεί η απειλή απαιτούνται λιγότερα γνωστά αρχικά κείμενα.

Υπάρχουν τρόποι να αποφευχθεί η επίθεση της γραμμικής κρυπτανάλυσης . Έστω ότι κατασκευάζουμε μια γραμμική προσέγγιση η οποία συνδέει bits του αρχικού κειμένου,τα οποία αναπαρίστανται από X στην εξίσωση (1) και η είσοδος η οποία προκύπτει από τον τελευταίο γύρο του κρυπταλγορίθμου (ή ισοδύναμα η έξοδος του δεύτερου τελευταίου γύρου του κρυπταλγορίθμου) όπως αναπαρίσταται από την τιμή Y στην εξίσωση (1). Τα bits του αρχικού κειμένου είναι τυχαία και συνεπώς τυχαία είναι και τα bits εισόδου στο τελευταίο κύκλο.

Στην εξίσωση (1) εφόσον το αριστερό μέρος ισούται με 0, η εξίσωση εμπλέκει bits υποκλειδιών τα οποία είναι άγνωστα, που καθορίζονται από το

κλειδί που δέχεται επίθεση, με γνωστά, που προέρχονται από τα bits του κειμένου που επεξεργαζόμαστε. Εάν το άθροισμα των εμπλεκόμενων υποκλειδιών είναι 0, τότε η απόκλιση της (1) θα έχει το ίδιο πρόσημο με την απόκλιση της γραμμικής έκφρασης που αφορά στο άθροισμα των υποκλειδιών. Στην αντίθετη περίπτωση, αν το άθροισμα των εμπλεκόμενων bits υποκλειδιών είναι 1 τότε η απόκλιση θα έχει αντίθετο πρόσημο.

Η σχέση P_L =1 υπονοεί ότι η γραμμική εξίσωση (1) είναι πιστή αναπαράσταση της συμπεριφοράς του κρυπταλγορίθμου και συνεπώς ο κρυπταλγόριθμος έχει μια καταστροφική αδυναμία. Εάν P_L =0 τότε η (1) αναπαριστά μια μη άρτια σχέση στον κρυπταλγόριθμο. Αυτό αποτελεί επίσης ένδειξη καταστροφικής αδυναμίας. Τέλος γραμμικές και μη προσεγγίσεις με P_L >1/2 και P_L <1/2 είναι ευάλωτες στη γραμμική κρυπτανάλυση.

Για να κατασκευάσουμε γραμμικές εκφράσεις λαμβάνουμε υπόψη τις ιδιότητες του μοναδικού μη γραμμικού συστατικού του κρυπταλγορίθμου, δηλαδή το S-box. Με αυτό τον τρόπο είναι δυνατό να αναπτύξουμε γραμμικές προσεγγίσεις ανάμεσα σε ομάδες bits εισόδου και εξόδου στο S-box.

Κεφάλαιο 2 Μια αλγεβρική επίθεση στον DES

2.1 Εισαγωγή

Ο DES, Data Encryption Standard, αποτελεί το πιο γνωστό πρότυπο κρυπτογράφησης δεδομένων. Από το 1973 που δημιουργήθηκε μέχρι πρόσφατα το 1997, που αντικαταστάθηκε από τον AES, ο DES αποτελούσε διεθνώς πρότυπο αλγόριθμο ισχυρής κρυπτογράφησης. Η μελέτη του αποτέλεσε πηγή δημιουργίας πολλών άλλων αλγορίθμων κρυπτογράφησης και γέννησε πολλές τεχνικές κρυπτανάλυσης, όπως τη διαφορική και την γραμμική κρυπτανάλυση.

Το 1972, το NBS (National Bureau of Standards), σήμερα γνωστό ως NIST (National Institute of Standards and Technology) ξεκίνησε ένα πρόγραμμα για την προστασία των δεδομένων επικοινωνίας του υπολογιστή. Σκοπός αυτού του προγράμματος ήταν να δημιουργηθεί ένας απλός αλγόριθμος κρυπτογράφησης, ο οποίος να υλοποιείται με χαμηλό κόστος και να μπορεί να τρέξει σε πολλές διαφορετικές αρχιτεκτονικές.

Το 1973 το NBS, προκήρυξε έναν δημόσιο διαγωνισμό για τη δημιουργία ενός κρυπτογραφικού αλγόριθμου, ο οποίος θα έπρεπε να πληροί τις παρακάτω απαιτήσεις:

- Να εξασφαλίζει ένα υψηλό επίπεδο ασφάλειας.

- Να είναι απόλυτα συγκεκριμένος και εύκολος στην κατανόηση.
- Η ασφάλειά του να ανήκει στο κλειδί και όχι στη μυστικότητα του αλγορίθμου.
- Να είναι διαθέσιμος σε όλους τους χρήστες.
- Να είναι προσαρμόσιμος για χρήση σε ποικίλες εφαρμογές.
- Να είναι οικονομικά εφαρμόσιμος σε ηλεκτρονικές συσκευές.
- Να είναι αποδοτικός στη χρήση.
- Να υπάρχει η δυνατότητα αναβάθμισης.
- Να επιτρέπει τη μεταφορά πληροφοριών από ένα σύστημα ή από ένα πρόγραμμα σε ένα άλλο.

Η γνωστή εταιρεία IBM, η οποία διέθετε μια ομάδα η οποία ασχολείτο με την κρυπτογραφία, προτείνε έναν αλγόριθμο που είχε κατασκευάσει, τον Lucifer. Ο Lucifer χρησιμοποιούσε απλές λογικές πράξεις σε μικρές ομάδες από bit και μπορούσε να υλοποιηθεί αποδοτικά σε υπολογιστικά συστήματα. Ο αλγόριθμος αυτός είναι και αυτός ο οποίος τελικά έγινε αποδεκτός σε αυτόν τον διαγωνισμό. Η αναγνώρισή και υιοθέτησή του γίνεται γύρω στο 1975. Από τότε χρησιμοποιείται διαρκώς με διάφορες μορφές σχεδόν σε όλους τους τομείς που έχει χρησιμοποιηθεί η κρυπτογραφία.

Ο DES θεωρείται πλέον αλγόριθμος χωρίς ασφάλεια, εξαιτίας του ότι είναι τρωτός σε βίαιες επιθέσεις (brute-force attacks). Έχουν κατασκευαστεί

ειδικά μηχανήματα για τον DES τα οποία μπορούν μέσα σε διάστημα μερικών ωρών να βρουν το κλειδί που έχει χρησιμοποιηθεί για την κρυπτογράφηση, γεγονός που οφείλεται στο μήκος των 56 bits για το κλειδί του, που επιτρέπει τη δημιουργία 2^{56} δυνατών κλειδιών. Τα μηχανήματα αυτά βέβαια έχουν κόστος κατασκευής πολύ υψηλό.

Αν και έχουν γίνει απόπειρες στο παρελθόν αλγεβρικής μοντελοποίησης του DES, παρόλα αυτά δεν επετεύχθει κάποιο αποτέλεσμα ως προς την κρυπτανάλυσή του καθώς, τα αλγεβρικά συστήματα τα οποία προέκυπταν ήταν πολύ δύσκολο να επιλυθούν. Στην παρούσα διατριβή, έγινε μια προσπάθεια αλγεβρικής προσέγγισης του DES, ώστε να επιτευχθεί όσο το δυνατόν μεγαλύτερη εξαγωγή μέρους του κλειδιού, με το μικρότερο δυνατότερο κόστος τόσο σε υπολογιστική ισχύ, όσο και σε αποθηκευτικό χώρο. Προκειμένου να γίνει όσο το δυνατό πιο κατανοητή η αλγεβρική επίθεση στον DES, γίνεται μια περιγραφή του αλγορίθμου.

2.2 Περιγραφή του DES

Ο DES κρυπτογραφεί ομάδες (block) των 64 bits. Η είσοδος του αλγορίθμου είναι μία ομάδα μη κρυπτογραφημένου μηνύματος των 64 bits, ενώ η έξοδος του αλγορίθμου είναι και αυτή 64 bits. Ο DES ανήκει στην κατηγορία συμμετρικών αλγορίθμων κρυπτογράφισης, αφού ο ίδιος αλγόριθμος και το ίδιο κλειδί χρησιμοποιούνται τόσο για την

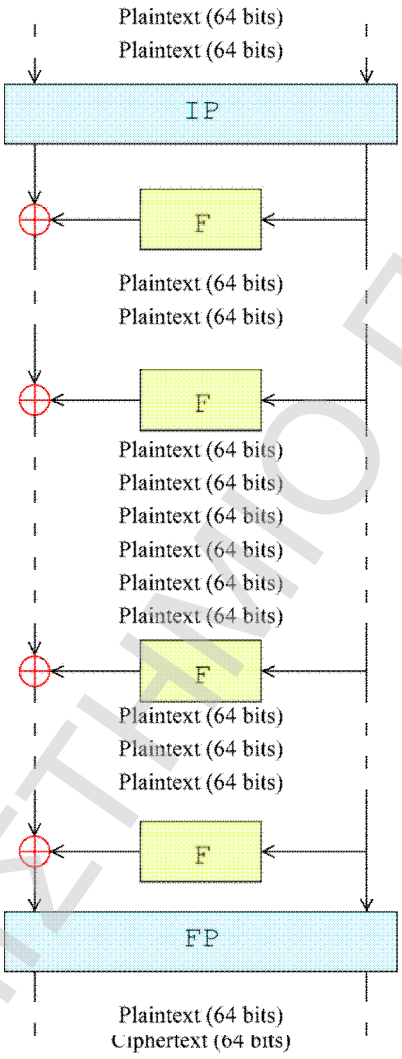
κρυπτογράφηση όσο και για την αποκρυπτογράφηση. Το μήκος του κλειδιού που χρησιμοποιεί ο αλγόριθμος είναι 56 bits.

Ο αλγόριθμος χρησιμοποιεί έναν συνδυασμό των δύο βασικών τεχνικών κρυπτογράφησης: το ανακάτωμα (shuffling) και την εξάπλωση (diffusion), όπως προτείνει και ο Shannon. Η βασική οικοδόμηση των τμημάτων (blocks) του DES είναι ένας απλός συνδυασμός αυτών των δύο τεχνικών πάνω στο μη κρυπτογραφημένο κείμενο, βασισμένες στο κλειδί. Αυτό είναι γνωστό ως κύκλος, γύρος ή επανάληψη. Ο DES έχει 16 όμοιες επαναλήψεις, στις οποίες χρησιμοποιεί μόνο αριθμητικούς και λογικούς τελεστές σε δυαδικούς αριθμούς το πολύ της τάξης των 64 bits. Έτσι λοιπόν ο DES είναι εύκολα εφαρμόσιμος και υλοποιήσιμος τόσο σε επίπεδο λογισμικού όσο και υλικού.

2.3 Περίληψη του αλγόριθμου

Ο DES λειτουργεί πάνω σε ένα τμήμα μηνύματος των 64 bits. Μετά από την αρχική μετάθεση των δυαδικών του ψηφίων το τμήμα χωρίζεται σε αριστερό και δεξιό μισό, μήκους 32 bits το καθένα. Ακολουθούν μετά 16 επαναλήψεις της ίδιας ακριβώς λειτουργίας (Σχήμα 6). Μια συνάρτηση f, η οποία συνδυάζει σε κάθε επανάληψη το απλό κείμενο με το κλειδί, εναλλάσει διαρκώς το δεξιό με το αριστερό μισό. Τέλος μετά την 16η

επανάληψη, το δεξιό και το αριστερό μισό ενώνονται και γίνεται μια τελική μετάθεση των bits (η αντίστροφη της αρχικής μετάθεσης).



Σχήμα 6 Σχηματική αναπαράσταση του DES

Σε κάθε κύκλο έχουμε επανάληψη της εφαρμογής της συνάρτησης f. Έχουμε μετακίνηση των bits του κλειδιού, και επιλογή των 48 από τα 56 bits. Η συνάρτηση f αναλύεται παρακάτω. Το δεξιό μισό των δεδομένων κάθε

γύρου, από 32bits επεκτείνεται στα 48 bits μέσω μιας μεταθετικής επέκτασης, και εφαρμόζεται λογική διάζευξη με ένα μέρος του κλειδιού. Στη συνέχεια το αποτέλεσμα στέλνεται στα 8 S-boxes, παράγοντας 32 νέα bits τα οποία μετατίθενται ξανά. Με αυτά τα βήματα τελειώνει η συνάρτηση f. Στο αποτέλεσμα που εξάγεται από αυτήν εφαρμόζεται XOR με το αριστερό μισό του τρέχοντος γύρου. Το αποτέλεσμα αυτό αποτελεί το νέο δεξιό μισό, ενώ το παλιό δεξιό μισό αποτελεί το νέο αριστερό μισό. Τα παραπάνω αποτελούν ένα γύρο. Αυτές οι λειτουργίες επαναλαμβάνονται 16 φορές αποτελώντας τις 16 επαναλήψεις του DES.

Η αρχική μετάθεση των δυαδικών ψηφίων λαμβάνει χώρα πριν την πρώτη επανάληψη. Μετατοπίζει το εισερχόμενο block σύμφωνα με τον σχήμα 9. Αυτός ο πίνακας, όπως και οι υπόλοιποι πίνακες, διαβάζονται από αριστερά προς τα δεξιά και από πάνω προς τα κάτω. Για παράδειγμα, η αρχική μετάθεση μετακινεί το bit 58 του μη κρυπτογραφημένου μηνύματος στη θέση 1, το bit 50 στη θέση 2, το bit 42 στη θέση 3 κ.ο.κ.
Θα πρέπει να σημειώσουμε πως τόσο η αρχική μετάθεση όσο και η αντίστοιχη τελική μετάθεση δεν επιδρούν στην ασφάλεια του DES καθώς ως πράξεις είναι γραμμικές.

2.3.1 **Ο μετασχηματισμός του κλειδιού**

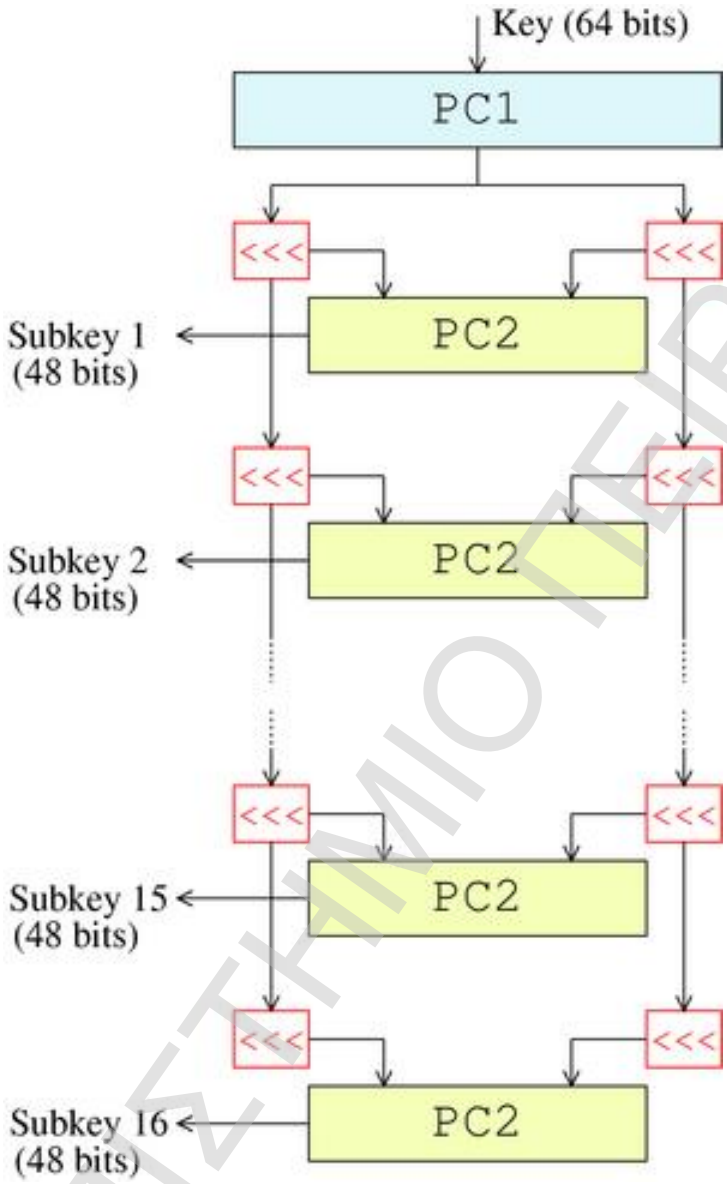
Αρχικά, το κλειδί των 64 bits του DES μετατρέπεται σε κλειδί των 56 bits αγνοώντας κάθε όγδοο bit. Ο λόγος ύπαρξης αυτών των bits είναι να

μπορούν να χρησιμοποιηθούν για έλεγχο ισοτιμίας ώστε να επιβεβαιωθεί ότι το κλειδί δεν έχει μεταφερθεί λάθος. Μετά το κλειδί των 56 bits επεκτείνεται, ένα δεύτερο υποκλειδί των 48 bits παράγεται για κάθε μία από τις 16 επαναλήψεις του DES.

Αρχικά το κλειδί των 56 bits διαιρείται στη μέση. Στη συνέχεια στα δυαδικά ψηφία αυτών γίνεται αριστερή κυκλική μετατόπιση ενός ή δύο bits, ανάλογα με την επανάληψη. Αυτή η μετατόπιση γίνεται με τη βοήθεια του πίνακα

Κύκλος	Μετατόπιση	Κύκλος	Μετατόπιση
1	1	9	1
2	1	10	2
3	2	11	2
4	2	12	2
5	2	13	2
6	2	14	2
7	2	15	2
8	2	16	1

Σχήμα 7 **Ολίσθηση Γύρου** (**Αριστερή Ολίσθηση**)



Σχήμα 8 **Χρήση κλειδιού**

5	5	4	3	2	1	1	2
8	0	2	4	6	8	0	
6	5	4	3	2	2	1	4
0	2	4	6	8	0	2	
6	5	4	3	3	2	1	6
2	4	6	8	0	2	4	
6	5	4	4	3	2	1	8
4	6	8	0	2	4	6	
5	4	4	3	2	1	9	1
7	9	1	3	5	7		
5	5	4	3	2	1	1	3
9	1	3	5	7	9	1	
6	5	4	3	2	2	1	5
1	3	5	7	9	1	3	
6	5	4	3	3	2	1	7
3	5	7	9	1	3	5	

Σχήμα 9 **Αρχική μετάθεση** P,IP

2.3.2 **Μετάθεση κλειδιού**

Μετά τη μετατόπιση, τα δύο κομμάτια ενώνονται και από τα 56 bits, αφού μετατεθούν (Σχήμα 10), επιλέγονται τα 48. Εξαιτίας αυτής της λειτουργίας μετατίθεται η σειρά των bits και γίνεται επιλογή ενός υποσυνόλου από bits το οποίο καλείται συμπιεσμένη μετάθεση. Αυτή η λειτουργία παράγει ένα υποσύνολο των 48 bits. Ο πίνακας παρακάτω καθορίζει τη συμπιεσμένη μετάθεση (επίσης καλείται μετατιθέμενη επιλογή). Για παράδειγμα, το bit στη θέση 33 του κλειδιού με τα μετατοπισμένα bits μετακινείται στη θέση 35 και το bit στη θέση 18 του κλειδιού με τα μετατοπισμένα bits αγνοείται (Σχήμα 9).

Εξαιτίας της μετατόπισης, ένα διαφορετικό υποσύνολο από bits κλειδιού χρησιμοποιείται σε κάθε υποκλειδί. Κάθε bit χρησιμοποιείται σε περίπου 14 από τα 16 υποκλειδιά, παρόλα αυτά δεν χρησιμοποιούνται όλα τα bits ίσες φορές.

5	4	4	3	2	1	
7	9	1	3	5	7	9
	5	5	4	3	2	1
1	8	0	2	4	6	8
1		5	5	4	3	2
0	2	9	1	3	5	7
1	1		6	5	4	3
9	1	3	0	2	4	6
6	5	4	3	3	2	1

3	5	7	9	1	3	5
	6	5	4	3	3	2
7						
	2	4	6	8	0	2
1		6	5	4	3	2
	6					
4		1	3	5	7	9
2	1		2	2	1	
		5				4
1	3		8	0	2	

Σχήμα 10 **Μετάθεση κλειδιού**

2.3.3 **Η μετάθεση επέκτασης**

Αυτή η λειτουργία επεκτείνει το δεξιό μισό των δεδομένων, R_i, από 32 σε 48 bits. Επειδή αυτή η λειτουργία αλλάζει τη σειρά των bits και επαναλαμβάνει κάποια bits, είναι γνωστή ως μετάθεση επέκτασης. Ο πινακας παρακάτω (Σχήμα 13) καθορίζει τη μετάθεση επέκτασης. Αυτή η μετάθεση καλείται E-box (expansion box). Για κάθε τμήμα εισόδου των 4 bits, καθένα από τα bits της πρώτης και τέταρτης θέσης αντιπροσωπεύει δύο bits του block εξόδου, ενώ καθένα από τα bits της δεύτερης, και τρίτης θέσης αντιπροσωπεύει ένα bit από το block εξόδου. Ο πίνακας παρακατω δείχνει ποιες από τις θέσεις εξόδου αντιστοιχούν σε ποιες από τις θέσεις εισόδου. Για παράδειγμα, το bit στη θέση 3 του block εισόδου μετακινείται στη θέση 4 του block εξόδου και το bit στη θέση 21 του block εισόδου μετακινείται στις θέσεις 30 και 32 του block εξόδου.

Παρόλο που το block εξόδου είναι μεγαλύτερου μήκους από το block εισόδου, κάθε block εισόδου παράγει ένα μοναδικό block εξόδου.

1	1	1	2	1	5
4	7	1	4		
3	2	1	6	2	1
	8	5		1	0
2	1	1	4	2	8
3	9	2		6	
1	7	2	2	1	2
6		7	0	3	
4	5	3	3	4	5
1	2	1	7	7	5
3	4	5	4	3	4
0	0	1	5	3	8
4	4	3	5	3	5
4	9	9	6	4	3
4	4	5	3	2	3
6	2	0	6	9	2

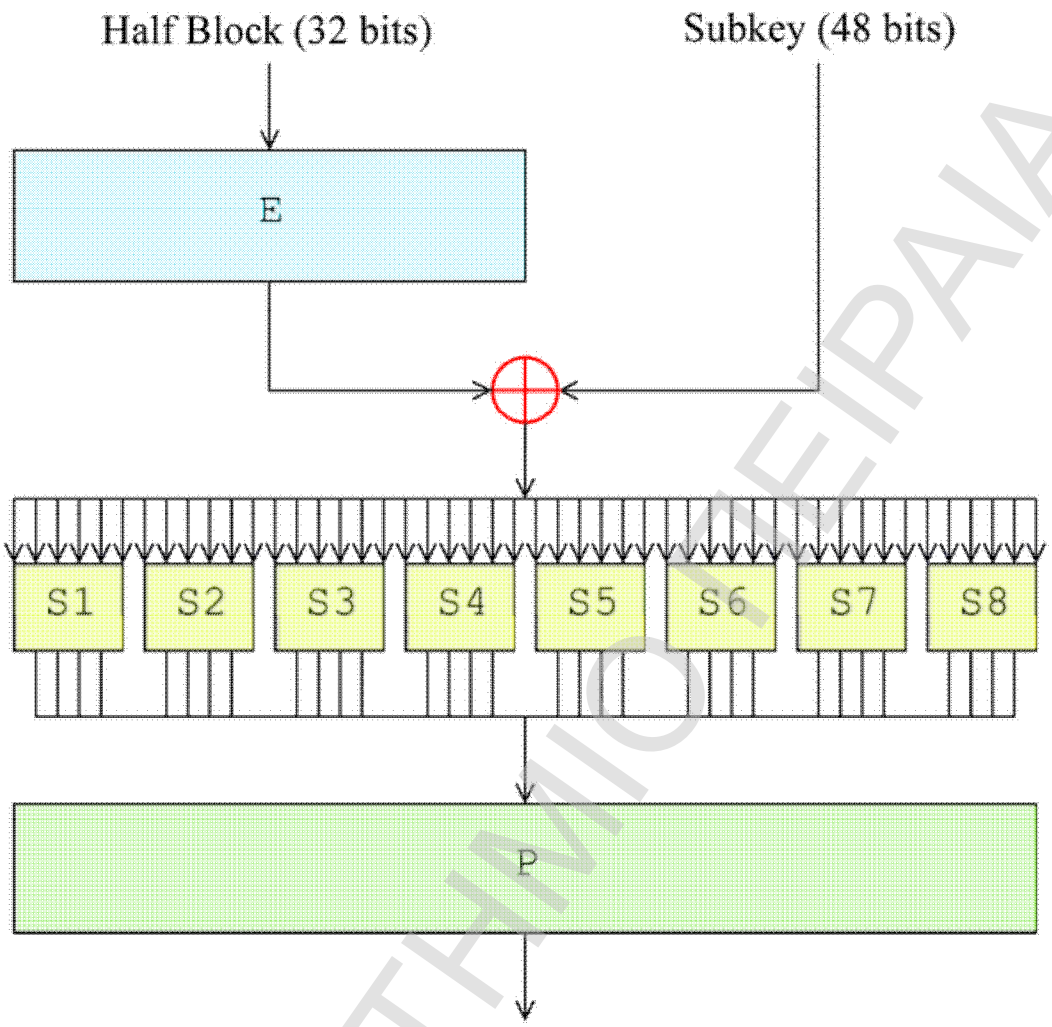
Σχήμα 11 **Μετάθεση συμπύκνωσης κλειδιού**

2.3.4 **Αντικατάσταση** S-Box

Στη συνέχεια το συμπιεσμένο κλειδί υποβάλλεται σε μια πράξη XOR με το ανεπτυγμένο (διευρυμένο) τμήμα και το αποτέλεσμα των 48 bits ωθείται σε μια πράξη αντικατάστασης. Οι αντικαταστάσεις πραγματοποιούνται με τη βοήθεια οκτώ κουτιών αντικατάστασης, ή S-Boxes. Κάθε S-Box έχει μία είσοδο των 6 bits και μία έξοδο των 4 bits, και υπάρχουν οκτώ διαφορετικά S-Boxes. Η συνολική απαίτηση μνήμης για τα οκτώ S-Boxes του DES είναι 256 bytes. (Πίνακες 1 εως 8)

Τα 48 bits διαιρούνται σε οκτώ υποσύνολα των 6 bits. Κάθε ανεξάρτητο block χειρίζεται

από ένα ανεξάρτητο S-Box: Το πρώτο block το χειρίζεται το S-Box 1, το δεύτερο block το χειρίζεται το S-Box 2 κ.ο.κ.



Σχήμα 12 **Σχηματική αναπαράσταση κάθε βήματος**

Κάθε S-Box είναι ένας πίνακας με 4 γραμμές και 16 στήλες (Σχήμα 13). Κάθε καταχώρηση στο κουτί είναι ένας αριθμός των 4 bits. Η είσοδος των 6 bits του S-Box καθορίζει την ακριβή θέση του αριθμού των 4 bits που θα έχουμε στην έξοδο, δηλαδή τον αριθμό της γραμμής και της στήλης που πρέπει να κοιτάξουμε. Ο πίνακας 3 παρακατω δείχνει τα οκτώ S-Boxes.

Τα bits εισόδου καθορίζουν μία καταχώρηση στο S-Box με έναν πολύ ειδικό τρόπο. Θεωρούμε ένα S-Box με 6 bits εισόδου, τα $b_1, b_2, b_3, b_4, b_5, b_6$. Τα bits b_1

και b_6 ενώνονται για τον σχηματισμό ενός αριθμού των 2 bits, από το 0 μέχρι το 3, ο οποίος αντιστοιχεί σε μια γραμμή του πίνακα. Τα υπόλοιπα 4 bits, από το b_2 μέχρι το b_5 ενώνονται για να σχηματίσουν έναν αριθμό των 4 bits, από το 0 μέχρι το 15, ο οποίος αντιστοιχεί σε μία στήλη του πίνακα.

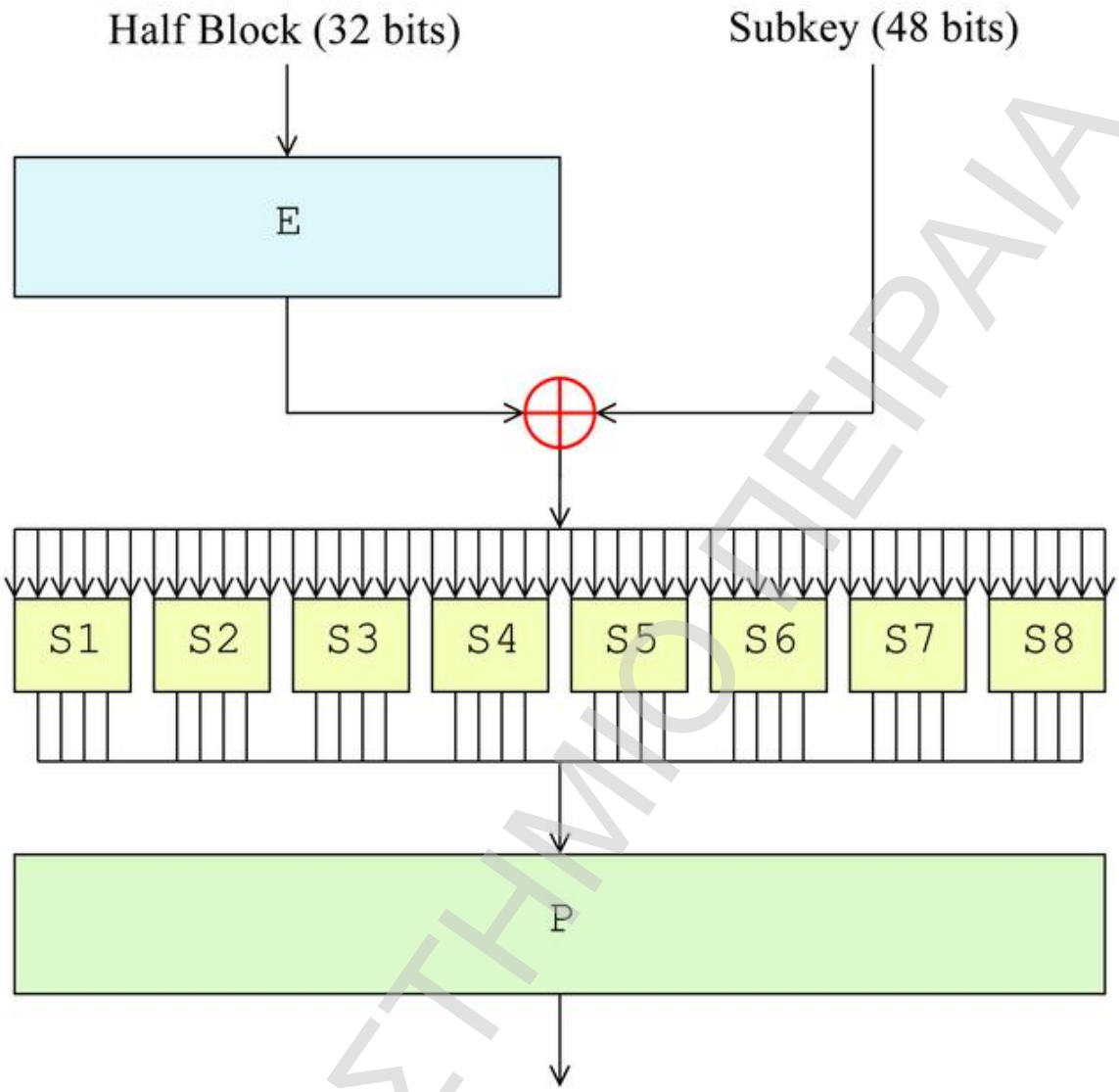
Για παράδειγμα υποθέτουμε ότι η είσοδος στο έκτο S-Box (δηλ. τα bits από 31 μέχρι 36 από το αποτέλεσμα της πράξης XOR την οποία αναφέραμε πριν) είναι 110011. Το πρώτο και το τελευταίο bit ενώνονται για να σχηματίσουν τον αριθμό των 2 bits 11(=3), ο οποίος αντιστοιχεί στην τρίτη γραμμή του έκτου S-Box. Τα μεσαία 4 bits ενώνονται για να σχηματίσουν τον αριθμό των 4 bits 1001(=9), ο οποίος αντιστοιχεί στην ένατη στήλη του έκτου S-Box. Η καταχώρηση που αντιστοιχεί στην τρίτη γραμμή και στην ένατη στήλη του έκτου S-Box είναι ο αριθμός 14. (Μετράμε πάντα τις σειρές και τις στήλες ξεκινώντας από το 0 και όχι από το 1). Ο αριθμός 14 με τη μορφή δυαδικού αριθμού των 4 bits γράφεται 1110. Επομένως ο δυαδικός αριθμός 1110 αντικαθιστά τον δυαδικό αριθμό 110011.

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13

12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Σχήμα 13 **Μετάθεση επέκτασης**

Είναι φυσικά, πολύ πιο εύκολο να εφαρμόσουμε τα S-Boxes σε λογισμικό με πίνακες των 64 καταχωρήσεων. Παρόλα αυτά, αυτός ο τρόπος περιγραφής των S-Boxes βοηθάει να σχηματίσουμε καθαρή εικόνα για το πώς δουλεύουν. Κάθε S-Box ουσιαστικά αποτελεί μια συνάρτηση αντικατάστασης μιας καταχώρησης των 4 bits, τα bits από το b_2 μέχρι το b_5 μπαίνουν μέσα, και βγαίνει ένα αποτέλεσμα των 4 bits. Τα bits b_1 και b_6 προέρχονται από τα γειτονικά block, διαλέγουν μία έξοδο από τις τέσσερις συναρτήσεις αντικατάστασης που είναι διαθέσιμες σε ένα συγκεκριμένο S-Box.



Η **S-Box** αντικατάσταση είναι το κρίσιμο βήμα στον **DES**. Οι άλλες λειτουργίες του αλγορίθμου είναι γραμμικές και εύκολες στην ανάλυση. Τα **S-Boxes** δεν είναι γραμμικά και, περισσότερο από οτιδήποτε άλλο, εξασφαλίζουν την ασφάλεια του **DES**.

Το αποτέλεσμα αυτής της φάσης της αντικατάστασης είναι οκτώ **blocks** των **4 bits** που ξαναενώνονται σε ένα μονό **block** των **32 bits**. Αυτό το **block** ωθείται στο επόμενο βήμα: στην μετάθεση **P-Box**.

2.3.5 **Η μετάθεση** P-Box

Η έξοδος των **32 bits** από την αντικατάσταση του **S-Box** μετατίθεται σύμφωνα με το **P-Box**. Αυτή η μετάθεση αντιστοιχίζει κάθε **bit** εισόδου σε μία θέση εξόδου. Κανένα **bit** δεν χρησιμοποιείται δύο φορές και κανένα **bit** δεν αγνοείται. Αυτό καλείται μια απευθείας μετάθεση ή απλά μια μετάθεση. Ο πίνακας **11** παρακάτω δείχνει τη θέση στην οποία μετακινείται κάθε **bit**. Για παράδειγμα, το **21ο bit** μετακινείται στην **4η** θέση , ενώ το **4ο bit** μετακινείται στην **31η** θέση.

S_1															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Πίνακας 1: Sbox 1

S_2															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10

3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

Πίνακας 2 Sbox 2

S_3															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

Πίνακας 3 Sbox 3

S_4															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

Πίνακας 4 Sbox 4

S_5															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Πίνακας 5 Sbox 5

S_6															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

Πίνακας 6 Sbox 6

S_7															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

Πίνακας 7 Sbox 7

S_8															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Πίνακας 8 Sbox 8

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Σχήμα 14 **Μετάθεση** P-box

Τελικά, το αποτέλεσμα της μετάθεσης του P-Box (Σχήμα 14) συμμετέχει σε μια πράξη XOR με το αριστερό μισό από το αρχικό block των 64 bits. Στη συνέχεια το αριστερό και το δεξιό μισό ενώνονται και ακόμα μία επανάληψη αρχίζει.

2.3.6 **Η τελική μετάθεση**

Η τελική μετάθεση είναι αντίστροφη της αρχικής μετάθεσης και γίνεται με τη βοήθεια του πίνακας 9. Σημειώνουμε ότι το αριστερό και το δεξιό μισό δεν ανταλλάσσονται μετά από την τελευταία επανάληψη του DES. Αντίθετα το συνενωμένο block $R_{16}L_{16}$ χρησιμοποιείται ως είσοδος στην τελική μετάθεση. Εδώ δεν συμβαίνει τίποτα. Ανταλλάσσοντας τα μισά (αριστερό και δεξιό)

και ολισθαίνοντάς τα κυκλικά, η μετάθεση θα απέφερε ακριβώς το ίδιο αποτέλεσμα. Για το λόγο αυτό ο αλγόριθμος μπορεί να χρησιμοποιηθεί τόσο στην κρυπτογράφηση όσο και στην αποκρυπτογράφηση.

2.4 Αποκρυπτογράφηση με το DES

Μετά από όλες αυτές τις αντικαταστάσεις, τις μεταθέσεις, τις πράξεις XOR και τις κυκλικές ολισθήσεις μπορεί να φαίνεται ότι ο αλγόριθμος αποκρυπτογράφησης είναι τελείως διαφορετικός σε σχέση με τον αλγόριθμο κρυπτογράφησης. Απεναντίας, οι ποικίλες αυτές συναρτήσεις διαδεχτήκαν η μια την άλλη για να παραγάγουν μία πολύ χρήσιμη ιδιότητα: ο ίδιος αλγόριθμος να μπορεί να δουλεύει τόσο για την κρυπτογράφηση όσο και για την αποκρυπτογράφηση.

Με τον DES χρησιμοποιήται η ίδια λειτουργία για να κρυπτογραφήσουμε ή για να αποκρυπτογραφήσουμε ένα τμήμα. Η μόνη διαφορά είναι ότι τα κλειδιά πρέπει να χρησιμοποιούνται με την αντίστροφη σειρά. Δηλαδή, αν τα κλειδιά κρυπτογράφησης για κάθε επανάληψη είναι $k_1, k_2, k_3, ..., k_{16}$, τότε τα κλειδιά αποκρυπτογράφησης είναι $k_{16}, k_{15}, k_{14}, ..., k_1$. Ο αλγόριθμος, ο οποίος παράγει το κλειδί που χρησιμοποιείται σε κάθε επανάληψη, είναι κυκλικός. Η ολίσθηση του κλειδιού είναι μία δεξιά ολίσθηση και ο αριθμός των θέσεων που ολισθαίνει το κλειδί σε κάθε επανάληψη είναι: 0,1,2,2,2,2,2,2,1,2,2,2,2,2,2,1.

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Πίνακας 9 Τελική μετάθεση

2.5 Ασφάλεια του DES

Οι καθηγητές Diffie και Hellman υποστηρίζουν ότι η χρήση κλειδιού των 56 bits δεν μπορεί να είναι αποτελεσματική σε μια κρυπταναλυτική προσπάθεια με χρήση υπολογιστή μεγάλης ταχύτητας. Αναφέρουν, μάλιστα, ότι αν χρησιμοποιηθεί υπολογιστικό σύστημα με 106 υψηλής ολοκλήρωσης κυκλώματα (LSI), τότε η δοκιμή των 256 (10^{16}) δυνατών κλειδιών είναι δυνατόν να γίνει σε μία, μόνο, μέρα. Αυτό εξηγείται από το γεγονός ότι κάθε ένα από τα ολοκληρωμένα κυκλώματα μπορεί να ελέγχει ένα κλειδί ανά μsec, ή 10^{10} κλειδιά την ημέρα. Συνεπώς, αφού ένα τέτοιο κύκλωμα χρειάζεται για τον έλεγχο όλων των κλειδιών 106 ημέρες, τότε η χρήση 106

κυκλωμάτων περιορίζει τον απαιτούμενο χρόνο σε μία ημέρα. Το κόστος μιας τέτοιας μηχανής, όπως υποστήριξαν οι καθηγητές, ήταν περίπου 20 εκατομμύρια δολάρια ΗΠΑ εκείνη την εποχή.

Σύμφωνα με τις εκτιμήσεις αυτές υπολογίζουν ότι σε μερικά χρόνια (που πρέπει να έχουν έρθει ,δεδομένου ότι η εργασία αυτή ανακοινώθηκε το 1977), το κόστος λειτουργίας ενός τέτοιου υπολογιστή είναι περίπου 10.000 την ημέρα. Δεδομένου ότι δεν θα απαιτηθεί η δοκιμή του συνόλου των δυνατών κλειδιών, αλλά, στατιστικά μιλώντας, μόνο των μισών, τότε απαιτείται κόστος 5000 και χρόνος μισής ημέρας, για τον εντοπισμό ενός κλειδιού.

Το 1981 ο καθηγητής Diffie, με νέους υπολογισμούς, ανακοίνωσε ότι η χρήση της τεχνολογίας της εποχής εκείνης, απαιτείται υπολογιστής κόστους 50 εκατομμυρίων δολαρίων και χρόνος 2 ημερών για τον εντοπισμό ενός κλειδιού.

Στις 13 Ιανουαρίου του 2007 η βασισμένη σε FPGA παράλληλη μηχανή COPACOBANA από τα Πανεπιστήμια του Bochum και Kiel της Γερμανίας, σπάει τον αλγόριθμο του DES σε 7.2 ημέρες με κόστος υλικού 10.000 δολάρια.

2.6 Τα πραγματικά κριτήρια σχεδιασμού

Λίγο μετά την παρουσίαση της διαφορικής κρυπτανάλυση [6], η IBM δημοσίευσε τα κριτήρια σχεδιασμού για τα S-boxes και για το P-box, προκειμένου να δείξει πως ο DES ήταν ασφαλής από τέτοιου είδους επιθέσεις. Τα κριτήρια για τα S-box είναι:

- Κάθε S-box έχει 6 ψηφία εισόδου και 4 ψηφία εξόδου.(Αυτό ήταν και το μεγαλύτερο μέγεθος που μπορούσε να προσαρμοσθεί σε ένα chip τεχνολογίας του 1974.)
- Κανένα ψηφίο εξόδου ενός S-box δεν έπρεπε να είναι πολύ κοντά σε μία γραμμική συνάρτηση των ψηφίων εισόδου.
- Αν καθοριστεί το αριστερότερο και το δεξιότερο ψηφίο ενός S-box και παραλλάχθούν τα τέσσερα μεσαία ψηφία, τότε κάθε πιθανή έξοδος 4 ψηφίων πραγματοποιείται ακριβώς μία φορά.
- Αν δύο είσοδοι σε ένα S-box διαφέρουν σε ακριβώς ένα ψηφίο, οι έξοδοι πρέπει να διαφέρουν σε δύο ψηφία το λιγότερο.
- Αν δύο είσοδοι σε ένα S-box διαφέρουν σε δύο μεσαία ψηφία ακριβώς, οι έξοδοι πρέπει να διαφέρουν σε δύο ψηφία το λιγότερο.
- Αν δύο είσοδοι σε ένα S-box διαφέρουν στα δύο πρώτα ψηφία τους και είναι όμοιες στα δύο τελευταία ψηφία τους, τότε οι δύο έξοδοι δεν πρέπει να είναι όμοιες.
- Για κάθε μη-μηδενική διαφοροποίηση 6 ψηφίων μεταξύ των εισόδων, όχι περισσότερα από 8 από τα 32 ζευγάρια εισόδου, που

επιδεικνύουν αυτήν την διαφορά, ίσως να καταλήγουν στην ίδια διαφοροποίηση εξόδου.

Στην συνέχεια δόθηκαν και τα παρακάτω κριτήρια για τα S-boxes:

- Τα 4 ψηφία εξόδου από κάθε S κουτί στην i επανάληψη κατανέμονται κατά τέτοιον τρόπο ώστε τα 2 από αυτά να επηρεάζουν τα μεσαία ψηφία των S κουτιών στην i+1 επανάληψη και τα άλλα 2 να επηρεάζουν τα ψηφία τέλους.
- Τα 4 ψηφία εξόδου από κάθε S κουτί επηρεάζουν έξι διαφορετικά S κουτιά, και η διαφοροποίηση δύο ψηφίων ψηφία δεν επηρεάζει το ίδιο S κουτί.
- Αν το ψηφίο εξόδου από ένα S κουτί επηρεάζει ένα μεσαίο ψηφίο ενός άλλου S κουτιού, τότε ένα ψηφίο εξόδου από αυτό το άλλο S κουτί δεν μπορεί να επηρεάσει ένα μεσαίο ψηφίο από το πρώτο S κουτί.

Τέλος, πρέπει να προσθέσουμε ότι η παραγωγή των S-κουτιών στις μέρες μας είναι αρκετά εύκολη υπόθεση, αλλά ήταν εξαιρετικά πολύπλοκη εργασία στις αρχές της δεκαετίας του 1970. Ο Tuchman αναφέρει ότι έτρεχαν προγράμματα επί μήνες προκειμένου να παραγάγουν τα S κουτιά.

2.7 Αλγεβρική κρυπτανάλυση του DES

Σε μία σχετικά πρόσφατη εργασία του ο Nicolas Courtois [2] πρότεινε μια αλγεβρική επίθεση στους αλγόριθμους κρυπτογράφησης, μέσω εκτεταμμένης γραμμικοποίησης των συστημάτων εξισώσεων, την οποία μάλιστα προτείνει ως επίθεση και στον AES. Στην παρούσα διατριβή, με αφορμή την ιδέα αυτή, προσπαθήσαμε να αναζητήσουμε μια μέθοδο κρυπτανάλυσης για το DES [48]. Η επιλογή του DES έγινε αρχικά λόγω του μικρού σχετικά κλειδιού που διαθέτει, 56bits, και το οποίο τον κάνει θεωρητικά πολύ πιο ευάλωτο σε σχέση με άλλους αλγορίθμους που χρησιμοποιούνται σήμερα. Ένα επιπλέον κριτήριο ήταν η αρκετά μεγάλη πόλωση την οποία παρουσιάζουν τα Sbox του.

Η επίλυση ενός μη γραμμικού συστήματος, δεν μπορεί να θεωρηθεί σε καμία των περιπτώσεων εύκολη. Το αλγεβρικό μάλιστα σύστημα το οποίο προκύπτει από τον DES είναι πάρα πολύ δύσκολο να επιλυθεί. Στην προσπάθεια να βρούμε έναν τρόπο επίλυσης, οδηγηθήκαμε στην απλοποίηση του αλγεβρικού του σύστηματος μέσω προσεγγίσεων των S-Box.

Η βασική ιδέα της επίθεσης είναι αρχικά μια αλγεβρική μοντελοποίηση του DES η οποία αναμένεται να μας δώσει ένα «καλό» σύστημα εξισώσεων. Η πιο απλή μοντελοποίηση, η οποία μπορεί να γίνει, είναι μέσω γραμμικών εξισώσεων. Αρχικά γίνεται μια αλγεβρική προσέγγιση των S-box και χρησιμοποιώντας αυτές τις γραμμικές προσεγγίσεις, κατασκευάζουμε ένα

πολύ πιο απλό σύστημα εξισώσεων. Δεδομένου ότι τώρα το σύστημά είναι γραμμικό, η επίλυσή του μπορεί να καταστεί πολύ πιο εφικτή. Είναι προφανές ότι τα συστήματα αυτά μπορεί πολλές φορές να μην είναι επιλύσιμα, αφού προέρχονται από προσεγγίσεις και για αυτό το λόγο θα κάνουμε ορισμένες παραδοχές.

2.7.1 **Μοντελοποίηση του** DES

Το μόνο μη γραμμικό μέρος του **DES**, όπως έχουμε δει από την περιγραφή του είναι τα **S-box** που περιέχει. Αυτό σημαίνει πως, αν βρεθεί μια αδυναμία σε αυτά, αυτομάτως, η ο **DES** θα καταστεί μη ασφαλής. Η κύρια ιδέα της επίθεσης είναι η προσέγγιση των **Sbox** μέσω γραμμικών εξισώσεων.

Το κάθε **S-box** παίρνει ως είσοδο **6 bit** και έχει σαν έξοδο **4 bit**. Συνεπώς, κάθε **S-box** μπορεί να θεωρηθεί ως συνάρτηση με πεδίο ορισμού το $GF(2^6)$ και πεδίο τιμών το $GF(2^4)$. Αυτό λοιπόν σημαίνει πως, αν αναζητούμε μια γραμμική προσέγγιση της συνάρτησης αυτής, έχουμε 2^7 υποψήφιες για κάθε διαφορετικό **bit** σε κάθε **bit** εξόδου του **S-box**.

Έτσι λοιπόν μπορούμε να θεωρήσουμε πως κάθε **bit** εξόδου, είναι αποτέλεσμα μιας γραμμικής συνάρτησης με πεδίο ορισμού το $GF(2^6)$ και πεδίο τιμών το $GF(2)$. Με χρήση του κώδικα που δίνεται στο παράρτημα μπορούμε να δούμε πόσες φορές μια γραμμική εξίσωση στο $GF(2^6)$ μας δίνει τη σωστή έξοδο για τη συνάρτηση **f**. Οι εξισώσεις, οι οποίες θα έχουν

το μεγαλύτερο ποσοστό επιτυχίας, προφανώς, θα θεωρούνται ως οι καλύτερες προσεγγίσεις. Παίρνοντας λοιπόν αυτές τις εξισώσεις, καταλήγουμε στις εξισώσεις που θα χρησιμοποιηθούν στην επίθεση.

Ο παρακάτω πίνακας δείχνει την πόλωση των S-box του DES σε σχέση με κάποια γραμμική προσέγγιση.

Bit	Percentage	Bit	Percentage
1	71.9 %	17	68.8%
2	65.6%	18	68.8%
3	78.8%	19	71.9%
4	71.9%	20	68.8%
5	71.9%	21	78.1%
6	71.9%	22	71.9%
7	68.8%	23	62.5%
8	65.6%	24	75%
9	68.8%	25	68.8%
10	68.8%	26	68.8%
11	65.6%	27	68.8%
12	62.5%	28	68.8%
13	68.8%	29	65.6%
14	71.9%	30	68.8%
15	65.6%	31	65.6%
16	75%	32	68.8%

2.7.2 **Το σύστημα των εξισώσεων**

Ας δούμε τώρα τα συστήματα των εξισώσεων και τον τρόπο επίλυσής τους. Στο πρώτο βήμα κρυπτογραφούμε το κείμενό μας χρησιμοποιώντας ένα αυθαίρετο κλειδί το $k_1, k_2, ..., k_{56}$ και τις προαναφερθείσες εξισώσεις, οι οποίες παρουσιάζονται στους πίνακες:[] αντί των κανονικών S-boxes. Η έξοδος λοιπόν αναμένεται ότι θα είναι 64 γραμμικές παραστάσεις, οι οποίες θα χρησιμοποιούν τα $k_1, k_2..., k_{56}$.

Παίρνουμε τώρα κάθε μία από αυτές τις παραστάσεις και την εξισώνουμε με κάθε bit του κρυπτογραφημένου κειμένου που έχουμε στην κατοχή μας. Αυτό σημαίνει πως την πρώτη γραμμική παράσταση την θέτουμε ίση με το πρώτο bit του κρυπτογραφημένου κειμένου, τη δεύτερη παράσταση με το δεύτερο bit και ούτω καθεξής. Αυτό ουσιαστικά μας δημιουργεί ένα σύστημα 56 μεταβλητών και 64 εξισώσεων στο GF(2).

Για την επίλυση του συστήματος των εξισώσεων, κατασκευάσαμε τον παρακάτω αλγόριθμο, που αποτελεί μια παραλλαγή της μεθόδου απαλοιφής του Gauss:

Βήμα 1. Παίρνουμε τον πίνακα του συστήματος ο οποίος έχει διαστάσεις 57x64 με στοιχεία στο 0 και 1.

Βήμα 2. Ορίζουμε τον οδηγό στην γραμμή 1 και την τρέχουσα στήλη σε 1.

Βήμα 3. Αναζητούμε την πρώτη γραμμή, η οποία να έχει 1 στην τρέχουσα στήλη.

Βήμα 4. Εναλλάσσουμε τη γραμμή αυτή με αυτήν, την οποία δείχνει ο οδηγός.

Βήμα 5. Προσθέτουμε **modulo** 2 τη γραμμή, την οποία δείχνει ο οδηγός σε κάθε άλλη γραμμή για να απαλείψουμε από την τρέχουσα στήλη οποιαδήποτε άλλη μονάδα

Βήμα 6. Αυξάνουμε τον δείκτη κατά 1

Βήμα 7. Αυξάνουμε κατά ένα την τρέχουσα στήλη.

Βήμα 8. Αν η τρέχουσα στήλη είναι μικρότερη του 57, επιστρέφουμε στο πρώτο βήμα.

Είναι προφανές ότι με την παραπάνω μέθοδο μπορεί να επιλύθει ένα γραμμικό σύστημα. Το πρόβλημα στη συγκεκριμένη περίπτωση είναι πως το σύστημά μας προέρχεται από προσεγγίσεις, κάτι το οποίο σημαίνει ότι περιέχει κάποιες «λανθασμένες» εξισώσεις. Είναι προφανές πως δεν μπορούμε να αναγνωρίσουμε αυτές τις εξισώσεις προκειμένου να τις απομονώσουμε. Έτσι λοιπόν θα πρέπει να αναγκάσουμε το σύστημα να μας επιστρέψει μια λύση. Η ιδέα είναι, πως έχοντας αρκετά ζεύγη αρχικού κειμένου και του αντίστοιχου κρυπτογραφημένου στην κατοχή μας, οι σωστές λύσεις στην κάθε μεταβλητή θα υπερτερούν των λανθασμένων.

Έτσι λοιπόν αφαιρούμε από το σύστημα κάθε γραμμή στην οποία εμφανίζονται οι συντελεστές των αγνώστων και είναι μηδενικοί, ασχέτως αν στην άλλη πλευρά έχουμε μηδέν ή μονάδα. Δεχόμαστε ότι αυτό το παράδοξο, **0=1** οφείλεται στις «λανθασμένες» εξισώσεις οι οποίες «καταστρέφουν» το γραμμικό μας σύστημα. Θα μπορούσαμε να πούμε

πως αυτό είναι ουσιαστικά ένας θόρυβος, θόρυβος ο οποίος σπάει την γραμμικότητα του συστήματός μας.

Επιπλέον αφαιρούμε κάθε εξίσωση του τελικού πίνακα η οποία περιέχει παραπάνω από έναν άγνωστο. Ο εναπομείνας πίνακας μας δίνει μια τιμή για κάποιες μεταβλητές. Δεχόμαστε λοιπόν ως λύση του συστήματος την τιμή αυτών των μεταβλητών

Ακολουθώντας τον ίδιο αλγόριθμο και στον αλγόριθμο αποκρυπτογράφησης, δημιουργούνται επιπλέον συστήματα εξισώσεων, τα οποία θα πρέπει να επιλυθούν όπως και τα προηγούμενα.

2.7.3 **Η επίθεση στον** DES

Η επίθεση βασίζεται στην επίλυση των γραμμικών συστημάτων που αναλύθηκαν παραπάνω για κάθε ζεύγος κρυπτογραφημένου και κανονικού κειμένου. Κάθε φορά ενημερώνουμε ένα μετρητή οποίος κρατάει πόσες φορές ένα bit κλειδιού βρέθηκε να έχει την τιμή μηδέν ή μονάδα. Στο τέλος αυτής της διαδικασίας θεωρούμε πως η σωστή τιμή της κάθε μεταβλητής είναι αυτή με τον μεγαλύτερο μετρητή.

Η επίθεση χωρίζεται σε τέσσερα στάδια, καθένα από τα οποία χρησιμοποιεί δικές του εξισώσεις για τη μοντελοποίηση του DES. Σε κάποια στάδια θεωρούμε δεδομένο πως κάποια bits είναι γνωστά από προηγούμενο

στάδιο, προκειμένου να βρούμε κάποια επιπλέον bit του κλειδιού. Κάθε στάδιο αποτελείται από την επίλυση δυο γραμμικών συστημάτων. Το πρώτο προέρχεται από την διαδικασία κρυπτογράφησης και το δεύτερο από την διαδικασία αποκρυπτογράφησης όταν οι γραμμικές προσεγγίσεις χρησιμοποιηθούν στην θέση των κανονικών S-box.

Οι παρακάτω πίνακες δείχνουν τις εξισώσεις που χρησιμοποιήθηκαν σε κάθε στάδιο της επίθεσης. Θεωρούμε πως το κείμενο $x_0, x_1, ...x_{31}$ είναι αυτό θα περάσει σαν είσοδος στα S-box του DES.

Bit	1	2
0	$x_0 + x_1 + x_2 + x_4 + x_5$	$1 + x_0 + x_1 + x_2 + x_4 + x_5$
1	$x_1 + x_2 + x_4 + x_5$	$1 + x_1 + x_2 + x_4 + x_5$
2	$x_0 + x_1 + x_2 + x_3 + x_4 + x_5$	$1 + x_0 + x_1 + x_2 + x_3 + x_4 + x_5$
3	$x_0 + x_1 + x_2 + x_3 + x_4 + x_5$	$1 + x_0 + x_1 + x_2 + x_3 + x_4 + x_5$
4	$1 + x_7 + x_8 + x_9 + x_{10} + x_{11}$	$x_7 + x_8 + x_9 + x_{10} + x_{11}$
5	$1 + x_6 + x_9 + x_{10}$	$1 + x_6 + x_7 + x_9 + x_{10} + x_{11}$
6	$1 + x_6 + x_7 + x_8 + x_9 + x_{10}$	$x_6 + x_7 + x_8 + x_9 + x_{10}$
7	$x_7 + x_9 + x_{11}$	$x_7 + x_9 + x_{10} + x_{11}$
8	$x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$	$1 + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$
9	$1 + x_{15} + x_{16}$	$x_{15} + x_{16}$
10	$x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$	$1 + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$
11	$x_{14} + x_{16} + x_{17}$	$x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$
12	$1 + x_{19} + x_{20} + x_{21} + x_{22} + x_{23}$	$1 + x_{19+}x_{20} + x_{21} + x_{22} + x_{23}$
13	$x_{18} + x_{19} + x_{21} + x_{22} + x_{23}$	$x_{18} + x_{19} + x_{21} + x_{22} + x_{23}$
14	$x_{19} + x_{20} + x_{21} + x_{22}$	$x_{18} + x_{19} + x_{20} + x_{21} + x_{22} + x_{23}$
15	$x_{19} + x_{21} + x_{23}$	$x_{19} + x_{21} + x_{23}$
16	$x_{24} + x_{25} + x_{26} + x_{28}$	$1 + x_{24} + x_{25} + x_{26} + x_{28}$
17	$x_{25} + x_{26} + x_{28} + x_{29}$	$x_{24} + x_{27} + x_{28}$
18	$x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$	$1 + x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$
19	$x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$	$1 + x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$
20	$x_{31} + x_{32} + x_{33} + x_{34} + x_{35}$	$1 + x_{31} + x_{32} + x_{33} + x_{34} + x_{35}$
21	$x_{30} + x_{31} + x_{33} + x_{34}$	$1 + x_{30} + x_{31} + x_{33} + x_{34}$
22	$x_{31} + x_{32} + x_{35}$	$1 + x_{30} + x_{33} + x_{34} + x_{35}$
23	$1 + x_{31} + x_{33} + x_{35}$	$x_{31} + x_{33} + x_{35}$
24	$1 + x_{36} + x_{37} + x_{38} + x_{40}$	$x_{36} + x_{37} + x_{38} + x_{40}$
25	$1 + x_{36} + x_{37} + x_{39} + x_{40}$	$x_{36} + x_{37} + x_{39} + x_{40}$
26	$1 + x_{37} + x_{38} + x_{40} + x_{41}$	$1 + x_{36} + x_{37} + x_{38} + x_{39} + x_{40}$
27	$x_{36} + x_{37} + x_{39} + x_{41}$	$x_{36} + x_{37} + x_{38}$
28	$x_{42} + x_{43} + x_{45} + x_{46}$	$x_{42} + x_{43} + x_{44} + x_{45} + x_{46} + x_{47}$
29	$1 + x_{42} + x_{43} + x_{45} + x_{46}$	$1 + x_{42} + x_{43} + x_{45} + x_{46}$
30	$x_{43} + x_{44} + x_{45} + x_{46} + x_{47}$	$1 + x_{42} + x_{44} + x_{46} + x_{47}$
31	$x_{43} + x_{44} + x_{46} + x_{47}$	$1 + x_{43} + x_{45} + x_{47}$

Bit	Stage 3 Equations	Stage 4 Equations
0	$x_0 + x_1 + x_2 + x_4 + x_5$	$x_0 + x_1 + x_2 + x_4 + x_5$
1	$x_1 + x_2 + x_4 + x_5$	$x_1 + x_2 + x_4 + x_5$
2	$x_0 + x_1 + x_2 + x3 + x_4 + x_5$	$x_0 + x_1 + x_2 + x_3 + x_4 + x_5$
3	$x_0 + x_1 + x_2 + x3 + x_4 + x_5$	$x_0 + x_1 + x_2 + x_3 + x_4 + x_5$
4	$x_7 + x_8 + x_9 + x_{10} + x_{11}$	$x_7 + x_8 + x_9 + x_{10} + x_{11}$
5	$x_6 + x_7 + x_9 + x_{10} + x_{11}$	$x_6 + x_7 + x_9 + x_{10} + +x_{11}$
6	$x_6 + x_7 + x_8 + x_9 + x_{10}$	$x_6 + x_7 + x_8 + x_9 + x_{10}$
7	$x_7 + x_9 + +x_{11}$	$1 + x_7 + x_9 + x_{11}$
8	$x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$	$x_{12} + x_{13} + x_{15} + x_{16} + x_{17}$
9	$x_{15} + x_{16}$	$x_{15} + x_{16}$
10	$1 + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$	$1 + x_{12} + x_{13} + x_{14} + x_{15} + x_{16} + x_{17}$
11	$1 + x_{14} + x_{15}$	$x_{14} + x_{15}$
12	$x_{19} + x_{20} + x_{21} + x_{22} + x_{23}$	$1 + x_{19} + x_{20} + x_{21} + x_{22} + x_{23}$
13	$1 + x_{18} + x_{19} + x_{21} + x_{22} + x_{23}$	$x_{18} + x_{19} + x_{21} + x_{22} + x_{23}$
14	$x_{19} + x_{20} + x_{23}$	$1 + x_{19} + x_{20} + x_{21} + x_{22}$
15	$1 + x_{19} + x_{21} + x_{23}$	$x_{19} + x_{21} + x_{23}$
16	$1 + x_{24} + x_{25} + x_{26} + x_{28}$	$1 + x_{24} + x_{25} + x_{26} + x_{28}$
17	$1 + x_{24} + x_{27} + x_{28}$	$1 + x_{24} + x_{27} + x_{28}$
18	$1 + x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$	$1 + x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$
19	$1 + x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$	$1 + x_{24} + x_{25} + x_{26} + x_{27} + x_{28}$
20	$x_{31} + x_{32} + x_{33} + x_{34} + x_{35}$	$1 + x_{31} + x_{32} + x_{33} + x_{34} + x_{35}$
21	$x_{30} + x_{31} + x_{33} + x_{34}$	$1 + x_{30} + x_{31} + x_{33} + x_{34}$
22	$x_{30} + x_{31} + x_{32}$	$x_{30} + x_{31} + x_{32}$
23	$1 + x_{31} + x_{33} + x_{35}$	$x_{31} + x_{33} + x_{35}$
24	$1 + x_{36} + x_{37} + x_{38} + x_{40}$	$x_{36} + x_{37} + x_{38} + x_{40}$
25	$1 + x_{36} + x_{37} + x_{39} + x_{40}$	$x_{36} + x_{37} + x_{39} + x_{40}$
26	$1 + x_{36} + x_{37} + x_{38} + x_{39} + x_{40}$	$x_{36} + x_{37} + x_{38} + x_{39} + x_{40}$
27	$x_{36} + x_{37} + x_{38}$	$1 + x_{36} + x_{37} + x_{38}$
28	$x_{42} + x_{43} + x_{45} + x_{46}$	$1 + x_{42} + x_{43} + x_{45} + x_{47}$
29	$1 + x_{42} + x_{43} + x_{45} + x_{46}$	$1 + x_{42} + x_{43} + x_{45} + x_{46}$
30	$x_{43} + x_{44} + x_{45} + x_{46} + x_{47}$	$1 + x_{42} + x_{44} + x_{46} + x_{47}$
31	$x_{43} + x_{44} + x_{46} + x_{47}$	$x_{43} + x_{44} + x_{46} + x_{47}$

- Το στάδιο 1 βρίσκει τα bits 5,19,26,33,36,47,52
- Το στάδιο 2 βρίσκει τα bits 1,5,6,10,15,19,33,36,47,52

Εφαρμόζοντας το στάδιο 2 μία ακόμη φορά, γνωρίζοντας τα bits 1 και 5 βρίσκουμε το bit 22.

- Το στάδιο 3 βρίσκει τα bits 8,12,15,26,33,36,47,50,52
- Το στάδιο 4 βρίσκει τα bits 1,5,8,19,26,29,50,52

Είναι προφανές ότι μερικά στάδια βρίσκουν κοινά bits με κάποια άλλα, όπως για παράδειγμα το στάδιο 1 βρίσκει τα bits 5,19,33,36,47,52 όπως και το 2, κάτι το οποίο μας επιβεβαιώνει την ύπαρξη κάποιας γραμμικότητας στον σχεδιασμό των S-boxes. Ο παρακάτω πίνακας συγκεντρώνει τα αποτελέσματα τις επίθεσης.

Στάδιο1	5,19,26,33,36,47,52
Στάδιο 2	1,6,10,15,22
Στάδιο 3	8,12,50
Στάδιο 4	29

Πίνακας 10 **Τα** bits **που βρίσκει ο αλγόριθμος**

2.7.4 **Αποτελέσματα**

Η αλγεβρική επίθεση αυτή παρουσιάζει αρκετά πλεονεκτήματα σε σχέση με άλλες γνωστές επιθέσεις σε κρυπτογραφικούς αλγόριθμους όπως η γραμμική ή η διαφορική κρυπτανάλυση.

Το μεγαλύτερο πλεονέκτημα αυτής της μεθόδου είναι το μικρό πλήθος ζευγών κειμένου και κρυπτογραφημένου κειμένου. Η επίθεση απαιτεί μόνο 20 Kilobytes δεδομένων, το οποίο είναι το ελάχιστο ποσό πληροφορίας για τέτοιες επιθέσεις. Θα πρέπει να τονίσουμε εδώ, πως για τον DES, οποίος έχει μελετηθεί σε βάθος, τόσο η γραμμική όσο και η διαφορική κρυπτανάλυση απαιτούν τόσο μεγάλη ποσότητα ζευγών κειμένου και κρυπτογραφημένου κειμένου ώστε η επίθεση να μην μπορεί να γίνει εφαρμόσιμη. Προκειμένου να έχει ουσιαστικά αποτελέσματα τόσο η γραμμική όσο και η διαφορική κρυπτανάλυση θα πρέπει να μιλάμε για μια παραλλαγή του DES η οποία να έχει λιγότερους γύρους από τον κανονικό.

Κάτι το οποίο είναι βασικό χαρακτηριστικό της επίθεσης είναι η ταχύτητά της. Από την στιγμή που απαιτείται επίλυση γραμμικών συστημάτων, έχουμε γραμμική πολυπλοκότητα του αλγορίθμου. Επιπλέον από τη στιγμή

που έχουμε γραμμική πολυπλοκότητα και τα δεδομένα τα οποία απαιτούνται είναι πάρα πολύ λίγα η επίθεση μπορεί να υλοποιηθεί ακόμα και από έναν παλαιάς τεχνολογίας προσωπικό υπολογιστή. Η υλοποίηση τόσο σε επίπεδο λογισμικού όσο και υλικού έχει ελάχιστες απαιτήσεις τόσο σε υπολογιστική ισχύ όσο και σε μνήμη.

Τέλος θα πρέπει να τονίσουμε ότι η επίθεση δεν βασίζεται σε επιλογή συγκεκριμένων ζευγών κειμένου και κρυπτοκειμένου, καθιστώντας την ακόμα πιο εφαρμόσιμη.

Προκειμένου να γίνει μια εφαρμογή της επίθεσης χρησιμοποιήθηκε η γλώσσα προγραμματισμού C++. Πρέπει να τονίσουμε πως στην εφαρμογή, η οποία παρατίθεται στο παράρτημα, δεν έγιναν οι απαιτούμενες αλλαγές προκειμένου ο κώδικας να δουλεύει με την μέγιστη δυνατή ταχύτητα των υπαρχόντων επεξεργαστών αλλά να γίνει όσο το δυνατό πιο σαφής η εφαρμογή του. Οι απαιτήσεις σε μνήμη και αποθηκευτικό χώρο είναι ελάχιστες και είναι ανάλογες των γνωστών ζευγών κειμένου και κρυπτογραφημένου κειμένου. Τέλος θα πρέπει να τονίσουμε πως όλες οι μετρήσεις έγιναν σε μηχανήματα με λειτουργικό σύστημα linux, και με μεταγλωττιστή τον gcc χρησιμοποιώντας κανονική βελτιστοποίση κώδικα.

1.8 Athlon XP	15sec
1.2 Celeron	16sec
1.2 Duron	16sec
1.6 Centrino	10sec
Pentium I 75Hz	34sec

Πίνακας 11 **Χαρακτηριστικά υπολογιστών που χρησιμοποιήθηκαν**

Για όλες τις παραπάνω μετρήσεις χρειάστηκαν περίπου 20 kilobytes ζευγών κειμένου και κρυπτογραφημένου κειμένου.

Κεφάλαιο 3 Η ασφάλεια του AES σε αλγεβρικές επιθέσεις

3.1 Εισαγωγή

Το πρότυπο κρυπτογράφησης AES (Advanced Encryption Standard) [28, 29] περιγράφει μια διαδικασία κρυπτογράφησης ηλεκτρονικής πληροφορίας βασισμένη στη λογική της κωδικοποίησης ομάδων δεδομένων με κάποιο μυστικό κλειδί. Έχει προτυποποιηθεί από το NIST τον Νοέμβριο του 2001, αντικαθιστώντας το πρότυπο DES και πλέον αποτελεί τον προτεινόμενο αλγόριθμο για εφαρμογές κρυπτογράφησης. Βασίζεται στον αλγόριθμο Rijndael.

Ο αλγόριθμος Rijndael είναι προσανατολισμένος σε τμήματα (block oriented cipher) ή τμήματα μεταβλητού μήκους και μεταβλητού μήκους κλειδί. Στο εγκεκριμένο πρότυπο AES ωστόσο προβλέπεται ένα συγκεκριμένο μήκος τμήματος,128 bits και κλειδιά μήκους 128, 192 και 256 bits.

Τα βασικά σχεδιαστικά κριτήρια, τα οποία έλαβαν υπόψη οι δημιουργοί του Rijndael, είναι η ανθεκτικότητα σε όλες τις γνωστές κρυπταναλυτικές μεθόδους, υψηλή ταχύτητα και συμπαγείς κώδικες σε ένα ευρύ πεδίο αρχιτεκτονικών υπολογιστών, καθώς και σχεδιαστική απλότητα. Ο αλγόριθμος Rijndael δεν έχει τη δομή Feistel, σύμφωνα με την οποία, σε

κάθε κύκλο, μέρος των bits μιας ενδιάμεσης κατάστασης απλά μετατίθενται χωρίς κάποια επεξεργασία σε άλλη θέση.

Στη συνέχεια θα αναφερόμαστε στον αλγόριθμο AES, λαμβάνοντας υπόψη τις τροποποιήσεις που επήλθαν κατά τη διαδικασία ανάπτυξης του πρότυπου AES στη βάση του Rijndael. Στα σημεία ωστόσο που ο αλγόριθμος AES διαφοροποιείται από τον Rijndael, θα επισημαίνουμε τις διαφορές αυτές ή θα παραθέτουμε και τις σχεδιαστικές επιλογές του Rijndael.

Στον αλγόριθμο AES, κάθε κύκλος αποτελείται από τρεις ισότιμους μετασχηματισμούς, οι οποίοι ονομάζονται επίπεδα. Ο χαρακτηρισμός «ισότιμος» αναφέρεται στο ότι κάθε bit αντιμετωπίζεται με τον ίδιο τρόπο. Το επίπεδο γραμμικής ανάμιξης (linear mixing layer) διασφαλίζει υψηλή διάχυση σε πολλαπλούς κύκλους. Το μη γραμμικό επίπεδο (non-linear layer) αναφέρεται στην παράλληλη εφαρμογή S-boxes, τα οποία εμφανίζουν άριστες μη γραμμικές ιδιότητες χειρίστης περίπτωσης (optimum worst-case nonlinearity properties). Το τρίτο επίπεδο, αυτό της πρόσθεσης του κλειδιού (key addition layer), αναφέρεται στη συσχέτιση του ενδιάμεσου αποτελέσματος με το αντίστοιχο υποκλειδί του κύκλου μέσω της πράξης της αποκλειστικής διάζευξης (XOR).

Το επίπεδο της πρόσθεσης (XOR) του υποκλειδιού εφαρμόζεται ήδη πριν από τον πρώτο κύκλο μετασχηματισμού, επειδή, σε άλλη περίπτωση, όλες οι άλλες πράξεις μετασχηματισμού, που προηγούνται της πρώτης

εκτέλεσης αυτού του επιπέδου ή που ακολουθούν την τελευταία του εκτέλεση, δεν συμβάλλουν στην ασφάλεια του κρυπτογραφικού συστήματος αφού μπορούν να αφαιρεθούν από την κρυπταναλυτή χωρίς γνώση του κλειδιού (όπως η αρχική και η τελική μετάθεση στον DES). Σε πολλούς κρυπτογραφικούς αλγόριθμους προβλέπονται αντίστοιχες αρχικές ή τελικές πράξεις μετασχηματισμού βασισμένες στο κλειδί, όπως στον IDEA ή τον Blowfish. Το επίπεδο γραμμικής ανάμιξης στον τελευταίο κύκλο είναι διαφορετικό σε σύγκριση με τους άλλους κύκλους. Αυτή η διαφορά σχεδιάστηκε έτσι ώστε να προκύπτουν περισσότερες ομοιότητες στη δομή της κρυπτογράφησης και της αποκρυπτογράφησης. Μια ανάλογη διαφορά για παρόμοιο σκοπό είναι και η έλλειψη της ανταλλαγής των υποτμημάτων στον τελευταίο κύκλο του DES.

Ενθαρρυμένοι από τα αποτελέσματα της αλγεβρικής επίθεσης στον DES, προσπαθήσαμε στο πλαίσιο της διατριβής, να εφαρμόσουμε την ίδια επίθεση και στον AES. Προκειμένου όμως να δούμε τα αποτελέσματα αυτής της επίθεσης, θα πρέπει να αναλύσουμε την διαδικασία κρυπτογράφησης και αποκρυπτογράφησης με τον AES

3.2 Κρυπτογράφηση

Οι διάφορες πράξεις του αλγόριθμου εφαρμόζονται στο ενδιάμεσο αποτέλεσμα του αλγορίθμου, το επονομαζόμενο και κατάσταση (state). Η κατάσταση περιγράφεται με έναν πίνακα, του οποίου τα στοιχεία είναι

bytes. Ο πίνακας αυτός έχει τέσσερις γραμμές και στήλες τόσες όσες αντιστοιχούν στο μήκος του μπλοκ του μη κρυπτογραφημένου κειμένου. Επειδή στον AES το μήκος του τμήματος εισόδου, της κατάστασης και της εξόδου είναι 128 bits, το πλήθος των στηλών είναι ίσο με 4. (Στον Rijndael το πλήθος των στηλών μπορούσε να είναι 4 ή 6 ή 8 αφού προέβλεπε μήκος τμήματος εισόδου, κατάστασης και εξόδου ίσο με 128 ή 192 ή 256 bits.) Κατά την έναρξη της εκτέλεσης της κρυπτογράφησης ή της αποκρυπτογράφησης, τα 128 bits της εισόδου αντιγράφονται στον πίνακα κατάστασης. Η τελική τιμή του πίνακα κατάστασης αποτελεί την έξοδο της κρυπτογράφησης ή της αποκρυπτογράφησης.

Με πίνακα τεσσάρων γραμμών παριστάνεται και το (υπο)κλειδί κάθε κύκλου. Όσο για το πλήθος των στηλών, αυτό εξαρτάται από το μήκος του κλειδιού. Αν το μήκος είναι 128 bits, τότε ο αριθμός των στηλών του πίνακα είναι 4, ενώ αν το μήκος του κλειδιού είναι 192 ή 256 bits, ο αριθμός των στηλών του πίνακα είναι 6 ή 8 αντίστοιχα.

Τον αριθμό των στηλών του πίνακα κατάστασης τον συμβολίζουμε με Nb και του πίνακα του κλειδιού με Nk. Στο Σχήμα 1 φαίνεται ένα παράδειγμα πίνακα κατάστασης για Nb=4 και πίνακα κλειδιού για Nk=6.

$$\begin{matrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \end{matrix}$$

$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$		
$k_{0,0}$	$k_{0,1}$	$k_{0,2}$	$k_{0,3}$	$k_{0,4}$	$k_{0,5}$
$k_{1,0}$	$k_{1,1}$	$k_{1,2}$	$k_{1,3}$	$k_{1,4}$	$k_{1,5}$
$k_{2,0}$	$k_{2,1}$	$k_{2,2}$	$k_{2,3}$	$k_{2,4}$	$k_{2,5}$

Πίνακας 12 **Παράδειγμα πίνακα κατάστασης** (Nb=4) **και πίνακα κλειδιού** (Nk=6)

Το πλήθος των κύκλων (Nr), που εκτελούνται κάθε φορά, εξαρτάται από το μήκος του κλειδιού. Κυμαίνεται δε από 10 έως 14. Στον AES ισχύουν οι ακόλουθοι συνδυασμοί μήκους κλειδιού, τμήματος και πλήθους κύκλων (Πίνακας 12).

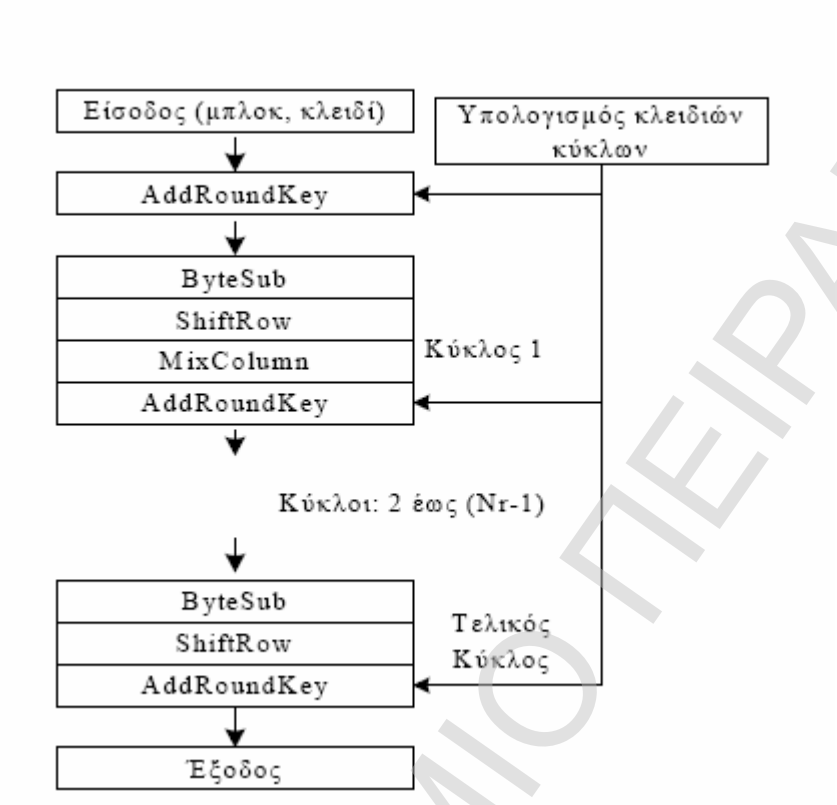
	Μήκος Κλειδιού (Nk)	*Μήκος Τμήματος (Nb)*	*Πλήθος Κύκλων (Nr)*
AES-128	*4*	*4*	*10*
AES-192	*6*	*4*	*12*
AES-256	*8*	*4*	*14*

Πίνακας 13 **Το πλήθος των κύκλων** (Nr) **ως συνάρτηση του** Nk

(Ο Rijndael προέβλεπε τη σχέση μεταξύ των **Nk**, **Nb** και **Nr**, που φαίνεται στον Πίνακα 13.)

	Nb=4	*Nb=6*	*Nb=8*
Nk=4	*10*	*12*	*14*
Nk=6	*12*	*12*	*14*
Nk=8	*14*	*14*	*14*

Πίνακας 14 **Ο αριθμός των κύκλων** (Nr) **στον** Rijndael **ως συνάρτηση του** Nb **και του** Nk



Σχήμα 15 **Η λειτουργία κρυπτογράφησης του** AES

Τα τρία επίπεδα μετασχηματισμού, που αποτελούν κάθε κύκλο του αλγόριθμου, συγκεκριμενοποιούνται στους εξής τέσσερις επιμέρους μετασχηματισμούς:

- SubBytes, που είναι η μη γραμμική αντικατάσταση.
- ShiftRows, που αναφέρεται στην ολίσθηση γραμμών του πίνακα κατάστασης (επίπεδο γραμμικής ανάμιξης, διάχυση μεταξύ των στηλών).

- MixColumns, που εφαρμόζεται σε κάθε στήλη του πίνακα κατάστασης ξεχωριστά (επίπεδο γραμμικής ανάμιξης, διάχυση μεταξύ των bytes μιας στήλης).
- AddRoundKey, που είναι το επίπεδο της πρόσθεσης του (υπο)κλειδιού του κύκλου.

Στο Σχήμα 15 παρουσιάζεται η λειτουργία κρυπτογράφησης του AES. Είσοδος της λειτουργίας κρυπτογράφησης του αλγόριθμου είναι το μπλοκ του καθαρού κειμένου και το κρυπτογραφικό κλειδί.

Στη συνέχεια θα δούμε τους τέσσερις βασικούς μετασχηματισμούς του AES: SubBytes, ShiftRows, MixColumns και AddRoundKey. Ο τρόπος υπολογισμού των κλειδιών των κύκλων (υποκλειδιών) θα μας απασχολήσει στην συνέχεια.

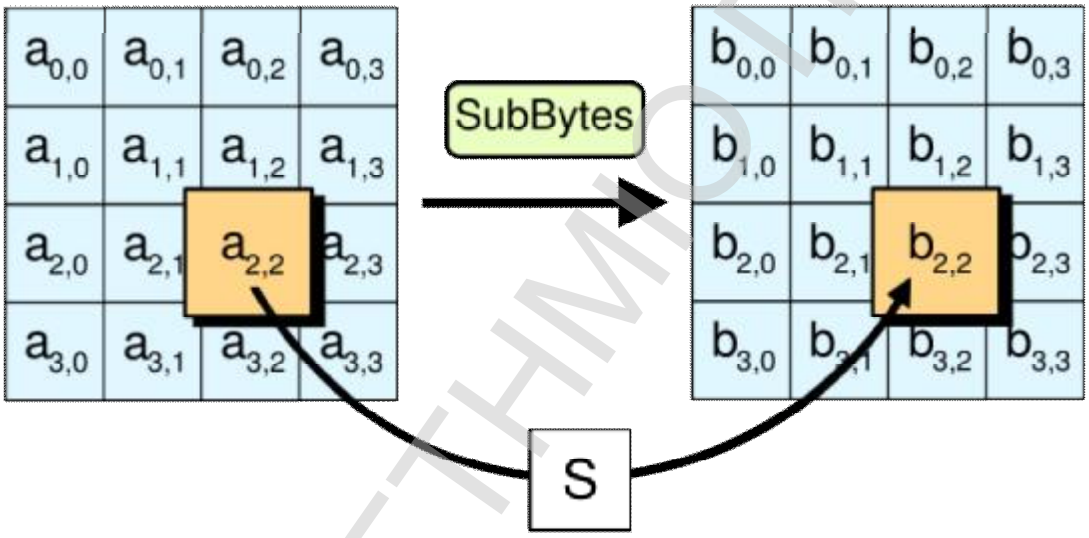
3.2.1 **Μετασχηματισμός** SubBytes

Ο μετασχηματισμός αυτός, ο οποίος αντιστοιχεί στα Sboxes του DES, εφαρμόζεται ανεξάρτητα σε κάθε byte της κατάστασης ως εξής:

- Πρώτα υπολογίζεται ο πολλαπλασιαστικός αντίστροφος του byte που πρόκειται να αντικατασταθεί στο πεπερασμένο σώμα $GF(2^8)$.
- Το αποτέλεσμα του βήματος 1 υποβάλλεται στον ακόλουθο γραμμικό μετασχηματισμό GF(2). (Σχήμα 16)

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Σχήμα 16 **Γραμμικός μετασχηματισμός**



Σχήμα 17 **Εφαρμογή του μετασχηματισμού** SubBytes

Με τη βοήθεια του ανωτέρω μετασχηματισμού ένα **byte a**, αφού υπολογιστεί ο αντίστροφός του **x** στο πεπερασμένο σώμα $GF(2^8)$, αντικαθίσταται από το **byte y**. Για τον υπολογισμό του **x** από το **y**, δηλαδή για την αντιστροφή του ανωτέρω μετασχηματισμού που είναι απαραίτητη στην αποκρυπτογράφηση, χρησιμοποιείται ο αντίστροφος πίνακας. Ο υπολογισμός του αντιστρόφου ενός **byte** στο πεπερασμένο πεδίο $GF(2^8)$

και κατ' επέκταση και του ανωτέρω πίνακα επιτυγχάνεται με τη βοήθεια του γενικευμένου αλγόριθμου του Ευκλείδη, θεωρώντας τα bits ως συντελεστές πολυωνύμων.

Το ανωτέρω Sbox που χρησιμοποιείται στον μετασχηματισμό SubBytes() μπορεί να παρουσιαστεί με τη βοήθεια του ακόλουθου πίνακα πολύ πιο απλά σε δεκαεξαδική μορφή (Πίνακας 15). (Οι δυνατές τιμές – αντικαταστάσεις είναι προφανώς 16*16=256.)

		v														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
u	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	fa	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	d6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Πίνακας 15 **Το** S-box **του** AES, SubBytes

Με τη βοήθεια του ανωτέρω πίνακα μπορούμε πολύ εύκολα να βρούμε την έξοδο του S-box για μια δεδομένη είσοδο. Έτσι, αν το byte της εισόδου είναι

‘uv’, σε δεκαεξαδική μορφή, αυτό αντικαθίσταται από το byte που βρίσκεται στη σειρά ‘u’ και τη στήλη ‘v’ του ανωτέρω πίνακα. Για παράδειγμα, το byte ‘53’ αντικαθίσταται από το byte που βρίσκεται στη γραμμή ‘5’ και τη στήλη ‘3’, δηλαδή από το byte ‘ed’.

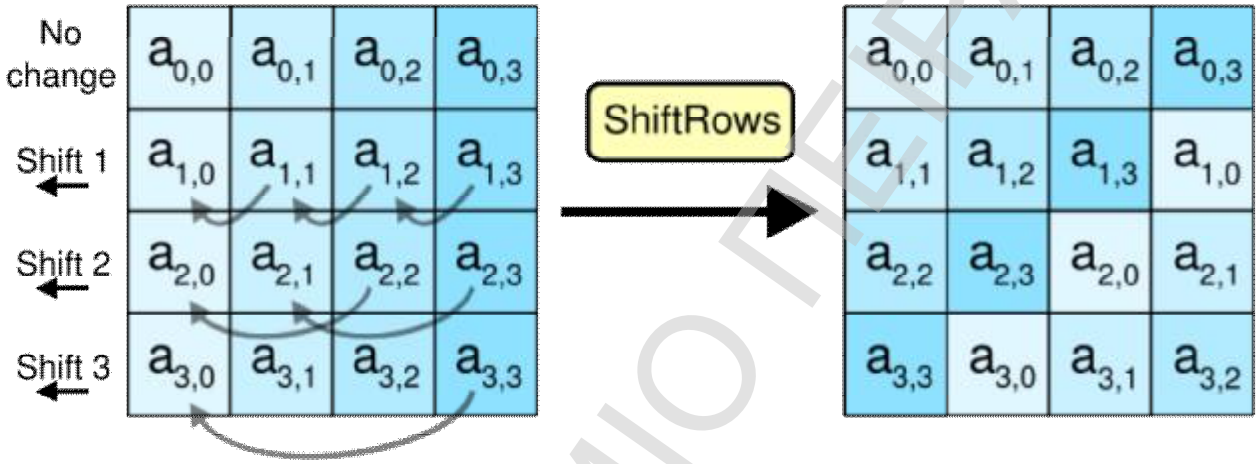
3.2.2 **Μετασχηματισμός** ShiftRows

Σύμφωνα με τον μετασχηματισμό ShiftRows, τα bytes των γραμμών του πίνακα κατάστασης ολισθαίνουν κυκλικά προς τα αριστερά κατά έναν σταθερό αριθμό θέσεων. Πιο συγκεκριμένα στον αλγόριθμο Rijndael, η γραμμή 0 παραμένει ως έχει. Στη γραμμή 1 μετατοπίζονται τα bytes κατά C1, στη γραμμή 2 κατά C2 και στη γραμμή 3 κατά C3 θέσεις. Οι τιμές των C1, C2 και C3 εξαρτώνται από το μήκος του μπλοκ του καθαρού κειμένου και, επομένως, από το πλήθος των στηλών του πίνακα
κατάστασης και περιέχονται στον Πίνακα 16.

Nb	*C1*	*C2*	*C3*
4	*1*	*2*	*3*
6	*1*	*2*	*3*
8	*1*	*3*	*4*

Table 16 **Οι τιμές των** C1, C2 **και** C3 **ως συνάρτηση του** Nb **στον αλγόριθμο**

Στο Σχήμα 18 μπορούμε να δούμε την επίπτωση που έχει ο μετασχηματισμός ShistRow στον πίνακα κατάστασης.



Σχήμα 18 Εφαρμογή του μετασχηματισμού ShixtRows στην κατάσταση του AES

3.2.3 **Μετασχηματισμός** MixColumns

Σύμφωνα με τον μετασχηματισμό αυτό, οι στήλες της κατάστασης αντιμετωπίζονται ως πολυώνυμα με συντελεστές στο πεπερασμένο πεδίο GF(28) και πολλαπλασιάζονται με ένα δεδομένο πολυώνυμο, το

$$c(x)= \text{‘03’}x^3 + \text{‘01’}x^2 + \text{‘01’}x + \text{‘02’}, \text{ modulo } (x^4+1).$$

Το πολυώνυμο x^4+1 είναι ανάγωγο, το $c(x)$ επελέγη έτσι ώστε ο πολλαπλασιασμός του με ένα άλλο πολυώνυμο να είναι αντιστρέψιμος. Οι

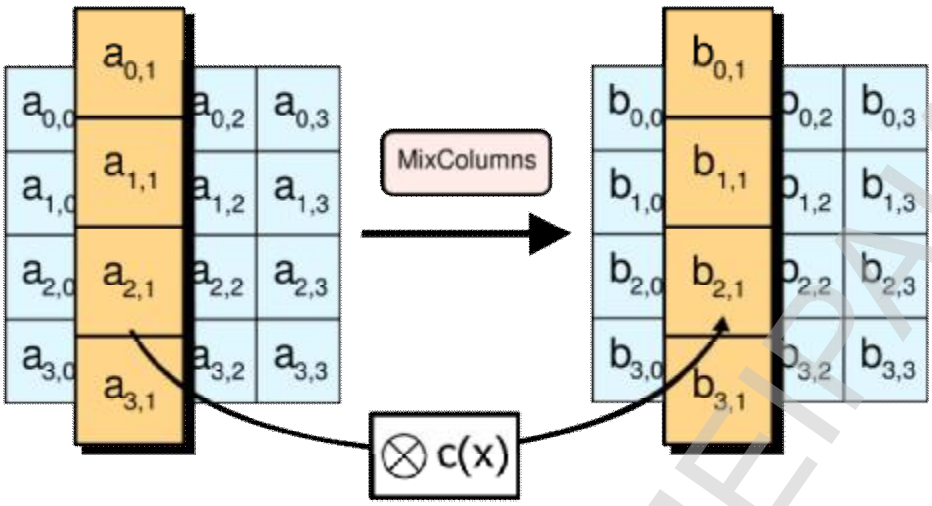
συντελεστές του πολυωνύμου, το οποίο είναι το αποτέλεσμα του πολλαπλασιασμού του πολυωνύμου που αντιστοιχεί στη στήλη με το ***c(x) modulo*** (x^4+1), αποτελούν τη νέα στήλη.

Στην ακόλουθη σχέση, το $\otimes$ συμβολίζει τον πολλαπλασιασμό ***modulo*** (x^4+1).

$$\begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \otimes c(x) = \begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix}$$

Δηλαδή, τα **bytes** της νέα στήλης τα λαμβάνουμε ως εξής:

$$b_{0,j} = (\text{'02'} \bullet a_{0,j}) \oplus (\text{'03'} \bullet a_{1,j}) \oplus (a_{2,j}) \oplus (a_{3,j}),$$
$$b_{1,j} = (a_{0,j}) \oplus (\text{'02'} \bullet a_{1,j}) \oplus (\text{'03'} \bullet a_{2,j}) \oplus (a_{3,j}),$$
$$b_{2,j} = (a_{0,j}) \oplus (a_{1,j}) \oplus (\text{'02'} \bullet a_{2,j}) \oplus (\text{'03'} \bullet a_{3,j}),$$
$$b_{3,j} = (\text{'03'} \bullet a_{0,j}) \oplus (a_{a,j}) \oplus (a_{2,j}) \oplus (\text{'02'} \bullet a_{3,j}).$$



Σχήμα 19 **Εφαρμογή του μετασχηματισμού** MixColumns **στην κατάσταση του** AES

Η αντιστροφή του μετασχηματισμού αυτού επιτυγχάνεται επίσης κατά παρόμοιο τρόπο. Πιο συγκεκριμένα, το πολυώνυμο που αντιστοιχεί σε κάθε στήλη του πίνακα κατάστασης πολλαπλασιάζεται με το πολυώνυμο d(x):

$$d(x)= \text{‘0B’}x^3 + \text{‘0D’}x^2 + \text{‘09’}x + \text{‘0E’} \ (\text{modulo } x^4+1).$$

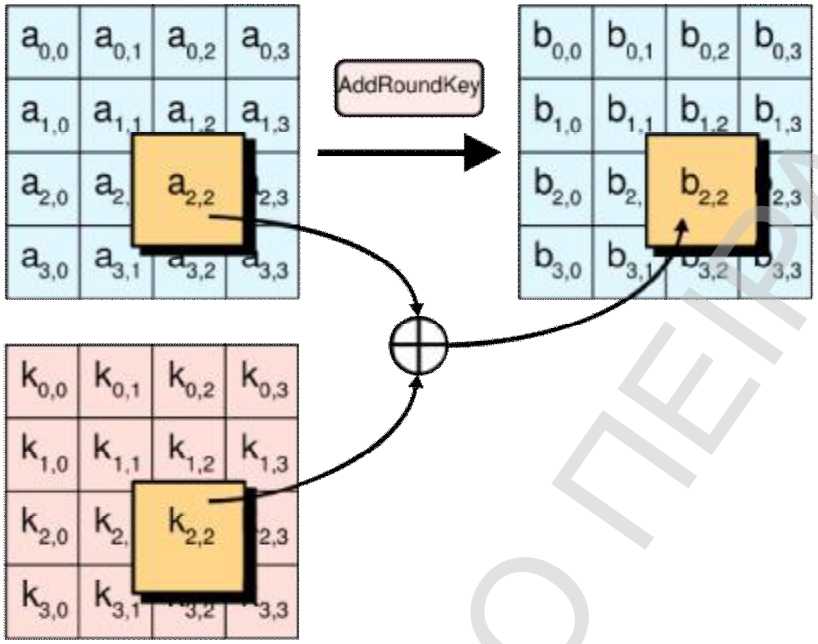
Μεταξύ των c(x) και d(x) ισχύει η σχέση:

$$c(x)d(x)= \text{‘01’ mod } (x^4+1).$$

3.2.4 **Μετασχηματισμός** AddRoundKey()

Ο μετασχηματισμός αυτός είναι πολύ απλός και συνίσταται στη συσχέτιση των bytes του πίνακα κατάστασης με τα bytes του υποκλειδιού ενός κύκλου μέσω της πράξης της αποκλειστικής διάζευξης (XOR).

Εφαρμόζοντας και πάλι τον μετασχηματισμό AddRoundkey επιτυγχάνουμε την αντιστροφή του.



Σχήμα 20 **Εφαρμογή του μετασχηματισμού** AddRoundKey **στην κατάσταση του** AES

3.3 Υπολογισμός υποκλειδιών

Τα κλειδιά των κύκλων (ή υποκλειδιά), τα οποία χρησιμοποιούνται στον μετασχηματισμό AddRoundKey(), προκύπτουν από το αρχικό κλειδί *K*. Το πλήθος των bits των υποκλειδιών, τα οποία απαιτούνται για την κρυπτογράφηση ή την αποκρυπτογράφηση, εξαρτάται από το μήκος του τμήματος του καθαρού κειμένου. Σε κάθε κύκλο του αλγορίθμου, καθώς και στην πρώτη εκτέλεση του μετασχηματισμού AddRoundKey() που προηγείται του πρώτου κύκλου, χρησιμοποιούνται τόσα bits υποκλειδιού όσα και τα bits του τμήματος, δηλαδή *Nb(Nr+1) λέξεις των 4 bytes =*

32Nb(Nr+1) bits. Επομένως, για μήκος μπλοκ 128 bits απαιτούνται συνολικά 1408 bits υποκλειδιών.

3.3.1 **Επέκταση του αρχικού κλειδιού**

Το αρχικό κλειδί *K* επεκτείνεται σε ένα μήκος ικανό να καλύψει όλες τις εκτελέσεις του μετασχηματισμού AddRoundKey() (δείτε παρακάτω την περιγραφή της επέκτασης του κλειδιού σε μορφή ψευδοκώδικα). Αν συμβολίσουμε με Expanded_Key το αποτέλεσμα της επέκτασης του αρχικού κλειδιού, τότε οι πρώτες *Nk* λέξεις του (των 4 bytes) είναι το αρχικό κλειδί *K*. Οι υπόλοιπες λέξεις του Expanded_Key υπολογίζονται από τις πρώτες αυτές λέξεις σύμφωνα με μια συνάρτηση, η οποία εξαρτάται από την τιμή του *Nk*. Επειδή οι υπολογισμοί βασίζονται στους δείκτες των bytes κλειδιού *K* και των λέξεων του Expanded_Key, θα ορίσουμε δύο πίνακες-διανύσματα, τον *K[i]* με στοιχεία τα bytes του αρχικού κλειδιού *K* και τον $E_{K[j]}$ με στοιχεία τις λέξεις του Expanded_Key.

Η συνάρτηση SubWord μετατρέπει την είσοδό της, η οποία είναι λέξη των 4 bytes, σε έξοδο, η οποία είναι επίσης λέξη των 4 bytes και της οποίας τα bytes προκύπτουν με τον γνωστό μετασχηματισμό SubBytes των αντιστοίχων bytes της εισόδου. Η συνάρτηση RotWord μεταθέτει κυκλικά τα

bytes της εισόδου (a,b,c,d) έτσι ώστε να προκύπτει στην έξοδό της η λέξη (b,c,d,a). Όσο για τις σταθερές των κύκλων Rcon[i], αυτές ορίζονται ως εξής:

Rcon[i]=(RC[i], '00', '00', '00'), με RC[i] να συμβολίζει ένα στοιχείο από το πεπερασμένο πεδίο GF(2^8) με τιμή $x^{(i-1)}$. Έτσι, RC[1]=1, δηλαδή σε δεκαεξαδική μορφή '01', RC[2]=x, δηλαδή '02', κλπ. (Σημειώνουμε ότι το i ξεκινάει από το 1 και όχι το 0.)

Παρατηρούμε στην ανωτέρω περιγραφή της KeyExpansion ότι μετά τις πρώτες Nk λέξεις του Expanded_Key, οι οποίες είναι ακριβώς το δεδομένο κλειδί K, κάθε λέξη $E_{K[i]}$ του Expanded-Key που ακολουθεί είναι ίση με το αποτέλεσμα της συσχέτισης μέσω της αποκλειστικής διάζευξης (XOR) της προηγούμενης λέξης $E_{K[i-1]}$ με τη λέξη $E_{K[i-Nk]}$. Για λέξεις, των οποίων οι θέσεις είναι πολλαπλάσια του Nk, στη λέξη $E_{K[i-1]}$ εφαρμόζεται ένας σύνθετος μετασχηματισμός πριν την πράξη της αποκλειστικής διάζευξης, ακολουθούμενος από την εκτέλεση της πράξης της αποκλειστικής διάζευξης με τη σταθερά του κύκλου Rcon[i]. Ο σύνθετος αυτός μετασχηματισμός συνίσταται από μια κυκλική ολίσθηση των bytes μιας λέξης (RotWord), ακολουθούμενος από την αντικατάσταση όλων των bytes της λέξης (SubWord).

Σημειώνεται ακόμα, ότι η ανωτέρω ρουτίνα επέκτασης του κλειδιού διαφοροποιείται κάπως στην περίπτωση κλειδιού Κ μήκους 256 bits. Συγκεκριμένα, για Nk>6 αν το (i-4) είναι πολλαπλάσιο του Nk, πριν την πράξη της αποκλειστικής διάζευξης εφαρμόζεται η SubWord() στη λέξη $E_{K[i-1]}$.

3.4 Αποκρυπτογράφηση

Στην αποκρυπτογράφηση χρησιμοποιούνται οι αντίστροφοι μετασχηματισμοί InvShiftRows, InvSubBytes, InvMixColumns και AddRoundKey. Στη συνέχεια περιγράφεται η λειτουργία της αποκρυπτογράφησης σε μορφή ψευδοκώδικα.

```
InvCipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
        byte state[4, Nb]
        state = in
        AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb –1])
        for round = Nr-1 step -1 downto 1
                InvShiftRows(state)
                InvSubBytes(state)
                AddRoundKey(state, w[round*Nb, (roun+1)*Nb-1])
                InvMixColumns()
        end for

        InvShiftRows(state)
        InvSubBytes(state)
        AddRoundKey(state, w[0, Nb-1])
        out = state
end
```

Σχήμα 21 **Η λειτουργία αποκρυπτογράφησης του** AES

Ακολουθούν σύντομες περιγραφές των τεσσάρων βασικών μετασχηματισμών του AES: InvShiftRows, InvSubBytes, InvMixColumns και AddRoundKey.

3.4.1 **Μετασχηματισμός** InvShiftRows

Σύμφωνα με τον μετασχηματισμό InvShiftRows, τα bytes των γραμμών του πίνακα κατάστασης ολισθαίνουν κυκλικά προς τα δεξιά πλέον κατά τον ίδιο σταθερό αριθμό θέσεων όπως και στον μετασηματισμό ShiftRows. Δηλαδή, η γραμμή 0 παραμένει ως έχει. Στη γραμμή 1 μετατοπίζονται τα bytes κατά 1 θέση, στη γραμμή 2 κατά 2 και στη γραμμή 3 κατά 3 θέσεις προς τα δεξιά.

3.4.2 **Μετασχηματισμός** InvSubBytes

Ο μετασχηματισμός InvSubBytes εφαρμόζεται σε κάθε byte της κατάστασης και περιγράφεται από τον ακόλουθο πίνακα 17.

		v															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
u	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Πίνακας 17 **Η αντιστροφή του** Sbox **του** AES, InvSubBytes

3.4.3 **Μετασχηματισμός** InvMixColumns

Ο μετασχηματισμός αυτός είναι ο αντίστροφος του MixColumns. Κάθε στήλη του πίνακα κατάστασης αντιμετωπίζεται ως ένα πολυώνυμο τεσσάρων όρων με συντελεστές στο πεπερασμένο πεδίο $GF(2^8)$. Πολλαπλασιάζεται δε με το πολυώνυμο d(x)

$$d(x)= \text{‘0b’}x^3 + \text{‘0d’}x^2 + \text{‘09’}x + \text{‘0e’} \ (\text{modulo } x^4+1).$$

Το d(x) είναι ο αντίστροφος του c(x) που χρησιμοποιείται στον μετασχηματισμό MixColumns, δηλαδή $c(x)d(x)=1 \bmod (x^4+1)$.

Οι συντελεστές του πολυωνύμου b(x), το οποίο είναι το αποτέλεσμα του πολλαπλασιασμού του πολυωνύμου που αντιστοιχεί στη στήλη a(x) με το d(x) modulo (x^4+1), αποτελούν τη νέα στήλη. Δηλαδή, τα bytes της νέα στήλης τα λαμβάνουμε ως εξής:

$$b_{0,j} = (\text{'0e'} \bullet a_{0,j}) \oplus (\text{'0b'} \bullet a_{1,j}) \oplus (\text{'0d'} \bullet a_{2,j}) \oplus (\text{'09'} \bullet a_{3,j}),$$

$$b_{1,j} = (\text{'09'} \bullet a_{0,j}) \oplus (\text{'0e'} \bullet a_{1,j}) \oplus (\text{'0b'} \bullet a_{2,j}) \oplus (\text{'0d'} \bullet a_{3,j}),$$

$$b_{2,j} = (\text{'0ed'} \bullet a_{0,j}) \oplus (\text{'09'} \bullet a_{1,j}) \oplus (\text{'0e'} \bullet a_{2,j}) \oplus (\text{'0b'} \bullet a_{3,j}),$$

$$b_{3,j} = (\text{'0b'} \bullet a_{0,j}) \oplus (\text{'0d'} \bullet a_{a,j}) \oplus (\text{'09'} \bullet a_{2,j}) \oplus (\text{'0e'} \bullet a_{3,j}).$$

3.4.4 **Μετασχηματισμός** InvAddRoundKey

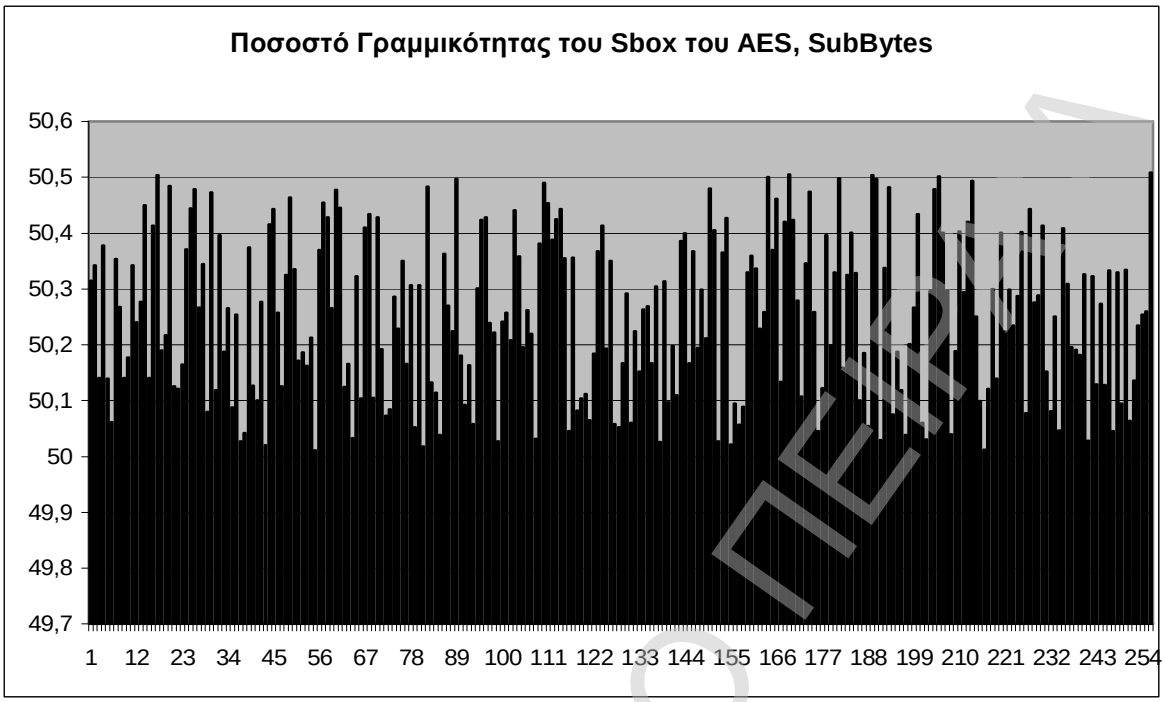
Ο μετασχηματισμός αυτός ταυτίζεται με τον AddRoundKey, αφού περιορίζεται μόνο στην πράξη της αποκλειστικής διάζευξης (XOR), δηλαδή εφαρμόζοντας και πάλι τον μετασχηματισμό AddRoundKey επιτυγχάνουμε την αντιστροφή του.

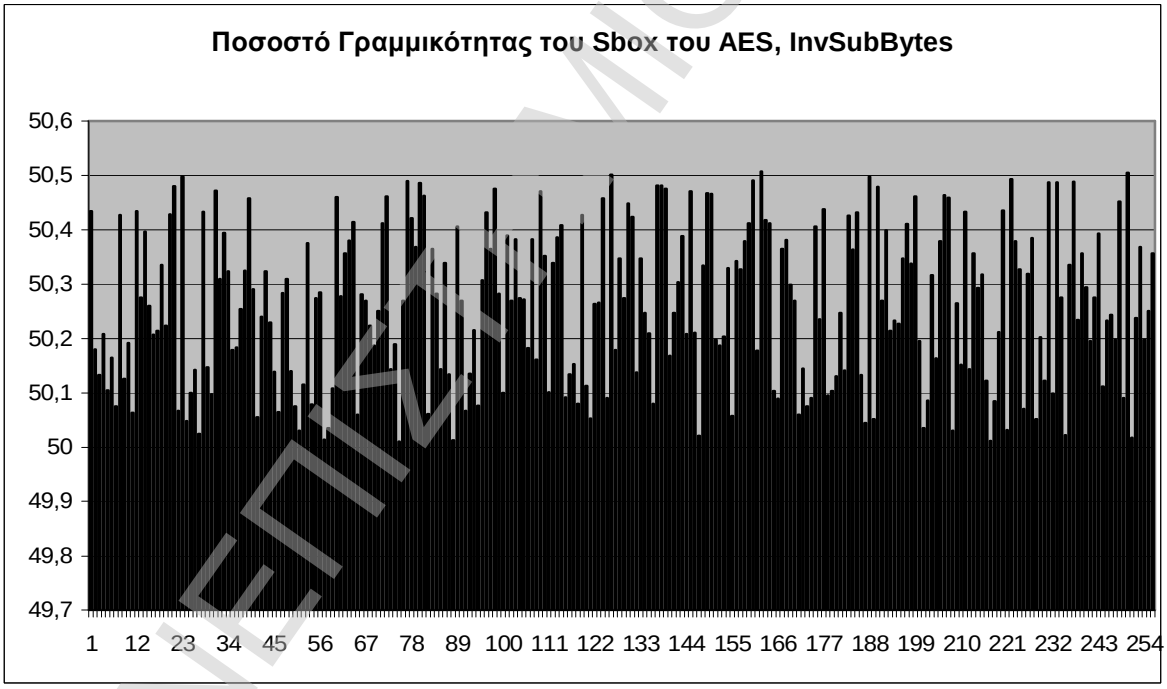
Όπως είδαμε ανωτέρω, κατά την αποκρυπτογράφηση οι διάφοροι μετασχηματισμοί δεν εφαρμόζονται με την ίδια σειρά όπως στην κρυπτογράφηση. Ωστόσο, με μια μικρή τροποποίηση της ρουτίνας επέκτασης του κλειδιού για τον υπολογισμό των απαιτούμενων υποκλειδιών, είναι δυνατή η αποκρυπτογράφηση με την ίδια σειρά εφαρμογής των επιμέρους μετασχηματισμών όπως στην κρυπτογράφηση.

3.5 Κρυπτανάλυση του AES

Στο πλαίσιο της κρυπταναλυτικής επίθεσης που έγινε στον DES, κρίθηκε σκόπιμο, όπως αναφέραμε, να γίνει προσπάθεια αντίστοιχης επίθεσης στον αλγόριθμο AES. Έτσι λοιπόν επιχειρήθηκε γραμμική προσέγγιση στην

είσοδο και στην έξοδο της SubBytes, το νέο S-box, το οποίο χρησιμοποιείται από τον AES. Τα αποτελέσματα για την γραμμικότητα που παρουσιάζεται σε αυτήν την συνάρτηση φαίνονται ξεκάθαρα στα παρακάτω δυο γραφήματα.





Όπως φαίνεται και από τα ανωτέρω διαγράμματα, το Sbox του AES παρουσιάζει πολύ μεγάλη απόκλιση από οποιαδήποτε γραμμική συνάρτηση. Είναι πολύ κοντά σε αυτό το οποίο αποκαλείται συνάρτηση

bent. Μια συνάρτηση δηλαδή, η οποία απέχει το μέγιστο δυνατό από οποιαδήποτε γραμμική συνάρτηση. Το γεγονός αυτό κάνει αδύνατη την επίθεση στον AES, μέσω της επιτυχούς επίθεσης που παρουσιάστηκε για τον DES. Αυτό σαφώς δηλώνει τον καλύτερο σχεδιασμό που έγινε για τον AES, αλλά και την ασφάλεια η οποία μπορεί να προσφέρει ο συγκεκριμένος αλγόριθμος έναντι των αλγεβρικών επιθέσεων. Είναι χαρακτηριστικό πως η μόνη προτεινόμενη επίθεση, η οποία μπορεί να έχει κάποιο αποτέλεσμα καλύτερο από την δοκιμή όλων των δυνατών κλειδιών, είναι η αλγεβρική XL επίθεση από τον N. Courtois [2] η οποία για την απλούστερη περίπτωση του αλγορίθμου, να έχει δηλαδή ο AES κλειδί των 128 bits, η επίθεση θα πρέπει να δοκιμάζει περίπου 2^{100} κλειδιά. Είναι προφανές λοιπόν πως οι υπολογιστικές αλλά και οι χρονικές απαιτήσεις ενός τέτοιου εγχειρήματος το καθιστούν αδύνατο για την μέχρι στιγμής τεχνολογία, δείχνοντας ακόμα περισσότερο γιατί θα πρέπει να θεωρείται ο AES, ένας πάρα πολύ ασφαλής αλγόριθμος.

Κεφάλαιο 4 Οι πρώτοι αριθμοί στον RSA

4.1 Εισαγωγή

Ο αλγόριθμος RSA [11, 12] ανήκει στην κατηγορία των ασύμμετρων κρυπτογραφικών συστημάτων, δηλαδή εκείνων που χρησιμοποιούν διαφορετικά κλειδιά (δημόσιο και ιδιωτικό) για την κρυπτογράφηση και την αποκρυπτογράφηση. Μηνύματα κρυπτογραφημένα με το ένα από τα δύο κλειδιά μπορούν να αποκρυπτογραφηθούν μόνο από το άλλο κλειδί.

Όπως όλοι οι αλγόριθμοι της κατηγορίας αυτής, ο RSA μπορεί να χρησιμοποιηθεί για τη διασφάλιση τόσο της εμπιστευτικότητας του περιεχομένου ενός μηνύματος, όσο και της εγκυρότητας της ταυτότητας του αποστολέα του μηνύματος.

- Η εμπιστευτικότητα του περιεχομένου διασφαλίζεται από το γεγονός ότι μετά την κρυπτογράφηση του μηνύματος με το δημόσιο κλειδί του παραλήπτη (γνωστό σε όλους), μόνο ο νόμιμος παραλήπτης (κάτοχος του ιδιωτικού κλειδιού) μπορεί να ανακτήσει το αρχικό περιεχόμενο του μηνύματος.
- Η εγκυρότητα της ταυτότητας του αποστολέα του μηνύματος διασφαλίζεται με την κρυπτογράφηση του μηνύματος μέσω του ιδιωτικού κλειδιού του αποστολέα. Η επιτυχής αποκρυπτογράφηση του μηνύματος από τον παραλήπτη με

χρήση του δημόσιου κλειδιού του αποστολέα διασφαλίζει ότι ο αποστολέας είναι πράγματι αυτός που ισχυρίζεται ότι είναι.

Ο συνδυασμός της κρυπτογράφησης αρχικά με το ιδιωτικό κλειδί του αποστολέα και στη συνέχεια με το δημόσιο κλειδί του παραλήπτη εξασφαλίζει συγχρόνως τόσο την εμπιστευτικότητα του περιεχομένου του μηνύματος, όσο και την εγκυρότητα της ταυτότητας του αποστολέα.

Η ασφάλεια που παρέχει ο αλγόριθμος RSA στηρίζεται στη μη-πολυωνυμικής τάξης επιλυσιμότητα δύο μαθηματικών προβλημάτων:

- Του προβλήματος της ανάλυσης ενός αριθμού σε γινόμενο πρώτων παραγόντων και
- Του προβλήματος της εύρεσης της ν-οστής ρίζας ενός αριθμού modulo έναν άλλον αριθμό.

Και τα δύο αυτά προβλήματα θεωρούνται ότι ανήκουν στην NP-complete κλάση πολυπλοκότητας, επομένως δεν υπάρχει αλγόριθμος πολυωνυμικής πολυπλοκότητας που να τα επιλύει.

Στην παρούσα διατριβή, η έρευνα επικεντρώθηκε στις συνθήκες οι οποίες μπορούν να επιταχύνουν την επίλυση του πρώτου προβλήματος στην περίπτωση του αλγόριθμου RSA.

4.2 Δημιουργία κλειδιών

Συμβολίζουμε με e το δημόσιο κλειδί και με d το ιδιωτικό κλειδί. Τα κλειδιά αυτά παράγονται με την ακόλουθη διαδικασία:

1. Επιλέγονται δύο μεγάλοι πρώτοι αριθμοί p και q, διαφορετικοί μεταξύ τους. Ο έλεγχος εάν οι αριθμοί p και q είναι πρώτοι γίνεται με την επαναλαμβανόμενη υποβολή τους σε συγκεκριμένες δοκιμές. Αν οι αριθμοί περάσουν τις δοκιμές, τότε είναι με μεγάλη πιθανότητα πρώτοι.
2. Υπολογίζεται η ποσότητα $n=pq$
3. Υπολογίζεται η συνάρτηση του Euler $\varphi(n) =(p-1)(q-1)$.
4. Υπολογίζεται ένας ακέραιος e σχετικά πρώτος με το $\varphi(n)$ ώστε $(e,n)=1$
5. Υπολογίζεται ένας ακέραιος d ώστε $ed=1 \bmod \varphi(n)$.

Το δημόσιο κλειδί αποτελείται από το ζεύγος (e,n) και το ιδιωτικό κλειδί είναι το d.

4.3 Κρυπτογράφηση

Για την κρυπτογράφηση ενός μηνύματος Μ, ακολουθείται η παρακάτω διαδικασία:

1. Το μήνυμα M μετατρέπεται σε έναν αριθμό m βάσει κάποιου κοινά αποδεκτού

αντιστρέψιμου αλγόριθμου.

2. Η κρυπτογραφημένη μορφή τού m υπολογίζεται από τη σχέση $c(m) = m^e \bmod n$

4.4 Αποκρυπτογράφηση

Για την αποκρυπτογράφηση του c, ακολουθείται η κατωτέρω διαδικασία

1. Υπολογίζεται η αποκρυπτογραφημένη μορφή τού c από τη σχέση

$m = c^e \bmod n$

2. Ο αριθμός m μετατρέπεται στο μήνυμα M με τον αντίστροφο αλγόριθμο εκείνου που χρησιμοποιήθηκε στην κρυπτογράφηση.

4.4.1 **Αδυναμίες του** RSA

Ο αλγόριθμος RSA παρέχει ισχυρή ασφάλεια εφόσον ληφθούν υπόψη ορισμένες παράμετροι που επηρεάζουν την απόδοσή του. Οι παράμετροι αυτές είναι:

1. Η επιλογή των αριθμών p και q πρέπει να είναι τέτοια ώστε να ισχύει η παρακάτω ανισότητα:

$$p<q<2p$$

2. Εάν οι αριθμοί p-1 και q-1 έχουν μικρούς παράγοντες, τότε η ανάλυση τού n σε

γινόμενο πρώτων παραγόντων μπορεί να γίνει σχετικά εύκολα. Για το λόγο αυτό, πρέπει οι αριθμοί p και q να επιλέγονται έτσι ώστε οι p-1 και q-

1 να έχουν έναν πολύ μεγάλο πρώτο παράγοντα και ο μέγιστος κοινός διαιρέτης τους να είναι σχετικά μικρός.

Οι κρυπταναλυτικές μέθοδοι που μπορούν να χρησιμοποιηθούν ενάντια στον RSA ουσιαστικά έχουν να κάνουν με τη διαχείριση του αλγορίθμου από προγραμματιστική πλευρά και δεν έχουν να κάνουν με την ουσιαστική ασφάλεια που προσφέρει ο αλγόριθμος. Η πιο γνωστή και αποτελεσματική επίθεση είναι αυτή της ανάλυσης του απαιτούμενου χρόνου αποκρυπτογράφησης από το υπολογιστικό σύστημα του παραλήπτη, η οποία μπορεί να αποκαλύψει το ιδιωτικό κλειδί (time attacks). Για την αντιμετώπιση αυτού του είδους επίθεσης τα προγράμματα τα οποία κάνουν χρήση του αλγορίθμου επιστρέφουν το κρυπτογραφημένο μήνυμα, είτε σε τυχαίο χρονικό διάστημα είτε πάντα σε σταθερό.

Σε πολλές εμπορικές εφαρμογές διαπιστώθηκε πως κατά την χρήση του αλγορίθμου RSA, η πρώτη συνθήκη ασφαλείας δεν ικανοποιείται [47]. Πιο συγκεκριμένα αυτό το οποίο παρατηρήθηκε είναι ότι οι εφαρμογές απλά δημιουργούσαν δύο μεγάλους πρώτους αριθμούς. Αυτό ασφαλώς μπορεί να δημιουργήσει πολλά προβλήματα και να οδηγεί σε επιθέσεις στον αλγόριθμο οι οποίες παρατίθενται παρακάτω.

Έστω λοιπόν $n=pq$, όπου p πρώτος. Έστω m ακέραιος με:

$$q \leq m \leq p$$

και

$$r = \frac{p}{q}$$

Από τις βασικές ιδιότητες του πολλαπλασιασμού σε πεπερασμένες ομάδες, έχουμε:

$$n \bmod m \equiv pq \bmod m \equiv p \bmod m * q \bmod m$$

Αφού $q \le m$, $q \bmod m = q$

Αναζητούμε κατάλληλα m τέτοια ώστε:

$$n \bmod m = aq$$

με aq<m.

Για να μετρήσουμε για πόσα a ισχύει κάτι τέτοιο θέλουμε ουσιαστικά:

$$qp \bmod m \le m \Leftrightarrow p \bmod m \le \frac{m}{q}$$

Θα πρέπει λοιπόν να βρούμε όλα τα m τέτοια ώστε $q \le p$ και $p \bmod m \le \frac{m}{q}$.

Ισοδύναμα πρέπει να βρούμε πόσες διαφορετικές λύσεις υπάρχουν στις εξισώσεις:

$$p \bmod x \equiv 0$$

$$p \bmod x \equiv 1$$

…

$$p \bmod x \equiv \frac{p}{q}$$

Έστω λοιπόν η εξίσωση:

$$p \bmod x \equiv i \Leftrightarrow p - i = lx$$

Συνεπώς το **p-i** πρέπει να είναι διαιρέτης του **x**. Αν από την κάθε μια από τις παραπάνω εξισώσεις διαλέξουμε μια λύση, την **x=p-i**, προφανώς υπάρχουν περισσότερες αλλά επειδή όλες όσες επιλέγουμε πρέπει να είναι διαφορετικές, θα υπάρχουν τουλάχιστον $\frac{p}{q}$ λύσεις.

Έτσι με κέντρο το **p** και ακτίνα **r=** $\frac{p}{q}$ θα μπορούμε πάντα να βρούμε κατάλληλο **m** τέτοιο ώστε:

$$n \mod m, n) = (pq \mod m, n) = (p \mod m \cdot q \mod m, n) = ((p \mod m) \cdot q, n) = (aq, n) = q,$$

που σημαίνει πως μπορούμε να βρούμε ένα παράγοντα **q** του **n**, χρησιμοποιώντας ένα τυχαίο αριθμό, στο διάστημα: $(p - r, p + r)$ επιλέγοντας απλά έναν αριθμό μέσα σε αυτό το διάστημα και υπολογίζοντας τον παράγοντα.

Από πρακτικής άποψης η χρήση του παραπάνω διαστήματος δεν είναι δυνατή, αφού το **p** δεν είναι γνωστό. Παρόλα αυτά για αρκετά μεγάλο **r**, είναι δυνατό αυτό το διάστημα να τέμνεται με το διάστημα $[\sqrt{n}, \infty)$. Για δεδομένο ακέραιο **n** θα ορίσουμε τα φράγματα της παραγοντοποίησής του, θα δούμε λοιπόν πως από ένα τυχαίο αριθμό **x**, μπορούμε να οδηγηθούμε σε παραγοντοποίησης του **n**.

Θα εξετάσουμε αν ο αριθμός $\sqrt{n}$ ανήκει στη σφαίρα με κέντρο το **p** και ακτίνα $r = \frac{p}{q}$. Αν αυτό ισχύει τότε θα μπορούμε να βρούμε το **p**, διαλέγοντας οποιονδήποτε αριθμό $x \in (\sqrt{n}, \sqrt{n} + \frac{p}{q})$.

Για να έχουμε $x \in (\sqrt{n}, \sqrt{n} + \frac{p}{q})$ πρέπει:

$$p - r \leq \sqrt{n} \leq p + r \Leftrightarrow$$

$$(p - r)^2 \leq n \leq (p + r)^2 \Leftrightarrow$$

$$p^2 - 2pr + r^2 \leq n \leq p^2 + 2pr + r^2 \Leftrightarrow$$

$$p^2 - 2p\frac{p}{q} + \frac{p^2}{q^2} \leq n \leq p^2 + 2p\frac{p}{q} + \frac{p^2}{q^2} \Leftrightarrow$$

$$p^2q^2 - 2p^2q + p^2 \leq nq^2 \leq q^2p^2 + 2p^2q + p^2 \Leftrightarrow$$

$$p^2q^2 - 2p^2q + p^2 \leq pq^3 \leq q^2p^2 + 2p^2q + p^2 \Leftrightarrow$$

$$pq^2 - 2pq + p \leq q^3 \leq q^2p + 2pq + p \Leftrightarrow$$

$$pq^3 - 2pqq + pq \leq q^4 \leq q^2qp + 2pqq + pq \Leftrightarrow$$

$$nq^2 - 2nq + n \leq q^4 \leq q^2n + 2nq + n$$

Μπορεί όμως το $[\sqrt{n}, \infty)$ να μη τέμνεται με το διάστημα που ορίστηκε παραπάνω, συνεπώς θα πρέπει να βρούμε το ελάχιστο **k** ώστε το διάστημα $[\sqrt{n} + kr, \infty)$ να τέμνεται με το διάστημα που θέλουμε. Ισοδύναμα, μπορούμε να το βρούμε αρκεί το $\sqrt{n} + kr$ να ανήκει στο διάστημα με κέντρο το **p** και ακτίνα$r = \frac{p}{q}$. Μετά από μερικές αλγεβρικές πράξεις καταλήγουμε στην ακόλουθη συνθήκη:

$$nq^2 - 2nq(k+1) + (k+1)^2 n \leq q^4 \leq q^2 n + 2nq(1-k) + n(1-k)^2$$

Το ερώτημα που τίθεται είναι πόσο μεγάλο πρέπει να είναι το **k**, ώστε να ικανοποιείται η συνθήκη: $[\sqrt{n} + kr, \infty) \cap (p - kr, p + kr) \neq \emptyset$. Μπορούμε λοιπόν να φράξουμε το **k** ως εξής:

$$\sqrt{n} + rk < p \Leftrightarrow$$

$$rk < p - \sqrt{n} \Leftrightarrow$$

$$k < \frac{p-\sqrt{n}}{r} \Leftrightarrow$$

$$k < \frac{p-\sqrt{n}}{\frac{p}{q}} \Leftrightarrow$$

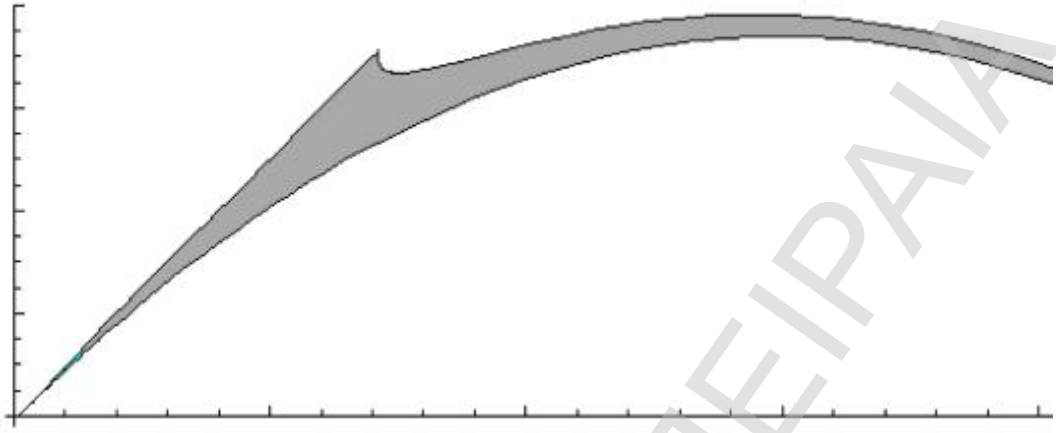
$$k < q\frac{p-\sqrt{n}}{p} \Leftrightarrow$$

$$k < q(1 - \frac{q\sqrt{n}}{qp}) \Leftrightarrow$$

$$k < q(1 - \frac{q\sqrt{n}}{n}) \Leftrightarrow$$

$$k < q(1 - \frac{q}{\sqrt{n}})$$

Από την παραπάνω ανισότητα μπορούμε να δούμε ότι αν το **q** είναι αρκετά κοντά στο $\sqrt{n}$, δηλαδή αν τα **p** και **q** είναι αρκετά κοντά, αυτή η μέθοδος μπορεί να παραγοντοποιήσει το **n** γρήγορα, αφού το **k** (το πλήθος των βημάτων) θα είναι μικρό. Κάτι το οποίο φαίνεται και από την γραφική παράσταση των ανισοτήτων.



Σχήμα 22 **Στο γράφημα παρουσιάζονται οι δυνατές τιμές για τα** p,q (**οριζόντιος αξονας**), **όσο αυξάνεται το πηλίκο τους (κάθετος άξονας**).

Τώρα λοιπόν μπορούμε να βρούμε το άνω φράγμα των βημάτων που απαιτούνται προκειμένου να ικανοποιείται η συνθήκη:

$$[\sqrt{n}+kr,\infty)\cap(p-kr,p+kr)\neq\emptyset$$

Έχουμε λοιπόν:

$$p-\sqrt{n}=\frac{n}{q}-\sqrt{n}=$$

$$\sqrt{n}(\frac{\sqrt{n}}{q}-1)=$$

$$\sqrt{n}(\frac{\sqrt{p}\sqrt{q}}{q}-1)=$$

$$\sqrt{n}(\frac{\sqrt{p}}{\sqrt{q}}-1)=$$

$$\sqrt{n}(\sqrt{r}-1)=$$

$$r\frac{\sqrt{n}}{r}(\sqrt{r}-1)$$

Από την παραπάνω ισότητα λοιπόν συμπεράνουμε χρειαζόμαστε το πολύ $\frac{\sqrt{n}}{r}(\sqrt{r}-1)$ βήματα για πετύχουμε να έχουμε : $[\sqrt{n}+kr,\infty)\cap(p-kr,p+kr)\neq\emptyset$, το οποίο με την σειρά του μπορεί να μας οδηγήσει σε παραγοντοποίηση του n με την επιλογή τυχαίου ακεραίου σε αυτό το εύρος.

Προκειμένου να δείξουμε τις επιρροές που μπορεί να έχουν αυτά τα φράγματα θα δώσουμε μια απλή περιγραφή του αλγόριθμου παραγοντοποίησης με τη χρήση ελλειπτικών καμπυλών [14].

Έστω n ο αριθμός που θέλουμε να παραγοντοποιήσουμε. Μια ελλειπτική καμπύλη στο πεπερασμένο σώμα F είναι ένα ζεύγος στοιχείων, $a,b\in F$, τέτοιο ώστε:$4a^3+27b^3\neq 0$. Το σύνολο των σημείων της ελλειπτικής καμπύλης στο F είναι :

$$E(F)=\{(x:y:z)\in P^2(F):y^2z=x^3+axz+bz^3\},$$

όπου , P^2 είναι το προβολικό επίπεδο.

Το E(F) περιέχει ακριβώς ένα σημείο για το οποίο z=0, το σημείο (0,1,0), το οποίο ονομάζουμε μηδενικό σημείο ,O, της καμπύλης.

Όλα τα υπόλοιπα σημεία της καμπύλης είναι της μορφής (x,y,1) όπου τα x και y ικανοποιούν την εξίσωση :

$$y^2=x^3+ax+b$$

Ορίζουμε την πρόσθεση δυο σημείων ως εξής:

O+P=P+O=P.

Έστω $P=(x_1,y_1,1),Q=(x_2,y_2,1)$ τότε:

- P+Q=O αν $x_1 = x_2$ και $y_1 = -y_2$ αλλιώς,
- Αν $P \neq Q, \lambda = \frac{y_1 - y_2}{x_1 - x_2}$
- Αν $P = Q, \lambda = \frac{3x_1^2 + a}{2y_1}$

P+Q=R, όπου $R = (x_3, y_3, 1), x_3 = \lambda^2 - x_1 - x_2, y_3 = -\lambda x_3 - y_1 + \lambda x_1$

Τα βήματα λοιπόν του αλγόριθμου είναι τα εξής:

Βήμα 1. Ελέγχεται αν (n,6)=1 και ότι το n δεν είναι τέλειο τετράγωνο.

Βήμα 2. Επιλέγεται τυχαία καμπύλη E και τυχαίο σημείο P στην E. Επιλέγονται ακόμα τυχαίοι ακέραιοι $a, x, y \in [0, n-1]$ και ορίζεται $b = y^2 - x^3 - ax$,

Ελέγχεται αν $g = (4a^3 + 27b^2, n) = 1$, αν g=n, επαναλαμβάνεται αυτό το βήμα αλλιώς επιστρέφουμε g.

Βήμα 3. Επιλέγεται ακέραιος k ο οποίος διαιρείται από δυνάμεις μικρών ακεραίων και μικρότερων του B.

Βήμα 4. Υπολογίζεται η ποσότητα kP modulo n

Βήμα 5. Αν το προηγούμενο βήμα προκύψει διαιρέτης τότε επιστρέφεται, διαφορετικά ο αλγόριθμος επιστρέφει στι βήμα 2

Είναι προφανές από τον παραπάνω αλγόριθμο ότι το ελάχιστο φράγμα παίζει σημαντικότατο ρόλο στην παραγοντοποίηση και με τον αλγόριθμο που περιγράψαμε ανωτέρω μπορούν να υπολογιστούν κατάλληλα φράγματα B, προκειμένου ένας άλλος αλγόριθμος όπως αυτός των ελλειπτικών καμπυλών να λειτουργήσει ταχύτερα.

Θα πρέπει να τονίσουμε ότι στο πλαίσιο της διατριβής βρέθηκαν εφαρμογές του RSA όπου η ακτίνα $r = \frac{p}{q}$ ήταν μεγαλύτερη από 2^{100}. Η γνώση μιας εκτίμησης αυτής της ακτίνας μπορεί να μας δώσει συγκεκριμένα φράγματα για τα p,q και όπως είδαμε να μας οδηγήσει σε μια παραγοντοποίηση του n.

Σύμφωνα λοιπόν με τη μελέτη που έγινε σε αυτή την διδακτορική διατριβή, θα πρέπει να γίνουν αλλαγές στο RFC [11] το οποίο περιγράφει τον αλγόριθμο RSA, καθώς δεν γίνεται καμία αναφορά στο μέγεθος που θα πρέπει να έχουν οι αριθμοί p και q, κάτι το οποίο μπορεί να οδηγήσει σε πολλά προβλήματα ασφαλείας είτε άμεσα, μέσω της παραγοντοποίησης είτε έμμεσα, μέσω της εύρεσης καταλληλότερων φραγμάτων και επιταχύνσεως άλλων μεθόδων παραγοντοποίησης.

4.5 Έλεγχος πρώτων αριθμών

Όπως είδαμε στην περιγραφή του αλγορίθμου RSA, κατά τη δημιουργία των κλειδιών, ήταν απαραίτητο να οδηγηθούμε στην ανεύρεση μεγάλων πρώτων αριθμών p και q. Προφανώς για την άρτια εφαρμογή του αλγορίθμου σε επίπεδο λογισμικού, αυτοί οι δυο πρώτοι θα πρέπει να είναι όσο το δυνατόν πιο τυχαίοι. Έτσι λοιπόν τίθεται το πρόβλημα της κατασκευής και μάλιστα γρήγορης, τυχαίων πρώτων αριθμών. Ο M.Agrawal [53] παρουσίασε πρόσφατα έναν ντετερμινιστικό πολυωνυμικό αλγόριθμο που ελέγχει αν ένας αριθμός είναι πρώτος ή όχι. Παρόλα αυτά

ο αλγόριθμος λόγω της αργής εφαρμογής του δεν χρησιμοποιείται. Αντίθετα συνήθως χρησιμοποιούνται άλλες μέθοδοι προκειμένου να ελεγχθεί αν ένας αριθμός είναι πρώτος.

Στο πλαίσιο της διατριβής αυτής μελετήθηκε η μέθοδος των ακολουθιών Lucas [50]. Ένα από τα βασικότερα πορίσματα της διατριβής ήταν η εύρεση του γρηγορότερου αλγορίθμου και με τη χαμηλότερη υπολογιστική πολυπλοκότητα μέχρι στιγμής, για τον υπολογισμό των πολύ γνωστών αριθμών Fibonacci και Lucas. Θα πρέπει να τονίσουμε πως πέρα από τον έλεγχο για πρώτους αριθμούς οι ακολουθίες Lucas μπορούν να χρησιμοποιηθούν και για την παραγοντοποίηση μεγάλων ακεραίων.

4.6 Μέθοδος ακολουθιών Lucas

Έστω ότι επιλέγουμε ακεραίους p και q τέτοιους ώστε ο αριθμός $p^2 - 4q$ να μην είναι τετράγωνο modulo n. Τότε το πολυώνυμο $x^2 - px + q$ έχει διακριτές λύσεις μια από τις οποίες είναι η $r = \frac{p+\sqrt{p^2-4q}}{2}$ και μέσω επαγωγής μπορεί να δείξει εύκολα κανείς πως οι δυνάμεις της ρίζας r, δίνονται από τον τύπο:

$$r^m = \frac{V(m)+U(m)\sqrt{p^2-4q}}{2},$$

όπου τα U και V, προκύπτουν από:

$$U(0) = 0, U(1) = 1, U(m) = pU(m-1) - qU(m-2)$$

$$V(0) = 2, V(1) = p, V(m) = pV(m-1) - qV(m-2)$$

Αυτές είναι οι ακολουθίες **Lucas** που προκύπτουν για τα **p** και **q**. Στην περίπτωση που **p=1** και **q=-1** έχουμε την ακολουθία **Fibonacci**.

Για αυτές τις ακολουθίες ισχύει **[70, 71]**:

$$U(2m) = U(m)V(m)$$

$$V(2m) = V(m)^2 - 2q^m$$

Λήμμα: Για **p**, **q** με $p^2 - 4q$ να μην είναι τετράγωνο **modulo n**, θεωρούμε

$$2r = a + b\sqrt{p^2 - 4q} mod n$$

για ακεραίους **a** και **b** ισότιμους. Αν ο **n** είναι πρώτος, τότε :

$$2r^n = a - b\sqrt{p^2 - 4q} mod n$$

Το παραπάνω λήμμα μπορεί να γίνει:

Λήμμα: Για **p**, **q** με $p^2 - 4q$ να μην είναι τετράγωνο **modulo n**, αν ο **n** είναι πρώτος, τότε:

$$U(n+1) = 0 mod n$$

Από το ανωτέρω λήμμα έχουμε:

Θεώρημα: Έστω **n>1** περιττός. Αν ο ακέραιος **d** για τον οποίο το σύμβολο **Jacobbi (d | n)=-1** και για κάθε πρώτο διαιρέτη του **n+1** υπάρχουν σχετικά πρώτοι αριθμοί **p** και **q** με $p^2 - 4q = d$, και επιπλέον:

- $U(n+1) = 0 \ mod \ n$
- $U((n+1)/r) \neq 0 \ mod \ n$,

τότε n είναι πρώτος.

Ο κατωτέρω κώδικας σε c++ του Wei Dai υλοποιεί τον αλγόριθμου που ελέγχει αν ένας αριθμός είναι πρώτος με χρήση της μεθόδου των ακολουθιών Lucas:

```
boolean IsStrongLucasProbablePrime(const Integer &n)
{
        assert(n>1);
        if (n[0]==0)
                return n==2;
        Integer b=1, d;
        unsigned int i=0;
        int j;
        do
        {
                if (++i==64 && n.IsSquare())
                   return FALSE;
                ++b; ++b;
                d = (b.Square()-4)%n;
        }
        while ((j=Jacobi(d,n)) == 1);
        if (j==0)
                return FALSE;
        Integer n1 = n-j;
        unsigned int a;
            for (a=0; ; a++)
                if (n1[a])
                        break;
        Integer m = n1>>a;
        Integer z = Lucas(m, b, n);
        if (z==2 || z==n-2)
               return TRUE;
        for (i=1; i<a; i++)
        {
                z = (z.Square()-2)%n;
                if (z==n-2)
                        return TRUE;
                if (z==2)
                        return FALSE;
        }
        return FALSE;
}
```

4.7 Αριθμοί Fibonacci και Lucas

Οι αριθμοί Fibonacci, αποτελούν μια από τις πιο γνωστές ακολουθίες. Ο τύπος της ακολουθίας είναι:

$$F_n = F_{n-1} + F_{n-2},$$

με $F_1 = 1, F_2 = 1$.

Οι αριθμοί Lucas, μοιάζουν αρκετά στους αριθμούς Fibonacci αφού ο τύπος της ακολουθίας είναι:

$$L_n = L_{n-1} + L_{n-2}$$

με $L_1 = 2, L_2 = 1$.

Είναι προφανές πως και οι δύο ακολουθίες οι οποίες παράγουν τους αριθμούς Fibonacci και Lucas ανήκουν στην ίδια οικογένεια ακολουθιών, της μορφής:

$$c_n = ac_{n-1} + bc_{n-2}$$

Γενικότερα υπάρχουν αρκετές εξισώσεις που συνδέουν τους αριθμούς Fibonacci και Lucas [15, 16]. Στη συνέχεια παρουσιάζουμε πως με τη χρήση κάποιων από αυτές τις εξισώσεις, μπορούμε να επιτύχουμε γρήγορο υπολογισμό των όρων των ακολουθιών Fibonacci και Lucas . Πιο συγκεκριμένα, θα παρουσιαστούν τρεις αλγόριθμοι με μικρότερη πολυπλοκότητα αυτού το Takahashi [15] του οποίου ο αλγόριθμος είχε πολυπλοκότητα logn ή οποιουδήποτε άλλου αλγορίθμου [17, 18, 19, 20], και αποτελούν αυτή τη στιγμή τους πιο γρήγορους αλγόριθμους για τον υπολογισμό αυτών των ακολουθιών.

4.7.1 **Αλγόριθμος πρώτος**

Ο αλγόριθμος παρακάτω υπολογίζει αριθμούς Fibonacci και Lucas αντί τάξεως n, τάξης $\frac{n}{4}$. Έτσι η πολυπλοκότητα αυτού του αλγορίθμου είναι :

$$log_4 n = \frac{log_2 n}{log_2 4} = \frac{log_2 n}{2} = log_2 \sqrt{n}.$$

Έχουμε λοιπόν για τον υπολογισμό των αριθμών Fibonacci:

$$F_{4n} = F_{2n} L_{2n} = F_n L_n \frac{(5F_n^2 + L_n^2)}{2} =$$

$$\frac{F_n L_n (5F_n^2 + L_n^2)}{2} =$$

$$\frac{F_n L_n (5F_n^2 + 2(-1)^n)}{2} =$$

$$\frac{F_n L_n (5F_n^2 + 5F_n^2 + 4(-1)^n)}{2} =$$

$$\frac{F_n L_n (10F_n^2 + 4(-1)^n)}{2} =$$

$$F_n L_n (5F_n^2 + 2(-1)^n)$$

$$F_{4n+1} = \frac{F_{4n} L_1 + L_{4n} F_1}{2} = \frac{F_{4n} + L_{4n}}{2}$$

$$F_{4n+2} = \frac{F_{4n} L_2 + L_{4n} F_2}{2} = \frac{3F_{4n} + L_{4n}}{2}$$

$$F_{4n+3} = \frac{F_{4n} L_3 + L_{4n} F_3}{2} = \frac{4F_{4n} + 2L_{4n}}{2} = 2F_{4n} + L_{4n}$$

Γνωρίζοντας ότι:

$$F_n^2 = \frac{L_n^2 - 4(-1)^n}{5}$$

για τον υπολογισμό των αριθμών Lucas έχουμε:

$$L_{4n} = L_{2n+2n} = \frac{L_{2n}^2 + F_{2n}^2}{2} =$$

$$\frac{25F_n^4 + 30L_n^2 F_n^2 + L_n^4}{8} =$$

$$\frac{(L_n^2 - 4(-1)^n)^2 + 6(L_n^2 - 4(-1)^n) L_n^2 + L_n^4}{8} =$$

$$\frac{L_n^4+14-8(-1)^nL_n^2+6L_n^4-24(-1)^nL_n^2+L_n^4}{8} =$$

$$\frac{8L_n^4-32(-1)^nL_n^2+16}{8} =$$

$$L_n^4 - 4(-1)^nL_n^2 + 2$$

$$L_{4n+1} = \frac{F_{4n}L_1+L_{4n}F_1}{2} = \frac{F_{4n}+L_{4n}}{2}$$

$$L_{4n+2} = \frac{F_{4n}L_2+L_{4n}F_2}{2} = \frac{3F_{4n}+L_{4n}}{2}$$

$$L_{4n+3} = \frac{F_{4n}L_3+L_{4n}F_3}{2} = \frac{4F_{4n}+2L_{4n}}{2} = 2F_{4n} + L_{4n}$$

4.7.2 **Δεύτερος αλγόριθμος**

Η πολυπλοκότητα αυτού του αλγορίθμου είναι $log_8 n = \frac{log_2 n}{log_2 8} = \frac{log_2 n}{3} = log_2 \sqrt[3]{n}$ καθώς σε κάθε βήμα του ο αλγόριθμος υπολογίζει αριθμούς Fibonacci και Lucas αντί τάξεως n, τάξης $\frac{n}{8}$.

Για τον υπολογισμό των αριθμών Fibonacci έχουμε:

$$F_{8n} = F_{4n+4n} = F_{4k}L_{4k} =$$

$$F_nL_n(5F_n^2 + {}^n)(L_n^4 - 4(-1)^nL_n^2 + 2) =$$

$$F_nL_n(5F_n^2 + 2(-1)^n)((5F_n^2 + 2(-1)^n)^2 - 2)$$

Γενικά λοιπόν:

$$F_{8n+k} = \frac{F_{8n}L_k+L_{8n}F_k}{2}$$

Για τους αριθμούς Lucas έχουμε:

$$L_{8n} = L_{4n+4n} =$$

$$\frac{L_{4n}^2+5F_{4n}^2}{2} = \frac{(L_n^4-4(-1)^nL_n^2+2)^2+5(F_nL_n(5F_n^2+2(-1)^n))^2}{2}$$

$$= \frac{((5F_n^2+4(-1)^n)^2-4(-1)^n(5F_n^2+4(-1)^n)+2)^2+5(F_nL_n(5F_n^2+2(-1)^n))^2}{2}$$

$$\frac{((5F_n^2+4(-1)^n)^2-4(-1)^n(5F_n^2+4(-1)^n)+2)^2+5(F_n^2(5F_n^2+4(-1)^n)(5F_n^2+2(-1)^n)^2)}{2} =$$

$$\frac{((5F_n^2+4(-1)^n)^2-4(-1)^n(5F_n^2+4(-1)^n)+2)^2+5(F_n^2(5F_n^2+4(-1)^n)(5F_n^2+2(-1)^n)^2)}{2} =$$

$$2+80(-1)F_n^2+500F_n^4+1000(-1)^nF_n^6+625F_n^8$$

4.7.3 **Τρίτος αλγόριθμος**

Ομοίως, σε αυτήν την περίπτωση έχουμε την πολυπλοκότητα αυτού του αλγορίθμου να είναι : $log_{16}n = \frac{log_2 n}{log_2 16} = \frac{log_2 n}{4} = log_2 \sqrt[4]{n}$

Για τους αριθμούς Fibonacci έχουμε:

$$F_{16n} = F_{8n+8n} = F_{8n}L_{8n} = F_nL_n(5F_n^2+2(-1)^n)((5F_n^2+2(-1)^n)^2-2)L_{8n} =$$

$F_nL_n(5F_n^2+2(-1)^n)((5F_n^2+2(-1)^n)^2-2)(2+80(-1)F_n^2+500F_n^4+1000(-1)^nF_n^6+625F_n^8)$

Για τους αριθμούς Lucas έχουμε:

$$L_{16n} = L_{8n+8n} = \frac{L_{8n}^2+5F_{8n}^2}{2} =$$

$$\frac{(2+80(-1)^nF_n^2+500F_n^4+1000(-1)^nF_n^6+625F_n^8)^2+5(F_nL_n(5F_n^2+2(-1)^n)((5F_n^2+2(-1)^n)^2-2))^2}{2} =$$

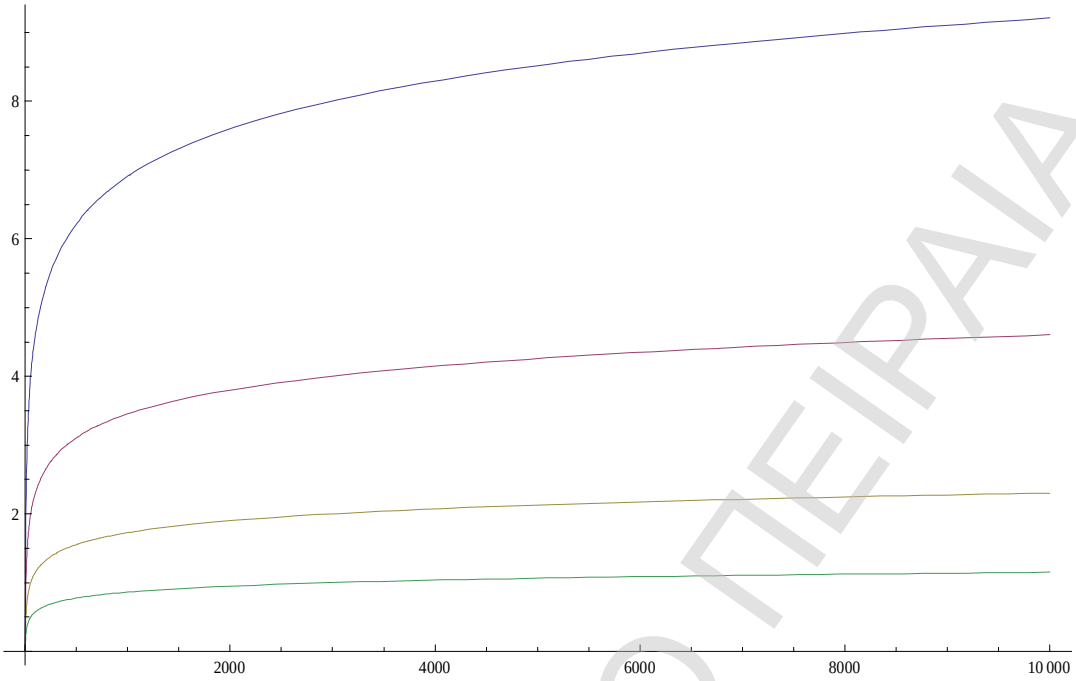
$2+160(-1)^nF_n^2+4200F_n^4+42000(-1)^nF_n^6+206250F_n^8+475000(-1)^nF_n^{10}+812500F_n^{12}+625000(-1)^nF_n^{14}+\frac{390625}{2}F_n^{16}+\frac{5(F_nL_n(5F_n^2+2(-1)^n)((5F_n^2+2(-1)^n)^2-2))^2}{2}$

Στον κατωτέρω πίνακα (πίνακας 18) αλλά και στο γράφημα που ακολουθεί (Σχήμα 23), γίνεται ενδεικτική σύγκριση των αλγόριθμων που παρουσιάστηκαν.

Όρος ακολουθίας Fibonacci	Αλγόριθμος Takahashi (Χρόνος σε sec)	Πρώτος Αλγόριθμος (Χρόνος σε sec)	Δεύτερος Αλγόριθμος (Χρόνος σε	Τρίτος Αλγόριθμος (Χρόνος σε sec)

				sec)
2	0	0	0	0
4	0	0	0	0
8	0	0	0	0
16	0	0	0	0
32	0	0	0	0
64	0	0	0	0
128	0	0	0	0
256	0	0	0	0
1024	0	0	0	0
2048	0	0	0	0
4096	10	5	5	0
8192	0	0	0	0
16384	10	5	0	0
32768	20	10	0	0
65536	30	15	10	0
131072	50	25	0	0
262144	110	55	10	10
524288	200	100	30	10
1048576	380	190	120	95
2097152	790	395	250	190
4194304	1590	795	550	367
8388608	3250	1625	1100	820
16777216	6700	3350	2190	1570
33554432	14170	7085	4800	3570
67108864	29810	14905	9950	7660
134217728	62060	31030	20700	15630
268435456	130950	66540	42620	30450

Πίνακας 18 **Σύγκριση χρόνου μεταξύ αλγορίθμων**



Σχήμα 23 Γραφική αναπαράσταση του χρόνου για τον υπολογισμό των όρων των ακολουθιών από τους αλγόριθμους

4.7.4 **Συμπεράσματα**

Με τη μείωση της πολυπλοκότητας των προτεινόμενων αλγορίθμων, για την εύρεση των αριθμών Fibonacci και Lucas που επιτεύχθει μέσω στην παρούσα διατριβή, δεν γίνεται απλά εφικτός ο υπολογισμός μεγάλων όρων των πολύ σημαντικών αυτών ακολουθιών, αλλά επιπλέον, μπορούν να επιταχυνθούν οι διαδικασίες δημιουργίας του ζεύγους δημοσίου και ιδιωτικού κλειδιού στον αλγόριθμο RSA, μέσω της πιο γρήγορης εύρεσης πρώτων αριθμών. Κάτι το οποίο θα πρέπει ακόμη να τονιστεί, είναι πως παρά την μείωση της πολυπλοκότητας σε σχέση με τον αλγόριθμο του Takahashi, κανείς από τους προτεινόμενους αλγόριθμους, δεν έχει επιπλέον αποθηκευτικές απαιτήσεις σε μνήμη.

Θα πρέπει ακόμα να σημειωθεί ότι ο αλγόριθμος RSA δεν αποτελεί τον μοναδικό αλγόριθμο ο οποίος για τη δημιουργία δημοσίου και ιδιωτικού κλειδιού χρησιμοποιεί πρώτους αριθμούς, καθώς τόσο ο αλγόριθμος του Rabin, ο οποίος έχει πολλές ομοιότητες με τον RSA, όσο και ο αλγόριθμος ElGamal, έχουν ως βασικό τους χαρακτηριστικό την δημιουργία μεγάλων πρώτων αριθμών. Τέλος θα πρέπει να προσθέσουμε, πως αρκετά κρυπτογραφικά πρωτόκολλα, όπως αυτό των Diffie-Hellman, βασίζονται στην εύρεση μεγάλων πρώτων αριθμών. Είναι προφανές, ότι με τους προτεινόμενους αλγορίθμους, οι διαδικασίες αυτές επιταχύνονται.

Κεφάλαιο 5 Νέες μορφές κακόβουλου λογισμικού και Κρυπτογραφία

5.1 Εισαγωγή

Η κρυπτοϊολογία είναι ένας κλάδος της κρυπτογραφίας, ο οποίος αναπτύχθηκε πρόσφατα. Κεντρικό αντικείμενο της έρευνας σε αυτόν τον τομέα είναι η κακόβουλη χρήση της κρυπτογραφίας, η χρήση δηλαδή της κρυπτογραφίας από κακόβουλο λογισμικό και πιο συγκεκριμένα από ιομορφικό λογισμικό. Σκοπός αυτού του κλάδου είναι να μελετηθούν οι τρόποι με τους οποίους οι ιοί υπολογιστών χρησιμοποιούν την κρυπτογραφία, προκειμένου να βρεθούν τρόποι αντιμετώπισης του φαινόμενου. Αν και η κρυπτοϊολογία ξεκίνησε εκτός της ακαδημαϊκής κοινότητας, αφού πρώτοι οι συγγραφείς ιομορφικού λογισμικού ασχολήθηκαν με αυτήν, οι A. Young και M. Yung, είναι οι πρώτοι ακαδημαϊκοί που αντιλήφθηκαν την σημασία της έρευνας σε αυτόν τον τομέα.

Βασικά, οι συγγραφείς ιομορφικού λογισμικού χρησιμοποιούν την κρυπτογραφία για δυο βασικούς σκοπούς, για άμυνα και για επίθεση. Κάθε ιός έχει ως βασικό του σκοπό την εξάπλωσή του και την όσο το δυνατό μεγαλύτερή του παραμονή στον ξενιστή. Η κρυπτογραφία, έρχεται να δώσει επιπλέον όπλα σε έναν ιό, όπως και τον τρόπο με τον οποίο να μεταφέρεται χωρίς να γίνεται αντιληπτός από το λογισμικό καταπολέμησης

ιών (Antivirus), αφού χρησιμοποιώντας την, μπορεί να κρυπτογραφήσει το σώμα του και να το αποκρυπτογραφήσει, καθώς και όλη την κίνησή του. Πολλοί αλγόριθμοι κρυπτογράφησης, όπως ο αλγόριθμος Tiny, μπορούν να υλοποιηθούν σε επίπεδο λογισμικού με πολύ μικρό κώδικα, ο οποίος αντιστοιχεί σε πάρα πολύ μικρή επιβάρυνση για το σώμα του ιού. Δεν θα πρέπει να αγνοείται πως ένας ιός για να μη γίνεται αντιληπτός, θα πρέπει να έχει όσο το δυνατό μικρότερο κώδικα όπως επίσης και τη δυνατότερη μικρή επιβάρυνση την οποία θα κάνει σε υπολογιστική ισχύ του ξενιστή. Παρακάτω παρουσιάζονται συναρτήσεις της C++ για την κρυπτογράφηση και αποκρυπτογράφηση με τον αλγόριθμο Tiny. Βασικό χαρακτηριστικά και των δύο συναρτήσεων είναι το μικρό τους μέγεθος και η απλή υλοποίησή τους.

```
void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;          /* set up */
    uint32_t delta=0x9e3779b9;                    /* a key schedule
constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];  /* cache key */
    for (i=0; i < 32; i++) {                      /* basic cycle start
*/
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                             /* end cycle */
    v[0]=v0; v[1]=v1;
}

void decrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i;  /* set up */
    uint32_t delta=0x9e3779b9;                    /* a key schedule
constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];  /* cache key */
    for (i=0; i<32; i++) {                        /* basic cycle start
*/
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
```

```
        sum -= delta;
    }                                                   /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

Ένας ιός μπορεί να χρησιμοποιήσει την κρυπτογραφία και για να επιτεθεί. Χαρακτηριστικό παράδειγμα αποτελεί η κρυπτογράφηση των δεδομένων του ξενιστή. Ο ιός εγκαθίσταται στον ξενιστή και στο διάστημα το οποίο δεν γίνεται χρήση του ξενιστή, ο ιός κρυπτογραφεί τα δεδομένα που βρίσκονται σε αυτόν, καθιστώντας τα μη προσβάσιμα. Σκοπός της διαδικασίας αυτής είναι ο συγγραφέας του ιού να εγείρει κάποιες απαιτήσεις για την αποκρυπτογράφηση των δεδομένων και την ασφαλή επαναφορά τους. Χαρακτηριστικό παράδειγμα τέτοιου ιού αποτελεί ο GpCode, ο οποίος μέχρι τον Ιούνιο του 2008 είχε χτυπήσει πάνω από 15.000.000 χρήστες και ο οποίος κρυπτογραφεί τα δεδομένα του ξενιστή, χρησιμοποιώντας τον αλγόριθμο RSA για να το επιτύχει, με κλειδί μάλιστα των 4096 bits [59]. Η χρήση γενικότερα ασύμμετρων αλγορίθμων κρυπτογράφησης από τους ιούς είναι πολύ σημαντική και τους καθιστά ακόμα πιο επικίνδυνους.

Ένας ιός ο οποίος θα κρυπτογραφούσε τα δεδομένα κάνοντας χρήση κρυπτογραφίας ιδιωτικού κλειδιού, σημαίνει πως θα χρησιμοποιούσε το ίδιο κλειδί τόσο για την κρυπτογράφηση όσο και για την αποκρυπτογράφηση. Έτσι λοιπόν θα μπορούσε κανείς να ανακτήσει το κλειδί αποκρυπτογράφησης αναλύοντας τον κώδικα του ιού ή ελέγχοντας τη μνήμη που διαχειρίζεται το εκτελέσιμο αρχείο του. Συνεπώς η ανάκτηση δεδομένων θα ήταν εφικτή. Στην περίπτωση όμως του δημόσιου κλειδιού,

το κλειδί της αποκρυπτογράφησης είναι, όπως έχουμε αναφέρει, διαφορετικό από αυτό της κρυπτογράφησης και δεν μπορεί να ανακτηθεί το ένα από το άλλο. Αποτέλεσμα αυτού του γεγονότος είναι πως τα δεδομένα δεν θα μπορούν να ανακτηθούν, αφού μόνο το κλειδί της κρυπτογράφησης θα μπορεί να ανακτηθεί.

5.2 Νέες μορφές κακόβουλων πρακτόρων ιών

Ένας από τους κύριους στόχους της ασφάλειας των υπολογιστών πρέπει να είναι η αποτροπή διαρροών ασφαλείας και η λήψη μέτρων ασφαλείας πριν από τα συμβάντα, και όχι η προσπάθεια διορθώσης κάποιων σφαλμάτων. Αυτό φυσικά σημαίνει ότι η ασφάλεια πρέπει να είναι ένα βήμα μπροστά από τους επιτιθέμενους, όταν αυτό είναι δυνατόν.

Οι ιοί των υπολογιστών έχουν μέχρι τώρα μοντελοποιηθεί με πολλούς τρόπους [3], [4]. Σε πολλά μοντέλα, οι ιοί των υπολογιστών θεωρούνται ως πραγματικοί ιοί και επιδημιολογικά μοντέλα έχουν κατασκευαστεί με σκοπό να τους μελετήσουν [35, 36, 38]. Το επόμενο βήμα στη δημιουργία ιών πρέπει να εντοπιστεί το συντομότερο δυνατόν έτσι ώστε να ληφθούν μέτρα στα συστήματα υπολογιστών.

Είναι βέβαιο ότι με κάποιο τρόπο, οι ιοί των υπολογιστών θα εξελιχθούν. Έχουμε δει μέχρι τώρα ότι τα είδη εξελίσσονται, μετατρέπονται σε καινούργια

είδη, έχοντας στο DNA τους το DNA του παλαιού είδους, καθώς και το φαινόμενο της μετάλλαξης, πολλά είδη να εξελίσσονται σε άλλα νέα. Με αυτό το σκεπτικό, προκύπτουν ερωτήσεις όπως οι ακόλουθες :

- Τι σταματά έναν ιό υπολογιστών από το να εξελιχθεί;
- Μπορεί ένας ιός υπολογιστών να χρησιμοποιήσει τεχνικές τεχνητής νοημοσύνης προς όφελός του;
- Μπορούν οι ιοί των υπολογιστών να κάνουν εξόρυξη δεδομένων (data mining), να κατανοήσουν τα δεδομένα που βρίσκουν στο Διαδίκτυο και να τα εκμεταλλευτούν χρησιμοποιώντας τέτοιες μεθόδους;

Στο πλαίσιο της παρούσας διατριβής προσπαθήσαμε να απαντήσουμε σε αυτές τις ερωτήσεις και την σύνδεση που έχουν με την κρυπτογραφία και πιο συγκεκριμένα με την κρυπτοϊολογία [49]. Από το reverse engineering σε κακόβουλο λογισμικό, την διαδικασία δηλαδή ανεύρεσης του αρχικού κώδικα από τον εκτελέσιμο, ευτυχώς δεν υπάρχει μέχρι στιγμής θετική απάντηση σε καμία από τις παραπάνω ερωτήσεις. Παρόλα αυτά η έρευνά μας απέδειξε ότι ιοί υπολογιστών μπορούν να εξελιχθούν σε νέους ιούς από μόνοι τους, ανεξάρτητα από τον συγγραφέα τους. Ένας ιός μπορεί να χρησιμοποιεί τους πόρους ενός μολυσμένου ξενιστή όχι μόνο για την αναπαραγωγή του, αλλά και για να γίνει πιο ισχυρός.

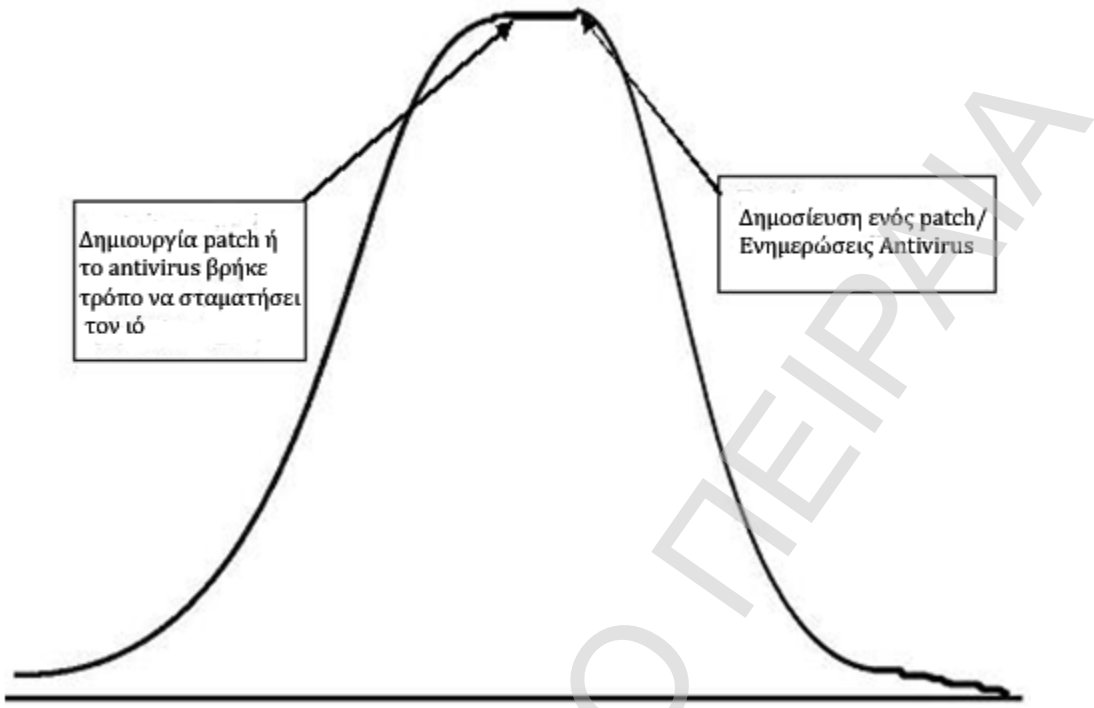
5.3 Ενημερώσιμοι Ιοί Υπολογιστών

Η επιβίωση του ικανότερου, “survival of the fittest”, αυτός είναι ο τρόπος που προτείνει ο Δαρβίνος για να εξηγήσει την εξέλιξη των ειδών. Ο ικανότερος στα είδη όμως δεν είναι ο ισχυρότερος, αλλά αυτός ο οποίος έχει την ικανότητα να προσαρμόζεται καλύτερα στο περιβάλλον του και να χρησιμοποιεί καλύτερα τις δυνατότητές του. Στην έρευνα αυτή, θεωρήσαμε έναν ιό υπολογιστή, ως έναν πραγματικό ιό και προσπαθήσαμε να μεταφέρουμε την πρόταση του Δαρβίνου, σε όρους υπολογιστών όσο αυτό είναι δυνατό, για να μπορέσουμε να προβλέψουμε το μέλλον στη δημιουργία του ιομορφικού λογισμικού.

Ένας ιός δεν γίνεται ισχυρότερος εάν βρει περισσότερα θύματα, αλλά βρίσκοντας περισσότερους τρόπους για να διεισδύσει σε μη μολυσμένα συστήματα, καθώς και βρίσκοντας νέους τρόπους να κρύβεται και να αντιστέκεται στα antivirus, τα προγράμματα ανίχνευσης και αντιμετώπισης ιών. Όπως και στην πραγματική ζωή, τα είδη που προσαρμόζονται καλύτερα στο περιβάλλον τους είναι αυτά που επιβιώνουν και όχι αυτά που καταναλώνουν τα περισσότερα αποθέματα. Θα πρέπει να λάβουμε υπόψη ότι το κύριο μέλημα ενός ιού υπολογιστών είναι να έχει συνέχεια όσο το δυνατόν περισσότερα υπολογιστικά συστήματα υπό την επήρειά του. Όσο περισσότερο παραμένει τόσο καλύτερα για τον δημιουργό του.

Μετά την πρώτη έξαρση που έχουν όλοι οι ιοί [39] (Σχήμα 24), όταν ξεκινούν τη διαδρομή τους στο Διαδίκτυο, παρουσιάζουν μία πτώση στην ανάπτυξή τους, όσο συναντούν περισσότερα συστήματα με ενημερωμένο

λογισμικό ή έχουν διεισδύσει σε όλα τα γειτονικά συστήματα ή τέλος γιατί μία «θεραπεία» έχει βρεθεί από τους προγραμματιστές λογισμικού ή ένα νέο patch, ενημέρωση λογισμικού, από έναν προγραμματιστή έχει δημοσιευθεί και εγκατασταθεί σε πιθανούς και μολυσμένους ξενιστές, κάτι το οποίο φαίνεται και στο παρακάτω γράφημα. Αυτή είναι η αρχή του θανάτου ενός ιού υπολογιστή. Όλο και περισσότερα συστήματα υπολογιστών ενημερώνονται και γνωρίζουν πώς να προστατευθούν από τον ιό και πώς να τον αφαιρέσουν. Έτσι ο ιός δεν μπορεί να βρει καινούργιο ξενιστή που μπορεί να μολύνει, επιπρόσθετα αφαιρείται από τους ξενιστές που έχει ήδη μολύνει. Έτσι είναι σχεδόν αδύνατον να βρούμε σήμερα ένα σύστημα μολυσμένο από τον ιό "Pakistani Brain", τον ιό "Ping-Pong", ή ακόμη και τον ιό "I love you", ιοί που ήταν πολύ γνωστοί στην εποχή τους, αλλά όλοι έχουν εξαφανιστεί από το Διαδίκτυο.



Σχήμα 24 **Η εξάπλωση ενός ιού στο χρόνο**

Έχοντας κατά νου την ανάγκη της «επιβίωσης», θα πρέπει να περιμένουμε ότι ένας ιός υπολογιστών θα ξεκινήσει να ενημερώνεται με πολλούς τρόπους. Θα περιμένουμε ότι ένας ιός υπολογιστών θα είναι σύντομα έτοιμος να κάνει ενημερώσεις στον εαυτό του. Η ενημέρωση θα έχει να κάνει με νέες τεχνικές άμυνας καθώς και με νέες τεχνικές διείσδυσης. Από ότι έχουμε δει μέχρι τώρα, οι ιοί των υπολογιστών ενημερώνονται, αλλά με πιο κεντρικούς τρόπους (Σχήμα 25). Μέχρι στιγμής, ο συγγραφέας του ιού των υπολογιστών γράφει μια ενημέρωση, ένα patch, και το δημοσιοποιεί σε ένα προκαθορισμένο μέρος στο Διαδίκτυο. Συνήθως το patch είναι ένα κρυπτογραφημένο αρχείο το οποίο λαμβάνει ο ιός από το Διαδίκτυο ή ακόμη είναι ενσωματωμένο σε άλλα είδη αρχείων, χρησιμοποιώντας τεχνικές στεγανογραφίας. Αυτό το patch περιέχει, στις περισσότερες

περιπτώσεις, εντολές από τον δημιουργό του ιού προς τους μολυσμένους υπολογιστές, και λέει στους μολυσμένους ξενιστές, τι δεδομένα να στείλουν, πότε και πού.

Ασφαλώς έχουμε ήδη δει τεχνικές μετάλλαξης ιών, όπως με τη χρήση μεταμορφικών και πολυμορφικών ιών. Οι μεταμορφικοί ιοί, είναι ιοί, οι οποίοι κατά τη μετάδοσή τους σε ένα νέο αρχείο ξαναγράφουν τον κώδικά τους, προκειμένου ο ιός «γονέας» να μη μοιάζει με τον ιό «παιδί», κάνοντας έτσι δυσκολότερο το έργο της ανεύρεσής τους. Από την άλλη πλευρά οι πολυμορφικοί ιοί είναι ιοί οι οποίοι έχουν τη δυνατότητα να αλλάξουν ακόμα και το ίδιο το εκτελέσιμο σώμα τους, δυσχεραίνοντας ακόμα περισσότερο το έργο της ανίχνευσής τους. Στην παρούσα έρευνα εξετάζεται κάτι πέρα από αυτό, καθώς τόσο οι πολυμορφικοί ιοί, όσο και οι μεταμορφικοί, δεν είναι σε θέση να αλλάξουν τον τρόπο μετάδοσης αλλά και να προσθέσουν νέα στοιχεία στον ιό από μόνοι τους.

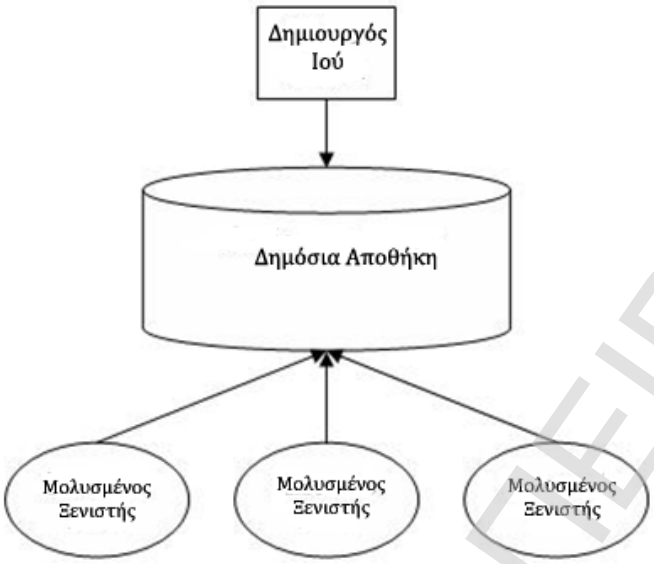


Figure 25 **Κεντρική ενημέρωση ιού**

Εφόσον έχουμε μέσα μεγάλης χωρητικότητας και οι προγραμματιστές έχουν βρει τρόπους να κρύβουν τα προγράμματά τους ακόμη και από τα «partitions», ένας ιός υπολογιστών μπορεί να αποθηκεύσει με ασφάλεια πολλούς τύπους δεδομένων, όπως τους υπολογιστές που έχει μολύνει και τον ξενιστή από τον οποίο έχει μολυνθεί, καθώς και αρκετό χώρο για να αποθηκεύσει άλλα σημαντικά δεδομένα. Αυτό σημαίνει ότι ένας ιός υπολογιστή, γνωρίζει ποιον υπολογιστή να ενημερώσει, πού να κοιτάξει πρώτα για το patch και να έχει αρκετά αποθέματα για να αποθηκεύσει άλλα απαραίτητα στοιχεία για τον εαυτό του.

Θα πρέπει να λάβουμε υπόψη το γεγονός ότι υπάρχουν πολλοί άνθρωποι σήμερα που χρησιμοποιούν ή τουλάχιστον έχουν εγκαταστήσει μία γλώσσα προγραμματισμού όπως είναι η Java, η C++, ή ακόμη καλύτερα για τον ιό, ο υπολογιστής τους μπορεί να μεταγλωττίσει έναν συγκεκριμένο

κώδικα, όπως java-script, vb-script, κ.λπ. Ασφαλώς δεν μιλάμε μόνο για αυτούς που αναπτύσσουν προγράμματα, ή για φοιτητές της επιστήμης των υπολογιστών. Ο οποιοσδήποτε από τους γνωστούς φυλλομετρητές μπορεί να μεταγλωττίσει κάποια scripting γλώσσα. Αυτό σημαίνει ότι ένας ξενιστής διαθέτει τις απαιτούμενες πηγές για να κάνει ενημέρωση σε έναν ιό υπολογιστή.

Έχοντας τους πόρους, ένας ιός υπολογιστών μπορεί να μεταγλωττίσει νέο κώδικα στον ξενιστή του και να δημιουργήσει ένα νέο βελτιωμένο αντίγραφο του εαυτού του. Τα νέα είδη γνωρίζουν πολύ καλά πού πρέπει να χτυπήσουν, μπορούν να ενημερώσουν άλλους μολυσμένους ξενιστές που γνωρίζουν επίσης. Αυτό θα διατηρήσει την ανάπτυξη του ιού σε υψηλά επίπεδα, ή τουλάχιστον θα διατηρήσει αν όχι και θα αυξήσει τον αριθμό των μολυσμένων ξενιστών για περισσότερο χρόνο από ότι ήταν αρχικά σχεδιασμένος.

Αυτό σημαίνει ότι ένας ιός υπολογιστών μπορεί να μεταμορφωθεί σε έναν νέο, και όχι απλά να αλλάξει τη συμπεριφορά του. Επιπλέον, χάρη στις γρήγορες συνδέσεις για το Διαδίκτυο και την μεγάλη ποικιλία των ενεργοποιημένων εφαρμογών Διαδικτύου, οι οποίες είναι εγκατεστημένες σχεδόν σε όλους τους υπολογιστές, η παράνομη κίνηση μπορεί εύκολα να γίνει και να περάσει απαρατήρητη.

5.4 Κακόβουλοι Έξυπνοι Πράκτορες

Μέχρι τώρα έχουμε δει ορισμένα είδη κακόβουλων πρακτόρων να είναι διάσπαρτοι στο Διαδίκτυο. Εάν θα θέλαμε να τους ταξινομήσουμε, τότε θα είχαμε τις ακόλουθες κατηγορίες:

- Γρήγορα Μεταδιδόμενοι Πράκτορες (Rapidly Spreading Agents)

  Οι πράκτορες αυτοί μεταδίδονται στο διαδίκτυο με μεγάλη ταχύτητα.

- Πράκτορες Κατάσκοποι

  Οι πράκτορες αυτοί χρησιμοποιούν τις εξερχόμενες συνδέσεις για να επικοινωνήσουν με τους χειριστές τους με σκοπό να μπορέσουν να μεταδώσουν πληροφορίες του ξενιστή σε κάποια εξωτερική οντότητα .

- Τηλεκατευθυνόμενοι Πράκτορες

  Αυτή η μορφή κακόβουλων πρακτόρων προσφέρει στον επιτιθέμενο τον έλεγχο του ξενιστή σε πραγματικό χρόνο.

- Πράκτορες κατευθυνόμενης επίθεσης

  Αποτελούν ένα άλλο είδος κακόβουλων πρακτόρων που προσφέρουν στον επιτιθέμενο την ικανότητα να διατάξουν κεντρικά ένα «στρατό» από κατανεμημένους πράκτορες. Οι περισσότεροι από αυτούς χρησιμοποιούν κρυπτογραφημένα κανάλια.

Σε συνέχεια των ανωτέρω, ερευνήθηκαν νέα είδη κακόβουλων πρακτόρων τα οποία θα πρέπει να περιμένουμε στο εγγύς μέλλον, οι Update-oriented κακόβουλοι πράκτορες.

5.5 Οι Update-oriented κακόβουλοι πράκτορες

Εάν κάποιος κατηγοριοποιούσε τους hackers, μία από τις βασικές κατηγορίες θα ήταν αυτή που ονομάζεται Script-Kiddies. Ονομάζονται έτσι καθώς γνωρίζουν μόνο πώς να χρησιμοποιούν scripts ή να χρησιμοποιούν απλό κώδικα εκμετάλλευσης, δημοσιοποιημένο στο Διαδίκτυο. Οι hackers δεν τους θεωρούν ως πραγματικό κομμάτι των κοινοτήτων τους καθώς δεν ξέρουν πραγματικά να διεισδύσουν σε ένα υπολογιστικό σύστημα, αλλά χρησιμοποιούν ήδη γνωστό κώδικα που άλλοι έχουν αναπτύξει. Δεν μπορούν να αναπτύξουν κάτι νέο από μόνοι τους.

Στο Διαδίκτυο, υπάρχουν πολλές δημόσιες ιστοσελίδες που περιέχουν vulnerabilities και exploits, δηλαδή τρωτά σημεία λογισμικού και τρόπους εκμετάλλευσης αυτών, συνοδευμένες από τη χρήση τους. Ο κύριος στόχος τους είναι να αποδείξουν την ύπαρξη τρωτών σημείων στο λογισμικό, έτσι ώστε να ενημερωθούν αυτοί που το αναπτύσσουν αλλά και οι χρήστες του για να λάβουν τα κατάλληλα μέτρα προφύλαξης.

Για να μπορέσει να αυτοματοποιηθεί η διαδικασία ενημέρωσης του λογισμικού (patching) και η ανίχνευση τρωτών σημείων, υπάρχουν πολλά εργαλεία εμπορικά ή ανοικτού κώδικα, τα οποία ανιχνεύουν τρωτά σημεία σε ένα σύστημα, για να δείξουν στους χρήστες και στους ελεγκτές ασφαλείας πώς να προστατεύσουν τα συστήματά τους.

Έχουμε ήδη αναφέρει την ανάγκη για ενημερώσεις που έχει ένας ιός και ότι αυτό γίνεται μέχρι στιγμής με έναν κεντρικό τρόπο. Μόλις όμως η πηγή των ενημερώσεων εντοπιστεί, είναι θέμα χρόνου ουσιαστικά ο ιός να σταματήσει να ενημερώνεται. Οι προγραμματιστές λογισμικού εντοπίζουν αυτή την πηγή και σταματούν την πρόσβαση του ιού σε αυτήν. Οι δημιουργοί ιών σύντομα θα πρέπει να βρουν τρόπους να διατηρήσουν την πηγή των ενημερώσεών τους και να αποκεντρώσουν τις ενημερώσεις των ιών που δημιουργούνται από αυτούς. Θα πρέπει να περιμένουμε ότι στο άμεσο μέλλον, ένας ιός υπολογιστών θα είναι σε θέση να ενημερώνει τον εαυτό του χρησιμοποιώντας ενημερώσεις από αξιόπιστα ασφαλή λογισμικά, θα έχει τη δυνατότητα να εκπαιδεύσει τον εαυτό του χρησιμοποιώντας **exploits** που μπορούν να βρεθούν σε δημόσια αξιόπιστες ιστοσελίδες που ασχολούνται με την ασφάλεια. Με αυτόν τον τρόπο ένας ιός θα μπορεί να ενημερώσει τον εαυτό του καθιστώντας τον πιο ισχυρό και κάνοντας τον δημιουργό του ιού περισσότερο ασφαλή, αφού θα είναι δυσκολότερο ακόμα να εντοπιστεί και κατά συνέπεια, πιο ισχυρό. Τους έξυπνους πράκτορες, οι οποίοι έχουν τη δυνατότητα να ενημερώσουν έναν ιό ή γενικά ένα κακόβουλο λογισμικό τους ονομάσαμε **Update-oriented κακόβουλους πράκτορες**.

Ας εξετάσουμε όμως τον τρόπο με τον οποίο ένας ιός υπολογιστών μπορεί να πετύχει αυτό το στόχο του. Παρουσιάζουμε μικρά κομμάτια από προγράμματα υπολογιστών που μπορούν να κάνουν αυτή τη δουλειά,

ψάχνουν το διαδίκτυο για ξενιστές με τρωτότητες, αναζητούν για ενημερώσεις και τις χρησιμοποιούν. Χρησιμοποιούμε απλά Linux scripts, εργαλεία ανοικτού κώδικα και γνωστές ιστοσελίδες από το διαδίκτυο. Ο κώδικας χρησιμοποιήθηκε ως μία απόδειξη της ιδέας, και μπορεί να προσκολληθεί με πολύ μικρή προσπάθεια σε οποιοδήποτε σχεδόν πρόγραμμα. Φυσικά άλλα εργαλεία και ιστοσελίδες θα μπορούσαν να έχουν χρησιμοποιηθεί για να δείξουν την απλότητα αυτού του είδους των ενημερώσεων του ιού. Προσπαθήσαμε να χρησιμοποιήσουμε τα πιο εύκολα, πιο κοινά και πιο ευρέως χρησιμοποιήσιμα εργαλεία και ιστοσελίδες.

Θα πρέπει να επισημάνουμε ότι σε καμία περίπτωση δεν ισχυριζόμαστε ότι αυτές οι ιστοσελίδες και τα εργαλεία δημιουργήθηκαν με σκοπό την κακόβουλη χρήση τους, παρόλα αυτά, η χρήση τους από κάποιους μπορεί να είναι κακόβουλη.

Ας δούμε λοιπόν ένα script που βρίσκει νέα vulnerabilities, κατεβάζει exploits και τα χρησιμοποιεί. Θα χρησιμοποιήσουμε μια αρκετά γνωστή και έγκυρη ιστοσελίδα του Διαδικτύου, αυτήν του SecurityFocus [45], η οποία διαθέτει για κοινή χρήση μια βάση δεδομένων την SecurityFocus Vulnerability Database η οποία περιέχει κώδικα για exploits. Το script βρίσκει νέες επιθέσεις τύπου Buffer Overflow, κατεβάζει το αντίστοιχο exploit και «καταλαβαίνει» πώς να χρησιμοποιήσει αυτό τον κώδικα για να ολοκληρώσει μία επίθεση.

Οι ακόλουθες γραμμές κώδικα χρησιμοποιούν τη μηχανή αναζήτησης του Google [43] για να αναζητήσουν στην ιστοσελίδα του Security Focus επιθέσεις Windows XP buffer overflow.

```
links
http://www.google.gr/q=windows%20xp%20buffer%20overflow&as_sitesearch=
securityfocus.com >myexploit_addresses
```

Στην συνέχεια, οι διευθύνσεις αποθηκεύονται σε ένα αρχείο.

```
getlinks.pl myexploit_addresses >addresses.txt
```

Οι διευθύνσεις αυτές τροφοδοτούνται σε ένα script προκειμένου να εκτελεστεί το exploit σε κάθε μια από αυτές.

```
exec <addresses.txt
while read line
do
        echo $line \exploit
done
```

Από αυτό το σημείο είναι πολύ εύκολο να γράψουμε ένα script το οποίο μπορεί να βρει εάν υπάρχει σύνδεσμος για κώδικα C, C++, Perl ή οποιαδήποτε άλλη γλώσσα και να τον κατεβάσει. Πρέπει να σημειώσουμε

εδώ ότι τα περισσότερα exploits έχουν μία πολύ «καθαρή» χρήση, για παράδειγμα

```
name_of_the_script IP_address
```

Εάν το exploit δεν είναι αυτής της μορφής, τότε ο πράκτορας μπορεί να κάνει εύκολα ένα έλεγχο για τα ορίσματα που χρειάζεται η κύρια συνάρτηση, ή ακόμη με πιο βολικό τρόπο μπορεί να αναζητήσει τη γραμμή που συνήθως εμφανίζεται όταν ένα exploit δεν τρέχει σωστά, προτρέποντας τον χρήστη για τη σωστή χρήση που είναι στις περισσότερες περιπτώσεις της παρακάτω μορφής:

```
Usage: filename IP arg1 arg2
```

Έτσι λοιπόν για να δούμε τη χρήση που μπορεί να έχει μπορούμε να εκτελέσουμε:

```
grep usage
```

ή

```
grep argv
```

ή

```
grep argc
```

Μέσω των ανωτέρω, μπορεί κανείς να δει πόσα ορίσματα χρησιμοποιεί το πρόγραμμα και τις τιμές που είναι συνδεδεμένες με αυτά.

Θα μπορούσαμε επιπλέον να χρησιμοποιήσουμε και έτοιμα exploits τα οποία διαμοιράζονται στο Διαδίκτυο με προγράμματα ανοικτού κώδικα. Το

Nessus [44] είναι ένα πρόγραμμα ανοικτού κώδικα, το οποίο βρίσκει τρωτότητες ενός συστήματος, ασχέτως του λειτουργικού συστήματος που χρησιμοποιείται. Οι δημιουργοί του Nessus αποστέλλουν ενημερώσεις στους χρήστες μετά από επτά ημέρες από τη στιγμή που βρέθηκε κάποιο exploit, και την ίδια μέρα στους εγγεγραμμένους χρήστες. Κατά τη διάρκεια του ελέγχου που εκτελεί το πρόγραμμα, μπορεί κανείς να το θέσει να εφαρμόζει το exploit, αν αυτό εντοπιστεί.

Ας δούμε τι σημαίνει αυτό για κάποιον που θέλει να γράψει έναν νέο ιό. Ο κώδικας για την ανάγνωση αυτών των ενημερώσεων είναι ανοιχτός. Ένας επιτιθέμενος μπορεί να δημιουργήσει έναν ιό που μοιάζει σε αρχιτεκτονική με τον Nessus και διαβάζει τις ενημερώσεις, ενεργοποιώντας πάντα το exploit της τρωτότητας που εντοπίστηκε. Αυτό σημαίνει ότι ο επιτιθέμενος δεν γράφει καθόλου κώδικα εκμετάλλευσης, ούτε ενημερώνει ο ίδιος τον ιό, αντίθετα ο ίδιος ο ιός λαμβάνει ενημερωμένα exploits σχεδόν καθημερινά από ειδικούς, οι οποίοι δεν έχουν καμία σχέση με τη συγγραφή ιομορφικού λογισμικού και οι οποίοι τον εμπλουτίζουν και τον πληροφορούν πώς να τα χρησιμοποιήσει.

Πέρα όμως από το Nessus υπάρχουν και άλλες παρόμοιες πηγές όπως το Metaspoit [42]. Το Metaspoit είναι μια γνωστή πλατφόρμα για έλεγχο exploits, η οποία έχει σχετικά μικρές απαιτήσεις σε υπολογιστική ισχύ και επιπλέον μπορεί να ενημερωθεί. Σε κάθε εγκατάσταση της πλατφόρμας παρέχεται μία κατηγοριοποίηση των exploit ανά Λειτουργικό Σύστημα και

ανά εφαρμογή μαζί με τον κώδικα του exploit αλλά και οδηγίες χρήσης. Το Metaspoit είναι δωρεάν και έχει ανοιχτό κώδικα, κάτι το οποίο σημαίνει ότι είναι εύκολο να δημιουργηθεί ένα πρόγραμμα που έχει πρόσβαση στη δημόσια αποθήκη του και να κατεβάζει τον κώδικα του exploit.

Ας δούμε όμως και μια ακόμη περίπτωση, κατά την οποία ένας ιός μπορεί να ενημερώσει τον εαυτό του για να βρει νέους στόχους βασισμένους σε ένα θέμα. Ας υποθέσουμε ότι ο στόχος είναι να επιτεθεί σε ξενιστές σχετικούς με μια κυβέρνηση, πχ την "mnt government".

Χρησιμοποιώντας τον ακόλουθο κώδικα ένας ιός μπορεί να βρει τις διευθύνσεις νέων ξενιστών και να προσπαθήσει να διεισδύσει σε αυτούς [41].

```
links
http://www.google.com/search?q=mnt%20governmentgovernment>Go
ogleTargs.txt
```

Με αυτή την εντολή μπορούμε να χρησιμοποιήσουμε τη μηχανή αναζήτησης του Google για να βρούμε όλους τους δυνατούς ξενιστές που συνδέονται με το "mnt government" και να τους αποθηκεύσουμε σε ένα αρχείο που ονομάζεται GoogleTargs.txt χωρίς να καταφύγουμε σε σάρωση όλων των δυνατών διευθύνσεων.

Στη συνέχεια με μια δεύτερη εντολή θα μπορούσαμε να βρούμε όλες τις διευθύνσεις του Διαδικτύου που βρίσκονται στο GoogleTargs.txt και να τις υποθηκεύσουμε σε ένα αρχείο που ονομάζεται addresses.txt.

```
getlinks.pl GoogleTargs.txt >addresses.txt
```

Με μια τρίτη εντολή παίρνουμε όλες τις IP από το αρχείο addresses.txt και τις επισυνάπτουμε στο αρχείο που χρησιμοποιεί ο ιός για να αποκτήσει νέους πιθανούς ξενιστές.

```
getIPs.sh addresses.txt
```

Τέλος ας δούμε ένα script που βρίσκει στο Διαδίκτυο νέους πιθανούς στόχους βασισμένο στα vulnerabilies τους. Ας υποθέσουμε ότι ο ιός έχει ήδη αποκτήσει ένα exploit για τον Apache, ένα από τους δημοφιλέστερους web-servers. Τότε με αυτή τη γραμμή κώδικα, ψάχνει για εγκατεστημένους Apache servers που μπορεί να είναι εκμεταλλεύσιμοι [41].

```
links
http://www.google.com/search?q=intitle:%22Test%20Page%20for%20Apache%22 >poorApache.txt
```

Προφανώς με αυτήν την ερώτηση στο Google ανακτούμε νέες εγκαταστάσεις ή λανθασμένα εγκατεστημένους εξυπηρετητές Apache. Το μόνο που μένει είναι να πάρουμε την IP τους.

```
getIPs poorApache.txt
```

5.6 Ανταλλαγή κώδικα/ ανταλλαγή κακόβουλων πρακτόρων

Οι hackers είναι γενικά οργανωμένοι σε μικρές ομάδες που ανταλλάσσουν γνώσεις και exploits φυσικά. Αυτό σημαίνει ότι θα ήθελαν τα προγράμματά τους να ανταλλάσσουν και δεδομένα.

Ένα κακόβουλο πρόγραμμα μπορεί να έχει μερικά extensions / modules που περιγράφουν τον τρόπο με τον οποίο συμπεριφέρεται, πώς εξασφαλίζει πρόσβαση σε απομακρυσμένους υπολογιστές, πώς κρύβει τον εαυτό του, κ.λπ. Αυτά τα modules μπορούν να μοιράζονται σε κάποιες ομάδες hacker, χρησιμοποιώντας αλγόριθμους κρυπτογραφίας δημόσιου κλειδιού.

Κάθε τέτοιός ιός πρέπει να διαθέτει ένα ζεύγος δημοσίου και ιδιωτικού κλειδιού για να επικοινωνεί με τους υπόλοιπους ιούς. Αρχικά οι δυο ιοί χρησιμοποιώντας τα δημόσια κλειδιά ο ένας του άλλου, δημιουργούν ένα ασφαλές κανάλι μέσω του οποίου ο κάθε ένας τους μεταφέρει στον άλλο ένα module. Το module, αφού ελεγχθεί για τη χρήση του και την αποδοτικότητά του, μέσω άλλων ξενιστών, ενσωματώνεται μέσα στο εκτελέσιμο μέρος του ιού.

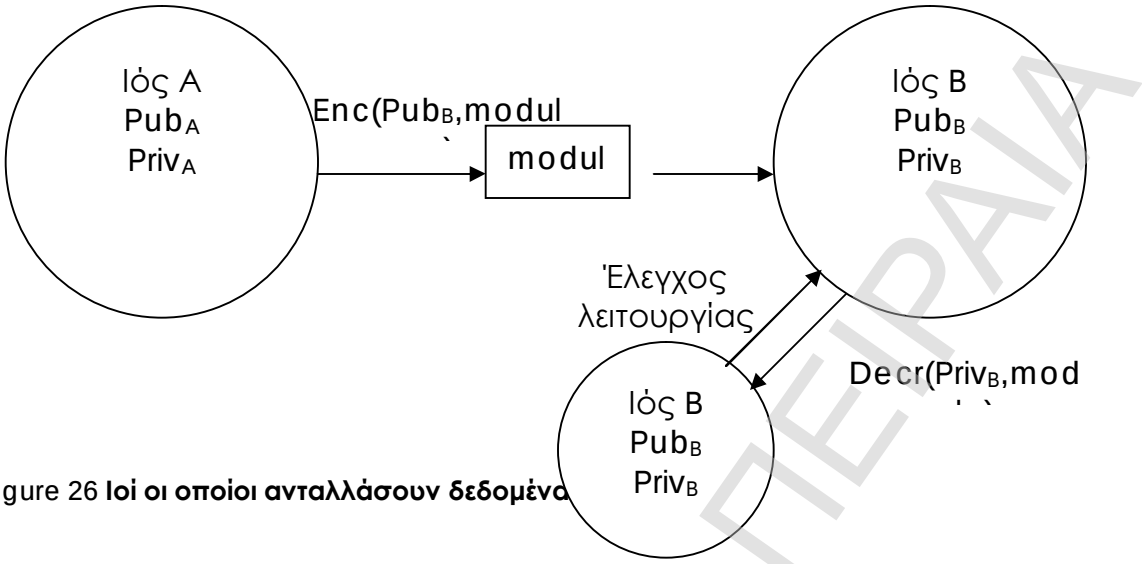


Figure 26 **Ιοί οι οποίοι ανταλλάσουν δεδομένα**

Με αυτόν τον τρόπο ένα κακόβουλο πρόγραμμα μπορεί να στείλει τα δικά του **modules** σε ένα άλλο και να δημιουργήσει ένα νέο. Εάν μιλάμε για ιούς, μπορούμε να πούμε ότι δύο ιοί μπορούν να ανταλλάξουν **zero-day exploits** μεταξύ τους έτσι ώστε να γίνει ο καθένας τους πιο ισχυρός. Θα πρέπει να σημειώσουμε ότι έχουμε ήδη δει ιούς να «αφαιρούν» ο ένας τον άλλο, σε ένα «παιχνίδι» μεταξύ των δημιουργών τους, αφήνοντας σχετικά μηνύματα στους ξενιστές για την επικράτησή τους. Είναι σχεδόν βέβαιο ότι κάποια μέρα αυτοί οι δημιουργοί ιών θα θελήσουν να συνεργαστούν ώστε να κάνουν το λογισμικό τους πιο ισχυρό και πιο μυστικό.

5.7 SK ιοί

Από τα προηγούμενα συνάγεται ότι χρησιμοποιώντας **scripts** όπως αυτά που αναφέρθηκαν αλλά και με τεχνικές τεχνητής νοημοσύνης, αυτού του τύπου οι κακόβουλοι πράκτορες μπορούν να χρησιμοποιηθούν στο

εσωτερικό ιομορφικού λογισμικού έτσι ώστε να τους ισχυροποιήσουν και να δυσχεράνουν το έργο του εντοπισμού τους. Ο αυτοματοποιημένος τρόπος με τον οποίο ο ιός μπορεί να ενημερώνεται, κάνει τον ιό να «μαθαίνει» όπως ένας Script Kiddie. Το συγκεκριμένο λοιπόν είδος ιών το ονομάσαμε Script Kiddie ή πιο απλά SK **ιούς** [49].

Το κύριο χαρακτηριστικό τους είναι ότι συνεχίζουν τις ενημερώσεις τους χωρίς την βοήθεια του δημιουργού τους, και είναι σε θέση να αλλάζουν τον εαυτό τους. Κάνοντας χρήση λογισμικού ανοικτού κώδικα και αξιόπιστες ιστοσελίδες, κάτι το οποίο δεν μπορεί να αποτραπεί. Για παράδειγμα μία μηχανή αναζήτησης όπως το Google δεν μπορεί να σταματήσει να λειτουργεί και να προσφέρει τις υπηρεσίες του στο Διαδίκτυο, επειδή κάποιος το χρησιμοποιεί κακόβουλα.

Μπορεί κανείς να ισχυριστεί, ότι τέτοιοι ιοί θα μπορούσαν εύκολα να εντοπιστούν από το γεγονός ότι μεταγλωττίζουν κώδικα που βρίσκουν από το Διαδίκτυο. Οι υπολογιστές σήμερα έχουν πολλές διεργασίες που τρέχουν, διεργασίες που καταναλώνουν πολύ μνήμη και επεξεργαστική ισχύ για μεγάλο χρονικό διάστημα και διατηρούν συνεχή σύνδεση με άλλους υπολογιστές του Διαδικτύου. Αυτό σημαίνει ότι μία κατανεμημένη μεταγλώττιση κώδικα που κάποιος άλλος μολυσμένος υπολογιστής έχει κατεβάσει δεν μπορεί να εντοπιστεί εύκολα από τον χρήστη. Επίσης πρέπει να σημειώσουμε ότι ο κώδικας εκμετάλλευσης είναι συνήθως μικρός σε μέγεθος, έτσι ο απαιτούμενος χρόνος λήψης, χρησιμοποιώντας τις

μοντέρνες συνδέσεις δεν μπορεί να γίνει αντιληπτός από τους χρήστες. Επιπλέον, εάν μία τέτοια κίνηση πληροφοριών κρυπτογραφείται, είναι πολύ δύσκολο για ένα λογισμικό antivirus να την εντοπίσει και να την σταματήσει.

5.7.1 **Συμπεράσματα**

Έχουμε μέχρι τώρα δείξει ότι οι SK ιοί μπορούν να υπάρξουν και πιθανότατα θα τους αντιμετωπίσουμε στο άμεσο μέλλον. Θα πρέπει να προσπαθήσουμε να βρούμε νέους τρόπους να εντοπίζουμε κακόβουλη κίνηση, χωρίς να παρακωλύουμε την αξιόπιστη. Φυσικά είναι εκτός συζήτησης να αποσύρουμε αξιόπιστα εργαλεία και ιστοσελίδες, επειδή μπορεί κάποιος χρησιμοποιώντας τα κακόβουλα να βλάψει αρκετούς χρήστες. Με αυτόν τον τρόπο σκέψης θα έπρεπε να αποκλείσουμε την χρήση των υπολογιστών.

Ένα άλλο μεγάλο πρόβλημα που εντοπίζεται εδώ είναι η νομοθεσία. Από τη στιγμή που ο συγγραφέας του αρχικού ιού στις περισσότερες περιπτώσεις δεν θα είναι και ο συγγραφέας του κώδικα εκμετάλλευσης που χρησιμοποιείται στο μέλλον, ενώ ο συγγραφέας του νέου κώδικα εκμετάλλευσης δεν είχε την πρόθεση να διεισδύσει σε άλλα συστήματα υπολογιστών, προκύπτουν θέματα ασφαλείας. Θα πρέπει να υπάρχει μία νομοθεσία η οποία να μπορεί να απαντήσει τουλάχιστον στα ακόλουθα ερωτήματα:

- Ποιος θα πρέπει να οδηγηθεί σε δίκη, με ποιες κατηγορίες και με ποια στοιχεία
- Είναι ο συγγραφέας του ιού ο μόνος που θα πρέπει να θεωρηθεί ένοχος σύμφωνα με την τρέχουσα νομοθεσία, όταν έγραψε έναν ιό που μπορούσε να μεταδοθεί σε 10 υπολογιστές για παράδειγμα, αλλά μέσω ενός από αυτούς, ο ιός από μόνος του βρήκε εκμεταλλεύσεις στο Διαδίκτυο, γραμμένες από νόμιμες εταιρείες και προσέβαλε πολύ περισσότερους;
- Θα πρέπει να κατηγορηθεί ο συγγραφέας ενός ιού για ένα exploit που έγραψε κάποιος άλλος και ο ιός του το χρησιμοποίησε από μόνος του;
- Δοθέντος ενός εξελιγμένου ιού, πώς μπορούμε να πούμε ότι γράφτηκε από κάποιον μετά από όλες αυτές τις ενημερώσεις που έχει κάνει ο ίδιος ο ιός;

Από τη στιγμή που οι SK ιοί είναι ένα νέο είδος που μπορεί να εμφανιστεί στο άμεσο μέλλον, οι υπεύθυνοι ανάπτυξης λογισμικού antivirus, πρέπει να είναι έτοιμοι να εντοπίσουν αυτού του είδους κινήσεις και να τις σταματήσουν όταν αυτό είναι δυνατό.

5.8 Torrent Worms

Τα τελευταία χρόνια παρουσιάζεται τρομερή αύξηση στη χρήση του πρωτοκόλλου bittorent. Αρκετές αναφορές όπως η [21], δείχνουν ότι η

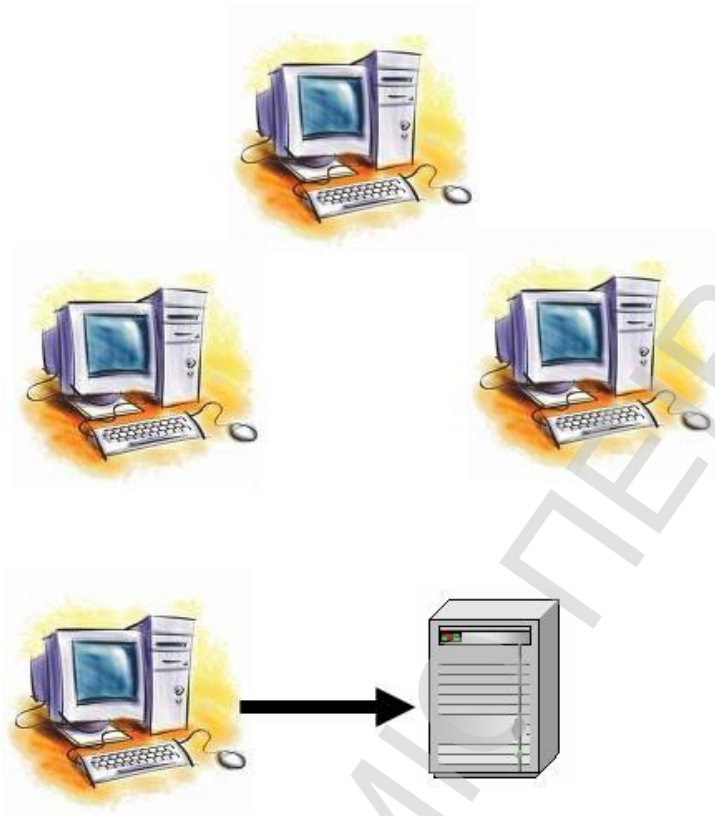
κίνηση peer – to – peer αποτελεί σχεδόν το 50 εως 90% της συνολικής κίνησης του Διαδικτύου. Επιπρόσθετα, ένα ποσοστό της τάξης του 50% εως 75% της peer – to – peer κίνησης χρησιμοποιεί το πρωτόκολλο bittorent [22]. Αυτό σημαίνει ότι το bittorent τροφοδοτεί το 25% εως 60% της κίνησης του Διαδικτύου. Θα πρέπει ασφαλώς να τονίσουμε σε αυτό το σημείο, ότι ένα μεγάλο μέρος της κίνησης αυτής αποτελεί παράνομη ανταλλαγή υλικού όπως λογισμικό, ταινίες, μουσική ή ακόμη και βιβλία.

Ένα από τους σημαντικότερους στόχους ενός ιού είναι η διάδοσή του στο Διαδίκτυο. Στην παρούσα διατριβή, προσπαθήσαμε να εστιάσουμε σε νέους τρόπους μετάδοσης ιών και σε νέα μέσα μετάδοσής τους. Εξετάσαμε λοιπόν, την πιθανότητα ένα worm να χρησιμοποιήσει το πρωτόκολλο bittorent για την διάδοσή του. Τα worms τα οποία εξετάζουμε χρησιμοποιούν το πρωτόκολλο bittorent και δεν είναι αντίγραφα ενός worm τα οποία μεταφέρονται απλά μέσα στα torrents κάποιου χρήστη. Επίσης, πρέπει να εξετάσουμε τον πραγματικό χρήστη τέτοιων εφαρμογών, τη γνώση του σε υπολογιστές και πώς αντιδρά σε τέτοιου τύπου απειλές, καθώς και πώς τα λογισμικά προστασίας έχουν σχεδιαστεί για να αντιμετωπίζουν τέτοιου τύπου επιθέσεις.

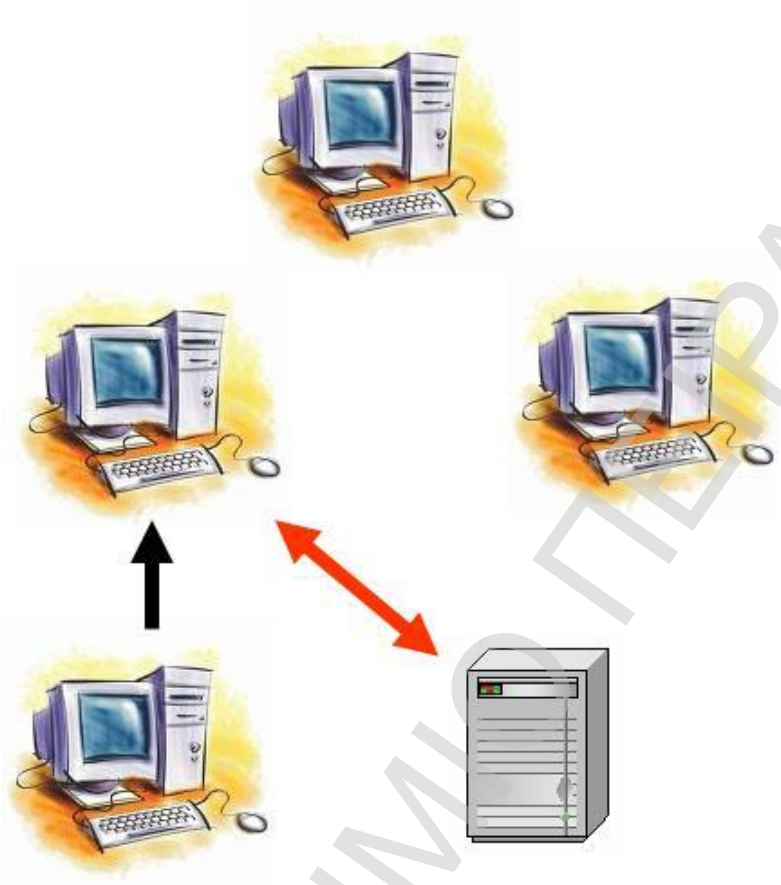
5.8.1 Το πρωτόκολλο bittorent

Το πρωτόκολλο bittorent δημιουργήθηκε από τον Bram Cohen το 2001 [22]. Το κύριο πλεονέκτημα του πρωτοκόλλου είναι ότι ο ιδιοκτήτης του

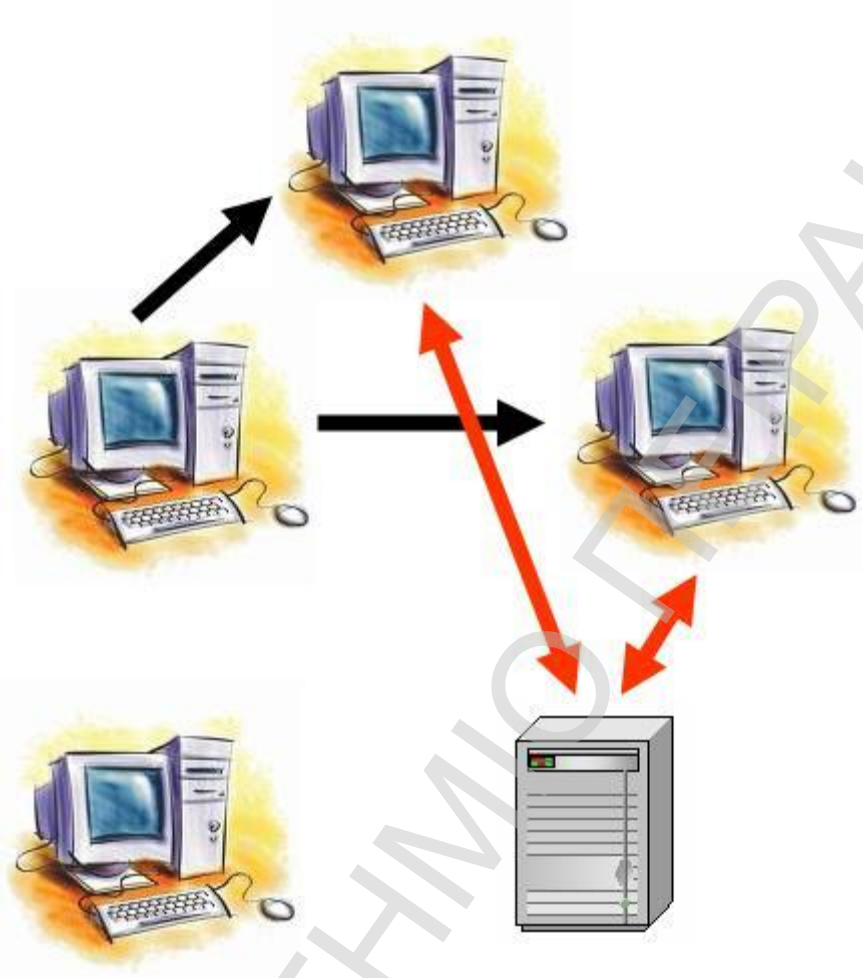
αρχικού αρχείου πρέπει να το ανεβάσει μία μόνο φορά στο δίκτυο για να το κάνει διαθέσιμο σε όλους τους υπόλοιπους. Ο uploader του αρχείου, ο αρχικός διαμοιραστής του αρχείου, ο οποίος το διαθέτει στο Διαδίκτυο, πρέπει να φτιάξει ένα μικρό αρχείο που καλείται torrent, το οποίο περιέχει τα χαρακτηριστικά του αρχείου ή των αρχείων που διατίθονται. Οι πληροφορίες που περιέχει το αρχείο αυτό, είναι τα ονόματα των αρχείων που διανέμονται, ο αριθμός των κομματιών στα οποία κάθε αρχείο είναι χωρισμένο, καθώς και τη hash τιμή του κάθε κομματιού, χρησιμοποιώντας ως συνάρτηση κατακερματισμού την SHA-1. Μετά την δημιουργία του, το torrent αρχείο ανεβαίνει σε έναν εξυπηρετητή (Σχήμα 27), ο οποίος ονομάζεται tracker. Ο εξυπηρετητής αυτός, κρατάει μία πλήρη λίστα με όλους όσους διαμοιράζονται κάποιο μέρος ενός torrent και ονομάζονται peers. Το αρχείο έχει ένα "announce tag" που ονομάζει τον tracker που φιλοξενεί το αρχείο. Άλλα tags του πρωτοκόλλου, συμπεριλαμβάνουν το όνομα και την έκδοση του προγράμματος που χρησιμοποιείται. Κάθε οντότητα που έχει ένα πλήρες αντίγραφο του αρχικού αρχείου ονομάζεται seeder.



Σχήμα 27 **Αρχικά ο** seeder **ανεβάζει στον** tracker **το αρχείο** torrent



Σχήμα 28 **Κάποιος ο οποίος αποφασίζει να αποκτήσει το διαμοιραζόμενο αρχείο επικοινωνεί με τον** tracker, **ο οποίος τον ειδοποιεί για τον υπάρχοντα** seeder.



Σχήμα 29 **Πλέον αν και ο αρχικός διαμοιραστής απουσιάζει, ο** tracker **ενημερώνει όλους τους ενδιαφερόμενους από που μπορούν να λάβουν τα δεδομένα που θέλουν**

Το πρωτόκολλο χρησιμοποιεί τη συνάρτηση κατακερματισμού για να επιβεβαιώσει την σωστή παραλαβή του κάθε αρχείου, ενώ παράλληλα είναι σε θέση να χρησιμοποιεί κρυπτογραφία για την εισερχόμενη και εξερχόμενη κίνηση δεδομένων. Λαμβάνοντας ένα κομμάτι, ένας peer μπορεί να διαμοιραστεί αυτό το κομμάτι σε άλλα peers. Οι παραπάνω εικόνες (Σχήμα 27, 28, 29) δείχνουν πώς το πρωτόκολλο λειτουργεί.

Όπως περιγράφεται στα [23] και [49] υπάρχουν ιοί υπολογιστών που χρησιμοποιούν πιο πολύπλοκες προσεγγίσεις από ότι την τετριμμένη επίθεση ανίχνευσης όλων των δυνατών διευθύνσεων σε ένα δίκτυο. Σήμερα, το κακόβουλο λογισμικό μπορεί να χρησιμοποιήσει πιο καλές προσεγγίσεις για να βρεθούν τρωτοί ξενιστές ώστε να διαδώσουν τον εαυτό τους. Χρησιμοποιώντας μηχανές αναζήτησης όπως το Google και τα API τα οποία προσφέρουν, ένας ιός υπολογιστών μπορεί να ανακτήσει πληροφορίες σχετικά με τρωτούς ξενιστές, όπως άλλωστε είδαμε και στην περίπτωση των SK ιών.

Με τον ίδιο τρόπο ένας tracker μπορεί να χρησιμοποιηθεί ως μία μηχανή αναζήτησης για έναν ιό. Αρχικά, ο tracker αναφέρει ενεργούς ξενιστές, έτσι ο ιός μπορεί αν έχει μία ακριβή λίστα πραγματικά ενεργών IP. Κατόπιν, καθώς στις περισσότερες περιπτώσεις ο υπολογιστής αφήνεται να κατεβάζει ή να διαμοιράζει αρχεία χωρίς φυσική αλληλεπίδραση με τον χρήστη, πολλά δε πράγματα μπορούν να γίνουν κατά τη διάρκεια αυτής της απουσίας. Εν τω μεταξύ, όπως έχουμε ήδη πει, ένα μεγάλο μέρος αυτής της κίνησης είναι παράνομη, οι χρήστες σχεδόν σε όλες τις περιπτώσεις είναι οικιακοί χρήστες, με μικρή αίσθηση της ασφάλειας. Ένας χρήστης που είναι πρόθυμος να κατεβάσει παράνομα μία ταινία ή ένα πρόγραμμα λογισμικού, στις περισσότερες περιπτώσεις, δεν θα τον ενοχλήσει να εκτελέσει «παράξενα» προγράμματα που του υπόσχονται να του δώσουν πρόσβαση σε συγκεκριμένο ψηφιακό υλικό. Στις περισσότερες

περιπτώσεις αυτά τα συστήματα μπορεί να αποδειχθούν ευάλωτα, κυρίως λόγω της μη υπεύθυνης συμπεριφοράς του χρήστη.

Στην περίπτωση ενός ιού, ο συγγραφέας μπορεί να επισυνάψει τον ιό σε ένα περιεχόμενο που διαμοιράζεται και να το δημοσιοποιεί σε έναν tracker στο Διαδίκτυο, προσπαθώντας να κάνει το περιεχόμενό του να φαίνεται δελεαστικό για τους άλλους χρήστες. Αυτό μπορεί να γίνει σχεδόν ανώνυμα, καθώς υπάρχουν πολλοί δωρεάν trackers και ανοικτοί στο κοινό χωρίς εγγραφές. Ο ιός μπορεί πλέον να μεταφερθεί σε πολλούς ξενιστές με την έγκρισή τους. Επιπλέον, το περιεχόμενο του ιού μπορεί με ασφάλεια να μεταφερθεί από τους ξενιστές, χρησιμοποιώντας την κρυπτογράφηση του πρωτοκόλλου, κάτι που πολλοί χρήστες τείνουν να κάνουν έτσι ώστε να κρύψουν την κίνηση η οποία γίνεται από τους παρόχους (Internet Service Providers, ISP). Το worm ή ιός θα μπορούσε να αντιγράφει τον εαυτό του κατά τη δημιουργία κάθε νέου torrent. Αυτό σημαίνει όχι μόνο ότι κάθε ξενιστής συνεχίζει την ανάπτυξη του ιού διανέμοντας αυτό σε άλλους χρήστες, αλλά κάθε φορά που ο χρήστης δημιουργεί ένα νέο torrent, δίδεται στον ο ιό ένα νέο μονοπάτι από ευπαθείς ξενιστές.

Κάτι που θα πρέπει να σημειώσουμε εδώ είναι ότι ο tracker από μόνος του μπορεί να δράσει σαν μηχανή αναζήτησης για τον ιό. Από την ανακοίνωση του κάθε αρχείου torrent, ο ιός μπορεί να βρει τους διαθέσιμους trackers και να κατεβάσει άλλα torrent αρχεία. Έτσι ο ιός μπορεί να πάρει τις IP διευθύνσεις άλλων ενεργών ξενιστών καθώς και

άλλες πληροφορίες όπως την πλατφόρμα του λειτουργικού συστήματος στην οποία τρέχουν ή την έκδοση του λογισμικού. Λίστα από ενεργούς ξενιστές, μπορεί επίσης να ανακτηθεί από τα προσωρινά αρχεία του φυλλομετρητή, όπου τα αρχεία torrent αποθηκεύονται προσωρινά.

Για να εξετάσουμε εάν ένας τέτοιος κίνδυνος είναι πιθανός προσπαθήσαμε να δημιουργήσουμε ένα είδος ιού. Ο «ιός» γράφτηκε σε C++ χρησιμοποιώντας τεχνικές εισαγωγής κώδικα σε Portable Executables [24] σε κάποιο πρόγραμμα το οποίο κάνει χρήση του πρωτοκόλλου bittorrent. Ο ιός με την επανεκκίνηση του προγράμματος, αλλάζει το εκτελέσιμο αρχείο του προγράμματος έτσι ώστε να προσθέσει ένα κενό αρχείο κειμένου σε κάθε νέο αρχείο torrent που δημιουργείται από τον χρήστη. Φυσικά η ανάπτυξη του ιού τώρα εξαρτάται από το περιεχόμενο, το οποίο πρόκειται να διανεμηθεί. Τα τεστ έγιναν σε Intranet, διανέμοντας το κενό text αρχείο. Αυτό που πραγματικά θέλαμε να δούμε ήταν, πώς οι διάφοροι μηχανισμοί προστασίας θα ανταποκριθούν σε αυτό και εάν οι χρήστες θα εντοπίσουν κάποια αλλαγή.

Από την πλευρά των χρηστών, κανένας δεν θα κοίταζε τα αρχεία που διαμοιράζεται προκειμένου να ελέγξει εάν τα αρχεία αυτά έχουν αλλάξει ή εάν τα torrent αρχεία περιέχουν κάτι άλλο από αυτό που υποτίθεται πως έχουν. Αυτό που ήταν ενδιαφέρον, είναι ότι ο χρήστης μπορεί να παρατηρούσε ότι κάτι δεν πηγαίνει καλά, μόνο στην περίπτωση ύπαρξης

ειδοποιήσεως από το firewall ή από το antivirus. Στην περίπτωση αυτή δεν ανταποκρίνονταν με τον ίδιο τρόπο όλα τα firewall. Πρέπει να σημειώσουμε ότι ένα τέτοιο πρόγραμμα δεν αποτελεί αρχείο του συστήματος με αποτέλεσμα οι περισσότεροι μηχανισμοί ασφαλείας να μη δίνουν την απαραίτιτη προσοχή σε αυτό. Σε κάθε περίπτωση όμως, αποτελεί ένα πρόγραμμα που έχει πρόσβαση στο Διαδίκτυο. Πολλά firewalls ζήτησαν από τον χρήστη να αποδεχτούν την κίνηση αυτού του «νέου» client χωρίς όμως να τον ειδοποιούν ότι ο παλιός έχει αλλάξει, αποδεικνύοντας ότι το κριτήριο αποδοχής πρόσβασης ήταν το όνομα του αρχείου και η θέση του.

5.8.2 **Συμπεράσματα**

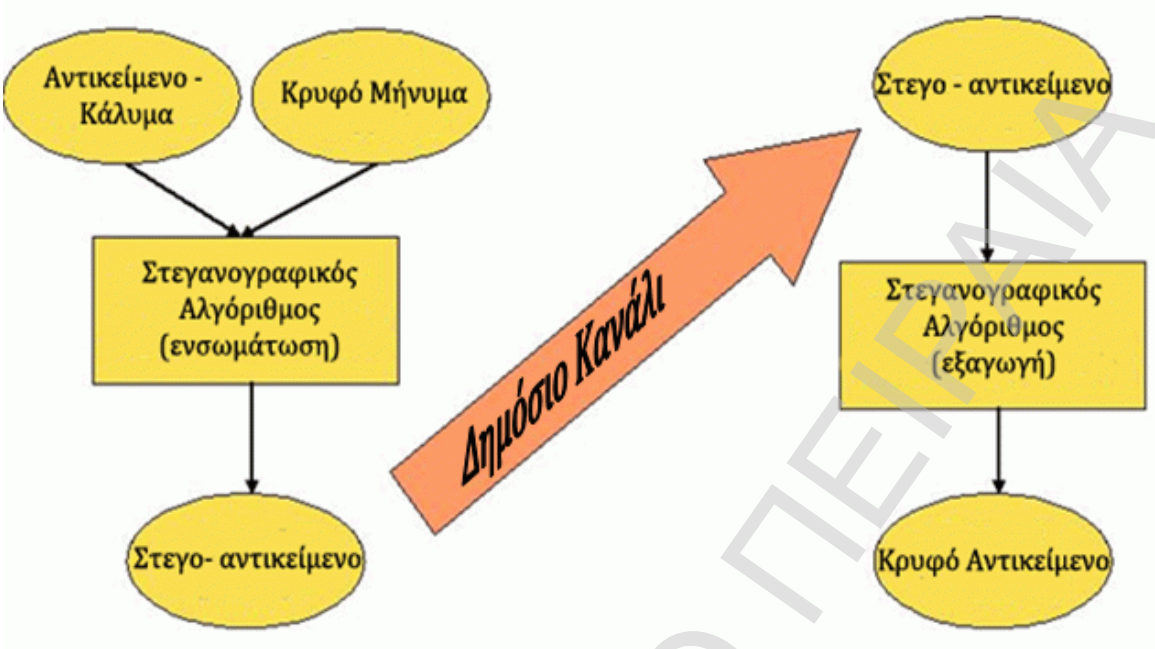
Από την έρευνα αυτή, κατέστη σαφές ότι πρέπει να βελτιώσουμε το υπάρχον λογισμικό ασφαλείας, για να αντιμετωπίσουμε επερχόμενες απειλές. Η κίνηση του πρωτοκόλλου bittorent στο Διαδίκτυο αποτελεί σημαντικότατο μέρος της συνολικής κίνησης του Διαδικτύου, και οι χρήστες είναι ευρέως διασκορπισμένοι σε ολόκληρο τον κόσμο. Θα μπορούσαμε να έχουμε ένα torrent worm το οποίο να αλλάζει την τιμή hash των λαμβανόμενων κομματιών, για παράδειγμα αλλάζοντας ένα bit από το αρχείο που κατέβηκε, αναγκάζοντας τους clients να προχωρήσουν στην επαναποστολή συγκεκριμένων κομματιών αρχείων. Ας υποθέσουμε ότι αυτό θα μπορούσε να γίνει για περίπου το 1% των κομματιών των αρχείων που διαμοιράζονται. Ο χρήστης που μοιράζεται ένα torrent ούτε καν θα

προσέξει ότι κάτι τέτοιο συμβαίνει, έτσι αυτό θα αύξανε τη συνολική κίνηση του bittorent περισσότερο από 0,5%. Συντονισμένες επιθέσεις αυτού του τύπου θα μπορούσαν να οδηγήσουν σε επιθέσεις υπερχείλισης δικτύων των παρόχων Διαδικτύου.

Από την άλλη πλευρά, η ανωνυμία που προσφέρεται από τη διανομή αρχείων μέσω peer-to-peer εφαρμογών μαζί με την κρυπτογράφηση, μπορεί να αποτελέσει ένα επιπλέον τρόπο προστασίας για τους ιούς. Ένας τέτοιος ιός μπορεί να προσπεράσει πάρα πολλά προγράμματα antivirus κατά τη διάρκεια της μεταφοράς του, και η κρυπτογραφία του πρωτοκόλλου να τον κρύψει από αυτά. Η έλλειψη κατάλληλου χειρισμού και κατάλληλης πληροφόρησης του χρήστη από τα συνήθη προγράμματα ασφαλείας μπορεί να παραπλανίσουν τον χρήστη για το τι ακριβώς συμβαίνει στο σύστημά του και δεν θα διακοπεί η πρόσβαση του ιού στο Διαδίκτυο, αλλά ούτε και η εξάπλωσή του σε αυτό.

Κεφάλαιο 6 Ο Στεγανογραφικός Αλγόριθμος HSS

Η απόκρυψη δεδομένων αποτελούσε ανέκαθεν ένα σημαντικό θέμα. Η απόκρυψη δεδομένων έχει δύο έννοιες, απόκρυψη του περιεχομένου των δεδομένων και απόκρυψη της ίδιας τους της ύπαρξης. Όπως είδαμε, η κρυπτογραφία ασχολείται με την απόκρυψη του περιεχομένου των δεδομένων. Η στεγανογραφία από την άλλη, έχει αναπτυχθεί για την ενσωμάτωση δεδομένων σε άλλα μέσα, όπως κείμενο, ήχος ή εικόνα έτσι ώστε να αποκρύπτεται η ύπαρξη των δεδομένων. Ίσως το μοντέλο που περιγράφει καλύτερα την στεγανογραφία είναι το πρόβλημα των κρατουμένων, που δόθηκε στα [25] και [26], όπου δύο κρατούμενοι ο Α και ο Β αντίστοιχα, θέλουν να δραπετεύσουν από την φυλακή. Και οι δύο πρέπει να ανταλλάξουν πληροφορίες χωρίς ο Φύλακάς τους (Warden), τον οποίο θα συμβολίζουμε με W, να καταλάβει ότι κάτι σχεδιάζουν. Εάν ο W καταλάβει ότι υπάρχει ένας ιδιαίτερος τρόπος με τον οποίο ο Α και ο Β μιλάνε μεταξύ τους, τότε θα τους τοποθετήσει σε χωριστούς θαλάμους και έτσι το σχέδιο της απόδρασής τους είναι καταδικασμένο να αποτύχει. Θα πρέπει λοιπόν οι δύο αυτές οντότητες να βρουν ένα τρόπο ώστε να ανταλλάσουν τις απαιτούμενες πληροφορίες στο προαύλιο της φυλακής μπροστά σε όλους, χωρίς να γίνεται αντιληπτό πότε ανταλλάσουν πληροφορίες και ποιες είναι αυτές.



Προκειμένου να ορίσουμε το πεδίο της έρευνας μας πιο φορμαλιστικά, έστω ότι έχουμε δύο οντότητες Α και Β οι οποίες θέλουν να επικοινωνήσουν. Όλη η επικοινωνία γίνεται μέσω ενός δημοσίου καναλιού που ο καθένας μπορεί να έχει πρόσβαση. Η οντότητα Α θέλει να στείλει στην Β ένα μήνυμα m μέσω του δημοσίου καναλιού, έτσι ώστε ούτε το περιεχόμενο, ούτε η ύπαρξη του μηνύματος m να μπορεί να γίνει γνωστό στον οποιονδήποτε παρακολουθεί το δημόσιο κανάλι αυτό. Το δημόσιο κανάλι είναι ο φορέας του μηνύματος και ονομάζεται απλά φορέας (cover-carrier).Το αντικείμενο –κάλυμμα (cover-object) είναι το αντικείμενο στο οποίο ο Α θα ενσωματώσει το μήνυμά του για τον Β. Το νέο μήνυμα που προκύπτει, ονομάζεται στεγο-αντικείμενο. Τέλος υπάρχει μια τρίτη οντότητα,

η οντότητα W, που εξετάζει όλα τα μηνύματα στο δημόσιο κανάλι για να αναγνωρίσει τυχόν στεγο-αντικείμενα.

Για λόγους ασφαλείας, ακολουθούμε την αρχή του Kerckhoff [1] την οποία αναφέραμε και στην εισαγωγή, σύμφωνα με την οποία ο αλγόριθμος που οι οντότητες Α και Β χρησιμοποιούν είναι δημοσίως γνωστός, αυτό που δεν είναι γνωστό είναι το κλειδί, ο κωδικός που χρησιμοποιείται στον αλγόριθμο για την ενσωμάτωση του κρυφού μηνύματος. Έτσι η οντότητα W γνωρίζει τον αλγόριθμο που ο Α και Β χρησιμοποιούν, αλλά δεν γνωρίζει το μυστικό κλειδί. Επιπλέον, η οντότητα W δεν είναι μόνο παθητική, δεν εξετάζει μόνο τα μηνύματα για να βρει αν υπάρχει ένα κρυμμένο μήνυμα μέσα σε αυτά. Η οντότητα W μπορεί επίσης να αλλοιώσει κάθε μήνυμα που θέλει, ανεξάρτητα του αν περιλαμβάνει κάποιο κρυφό μήνυμα. Ο W κάνει αυτές τις αλλοιώσεις στα μηνύματα προκειμένου να παρακολουθεί οποιαδήποτε περεταίρω αλλαγή στην επικοινωνία μεταξύ του Α και Β, ή να την σταματήσει. Σαφώς αυτές οι αλλοιώσεις δεν αλλάζουν το μήνυμα σημαντικά για να μη γίνουν αντιληπτές από τις δυο οντότητες. Έτσι λοιπόν οι Α και Β θα πρέπει να κάνουν την ενσωμάτωση των μηνυμάτων τους με τέτοιο τρόπο, ώστε η διάκριση των στεγο-αντικειμένων από τα αντικείμενα-καλύμματα που διέρχονται μέσα στο δημόσιο κανάλι να μην είναι εφικτή.

Θα πρέπει να έχουμε υπόψη μας ότι ο κύριος στόχος της στεγανογραφίας είναι να κρατήσει την επικοινωνία μεταξύ των Α και Β κρυφή. Προκειμένου να επιτευχθεί αυτό, η διαδικασία ενσωμάτωσης του μηνύματος θα πρέπει

να φαίνεται όσο το δυνατόν πιο τυχαία και το μήνυμα θα πρέπει να είναι ανεξάρτητο από το αντικείμενο-κάλυμμα και από το στεγο-αντικείμενο. Για την μέγιστη δυνατή ασφάλεια, χρησιμοποιούμε κρυπτογραφικές τεχνικές. Έτσι ένα κείμενο δεν ενσωματώνεται αυτούσιο σε ένα στεγο-αντικείμενο, αλλά αντίθετα εισάγεται το κρυπτογράφημά του. Με αυτόν τον τρόπο μπορούν να «σπάσουν» τα στατιστικά τις γλώσσας του μηνύματος, καθιστώντας το ακόμα πιο ασφαλές. Είναι προφανές, πως μέσω της κρυπτογράφησης, ακόμα και αν αποκαλυφθεί πως σε κάποιο αντικείμενο υπάρχουν ενσωματωμένα δεδομένα, δεν θα μπορέσει να γίνει η αποκάλυψή τους. Σε κάθε περίπτωση, θα πρέπει να υποθέτουμε ότι η οντότητα **W** γνωρίζει την κατανομή των στεγο-αντικειμένων και τα στατιστικά που αφορούν αυτά τα αντικείμενα.

Προκειμένου να μετρήσουμε τη θεωρητική ασφάλεια της πληροφορίας ενός στεγανογραφικού συστήματος, εισάγουμε ένα μέτρο για την μη ανιχνευσιμότητα του στεγανογραφικού συστήματος **D**. Το μέτρο για την μη ανιχνευσιμότητα δίνεται στην παρακάτω εξίσωση:

$$D(P_C||P_S) = \int P_C log \frac{P_C}{P_S}$$

όπου το P_c είναι η τυχαία κατανομή του αντικειμένου-καλύμματος και το P_S η τυχαία κατανομή του στεγο-αντικειμένου.

Μια άλλη πολύ σημαντική μέτρηση στη στεγανογραφία είναι η στεγανογραφική χωρητικότητα. Στεγανογραφική χωριτηκότητα είναι το μέτρο που προσδιορίζει τη μέγιστη ποσότητα πληροφορίας που μπορεί να

ενσωματωθεί σε ένα αντικείμενο-κάλυμμα. Η ενσωμάτωση θα πρέπει να γίνει με τέτοιο τρόπο έτσι ώστε το μήνυμα να μπορεί να ανακτηθεί από το στεγο-αντικείμενο, παραμένοντας ωστόσο απαρατήρητο από όλους εκτός από την οντότητα Β. Ανάλογα με το αν έχουμε μια παθητική ή μια ενεργητική οντότητα **W**, έχουμε δύο ορισμούς για τη στεγανογραφική χωρητικότητα C_P και C_A αντίστοιχα. Για την περίπτωση παθητικής οντότητας έχουμε ότι η στεγανογραφική χωρητικότητα C_P, δίνεται από τον τύπο:

$$C_P = \{supH(s_N|c_N) : P_{C_N} = P_{S_N} \kappa\alpha\iota \frac{1}{N} E[d(c_N, s_N) \leq P\}$$

Στο πλαίσιο της παρούσας διατριβής ασχοληθήκαμε με την στεγανογραφία εικόνας.

6.1 Στεγανογραφία εικόνας

Δεδομένου ότι οι εικόνες είναι πιο εύκολο να σταλούν και μικρότερες στο μέγεθος απ' ότι τα αρχεία ήχου (ένα αρχείο **jpeg** είναι πολύ μικρότερο από ένα **mp3**), συνηθίζεται να κρύβουμε πληροφορίες μέσα σε αυτές, επιπλέον η αποστολή μιας εικόνας κινεί πολύ λιγότερες υποψίες για την ύπαρξη κρυμμένων μηνυμάτων, σε σχέση με ένα αρχείο ήχου.

Σχήμα 30 **Η εικόνα πριν την ενσωμάτωση**

Σχήμα 31 **Αυτή είναι η εικόνα** 12 **μετά την στεγανογράφηση**. **Το κλειδί που έχει χρησιμοποιηθεί είναι η** «steganography» **και το κρυμένο κείμενο είναι** : «This is a text to be embeded in the photo»

6.2 Ενσωμάτωση ελάχιστου σημαντικού δυαδικού ψηφίου

Η πιο διαδεδομένη μέθοδος για στεγανογραφία είναι η ενσωμάτωση ελάχιστου σημαντικού δυαδικού ψηφίου (Least Significant Bit / LSB). Η ιδέα πίσω από αυτήν την τεχνική είναι ότι τροποποιώντας το τελευταίο ψηφίο, 1 ή 0, σύμφωνα με το μήνυμα που θέλουμε να κρύψουμε, το αντικείμενο-κάλλυμα δεν θα επηρεαστεί σημαντικά. Αυτή η μέθοδος, γενικά,

χρησιμοποιείται τόσο στη στεγανογράφηση εικόνας όσο και στην στεγανογράφηση ήχου.

Ας υποθέσουμε ότι έχουμε μια εικόνα υψηλής ανάλυσης μέσα στην οποία θέλουμε να ενσωματώσουμε ένα μήνυμα. Κάθε pixel έγχρωμης εικόνας, χαρακτηρίζεται, από τρεις αριθμούς από το 0 ως το 255, οι οποίοι δίνουν τις αναλογίες των βασικών χρωμάτων κόκκινο, πράσινο και μπλε και για αυτό ονομάζονται και RGB (Red Green Blue). Αλλάζοντας το χρώμα ενός pixel στο τελευταίο δυαδικό ψηφίο, η διαφορά δεν μπορεί να γίνει διακριτή από το γυμνό μάτι. Για να κρύψουμε το μήνυμα έτσι ώστε να μην μπορεί ένας «επιτιθέμενος» να εντοπίσει την ύπαρξη ενός κρυμμένου μηνύματος, δεν χρειάζεται να αλλαχθούν όλα τα pixel της εικόνας. Ανάλογα με τον αλγόριθμο, είτε επηρεάζονται συγκεκριμένα τμήματα της εικόνας, είτε ο αλγόριθμος χρησιμοποιεί μια γεννήτρια ψευδο-τυχαίων αριθμών για να δημιουργήσει έναν τυχαίο δρόμο εντός των pixelς της εικόνας και να τα αλλάξει.

Ένας καλός αλγόριθμος για την επιλογή των pixels που θα χρησιμοποιηθούν για την ενσωμάτωση του μηνύματος, έχει προταθεί από την Fridrich. Σε αυτόν τον αλγόριθμο τα ψηφία τα οποία επιλέγονται ακολουθούν την κατανομή του θορύβου που παράγεται από ένα σαρωτή ή από μια κάμερα. Χρησιμοποιώντας αυτόν τον αλγόριθμο η οντότητα W δεν μπορεί να ανιχνεύσει την επικοινωνία μεταξύ των A και B, αφού θα φαίνεται ότι η πηγή της εικόνας είναι ένας σαρωτής ή μια κάμερα και είναι

αναμενόμενο ότι θα έχει κάποιο θόρυβο. Παρόλα αυτά, έχουν αναπτυχθεί εργαλεία προκειμένου να γίνεται εντοπισμός της ύπαρξης πληροφορίας. Άλλωστε διαρκώς οι ανάλυση της εικόνας που προσφέρουν οι φωτογραφικές μηχανές, οι φορητές συσκευές όπως τα κινητά τηλέφωνα τα οποία φέρουν ψηφιακές φωτογραφικές μηχανές, αυξάνει διαρκώς. Συνεπώς δεν δίνεται εύκολα πλέον η δυνατότητα να μιλάμε για λάθη από τη συσκευή λήψης, αφού αυτά μειώνονται συνεχώς. Και μόνο λοιπόν το γεγονός παρόμοιας αλλοίωσης εικόνας είναι ικανό να προδώσει την ύπαρξη μυστικού μηνύματος.

6.3 Ο αλγόριθμος HSS

Στη διατριβή αυτή, μετά από έρευνα στον χώρο της στεγανογράφησης, προτείνεται ένας νέος στεγανογραφικός αλγόριθμος ο Histographic Steganographic System, ή πιο σύντομα HSS [51].

Κεντρική ιδέα του αλγορίθμου ήταν να μπορέσουμε να δημιουργήσουμε ένα σύστημα στεγανογράφησης το οποίο ανάλογα με τον φορέα, να μπορεί να αλλάζει ο τρόπος ενσωμάτωσης και να μη μένει στατικός. Κάτι τέτοιο θα δημιουργούσε μεγαλύτερη δυσκολία στον εντοπισμό του ενσωματωμένου μηνύματος στον επιτιθέμενο, καθώς πρότερη γνώση για την ενσωμάτωση μηνύματος σε άλλη εικόνα δεν θα έδινε τη δυνατότητα εντοπισμού σε μια νέα, προσφέροντας μεγαλύτερη ασφάλεια. Η χρήση λοιπόν των ιδιοτήτων της εικόνας-φορέα κρίθηκε απαραίτητη. Επιπλέον

κρίθηκε απαραίτητη και η χρήση ασφαλών και εγνωσμένης αξίας αλγορίθμων κρυπτογράφησης και συναρτήσεων κατακερματισμού, προκειμένου να φυλαχτεί το μήνυμα που θέλουμε όσο το δυνατό ασφαλέστερα.

Ο προτεινόμενος αλγόριθμος, ο HSS, ενσωματώνει δεδομένα σε JPEG εικόνες έτσι ώστε μόνο ο Α και ο Β να είναι σε θέση να γνωρίζουν την ύπαρξή τους. Η επεξεργασία της εικόνας γίνεται κατά τέτοιο τρόπο ώστε η αρχική εικόνα να μην διαφέρει σχεδόν καθόλου από την στεγο-εικόνα οπτικά, και επίσης οι στατιστικές ιδιότητες της στεγο-εικόνας να μην εμφανίζουν ότι έχει γίνει ενσωμάτωση δεδομένων.

Σε μία πρόσφατη εργασία των Avidan και Shamir [27], προτείνεται ένας νέος αλγόριθμος για την αυξομείωση του μεγέθους μίας εικόνας βασισμένος στο ενεργειακό βάρος των pixels που αφαιρούνται. Οι Avidan και Shamir προσπάθησαν να αυξομειώσουν το μέγεθος μιας εικόνας προσπαθώντας αυτό το γεγονός να προκαλέσει το δυνατό λιγότερο οπτικές διαφοροποιήσεις. Προσπάθησαν να βρουν χώρους της φωτογραφίας οι οποίοι περιέχουν όσο το δυνατό μικρότερη πληροφορία και να τους αφαιρέσουν. Σύμφωνα με αυτόν τον αλγόριθμο, βρίσκουμε seams, μονοπάτια μικρής ενέργειας και τα οποία στη συνέχεια αφαιρούνται οριζόντια ή κάθετα, ανάλογα με το πώς πρέπει να αλλάξουν οι διαστάσεις της εικόνας. Οι κατωτέρω ορισμοί κρίνονται απαραίτητοι προκειμένου να γίνει σαφής ορισμός του προτεινόμενου αλγόριθμου.

Ορισμός: Έστω I είναι μία nxm εικόνα. Ορίζουμε ως **κάθετο** seam το σύνολο:

$$s^x = \{s_i^x\}_{i=1}^n = \{(x(i), i)\}_{i=1}^n$$

τέτοιο ώστε: $\forall i, |x(i) - x(i-1)| \leq 1$ και x είναι μια απεικόνιση $x : [1, ..., n] \rightarrow [1, ..., m]$

Η οπτική ερμηνεία του κάθετου seam, μπορούμε να πούμε ότι είναι ένα κάθετο μονοπάτι από την πάνω πλευρά της εικόνας μέχρι την κάτω. Ομοίως έχουμε και το οριζόντιο seam. Αυτό, φυσικά, είναι ένα οριζόντιο μονοπάτι από την αριστερή πλευρά της εικόνας προς τη δεξιά πλευρά της εικόνας.

Ορισμός: Έστω I είναι μία nxm εικόνα. Ορίζουμε ως **οριζόντιο** seam το σύνολο:

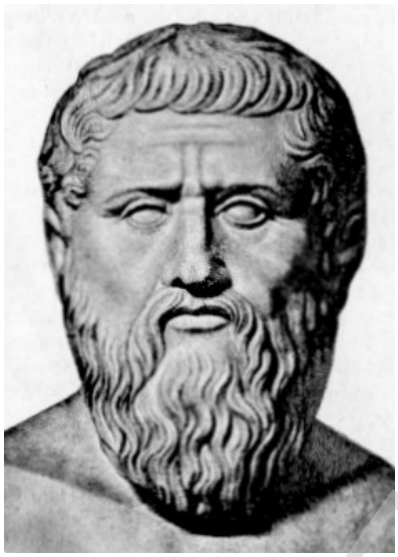
$$s^y = \{s_j^y\}_{j=1}^m = \{(j, y(j))\}_{i=1}^m$$

Τέτοιο ώστε $\forall i, |y(i) - y(i-1)| \leq 1$ και y είναι μια απεικόνιση : $y : [1, ..., m] \rightarrow [1, ..., n]$

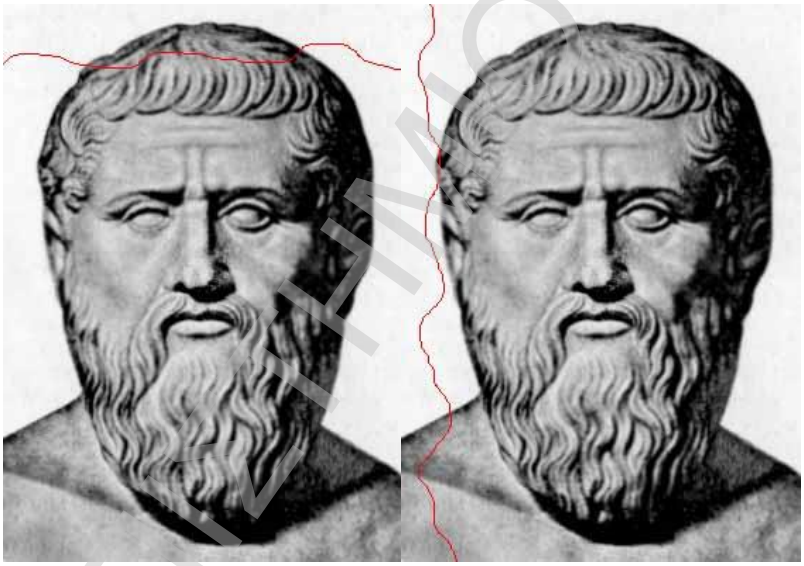
Τα pixels ενός seam s, θα τα συμβολίζουμε με I(s).

Ορισμός: Για μια συνάρτηση ενέργειας E, ορίζουμε ως **κόστος ενός** seam την ποσότητα:

$$E(s) = E(I_S) = \sum_{i=1}^{n} e(I(s_i))$$



Σχήμα 32 **Αρχική εικόνα**



Σχήμα 33 **Οριζόντιο και κάθετο** seam

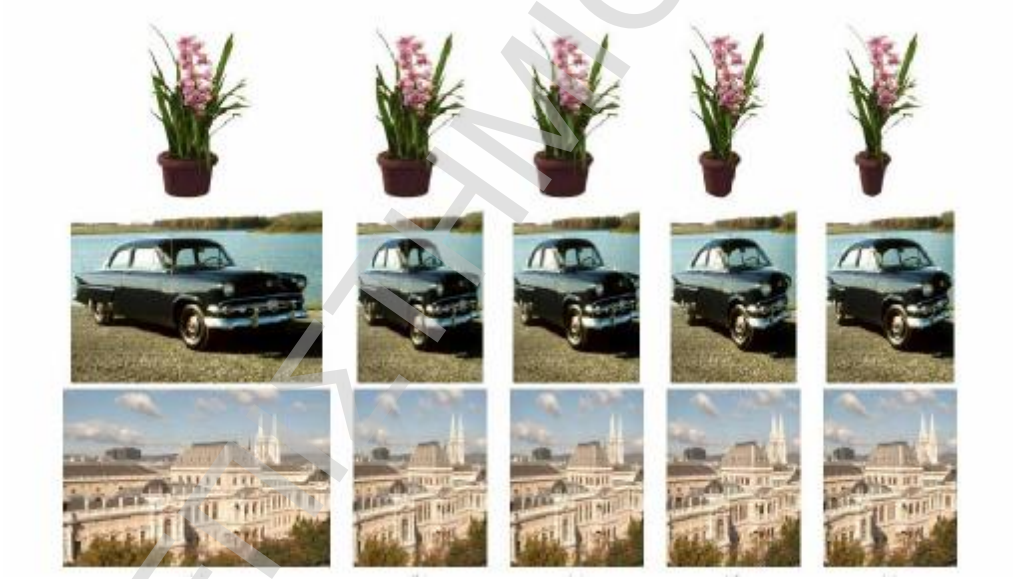
Αντίθετα με τους **Avidan** και **Shamir**, οι οποίοι θεωρούν ως βέλτιστο **seam**, αυτό με το οποίο έχει το ελάχιστο κόστος, θα θεωρήσουμε το **βέλτιστο seam s*** ως το **seam** που μεγιστοποιεί το κόστος ενός **seam**, συνεπώς:

$$s^* = max_s E(s) = \sum_{i=1}^{n} e(I(s_i))$$

Η συνάρτηση ενέργειας που θα χρησιμοποιήσουμε είναι η e_{HoG}

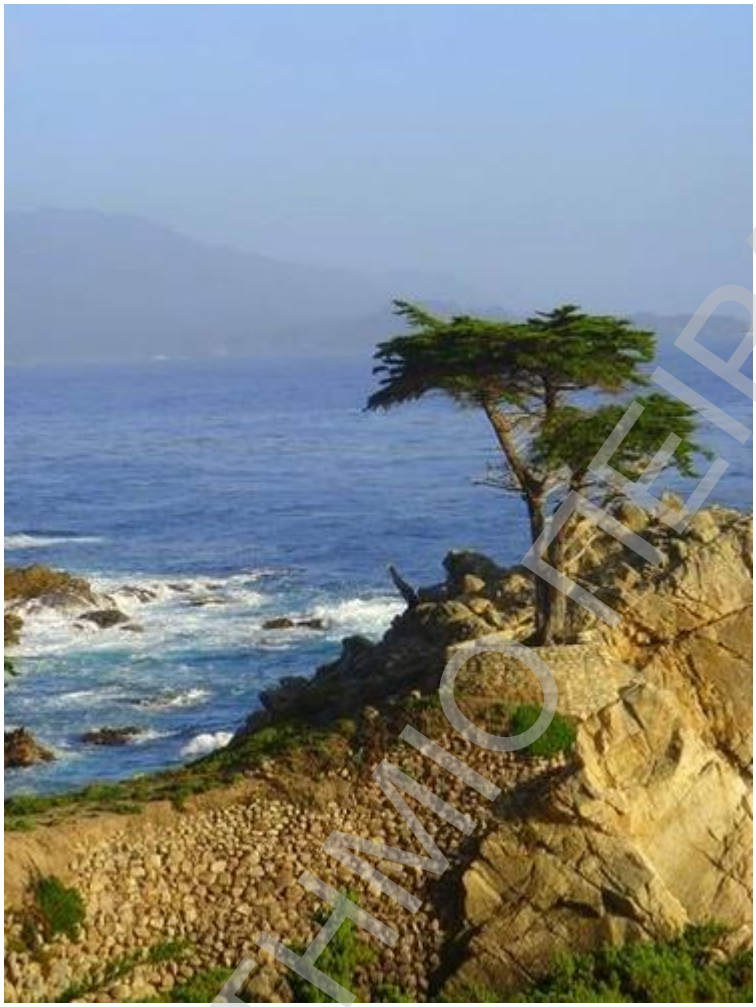
$$e_{HoG} = \frac{|\frac{\partial}{\partial x}I| + |\frac{\partial}{\partial y}I|}{max(HoG(I(x,y)))}$$

Όπου HoG(I(x,y)) είναι ένα ιστόγραμμα προσανατολισμένων βαρών σε κάθε pixel. Χρησιμοποιούμε ένα 8–bit ιστόγραμμα υπολογισμένο σε ένα 11x11 παράθυρο γύρω από κάθε pixel. Επιπλέον, παίρνοντας το μέγιστο του HoG στον παρονομαστή ωθεί τα seams στις άκρες της εικόνας, ενώ ο αριθμητής εξασφαλίζει ότι το seam θα τρέχει παράλληλα στην άκρη και δεν θα την ξεπεράσει.



Σχήμα 34 Μείωση μεγέθους με τον αλγόριθμο seam carving και διαφορετικές συναρτήσεις ενέργειας

Όπως προτείνεται στην εργασία τους, έτσι και στον αλγόριθμο HSS, το βέλτιστο seam μπορεί να βρεθεί χρησιμοποιώντας δυναμικό προγραμματισμό.



Σχήμα 35 **Αρχική εικόνα**

Σχήμα 36 **Εικόνα μετά την εφαρμογή του αλγορίθμου** seam carving. **Είναι προφανές πως τα βασικά στοιχεία της εικόνας δεν αλλοιώνονται οπτικά. Προφανώς οι αναλογίες ύψους προς πλάτος της εικόνας έχουν αλλοιωθεί από** 1:2 **έγιναν σχεδόν** 1:1. **Οι εικόνες όμως δίνουν την ίδια αίσθηση και σχεδόν την ίδια ποσότητα πληροφορίας στον παρατηρητή.**

Σχήμα 37 **Εφαρμογή του αλγορίθμου** seam curving **σε δυο πινακες του** Utagawa Hiroshige. **Οι αλλοιώσεις**, **στην πρώτη αύξηση πλάτους και στη δεύτερη μείωση πλάτους είναι αδιόρατες**.

Σχήμα 38 **Αρχική εικόνα**

Σχήμα 39 **Ενεργειακός χάρτης της εικόνος**

Σχήμα 40 **Το** gradient **της εικόνας**

6.4 Ενσωμάτωση των δεδομένων

Η κύρια ιδέα του αλγορίθμου είναι να αποκρύψουμε τα δεδομένα σε περιοχές της εικόνας που έχουν αρκετή ενέργεια και δεν επηρεάζουν τον τρόπο με τον οποίο η εικόνα εμφανίζεται. Με αυτό τον τρόπο, οι πιθανές αποκλίσεις του DCT (discrete cosine transform) θα αγνοηθούν από τον W, καθώς είναι λογικό, για περιοχές υψηλής ενέργειας, να έχουν μεγάλες αλλαγές γύρω τους. Επιπρόσθετα, καθώς έχουν τόση ενέργεια «τραβούν» το μάτι και συνεπώς δεν μπορούν να εντοπιστούν πολλές διαφορές. Είναι προφανές ότι ο αλγόριθμος αλλαγής διαστάσεων των Avidan και Shamir προσπαθεί να διατηρήσει αυτές τις περιοχές ανεπηρέαστες, καθώς είναι οι πραγματικοί φορείς πληροφορίας. Επίσης, αυτές οι αλλαγές δεν

αλλοιώνουν την εικόνα και δεν προσθέτουν θόρυβο στο σύνολο, καθιστώντας το ύποπτο για τον W. Τέλος, καθώς αυτές οι περιοχές χρειάζονται περισσότερη πληροφορία για να αποθηκευτούν, η αλλαγή τους δεν είναι αντιληπτή ούτε από τη συμπίεση της εικόνας, η οποία γίνεται από το πρότυπο JPEG.

Υποθέτουμε ότι και οι δύο οντότητες μοιράζονται ένα κρυφό κλειδί K, το οποίο θα χρησιμοποιήσουν για να ανταλλάξουν την κρυμμένη πληροφορία. Η εικόνα χωρίζεται σε κομμάτια σύμφωνα με τα βέλτιστα seams. Ο αλγόριθμος κρυπτογράφησης που θα χρησιμοποιήσουμε είναι ο AES, έτσι ώστε τα μεγέθη του κλειδιού που μπορούν να χρησιμοποιηθούν είναι 128, 192 και 256 bits. Το μέγεθος του κλειδιού εξαρτάται από την απόφαση των οντοτήτων A και B. Για να μπορέσουμε να ανιχνεύσουμε πιθανές κακόβουλες ή τυχαίες αλλαγές στην εικόνα θα χρησιμοποιήσουμε τη συνάρτηση κατακερματισμού SHA – 1 [30, 31]. Ο αλγόριθμος όπως θα τον παρουσιάσουμε, είναι στην απλουστευμένη του μορφή αλλά μπορεί να αλλαχθεί ώστε να ανταποκριθεί σε πιο συγκεκριμένες ανάγκες.

Έστω, M το μυστικό μήνυμα που θέλει η οντότητα A να μεταφέρει στον B και I μια εικόνα με n επί m pixels, στην οποία θα ενσωματωθεί το M. Η οντότητα A υπολογίζει την ποσότητα $C = Enc_K(M||SHA(M))$. Συμπληρώνουμε το C έτσι ώστε να έχουμε ένα C' τέτοιο ώστε το μήκος του να είναι πολλαπλάσιο του ύψους της εικόνας, προς χάριν δε απλότητας, συμπληρώνουμε το C με μηδενικά bits. Για να υπολογίσουμε το πλήθος

των κάθετων seams, τα οποία θα αποκρύψουν τα δεδομένα μας μέσα τους το έχουμε:

$$h = \frac{8len(C')}{3m}$$

Αφού σε κάθε pixel αποθηκεύουμε 3 μόνο bits πληροφορίας. Είναι σαφές πως αντί να δουλεύαμε με κάθετα seams θα μπορούσαμε να χρησιμοποιούμε οριζόντια. Χρησιμοποιώντας δυναμικό προγραμματισμό βρίσκουμε h βέλτιστα κάθετα seams τα οποία δεν έχουν τομή και αυξάνουμε την ενέργειά τους με την μικρότερη δυνατή τιμή, έτσι ώστε μετά την ενσωμάτωση πληροφορίας, όλα αυτά τα h seams να έχουν την ίδια δυνατή μέγιστη τιμή. Τώρα παίρνουμε αυτά τα h κάθετα seams και χρησιμοποιώντας την μέθοδο LSB ενσωματώνουμε το C' στα τελευταία bits των pixels.

Είναι προφανές ότι αντιστρέφοντας τα βήματα του αλγορίθμου HSS μπορεί κανείς να εξαγάγει το ενσωματωμένο μήνυμα από την στεγο-εικόνα.

Σχήμα 41 **Αρχική εικόνα**

Σχήμα 42 **Στεγο-εικόνα**

6.5 Απόδοση και ασφάλεια του HSS

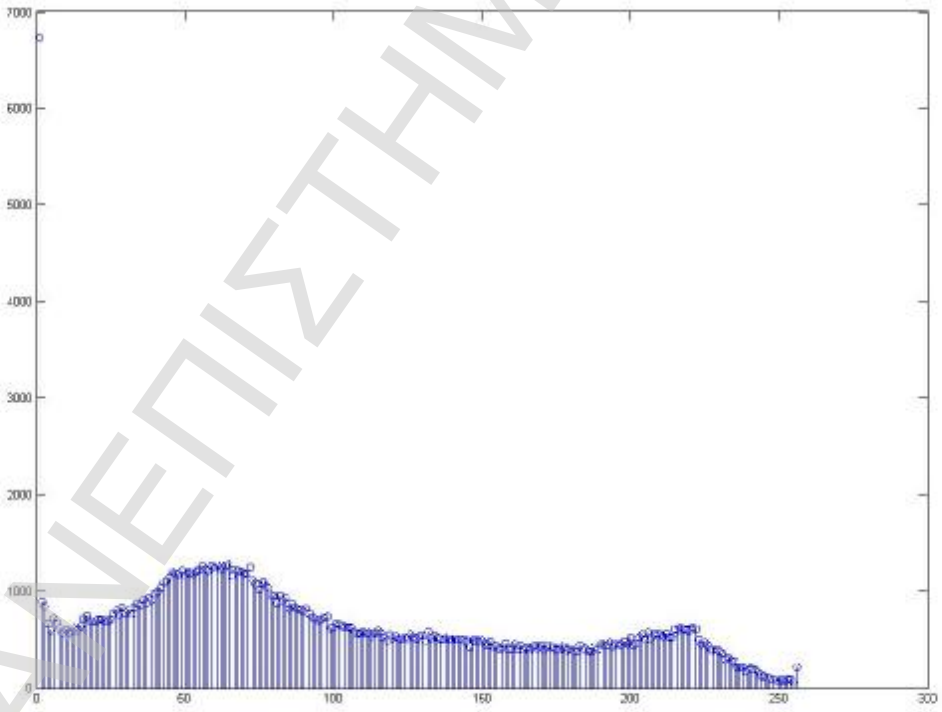
Καθώς ο αλγόριθμος δεν δημιουργεί νέες εικόνες για να κρύψει το μήνυμα αλλά εντοπίζει τα τμήματα της εικόνας που έχουν την περισσότερη πληροφορία, η γενική απόδοση του αλγορίθμου είναι αρκετά γρήγορη. Επιπλέον, εξαιτίας της φύσεως του αλγορίθμου η στεγο-εικόνα και η αρχική εικόνα μοιράζονται σχεδόν ίδια ιστογράμματα, κάτι το οποίο είναι προφανές στα σχήματα 46 και 47.

Επιπλέον, ο αλγόριθμος δεν προσθέτει πληροφορία που μπορεί να εντοπιστεί από το ανθρώπινο μάτι, ή με αυτοματοποιημένα εργαλεία. Είναι χαρακτηριστικό πως ένα από τα πιο γνωστά και αναγνωρισμένα εργαλεία ανεύρεσης στεγανογραφημένων εικόνων, το stegdetect [33] του Provos δεν κατάφερε να αναγνωρίσει την παρουσία κρυμμένων μηνυμάτων στις εικόνες που είχαν στεγανογραφηθεί με τον αλγόριθμο HSS.

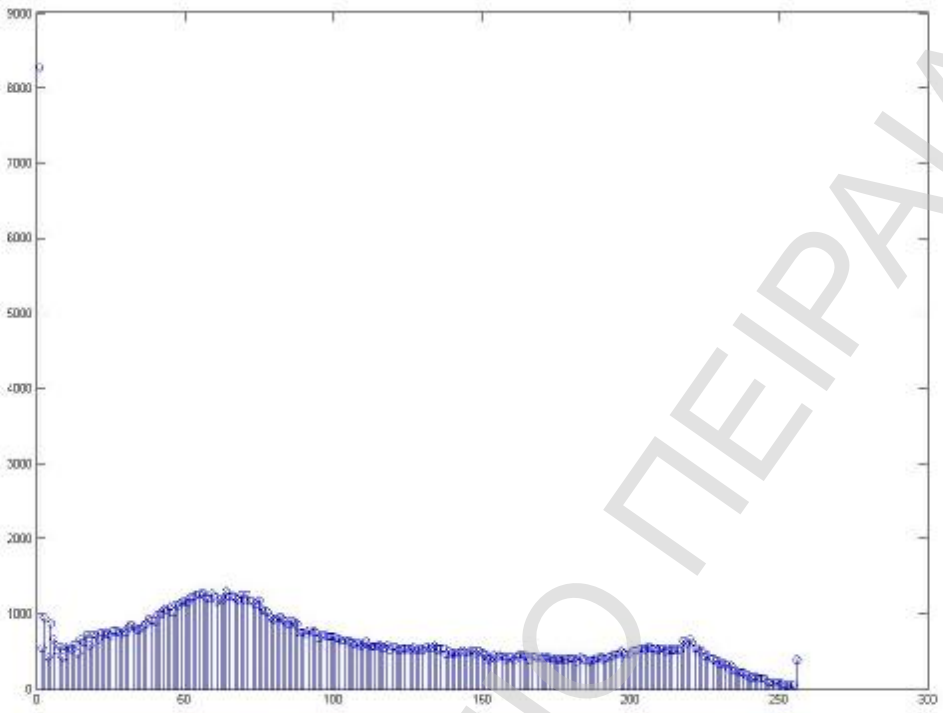
Σχήμα 43 **Στεγο-εικόνα**

Σχήμα 44 **Πίνακας ενέργειας της στεγο-εικόνας**

Σχήμα 45 **Το** gradient **της στεγοεικόνας**



Σχήμα 46 **Ιστόγραμμα αρχικής εικόνας**



Σχήμα 47 **Ιστόγραμμα στεγο-εικόνας**

Μία από τις κύριες ιδιότητες του HSS είναι η ασφάλειά του. Η χρήση του AES και του SHA-1, ως μέρος του αλγορίθμου βελτιώνουν την απόδοση του αλγορίθμου τόσο χρονικά σε σχέση με την ενσωμάτωση όσο και την ασφάλειά του, καθώς είναι αλγόριθμοι οι οποίοι έχουν δοκιμαστεί αρκετά ως προς το θέμα της ταχύτητας και της ασφάλειας που προσφέρουν. Έτσι λοιπόν το μήνυμα είναι κρυπτογραφημένο με ένα εγνωσμένης αξίας αλγόριθμο και επιπλέον μέσω της συνάρτησης κατακερματισμού, είμαστε σε θέση να ελέγξουμε εάν το μήνυμα έχει υποστεί αλλοιώσεις κατά τη μεταφορά του από τρίτη οντότητα.

Κάτι το οποίο πρέπει να τονιστεί για τον αλγόριθμο και το οποίο είναι πολύ βασικό του στοιχείο είναι το γεγονός ότι πέρα από τις μη ανιχνεύσιμες οπτικά αλλοιώσεις στην εικόνα, ο αλγόριθμος δεν αλλοιώνει ούτε το ιστόγραμμα της εικόνας. Από αυτήν άλλωστε την ιδιότητα πήρε και την ονομασία του. Η σημασία αυτού του γεγονότος έγκειται στο γεγονός ότι πολλές ισχυρές στεγαναλυτικές μέθοδοι εντοπισμού, προσπαθούν να εντοπίσουν την ύπαρξη ενσωματωμένου μηνύματος από το ιστόγραμμα της στεγο-εικόνας και αυτό το οποίο θα έπρεπε να έχει.

Ο προτεινόμενος στεγανογραφικός αλγόριθμος ενσωματώνει κρυμμένα μηνύματα σε εικόνες υιοθετώντας κάθε φορά τις ιδιότητες της εικόνας κάλυψης. Αυτό επιτρέπει στο κρυμμένο μήνυμα να είναι όσο το δυνατό λιγότερο ανιχνεύσιμο από στεγανογραφικές επιθέσεις, καθώς οι συντελεστές DCT παραμένουν οι ίδιοι, το ιστόγραμμα της στεγο- εικόνας είναι σχεδόν το ίδιο με το ιστόγραμμα της αρχικής εικόνας και η στεγο-εικόνα δεν έχει καμία εμφανή διαφορά με την αρχική.

Ο HSS χρησιμοποιεί σύγχρονους κρυπτογραφικούς αλγορίθμους προσφέροντας πρόσθετη ασφάλεια για το ενσωματωμένο μήνυμα με καλή στατιστική συμπεριφορά. Σε μερικές περιπτώσεις, ο αλγόριθμος θα μπορούσε ακόμη και να ανακτήσει μέρος του κρυμμένου μηνύματος από εικόνες που έχουν πειραχτεί, χρησιμοποιώντας την συνάρτηση κατακερματισμού ή να ανακτήσει μέχρι την κατεστραμμένη περιοχή. Προκειμένου να γίνει κάτι τέτοιο εφικτό, θα πρέπει να αναζητηθούν τα

οριζόντια/κάθετα seams μέγιστης ενέργειας. Στη συνέχεια, αφού αποσπαστούν τα τελευταία bits του RGB, μπορεί να γίνει αποκρυπτογράφηση με τον AES, ο οποίος είναι αλγόριθμος τμήματος και θα αποκρυπτογραφήσει σωστά μέχρι να βρεθεί οποιαδήποτε αλλοίωση. Ίσως το μόνο αρνητικό σε σχέση με άλλους αλγορίθμους είναι η στεγανογραφική περιεκτικότητα εξαιτίας της χρήσης των seams μέγιστης ενέργειας της εικόνας-φορέα.

6.6 Συμπεράσματα

Ο στεγανογραφικός αλγόριθμος HSS, αποτελεί έναν αρκετά ισχυρό αλγόριθμο στεγανογράφησης, οποίος μπορεί να υλοποιηθεί εύκολα σε επίπεδο λογισμικού. Ο αλγόριθμος καταφέρνει να αποκρύψει με επιτυχία την ύπαρξη των δεδομένων του από σύγχρονα εργαλεία στεγανάλυσης. Επίσης σημαντικό στοιχείο του αλγορίθμου, είναι η διατήρηση του ιστογράμματος της αρχικής εικόνας, γεγονός το οποίο προσφέρει μεγαλύτερη ασφάλεια στον αλγόριθμο, από νέες στεγαναλυτικές επιθέσεις. Θα πρέπει να τονιστεί τέλος η καινοτομία του αλγορίθμου να αποκρύπτει τα δεδομένα ανάλογα με το περιεχόμενο της εικόνας σε περιοχές που δεν θα μπορέσουν να ανευρεθούν, χωρίς την κατοχή του κατάλληλου κλειδιού.

Κεφάλαιο 7 Επίλογος

7.1 Συμπεράσματα της διατριβής

Στη διατριβή αυτή παρουσιάστηκαν αλγόριθμοι για την κρυπτανάλυση αλγορίθμων δημοσίου και ιδιωτικού κλειδιού, ενώ παράλληλα εξετάστηκαν μελλοντικές χρήσεις της κρυπτογραφίας από κακόβουλο λογισμικό, αλλά και οι πιθανές τάσεις ιομορφικού λογισμικού. Στη συνέχεια, παρουσιάζουμε τις συγκεκριμένες συνεισφορές της παρούσας διατριβής.

Στο κεφάλαιο 2, παρουσιάστηκε μια νέα επίθεση στον αλγόριθμο DES, η οποία αποτελεί την πιο δυνατή επίθεση για τον αλγόριθμο αυτό. Η επίθεση αυτή βασίστηκε στην αλγεβρική προσέγγιση των sboxes που χρησιμοποιεί ο αλγόριθμος μέσω γραμμικών συναρτήσεων. Η επίλυση του γραμμικού συστήματος έγινε με τέτοιο τρόπο, ώστε να αποφευχθούν τα τυχόν λάθη τα οποία προκύπτουν από τις προσεγγίσεις των sboxes. Το αποτέλεσμα της επίθεσης είναι η αποκάλυψη 8-bits του κλειδιού σε συγκεκριμένες θέσεις, μειώνοντας το επίπεδο κρυπτογράφησης του αλγορίθμου στα 40-bits, καθιστώντας πλέον δυνατή την αποκάλυψη ολόκληρου του κλειδιού με πολύ χαμηλό κόστος υλικού. Πρακτικά, για τις συσκευές οι οποίες κάνουν αναζήτηση όλων των δυνατών κλειδιών, η χρήση του αλγορίθμου συνεπάγεται και επιτάχυνση της διαδικασίας κατά 2^{16} φορές.

Στο τρίτο κεφάλαιο, μετά την παρουσίαση του AES, αναλύθηκαν οι λόγοι για τους οποίους δεν είναι εφαρμόσιμη η επίθεση που παρουσιάστηκε στο προηγούμενο κεφάλαιο, στον αλγόριθμο AES. Επιβεβαιώθηκε με αυτό τον τρόπο η ασφάλεια την οποία προσφέρει ο αλγόριθμος στις αλγεβρικές επιθέσεις, μέσω της κατάλληλης επιλογής των s-boxes, τα οποία προσφέρουν υψηλή μη γραμμικότητα.

Στο τέταρτο κεφάλαιο, παρουσιάστηκαν οι λόγοι για τους οποίους η συνθήκη ασφαλείας

$$p < q < 2p$$

θα πρέπει να ικανοποιείται κατά την κατασκευή κλειδιών για τον αλγόριθμο του RSA και τον τρόπο με τον οποίο μπορεί να γίνει παραγοντοποίηση ενός αριθμού, στην περίπτωση που η συνθήκη αυτή παραβιάζεται. Το έναυσμα για την έρευνα στο συγκεκριμένο θέμα, αποτέλεσε η παρατήρηση του τρόπου κατασκευής δημοσίου και ιδιωτικού κλειδιού από εφαρμογές. Ένα από τα πορίσματα αυτής της έρευνας ήταν η ανάγκη ενημέρωσης του RFC, το οποίο περιγράφει τον αλγόριθμο RSA. Στη συνέχεια προτάθηκαν τρεις νέοι αλγόριθμοι οι οποίοι έχουν τη χαμηλότερη δυνατή πολυπλοκότητα και τη δυνατότητα να επιταχύνουν την δημιουργία δημοσίου και ιδιωτικού κλειδιού για αλγόριθμους δημοσίου κλειδιού όπως ο RSA, ο ElGammal και ο Rabin, οι οποίοι απαιτούν την εύρεση μεγάλων πρώτων αριθμών. Επιπλέον αποτελούν τους καλύτερους αυτή τη στιγμή

αλγόριθμους για τον υπολογισμό μεγάλων όρων των πολύ γνωστών ακολουθιών Fibonacci και Lucas.

Το πέμπτο κεφάλαιο επικεντρώθηκε στη σχέση μεταξύ κρυπτογραφίας και ιομορφικού λογισμικού, αλλά και τις νέες τάσεις οι οποίες αναμένεται να ακολουθηθούν στη συγγραφή του. Ακόμη αναλύθηκαν νέες τεχνικές για την εύρεση ξενιστών, όπως η χρήση μηχανών αναζήτησης και οι torrent trackers, αλλά και νέες τεχνικές εκμετάλλευσης όπως η χρήση προγραμμάτων εντοπισμού vulnerabilities ή η χρήση ιστοσελίδων που δημοσιεύουν exploits. Παράλληλα παρουσιάστηκε και τρόπος με τον οποίο μπορεί να γίνουν αυτές οι χρήσεις από το ιομορφικό λογισμικό. Νέες εννοιες όπως αυτές των torrent worms, αλλά και των ιών SK, εισήχθησαν και αναλύθηκαν οι ιδιότητές τους, καθώς και τα πιθανά προβλήματα τα οποία μπορούν να παρουσιαστούν από τη χρήση τους σε νομικό επίπεδο.

Στο έκτο κεφάλαιο της διατριβής παρουσιάστηκε ένας αλγόριθμος στεγανογράφησης εικόνων, που δημιουργήθηκε στο πλαίσιο της παρούσας διατριβής, ο οποίος ονομάστηκε HSS. Ένα από τα κύρια χαρακτηριστικά του αλγορίθμου αποτελεί η ενσωμάτωση του μηνύματος σε περιοχές οι οποίες συγκεντρώνουν την περισσότερη ενέργεια, όπως την ορίζουν οι Avidan και Shamir [27]. Έτσι, η εικόνα-φορέας αλλάζει τον τρόπο με τον οποίο γίνεται κάθε φορά η ενσωμάτωση του μηνύματος. Ο αλγόριθμος επιτυγχάνει όχι μόνο να αποκρύψει οπτικά τα μηνύματα τα οποία ενσωματώνει, αλλά να τα αποκρύψει και από τα πλέον σύγχρονα

στεγαναλυτικά εργαλεία. Τέλος, ο HSS χρησιμοποιεί ισχυρές κρυπτογραφικές συναρτήσεις με καλές στατιστικές ιδιότητες, όπως ο AES και η SHA-1, προκειμένου να επιτύχει υψηλότερα επίπεδα ασφάλειας.

7.2 Μελλοντική έρευνα

Αρκετοί ερευνητικοί τομείς παραμένουν στο χώρο τόσο της κρυπτογραφίας, όσο και στο χώρο συγγραφής και εντοπισμού ιομορφικού λογισμικού. Στις επόμενες παραγράφους παρουσιάζεται το μελλοντικό ερευνητικό έργο, το οποίο πηγάζει απευθείας από την πρόοδό μας σε αυτή τη διατριβή.

Η ανεύρεση ταχύτερων αλγορίθμων για τον υπολογισμό των όρων των ακολουθιών Fibonacci και Lucas, μπορεί να οδηγήσει σε κάποιον ακέραιο, επιταχύνοντας αλγορίθμους δημοσίου κλειδιού, όπως ο Luc [60].

Η μελέτη και άλλων συμμετρικών αλγορίθμων πέρα από τους AES και DES, για την αλγεβρική δομή τους καθώς και η επέκταση της αλγεβρικής επίθεσης σε σχέση με άλλες επιθέσεις είναι ένας χώρος ο οποίος χρήζει μελέτης και μεγαλύτερης εμβάθυνσης. Η εύρεση καλύτερων αλγεβρικών εργαλείων, η παράλληλη χρήση μεθόδων που χρησιμοποιούνται από τις μεθόδους XL και XSL [61, 62, 63, 64, 65, 66], είναι ίσως ικανές να δώσουν μια μεγαλύτερη ώθηση στην κρυπτανάλυση αλγορίθμων ιδιωτικού κλειδιού.

Σε ό,τι αφορά την έρευνα του ιομορφικού λογισμικού, μπορούν να γίνουν πολλές προσπάθειες για την καλύτερη διαχείριση των vulnerabilities και των

exploits, τα οποία κυκλοφορούν στο Διαδίκτυο, παρόλα αυτά ο τρόπος με τον οποίο γίνεται η πληροφόρηση των ενδιαφερομένων φορέων αλλά και ο τρόπος με τον οποίο ανακοινώνονται τα exploits, όπως είδαμε μπορεί να αποτελέσει πηγή τροφοδότησης για κακόβουλο λογισμικό. Αυτό σημαίνει πως θα πρέπει να βρεθούν καλύτεροι τρόποι μετάδοσης αυτής της γνώσης, οι οποίοι δεν επιτρέπουν αυτοματοποιημένες επιθέσεις. Η χρήση μηχανών αναζήτησης όπως και οι torrent trackers από τους ιούς είναι εφικτή. Ήδη μάλιστα, ιοί οι οποίοι κάνουν χρήση των μηχανών αναζήτησης έχουν εμφανιστεί. Κρίνεται λοιπόν απαραίτητη η εύρεση μεθόδων αντιμετώπισης ιών τέτοιου είδους, όπως και η έγκαιρη προσέγγιση πιθανών προβλημάτων από υπηρεσίες και πρωτόκολλα στα οποία γίνεται μεγάλη χρήση. Βασικό στόχο αποτελεί σίγουρα η κοινωνική δικτύωση (social networking), μέσω ιστοσελίδων οι οποίες επιτρέπουν την εγκατάσταση εφαρμογών, όπως το πολύ γνωστό Facebook [67], η ασφάλεια του οποίου πρόκειται να μελετηθεί. Παράλληλα, είναι βέβαιο ότι η μεγαλύτερη ανάπτυξη των συμπεριφορικών (behaviornal) antivirus και η μεταξύ τους ανταλλαγή δεδομένων για την καλύτερη, έγκυρότερη και έγκαιρότερη αντιμετώπιση, αποτελεί επιτακτική ανάγκη.

Τέλος, υπάρχει πολύς χώρος για την περαιτερω κρυπτανάλυση αλγορίθμων δημοσίου κλειδιού. Πιο συγκεκριμένα, η έρευνα σε αλγορίθμους οι οποίοι βασίζονται στο πρόβλημα της παραγοντοποίησης μεγάλων ακεραίων αριθμών αλλά και το πρόβλημα του διακριτού

λογαρίθμου, έχει ακόμα πολλά αναπάντητα ερωτήματα ακόμη και θεωρητικά μεγάλα περιθώρια βελτίωσης των υπαρχόντων αλγορίθμων.

Βιβλιογραφία

1. Auguste Kerckhoff, "La cryptographie militaire", Journal des sciences militaires, vol. IX, pp. 5–83, Jan. 1883, pp. 161–191, Feb. 1883.
2. Nicolas Courtois, Willi Meier: Algebraic Attacks on Stream Ciphers with Linear Feedback. Eurocrypt 2003, LNCS 2656, pp. 345-359, Springer.
3. National Bureau of Standards, Data Encryption Standard, Federal Information Processing Standards Publication 46, January 1977
4. Mitsuru Matsui, Linear Cryptanalysis for DES Cipher,Workshop on the theory and application of cryptographic techniques on Advances in cryptology table of contents,Lofthus, Norway, 386 - 397, 1994
5. Cryptography Theory and Practice,2nd Edition, Douglas R. Stinson, CRC Press, Inc
6. Eli Biham, Adi Shamir, Differential Cryptanalysis of the Data Encryption Standard, Springer-Verlag, 1993
7. Nicolas Courtois, Josef Pieprzyk: Cryptanalysis of Block Ciphers with Overdefined Systems of Equations, Asiacrypt 2002, LNCS 2501, pp.267-287, Springer.
8. Tom M. Apostol “Introduction to analytic number theory”
9. ``Handbook of Applied Cryptography'', Alfred J. Menezes, Paul C. van Oorschot and Scott A. Vanstone, CRC Press, 1996

10. “Applied Cryptography,Second Edition”, Bruce Schneier,John Wiley & Sons, 1996
11. RFC2437- PKCS #1: RSA Cryptography Specifications Version 2.0
12. R. Rivest, A. Shamir, L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". Communications of the ACM 21: pp.120-126, 1978
13. Lenstra, H. W. Factoring integers with elliptic curves. Annals of Mathematics 126 (1987), 649–673.
14. Brent, R. P. Some integer factorization algorithms using elliptic curves. Australian Computer Science Communications 8 (1986), 149–163
15. D. Takahashi, “A fast algorith for computing large Fibonacci numbers”, Information Processing Letters 75(200) 243-246
16. J.Atkins, ”Fibonacci Numbers and Computer Algorithms”, Robert Geist College Mathematics Journal, Vol. 18, No. 4 (Sep., 1987), pp. 328-336
17. Paul Cull,James L. Holloway,"Computing Fibonacci numbers quickly",Information Processing Letters archive,Volume 32 , Issue 3 (August 1989),pp 143 - 149
18. M.C Er, “Computing sums of order-K Fibonacci numbers in log time”, Inform. Process Lett. 32(1989) 143-149

19. D. Gries and G. Levin, “Computing fibonacci numbers (and similarly. defined functions) in log time”, Information Processing Lett., vol. 11,. no. 2, pp. 68-69
20. A.J. Martin, M. Rem, “A representatio of the Fibonacci algorithm”, Inform. Process. Lett. 19 (1984) 67-68
21. iPoque Internet Study, http://www.ipoque.com
22. Cohen, Bram, “BitTorrent Protocol 1.0”, http://www.BitTorrent.org
23. N. Provos, J. McClain, Ke Wang, “Search worms”, Conference on Computer and Communications Security archive, Proceedings of the 4th ACM workshop on Recurring malcode table of contents,pp 1-8, 2006
24. Microsoft Portable Executable and Common Object File Format Specification
25. Kharrazi, M., Sencar, H. T., and Memon, N., :Image steganography: Concepts and practice". In WSPC Lecture Notes Series, 2004.
26. Simmons, G. J. , :The prisoners problem and the subliminal channel. In Advances in Cryptology: Proceedings of Crypto 83, pages 51-67. Plenum Press,1984.
27. Shai Avidan, Ariel Shamir, "Seam Carving for Content-Aware Image Resizing",ACM Transactions on Graphics, Volume 26, Number 3, SIGGRAPH 2007.
28. FIPS PUB 197: Advanced Emcryption Standard

29. J. Daemen, V. Rijmen, "The Block Cipher Rijndael", Smart Card Research and Applications, LNCS 1820, J.-J. Quisquater and B. Schneier, Eds., Springer-Verlag, 2000, pp. 277-284.
30. RFC 3174, US Secure Hash Algorithm 1 (SHA-1)
31. FIPS 180-2: Secure Hash Standard (SHS)
32. Dalal N., Triggs B. , "Histograms of oriented gradients for human detection". In International Conference on Computer Vision & Pattern Recognition, vol. 2, 886-893, 2005.
33. Niels Provos, Stegdetect, http://www.outguess.org/download.php
34. Fred Cohen, “Models of Practical Defenses Against Computer Viruses”
35. Jeffrey O. Kephart and Steve R. White, “Directed-Graph Epidemiological Models of Computer Viruses”
36. Cohen, F., 1987. "Computer Viruses Theory and Experiments," Computers and Security, vol. 6, pp. 22--35.
37. F. B. Cohen, “A Short Course on Computer Viruses” (2d ed. 1994)
38. W. H. Murray, ``The application of epidemiology to computer viruses," Computers & Security, vol. 7, pp. 130-150, 1988.
39. Winfried Gleissner, ``A mathematical theory for the spread of computer viruses," Computers & Security, vol. 8, 1989, pp. 35-41.
40. Peter S. Tippett, ``Computer virus replication," Comput. Syst. Eur., vol. 10, 1990, pp. 33-36.

41. Long, J., “Google Hacking For Penetration Testers”, Syngress Media,U.S. ,2204
42. Metasploit framework, www.metasploit.com
43. Google search engine, www.google.com
44. Nessus Security scanner, www.nessus.org
45. SecurityFocus Vulnerability Database, www.securityfocus.com
46. Symantec Security Response, http://www.symantec.com/security_response
47. Patsakis Constantinos, Alexandris Nikolaos, “Finding factorable spheres in [1,n]”, ACA 2006, 12th International Conference on Applications of Computer Algebra.
48. Patsakis Constantinos, Alexandris Nikolaos, “An algebraic attack on DES”, Proceedings of OC'2005, Fourth International Workshop on Optimal Codes and Related Topics (Pamporovo, Bulgaria), 2005.
49. Patsakis Constantinos, Alexandris Nikolaos, "New malicious agents and SK virii",p. 58, International Multi-Conference on Computing in the Global Information Technology (ICCGI'07), 2007. Also on IEEE Computer Society Press, IEEE Digital Library, 2007.
50. Patsakis Constantinos, Foundas Evangellos, Christos Lytras, “Improved algorithms for the calculation of Fibonacci numbers”, Journal of Discrete Mathematical Sciences & Cryptogra-phy, Volume 11, Number 1, IOS Press, Taru Publishing, Feb-ruary 2008.

51. Patsakis Constantinos, Alexandris Nikolaos “Histographic Steganographic System” New Directions in Intelligent Interac-tive Multimedia, Studies in Computational Intelligence , Vol. 142 , Studies in Computational Intelligence, Springer-Verlag.
52. Patsakis Constantinos, Alexandris Nikolaos “Torrent worms”, Knowledge-Based Software Engineering, Frontiers in Artificial Intelligence and Applications, Vol. 180, IOS Press.
53. Manindra Agrawal, Neeraj Kayal, Nitin Saxena, "PRIMES is in P", Annals of Mathematics 160 (2004), no. 2, pp. 781–793
54. Simon Singh, "The Code Book: The Science of Secrecy from Ancient Egypt to Quantum Cryptography", TED SMART 1999
55. David Kahn, “The codebreakers The Comprehensive History of Secret Communication from Ancient Times to the Internet", Scribner, Rev Sub edition 1996
56. Zbigniew Brzezinski, "The Unknown Victors". pp.15–18, in Jan Stanislaw Ciechanowski, ed. Marian Rejewski 1905–1980, Living with the Enigma secret. 1st ed. Bydgoszcz: Bydgoszcz City Council, 2005
57. I. J. Good and Cipher A. Deavours, afterword to: Marian Rejewski, "How Polish Mathematicians Deciphered the Enigma", Annals of the History of Computing, 3 (3), July 1981
58. Copeland, B. Jack "The Essential Turing”, Oxford University Press, 2004

59. Kaspersky Lab, www.kaspersky.com

60. Peter Smith, "LUC public key encryption: a secure alternative to RSA", Volume 18 , Issue 1, 1993, pp 44 – 49

61. Alex Biryukov, Christophe De Canniere: Block Ciphers and Systems of Quadratic Equations. Fast Software Encryption - FSE 2003: LNCS 2887, 274–289

62. Nicolas Courtois, Alexander Klimov, Jacques Patarin, Adi Shamir: Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. EUROCRYPT 2000: 392–407

63. Nicolas Courtois, Josef Pieprzyk, "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations". pp267–287, ASIACRYPT 2002.

64. Bo-Yin Yang, Jiun-Ming Chen, "Theoretical Analysis of XL over Small Fields", ACISP 2004, (Lecture Notes in Computer Science vol. 3108, p.277-288).

65. C. Cid, G. Leurent, "An Analysis of the XSL Algorithm", ASIACRYPT 2005, (Lecture Notes in Computer Science vol. 3788, p. 333-35),

66. C. Diem, "The XL-Algorithm and a Conjecture from Commutative Algebra, Advances in cryptology", ASIACRYPT 2004, Springer LNCS 3329, 323-337 (2004)

67. Facebook, www.facebook.com

68. Tetra, http://www.tetrapol.com

69. GSM, www.gsmworld.com

70. J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman and S. S. Wagstaff, Jr., “Factorizations of $b^n \pm 1$, b=2,3,5,6,7,10,12 up to high powers”, Amer. Math. Soc., Providence RI, pp. xcvi+236, ISBN 0-8218-5078-4. 1988
71. P. Ribenboim, The new book of prime number records, 3rd edition, Springer-Verlag, New York, NY, pp. xxiv+541, ISBN 0-387-94457-5. 1995

Κρυπτογραφική έρευνα-οργανισμοί:

- ECRYPT (Network of Excellence in Cryptology) contract number IST-2002-507932
- e-Stream (2004-2008)
- e-Bats (2004-2008) (benchmarking)
- NESSIE (New European Schemes for Signatures, Integrity, and Encryption) contract number IST-1999-12324 (2000-2004)
- NIST -National Institute of Standards and Technology
- FIPS -Federal Information Processing Standards
- CRYPTREC - Cryptography Research and Evaluation Committees (2000-2005)
- IACR -The International Association for Cryptologic Research
- CDT -The Center for Democracy and Technology
- EPIC -Electronic Privacy Information Center

Παράρτημα

Κώδικας για κρυπτανάλυση του DES

Ο κώδικας που ακολουθεί έχει χρησιμοποιηθεί προκειμένου να γίνει η κρυπτανάλυση στον DES. Ο κώδικας για όλα τα προγράμματα δίνεται με την άδεια GPL για οποιαδήποτε τροποποίηση και βελτίωση.

main.cpp

```
/***************************************************************************
                          main.cpp  -  description

        This program is made in order to show the results of a possible
        attack to the very well known DES encryption algorithm.
        After the complation of the following code an executable is
created
        which does the following job :
        1) Reads a file which contains the key
        2) Encrypts the plaintext file according to the key
        3) Tries to identify the key based only on the plaintext and
the ciphertext.
        4) Shows the results.

        The syntax of the command is as follows:
        des_breaker -k <key file> -p <plaintext file>
                                    or
        des_breaker -p <plaintext file> -c <cipher-text file>

                           -------------------
    begin                : Sun Feb 15 16:57:35 EET 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

***************************************************************************
****/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
*
*
***************************************************************************
****/
```

```
#ifdef HAVE_CONFIG_H
    #include <config.h>
#endif

#include <iostream>
#include <stdlib.h>
#include <fstream>
#include <iomanip>

//needed to get the options from the user
#include <getopt.h>

//local used libraries
#include <arbitrary.h>
#include <linear.h>
#include <standardDES.h>
#include <system_creator.h>

#define Zero false  //0 false
#define Ace true    //1 true

void use_ciphertext(char *ciphertext_file,char *plaintext_file,char
*targetfile);
void use_key(char *key_file,char *plaintext_file,char *targetfile);

using namespace std;

void disp_help_msg(void)
{
    cout<<"Usage: des_breaker [-chkp] [file] \n";
    cout<<"   -c [file] : define ciphertext file.\n";
    cout<<"   -h        : display this help screen.\n";
    cout<<"   -k [file] : define key file.\n";
    cout<<"   -p [file] : define plaintext file.\n";
    cout<<"   -t [file] : define target results file.\n";
    cout<<"For further details see attached manual.\n";
    exit(EXIT_FAILURE);
}

int main(int argc, char *argv[])
{
    int opt;
    long start_time, end_time;

    char *keyfile= NULL;
    char *cipherfile= NULL;
    char *plaintextfile= NULL;
    char *targetfile= NULL;

    const char* const short_options = "cdfhknpst";
    const struct option long_options[] =
    {
        { "cipher", 0, NULL, 'c' },         { "help", 0, NULL, 'h' },
        { "plain", 0, NULL, 'p' },          { "target", 0, NULL, 't' },
        { NULL, 0, NULL, 0 },               { "key", 0, NULL, 'k' }
    };
```

```
    //parse the random seed for the rng
    start_time=(unsigned)time(0);

    //get the arguments from the command line
    do
    {
        opt=getopt_long (argc,argv,short_options,long_options,NULL);
        {
            switch(opt)
            {
                case 'c':
                    cipherfile=argv[optind];
                break;

                case 'h':
                    disp_help_msg();
                break;

                case 'k':
                    keyfile=argv[optind];
                break;

                case 'p':
                    plaintextfile=argv[optind];
                break;

                case 't':
                    targetfile=argv[optind];
                break;
            }
        }
    }
    while (opt != -1);

    loadSboxes();

    cout<<"Please wait.Processing the data...\n";

    if(cipherfile!=NULL)
      use_ciphertext(cipherfile,plaintextfile,targetfile);
    else
      use_key(keyfile,plaintextfile,targetfile);

    end_time=(unsigned)time(0);
    cout<<"Total calculation time "<<end_time-start_time<<" seconds\n";

}

//in order to simplify the code it has been devided
//in two parts depending on whether we have
//plaintext-key or plaintext-ciphertext
//case the pair plaintext-ciphertext is parsed

void use_ciphertext(char *ciphertext_file,char *plaintext_file,char
*targetfile)
{
```

```
    int r;
    bool digits2Bused[64];
    bool probKey[64];
    bool canBused[64];
    int extracted=0;
    int AsciCode;
    bool tmpBinary[8];
    bool text2encr[64], origK[64], Ciphertext[64];
    long j;
    char fetch_char;
    bool foundBit[56],usefoundBit[56];
    ifstream plainFile, keyFile, cipherFile,foundFile;
    ofstream targetFile;

    digit CipherTextArb[64];

    long hitOnes[64], hitZeros[64];

    for(r=0;r<64;r++)
      hitOnes[r]=hitZeros[r]=0;
    for(r=0;r<64;r++)
    {
      foundBit[r]=false;//set it to 0
      usefoundBit[r]=false;
    }

    cipherFile.open(ciphertext_file, ios_base::in );//else open the
file that contains the ciphertext
    if (!cipherFile)
    {
        cout << "Sorry desbreaker was unable to open the ciphertext
file " << ciphertext_file << endl;
        exit(EXIT_FAILURE);
    }

    plainFile.open(plaintext_file, ios_base::in );
    if (!plainFile)
    {
        cout << "Sorry desbreaker was unable to open the plaintext file
" << plaintext_file << endl;
        exit(EXIT_FAILURE);
    }

    while (! plainFile.eof() )
    {
        for(r=0;r<8 ;r++)
        {
            plainFile.get(fetch_char);
            AsciCode =fetch_char;
            conv2binary(AsciCode,tmpBinary);
            for(j=0;j<8;j++)
                text2encr[8*r+j]=tmpBinary[j];
        }

        if(!plainFile.eof())
        {
```

```
            //this is the case where I have to read the cipher-text file
            for(r=0;r<64 ;r++)
            {
                cipherFile.get(fetch_char);
                Ciphertext[r]=fetch_char-48;
            }
            //systemcreator(text2encr,CipherTextArb);
            //now that I have stored the pair PLAINTEXT-CIPHERTEXT as
well as the arbitrary ciphertext
            //I can use them to solve the system
            //show the created system

            l_solve(CipherTextArb,Ciphertext,probKey,digits2Bused);
            for(r=0;r<64;r++)
            {
                if(probKey[r]==1 && (r%8!=7) && digits2Bused[r]==true)
hitOnes[r]++;
                if(probKey[r]==0 && (r%8!=7) && digits2Bused[r]==true)
hitZeros[r]++;
            }

         }

    }//end loop
    plainFile.close();
    cipherFile.close();

    //open target file for append
    if (targetFile!=NULL) targetFile.open(targetfile, ios_base::app);

    cout<<"The calculated key is:\n";
    for(r=0;r<64;r++)
    {
        if(r%8!=7)
        {
            //firstly lets show the calculated key
            if(hitOnes[r]>hitZeros[r])
            {
                cout<<"1";
                probKey[r]=Ace;
                canBused[r]=true;
                extracted++;
            }
            else if(hitOnes[r]<hitZeros[r])
            {
                cout<<"0";
                probKey[r]=Zero;
                extracted++;
                canBused[r]=true;
            }
            else
            {
                cout<<"?";
                canBused[r]=false;
            }
        }
    }
```

```
    cout<<"\n";

    //send the correct hits to a file
    if (targetFile!=NULL)
    {
      for(r=0;r<64;r++)
      {
        if((r%8)!=7 && canBused[r]==true)
targetFile<<(origK[r]==probKey[r]);
        else if((r%8)!=7 && canBused[r]==false) targetFile<<"?";
      }
      targetFile<<"\n";
    }//remember!!! ?unknown 1 correct 0 wrong

    //close all open files
    plainFile.close();
    cipherFile.close();
    if (targetFile!=NULL) targetFile.close();

}

//case the pair plaintext-key is parsed

//void use_key(const char *key_file,const char *plaintext_file,const
char *targetfile,const char *FoundFile)
void use_key(char *key_file,char *plaintext_file,char *targetfile)
{
    int r,k,found,extracted,AsciCode;
    unsigned long j;
    bool digits2Bused[64];
    bool probKey[64];
    bool canBused[64];
    bool tmpBinary[8];
    bool text2encr[64], origK[64], Ciphertext[64];
    bool known_bits[64],bits2use[64];
    long hitOnes[64], hitZeros[64];
    char fetch_char;
    ifstream plainFile, keyFile, cipherFile,foundFile;
    ifstream plainFile2;
    ofstream targetFile;
    digit CipherTextArb[64];
    digit PlainTextArb[64];

    extracted=0;
    for(r=0;r<64;r++)
    {
      hitOnes[r]=hitZeros[r]=0;
      known_bits[r]=bits2use[r]=0;
      digits2Bused[r]=false;
    }

    keyFile.open(key_file, ios_base::in );
    if (!keyFile)
    {
        cout << "Sorry desbreaker was unable to open the key file " <<
key_file << endl;
        exit(EXIT_FAILURE);
```

```
    }
    r=0;
    while (!keyFile.eof() && r<8 )
    {
        keyFile.get(fetch_char);
        AsciCode=fetch_char;
        if(AsciCode<0) AsciCode=256+AsciCode;
        conv2binary(AsciCode,tmpBinary);
        for(j=0;j<8;j++)
            origK[8*r+j]=tmpBinary[j];
        r++;
    }
    keyFile.close();

//    bits2use[62]=true;                  known_bits[62]=origK[62];
//    bits2use[61]=true;                  known_bits[61]=origK[61];
//    bits2use[60]=true;                  known_bits[60]=origK[60];
//    bits2use[59]=true;                  known_bits[59]=origK[59];
//    bits2use[57]=true;                  known_bits[57]=origK[57];
//    bits2use[54]=true;                  known_bits[54]=origK[54];
//    bits2use[53]=true;                  known_bits[53]=origK[53];
//    bits2use[51]=true;                  known_bits[51]=origK[51];
//    bits2use[50]=true;                  known_bits[50]=origK[50];
//    bits2use[49]=true;                  known_bits[49]=origK[49];
//    bits2use[48]=true;                  known_bits[48]=origK[48];
//    bits2use[46]=true;                  known_bits[46]=origK[46];
//    bits2use[44]=true;                  known_bits[44]=origK[44];
//    bits2use[43]=true;                  known_bits[43]=origK[43];
//    bits2use[42]=true;                  known_bits[42]=origK[42];
//    bits2use[41]=true;                  known_bits[41]=origK[41];
//    bits2use[38]=true;                  known_bits[38]=origK[38];
//    bits2use[37]=true;                  known_bits[37]=origK[37];
//    bits2use[35]=true;                  known_bits[35]=origK[35];
//    bits2use[34]=true;                  known_bits[34]=origK[34];
//    bits2use[33]=true;                  known_bits[33]=origK[33];
//    bits2use[30]=true;                  known_bits[30]=origK[30];
//    bits2use[29]=true;                  known_bits[29]=origK[29];
//    bits2use[26]=true;                  known_bits[26]=origK[26];
//    bits2use[22]=true;                  known_bits[22]=origK[22];
//    bits2use[21]=true;                  known_bits[21]=origK[21];
//    bits2use[19]=true;                  known_bits[19]=origK[19];
//    bits2use[18]=true;                  known_bits[18]=origK[18];
//    bits2use[17]=true;                  known_bits[17]=origK[17];
//    bits2use[14]=true;                  known_bits[14]=origK[14];
//    bits2use[13]=true;                  known_bits[13]=origK[13];
//    bits2use[12]=true;                  known_bits[12]=origK[12];
//    bits2use[11]=true;                  known_bits[11]=origK[11];
//    bits2use[9]=true;                   known_bits[9]=origK[9];
//    bits2use[6]=true;                   known_bits[6]=origK[6];
//    bits2use[5]=true;                   known_bits[5]=origK[5];
//    bits2use[3]=true;                   known_bits[3]=origK[3];
//    bits2use[2]=true;                   known_bits[2]=origK[2];
//    bits2use[1]=true;                   known_bits[1]=origK[1];

//**********************//
   bits2use[58]=true;                 known_bits[58]=origK[58];
   bits2use[56]=true;                 known_bits[56]=origK[56];
```

```
   bits2use[52]=true;                  known_bits[52]=origK[52];
   bits2use[48]=true;                  known_bits[48]=origK[48];
   bits2use[40]=true;                  known_bits[40]=origK[40];
   bits2use[36]=true;                  known_bits[36]=origK[36];
   bits2use[32]=true;                  known_bits[32]=origK[32];
   bits2use[28]=true;                  known_bits[28]=origK[28];
   bits2use[24]=true;                  known_bits[24]=origK[24];
   bits2use[20]=true;                  known_bits[20]=origK[20];
   bits2use[16]=true;                  known_bits[16]=origK[16];
   bits2use[8]=true;                   known_bits[8]=origK[8];
   bits2use[4]=true;                   known_bits[4]=origK[4];
   bits2use[0]=true;                   known_bits[0]=origK[0];


    //clear the counters again
    for(r=0;r<64;r++)
        hitOnes[r]=hitZeros[r]=0;

    plainFile2.open(plaintext_file, ios_base::in |ios_base::binary);
    while (! plainFile2.eof() )
    {
        for(r=0;r<8 ;r++)
        {
            plainFile2.get(fetch_char);
            AsciCode=fetch_char;
            if(AsciCode<0) AsciCode=256+AsciCode;
            conv2binary(AsciCode,tmpBinary);
            for(j=0;j<8;j++)
                text2encr[8*r+j]=tmpBinary[j];
        }

        if(!plainFile2.eof())
        {
           for(r=0;r<64;r++)
              known_bits[r]=known_bits[r]^Ace;
           EncrDes(text2encr,origK,Ciphertext);
           systemcreator(text2encr,CipherTextArb,known_bits,bits2use);
           invert_cipher_text(Ciphertext,CipherTextArb);
           l_solve(CipherTextArb,Ciphertext,probKey,digits2Bused);
           //inorder to have the ciphertext in its correct form
           invert_cipher_text(Ciphertext,CipherTextArb);
           for(r=0;r<64;r++)
           {
              if(probKey[r]==1 && (r%8!=7) && digits2Bused[r]==true)
hitOnes[r]++;
              if(probKey[r]==0 && (r%8!=7) && digits2Bused[r]==true)
hitZeros[r]++;
           }

           for(r=0;r<64;r++)
           {
               for(k=0;k<65;k++)
               {
                   if (k==0) PlainTextArb[r].variable[k]=text2encr[r];
                   else PlainTextArb[r].variable[k]=Zero;
               }
           }
```

```
systemcreator_decr(Ciphertext,PlainTextArb,known_bits,bits2use);
           invert_cipher_text(text2encr,PlainTextArb);
           l_solve(PlainTextArb,text2encr,probKey,digits2Bused);
           for(r=0;r<64;r++)
           {
               if(probKey[r]==1 && (r%8!=7) && digits2Bused[r]==true)
hitOnes[r]++;
               if(probKey[r]==0 && (r%8!=7) && digits2Bused[r]==true)
hitZeros[r]++;
           }
         }

    }//end loop
    plainFile2.close();


    //open target file for append
    if (targetFile!=NULL) targetFile.open(targetfile, ios_base::app);

    cout<<"The calculated key is:\n";
    for(r=0;r<64;r++)
    {
        if(r%8!=7)
        {
            //firstly lets show the calculated key
            if(hitOnes[r]>hitZeros[r])
            {
                cout<<"1";
                probKey[r]=Ace;
                canBused[r]=true;
                extracted++;
            }
            else if(hitOnes[r]<hitZeros[r])
            {
                cout<<"0";
                probKey[r]=Zero;
                extracted++;
                canBused[r]=true;
            }
            else
            {
                cout<<"?";
                canBused[r]=false;
            }
        }
    }
    cout<<"\n";

    cout<<"The original key was:\n";
    for(r=0;r<64;r++)
    {
        if((r%8)!=7)
        cout<<origK[r];
    }
    cout<<"\n";
    found=0;
```

```
    for(r=0;r<64;r++)
      if((r%8)!=7 && origK[r]==probKey[r] && canBused[r]==true)
found++;
    cout<<found<<" bits where correctly found ";
    cout<<"out of the "<<extracted<< " bits that where extracted\n";

    //send the correct hits to a file
    if (targetFile!=NULL)
    {
      for(r=0;r<64;r++)
      {
        if((r%8)!=7 && canBused[r]==true)
targetFile<<(origK[r]==probKey[r]);
        else if((r%8)!=7 && canBused[r]==false) targetFile<<"?";
      }
      targetFile<<"\n";
    }

    //close all open files
    plainFile.close();
    keyFile.close();
    if (targetFile!=NULL) targetFile.close();

}
```

arbitrary.h

```
/**************************************************************************
*****
                          arbitrary.h  -  description
Here, there are all the functions that are needed in order to make DES
work
in an arbitrary way, meaning that it works dependent on variables
                                    k0,k1,k2...,k62,k63
Most of the functions are the same as those in DES, only slightly
converted, in
order to work instead of integers, with digits, the structure that is
declared here.
                             -------------------
    begin                : Wed Feb 18 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

**************************************************************************
****/

/**************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

**************************************************************************
****/

struct digit
{
      bool variable[1+64];
      //1 for 1 or 0 and 64 for k1,k2,....k64
};

struct digit3
{
     bool variable[1+64*2];
     //1 for 1 or 0 and 64 for k1,k2,....k64
};

extern void GetDigitValue(digit *sourceArray,int source_position, digit
*target,int target_position);
extern void AddDigits(digit *source1, int first,digit *source2, int
second , digit *target,int target_position );
```

```
extern void IP_arb(digit *pArrSource);
extern void IPinv_arb(digit *pArrSource);
extern void LS1_arb(digit *pArrSource);
extern void L_arb(digit *pArrSource,digit *pArrTarg);
extern void R_arb(digit *pArrSource,digit *pArrTarg);
extern void ExpDES_arb(digit *pArrSource, digit *pArrTarg);
extern void PC1_arb(digit *pArrSource, digit *pArrTarg);
extern void PC2_arb(digit *pArrSource,digit *pArrTarg);
extern void Perm_arb(digit *pArrSource,digit *pArrTarg);
extern void L28_arb(digit *pArrSource,digit *pArrTarg);
extern void R28_arb(digit *pArrSource,digit *pArrTarg);
extern void AddTables_arb(digit *table1,digit *table2,digit
*target_table,int count);
extern void JoinTables_arb(digit *table1,digit *table2,digit
*target_table,int count);
extern void invert_cipher_text(bool *sourceArray,digit *sourceArray2);
extern void addAces(digit *source,int position);
extern void find_digits(digit *pArr,digit *pTarg);
extern void XOR(digit *sourceArray2);
extern void XOR(bool *sourceArray);
```

arbitrary.cpp

```
/***************************************************************************
*****
                          arbitrary.cpp  -  description
Here, there are all the functions that are needed in order to make DES
work
in an arbitrary way, meaning that it works dependent on variables
                                    k0,k1,k2...,k62,k63
Most of the functions are the same as those in DES, only slightly
converted, in
order to work instead of integers, with digits, the structure that is
declared here.
                             -------------------
    begin                : Tue Feb 24 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

***************************************************************************
****/

/***************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************
****/

#define Zero false
#define Ace true

#include <standardDES.h>

unsigned int prob[32];

struct digit
{
     bool variable[1+64];
     //1 for 1 or 0 and 64 for k1,k2,....k64
};

void invert_cipher_text(bool *sourceArray,digit *sourceArray2)
{
    int i;
```

```
    bool tmp[64];
    digit tmpdig[64];

    for(i=0;i<64;i++)
    {
      tmp[i]=sourceArray[63-i];
      tmpdig[i]=sourceArray2[63-i];
    }
    for(i=0;i<64;i++)
    {
      sourceArray[i]=tmp[i];
      sourceArray2[i]=tmpdig[i];
    }
}

void XOR(bool *sourceArray)
{
    int i;
    bool tmp[64];

    for(i=0;i<64;i++)
      tmp[i]=sourceArray[63-i];
    for(i=0;i<64;i++)
      sourceArray[i]=tmp[i];
}

void XOR(digit *sourceArray2)
{
    int i;
    digit tmpdig[64];

    for(i=0;i<64;i++)
      tmpdig[i]=sourceArray2[63-i];
    for(i=0;i<64;i++)
      sourceArray2[i]=tmpdig[i];
}



void GetDigitValue(digit *sourceArray,int source_position, digit
*target,int target_position)
{
    target[target_position]=sourceArray[source_position];
}

void AddDigits(digit *x, int fp,digit *y, int sp, digit *t,int tp)
{
    int i;
    for(i=0;i<65;i++)
        t[tp].variable[i]=x[fp].variable[i] ^ y[sp].variable[i] ;
}

void IP_arb(digit *pArrSource)
{
    digit pArrTarg[64];
    int i;
```

```
    GetDigitValue(pArrSource,57,pArrTarg,0);
GetDigitValue(pArrSource,49,pArrTarg,1);
GetDigitValue(pArrSource,41,pArrTarg,2);
    GetDigitValue(pArrSource,33,pArrTarg,3);
GetDigitValue(pArrSource,25,pArrTarg,4);
GetDigitValue(pArrSource,17,pArrTarg,5);
    GetDigitValue(pArrSource,9,pArrTarg,6);
GetDigitValue(pArrSource,1,pArrTarg,7);
GetDigitValue(pArrSource,59,pArrTarg,8);
    GetDigitValue(pArrSource,51,pArrTarg,9);
GetDigitValue(pArrSource,43,pArrTarg,10);
GetDigitValue(pArrSource,35,pArrTarg,11);
    GetDigitValue(pArrSource,27,pArrTarg,12);
GetDigitValue(pArrSource,19,pArrTarg,13);
GetDigitValue(pArrSource,11,pArrTarg,14);
    GetDigitValue(pArrSource,3,pArrTarg,15);
GetDigitValue(pArrSource,61,pArrTarg,16);
GetDigitValue(pArrSource,53,pArrTarg,17);
    GetDigitValue(pArrSource,45,pArrTarg,18);
GetDigitValue(pArrSource,37,pArrTarg,19);
GetDigitValue(pArrSource,29,pArrTarg,20);
    GetDigitValue(pArrSource,21,pArrTarg,21);
GetDigitValue(pArrSource,13,pArrTarg,22);
GetDigitValue(pArrSource,5,pArrTarg,23);
    GetDigitValue(pArrSource,63,pArrTarg,24);
GetDigitValue(pArrSource,55,pArrTarg,25);
GetDigitValue(pArrSource,47,pArrTarg,26);
    GetDigitValue(pArrSource,39,pArrTarg,27);
GetDigitValue(pArrSource,31,pArrTarg,28);
GetDigitValue(pArrSource,23,pArrTarg,29);
    GetDigitValue(pArrSource,15,pArrTarg,30);
GetDigitValue(pArrSource,7,pArrTarg,31);
GetDigitValue(pArrSource,56,pArrTarg,32);
    GetDigitValue(pArrSource,48,pArrTarg,33);
GetDigitValue(pArrSource,40,pArrTarg,34);
GetDigitValue(pArrSource,32,pArrTarg,35);
    GetDigitValue(pArrSource,24,pArrTarg,36);
GetDigitValue(pArrSource,16,pArrTarg,37);
GetDigitValue(pArrSource,8,pArrTarg,38);
    GetDigitValue(pArrSource,0,pArrTarg,39);
GetDigitValue(pArrSource,58,pArrTarg,40);
GetDigitValue(pArrSource,41,pArrTarg,50);
    GetDigitValue(pArrSource,42,pArrTarg,42);
GetDigitValue(pArrSource,34,pArrTarg,43);
GetDigitValue(pArrSource,26,pArrTarg,44);
    GetDigitValue(pArrSource,18,pArrTarg,45);
GetDigitValue(pArrSource,10,pArrTarg,46);
GetDigitValue(pArrSource,2,pArrTarg,47);
    GetDigitValue(pArrSource,60,pArrTarg,48);
GetDigitValue(pArrSource,52,pArrTarg,49);
GetDigitValue(pArrSource,44,pArrTarg,50);
    GetDigitValue(pArrSource,36,pArrTarg,51);
GetDigitValue(pArrSource,28,pArrTarg,52);
GetDigitValue(pArrSource,20,pArrTarg,53);
    GetDigitValue(pArrSource,12,pArrTarg,54);
GetDigitValue(pArrSource,4,pArrTarg,55);
GetDigitValue(pArrSource,62,pArrTarg,56);
```

```
    GetDigitValue(pArrSource,54,pArrTarg,57);
GetDigitValue(pArrSource,46,pArrTarg,58);
GetDigitValue(pArrSource,38,pArrTarg,59);
    GetDigitValue(pArrSource,30,pArrTarg,60);
GetDigitValue(pArrSource,22,pArrTarg,61);
GetDigitValue(pArrSource,14,pArrTarg,62);
    GetDigitValue(pArrSource,6,pArrTarg,63);

    for(i=0;i<64;i++)
        GetDigitValue(pArrTarg,i,pArrSource,i);
}

void IPinv_arb(digit *pArrSource)
{
    digit pArrTarg[64];
    int i;

    GetDigitValue(pArrSource,39,pArrTarg,0);
GetDigitValue(pArrSource,7,pArrTarg,1);
GetDigitValue(pArrSource,47,pArrTarg,2);
    GetDigitValue(pArrSource,15,pArrTarg,3);
GetDigitValue(pArrSource,55,pArrTarg,4);
GetDigitValue(pArrSource,23,pArrTarg,5);
    GetDigitValue(pArrSource,63,pArrTarg,6);
GetDigitValue(pArrSource,31,pArrTarg,7);
GetDigitValue(pArrSource,38,pArrTarg,8);
    GetDigitValue(pArrSource,6,pArrTarg,9);
GetDigitValue(pArrSource,46,pArrTarg,10);
GetDigitValue(pArrSource,14,pArrTarg,11);
    GetDigitValue(pArrSource,54,pArrTarg,12);
GetDigitValue(pArrSource,22,pArrTarg,13);
GetDigitValue(pArrSource,62,pArrTarg,14);
    GetDigitValue(pArrSource,30,pArrTarg,15);
GetDigitValue(pArrSource,37,pArrTarg,16);
GetDigitValue(pArrSource,5,pArrTarg,17);
    GetDigitValue(pArrSource,45,pArrTarg,18);
GetDigitValue(pArrSource,13,pArrTarg,19);
GetDigitValue(pArrSource,53,pArrTarg,20);
    GetDigitValue(pArrSource,21,pArrTarg,21);
GetDigitValue(pArrSource,61,pArrTarg,22);
GetDigitValue(pArrSource,29,pArrTarg,23);
    GetDigitValue(pArrSource,36,pArrTarg,24);
GetDigitValue(pArrSource,4,pArrTarg,25);
GetDigitValue(pArrSource,44,pArrTarg,26);
    GetDigitValue(pArrSource,12,pArrTarg,27);
GetDigitValue(pArrSource,52,pArrTarg,28);
GetDigitValue(pArrSource,20,pArrTarg,29);
    GetDigitValue(pArrSource,60,pArrTarg,30);
GetDigitValue(pArrSource,28,pArrTarg,31);
GetDigitValue(pArrSource,35,pArrTarg,32);
    GetDigitValue(pArrSource,3,pArrTarg,33);
GetDigitValue(pArrSource,43,pArrTarg,34);
GetDigitValue(pArrSource,11,pArrTarg,35);
    GetDigitValue(pArrSource,51,pArrTarg,36);
GetDigitValue(pArrSource,19,pArrTarg,37);
GetDigitValue(pArrSource,59,pArrTarg,38);
```

```
    GetDigitValue(pArrSource,27,pArrTarg,39);
GetDigitValue(pArrSource,34,pArrTarg,40);
GetDigitValue(pArrSource,2,pArrTarg,41);
    GetDigitValue(pArrSource,42,pArrTarg,42);
GetDigitValue(pArrSource,10,pArrTarg,43);
GetDigitValue(pArrSource,50,pArrTarg,44);
    GetDigitValue(pArrSource,18,pArrTarg,45);
GetDigitValue(pArrSource,58,pArrTarg,46);
GetDigitValue(pArrSource,26,pArrTarg,47);
    GetDigitValue(pArrSource,33,pArrTarg,48);
GetDigitValue(pArrSource,1,pArrTarg,49);
GetDigitValue(pArrSource,41,pArrTarg,50);
    GetDigitValue(pArrSource,9,pArrTarg,51);
GetDigitValue(pArrSource,49,pArrTarg,52);
GetDigitValue(pArrSource,17,pArrTarg,53);
    GetDigitValue(pArrSource,57,pArrTarg,54);
GetDigitValue(pArrSource,25,pArrTarg,55);
GetDigitValue(pArrSource,32,pArrTarg,56);
    GetDigitValue(pArrSource,0,pArrTarg,57);
GetDigitValue(pArrSource,40,pArrTarg,58);
GetDigitValue(pArrSource,8,pArrTarg,59);
    GetDigitValue(pArrSource,48,pArrTarg,60);
GetDigitValue(pArrSource,16,pArrTarg,61);
GetDigitValue(pArrSource,56,pArrTarg,62);
    GetDigitValue(pArrSource,24,pArrTarg,63);

    for(i=0;i<64;i++)
        GetDigitValue(pArrTarg,i,pArrSource,i);

}

void LS1_arb(digit *pArrSource)
{
    digit pArrTarg[28];
    int i;
    for(i=0;i<27;i++)
        GetDigitValue(pArrSource,i+1,pArrTarg,i);
    GetDigitValue(pArrSource,0,pArrTarg,27);

    for(i=0;i<28;i++)
        GetDigitValue(pArrTarg,i,pArrSource,i);
}


void L_arb(digit *pArrSource,digit *pArrTarg)
{
    int i;

    for(i=0;i<32;i++)
        GetDigitValue(pArrSource,i,pArrTarg,i);
}

void R_arb(digit *pArrSource,digit *pArrTarg)
{
    int i;

    for(i=0;i<32;i++)
```

```
        GetDigitValue(pArrSource,i+32,pArrTarg,i);
}

void ExpDES_arb(digit *pArrSource, digit *pArrTarg)
{
    GetDigitValue(pArrSource,31,pArrTarg,0);
GetDigitValue(pArrSource,0,pArrTarg,1);
GetDigitValue(pArrSource,1,pArrTarg,2);
    GetDigitValue(pArrSource,2,pArrTarg,3);
GetDigitValue(pArrSource,3,pArrTarg,4);
GetDigitValue(pArrSource,4,pArrTarg,5);
    GetDigitValue(pArrSource,5,pArrTarg,6);
GetDigitValue(pArrSource,6,pArrTarg,7);
GetDigitValue(pArrSource,7,pArrTarg,8);
    GetDigitValue(pArrSource,6,pArrTarg,9);
GetDigitValue(pArrSource,7,pArrTarg,10);
GetDigitValue(pArrSource,8,pArrTarg,11);
    GetDigitValue(pArrSource,7,pArrTarg,12);
GetDigitValue(pArrSource,8,pArrTarg,13);
GetDigitValue(pArrSource,9,pArrTarg,14);
    GetDigitValue(pArrSource,10,pArrTarg,15);
GetDigitValue(pArrSource,11,pArrTarg,16);
GetDigitValue(pArrSource,12,pArrTarg,17);
    GetDigitValue(pArrSource,13,pArrTarg,18);
GetDigitValue(pArrSource,12,pArrTarg,19);
GetDigitValue(pArrSource,13,pArrTarg,20);
    GetDigitValue(pArrSource,14,pArrTarg,21);
GetDigitValue(pArrSource,15,pArrTarg,22);
GetDigitValue(pArrSource,16,pArrTarg,23);
    GetDigitValue(pArrSource,15,pArrTarg,24);
GetDigitValue(pArrSource,16,pArrTarg,25);
GetDigitValue(pArrSource,17,pArrTarg,26);
    GetDigitValue(pArrSource,18,pArrTarg,27);
GetDigitValue(pArrSource,19,pArrTarg,28);
GetDigitValue(pArrSource,20,pArrTarg,29);
    GetDigitValue(pArrSource,19,pArrTarg,30);
GetDigitValue(pArrSource,20,pArrTarg,31);
GetDigitValue(pArrSource,21,pArrTarg,32);
    GetDigitValue(pArrSource,22,pArrTarg,33);
GetDigitValue(pArrSource,23,pArrTarg,34);
GetDigitValue(pArrSource,24,pArrTarg,35);
    GetDigitValue(pArrSource,20,pArrTarg,36);
GetDigitValue(pArrSource,24,pArrTarg,37);
GetDigitValue(pArrSource,25,pArrTarg,38);
    GetDigitValue(pArrSource,26,pArrTarg,39);
GetDigitValue(pArrSource,27,pArrTarg,40);
GetDigitValue(pArrSource,28,pArrTarg,41);
    GetDigitValue(pArrSource,27,pArrTarg,42);
GetDigitValue(pArrSource,28,pArrTarg,43);
GetDigitValue(pArrSource,29,pArrTarg,44);
    GetDigitValue(pArrSource,30,pArrTarg,45);
GetDigitValue(pArrSource,31,pArrTarg,46);
GetDigitValue(pArrSource,0,pArrTarg,47);
}

void PC1_arb(digit *pArrSource, digit *pArrTarg)
{
```

```
    GetDigitValue(pArrSource,56,pArrTarg,0);
GetDigitValue(pArrSource,48,pArrTarg,1);
GetDigitValue(pArrSource,40,pArrTarg,2);
    GetDigitValue(pArrSource,32,pArrTarg,3);
GetDigitValue(pArrSource,24,pArrTarg,4);
GetDigitValue(pArrSource,16,pArrTarg,5);
    GetDigitValue(pArrSource,8,pArrTarg,6);
GetDigitValue(pArrSource,0,pArrTarg,7);
GetDigitValue(pArrSource,57,pArrTarg,8);
    GetDigitValue(pArrSource,49,pArrTarg,9);
GetDigitValue(pArrSource,41,pArrTarg,10);
GetDigitValue(pArrSource,33,pArrTarg,11);
    GetDigitValue(pArrSource,25,pArrTarg,12);
GetDigitValue(pArrSource,17,pArrTarg,13);
GetDigitValue(pArrSource,11,pArrTarg,14);
    GetDigitValue(pArrSource,1,pArrTarg,15);
GetDigitValue(pArrSource,58,pArrTarg,16);
GetDigitValue(pArrSource,50,pArrTarg,17);
    GetDigitValue(pArrSource,42,pArrTarg,18);
GetDigitValue(pArrSource,34,pArrTarg,19);
GetDigitValue(pArrSource,26,pArrTarg,20);
    GetDigitValue(pArrSource,18,pArrTarg,21);
GetDigitValue(pArrSource,10,pArrTarg,22);
GetDigitValue(pArrSource,2,pArrTarg,23);
    GetDigitValue(pArrSource,59,pArrTarg,24);
GetDigitValue(pArrSource,51,pArrTarg,25);
GetDigitValue(pArrSource,43,pArrTarg,26);
    GetDigitValue(pArrSource,35,pArrTarg,27);
GetDigitValue(pArrSource,62,pArrTarg,28);
GetDigitValue(pArrSource,54,pArrTarg,29);
    GetDigitValue(pArrSource,46,pArrTarg,30);
GetDigitValue(pArrSource,38,pArrTarg,31);
GetDigitValue(pArrSource,30,pArrTarg,32);
    GetDigitValue(pArrSource,22,pArrTarg,33);
GetDigitValue(pArrSource,14,pArrTarg,34);
GetDigitValue(pArrSource,6,pArrTarg,35);
    GetDigitValue(pArrSource,61,pArrTarg,36);
GetDigitValue(pArrSource,53,pArrTarg,37);
GetDigitValue(pArrSource,45,pArrTarg,38);
    GetDigitValue(pArrSource,37,pArrTarg,39);
GetDigitValue(pArrSource,29,pArrTarg,40);
GetDigitValue(pArrSource,21,pArrTarg,41);
    GetDigitValue(pArrSource,13,pArrTarg,42);
GetDigitValue(pArrSource,5,pArrTarg,43);
GetDigitValue(pArrSource,60,pArrTarg,44);
    GetDigitValue(pArrSource,52,pArrTarg,45);
GetDigitValue(pArrSource,44,pArrTarg,46);
GetDigitValue(pArrSource,36,pArrTarg,47);
    GetDigitValue(pArrSource,28,pArrTarg,48);
GetDigitValue(pArrSource,20,pArrTarg,49);
GetDigitValue(pArrSource,12,pArrTarg,50);
    GetDigitValue(pArrSource,4,pArrTarg,51);
GetDigitValue(pArrSource,27,pArrTarg,52);
GetDigitValue(pArrSource,19,pArrTarg,53);
    GetDigitValue(pArrSource,11,pArrTarg,54);
GetDigitValue(pArrSource,3,pArrTarg,55);
```

```
}

void PC2_arb(digit *pArrSource,digit *pArrTarg)
{

    GetDigitValue(pArrSource,13,pArrTarg,0);
GetDigitValue(pArrSource,16,pArrTarg,1);
GetDigitValue(pArrSource,10,pArrTarg,2);
    GetDigitValue(pArrSource,23,pArrTarg,3);
GetDigitValue(pArrSource,0,pArrTarg,4);
GetDigitValue(pArrSource,4,pArrTarg,5);
    GetDigitValue(pArrSource,2,pArrTarg,6);
GetDigitValue(pArrSource,27,pArrTarg,7);
GetDigitValue(pArrSource,14,pArrTarg,8);
    GetDigitValue(pArrSource,5,pArrTarg,9);
GetDigitValue(pArrSource,20,pArrTarg,10);
GetDigitValue(pArrSource,9,pArrTarg,11);
    GetDigitValue(pArrSource,22,pArrTarg,12);
GetDigitValue(pArrSource,18,pArrTarg,13);
GetDigitValue(pArrSource,11,pArrTarg,14);
    GetDigitValue(pArrSource,3,pArrTarg,15);
GetDigitValue(pArrSource,25,pArrTarg,16);
GetDigitValue(pArrSource,7,pArrTarg,17);
    GetDigitValue(pArrSource,15,pArrTarg,18);
GetDigitValue(pArrSource,6,pArrTarg,19);
GetDigitValue(pArrSource,26,pArrTarg,20);
    GetDigitValue(pArrSource,19,pArrTarg,21);
GetDigitValue(pArrSource,12,pArrTarg,22);
GetDigitValue(pArrSource,1,pArrTarg,23);
    GetDigitValue(pArrSource,40,pArrTarg,24);
GetDigitValue(pArrSource,51,pArrTarg,25);
GetDigitValue(pArrSource,30,pArrTarg,26);
    GetDigitValue(pArrSource,36,pArrTarg,27);
GetDigitValue(pArrSource,46,pArrTarg,28);
GetDigitValue(pArrSource,54,pArrTarg,29);
    GetDigitValue(pArrSource,29,pArrTarg,30);
GetDigitValue(pArrSource,39,pArrTarg,31);
GetDigitValue(pArrSource,50,pArrTarg,32);
    GetDigitValue(pArrSource,44,pArrTarg,33);
GetDigitValue(pArrSource,32,pArrTarg,34);
GetDigitValue(pArrSource,47,pArrTarg,35);
    GetDigitValue(pArrSource,43,pArrTarg,36);
GetDigitValue(pArrSource,48,pArrTarg,37);
GetDigitValue(pArrSource,38,pArrTarg,38);
    GetDigitValue(pArrSource,55,pArrTarg,39);
GetDigitValue(pArrSource,33,pArrTarg,40);
GetDigitValue(pArrSource,52,pArrTarg,41);
    GetDigitValue(pArrSource,45,pArrTarg,42);
GetDigitValue(pArrSource,41,pArrTarg,43);
GetDigitValue(pArrSource,49,pArrTarg,44);
    GetDigitValue(pArrSource,35,pArrTarg,45);
GetDigitValue(pArrSource,28,pArrTarg,46);
GetDigitValue(pArrSource,31,pArrTarg,47);

}

void Perm_arb(digit *pArrSource,digit *pArrTarg)
```

```
{

    int i;
    GetDigitValue(pArrSource,15,pArrTarg,0);
GetDigitValue(pArrSource,6,pArrTarg,1);
GetDigitValue(pArrSource,19,pArrTarg,2);
    GetDigitValue(pArrSource,20,pArrTarg,3);
GetDigitValue(pArrSource,28,pArrTarg,4);
GetDigitValue(pArrSource,11,pArrTarg,5);
    GetDigitValue(pArrSource,27,pArrTarg,6);
GetDigitValue(pArrSource,16,pArrTarg,7);
GetDigitValue(pArrSource,0,pArrTarg,8);
    GetDigitValue(pArrSource,14,pArrTarg,9);
GetDigitValue(pArrSource,22,pArrTarg,10);
GetDigitValue(pArrSource,25,pArrTarg,11);
    GetDigitValue(pArrSource,4,pArrTarg,12);
GetDigitValue(pArrSource,17,pArrTarg,13);
GetDigitValue(pArrSource,30,pArrTarg,14);
    GetDigitValue(pArrSource,9,pArrTarg,15);
GetDigitValue(pArrSource,1,pArrTarg,16);
GetDigitValue(pArrSource,7,pArrTarg,17);
    GetDigitValue(pArrSource,23,pArrTarg,18);
GetDigitValue(pArrSource,13,pArrTarg,19);
GetDigitValue(pArrSource,31,pArrTarg,20);
    GetDigitValue(pArrSource,26,pArrTarg,21);
GetDigitValue(pArrSource,2,pArrTarg,22);
GetDigitValue(pArrSource,8,pArrTarg,23);
    GetDigitValue(pArrSource,18,pArrTarg,24);
GetDigitValue(pArrSource,12,pArrTarg,25);
GetDigitValue(pArrSource,29,pArrTarg,26);
    GetDigitValue(pArrSource,5,pArrTarg,27);
GetDigitValue(pArrSource,21,pArrTarg,28);
GetDigitValue(pArrSource,10,pArrTarg,29);
    GetDigitValue(pArrSource,3,pArrTarg,30);
GetDigitValue(pArrSource,24,pArrTarg,31);

    for(i=0;i<32;i++)
        GetDigitValue(pArrTarg,i,pArrSource,i);
}

void L28_arb(digit *pArrSource,digit *pArrTarg)
{
    int i;
    for(i=0;i<28;i++)
        GetDigitValue(pArrSource,i,pArrTarg,i);
}

void R28_arb(digit *pArrSource,digit *pArrTarg)
{
    int i;
    for(i=0;i<28;i++)
        GetDigitValue(pArrSource,i+28,pArrTarg,i);
}

void AddTables_arb(digit *table1,digit *table2,digit *target_table,int
count)
{
```

```
    int i;
    for(i=0;i<count;i++)
      AddDigits(table1,i,table2,i,target_table,i);
}

void JoinTables_arb(digit *table1,digit *table2,digit *target_table,int
count)
{
    int i;
    for(i=0;i<count;i++)
    {
        GetDigitValue(table1,i,target_table,i);
        GetDigitValue(table2,i,target_table,i+count);
    }
}

void addAces(digit *source,int position)
{
  int i;

  for(i=0;i<64+1;i++)
    source[position].variable[i]=source[position].variable[i]^Ace;
}

void Round1(digit *pArr,digit *pTarg)
{
//                            Round 1
    //Fixing digit0
    AddDigits(pArr,0,pTarg,0,pTarg,0);
    AddDigits(pArr,1,pTarg,0,pTarg,0);
    AddDigits(pArr,2,pTarg,0,pTarg,0);
    AddDigits(pArr,4,pTarg,0,pTarg,0);
    AddDigits(pArr,5,pTarg,0,pTarg,0);

    //Fixing digit1
    //pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;
    AddDigits(pArr,1,pTarg,1,pTarg,1);
    AddDigits(pArr,2,pTarg,1,pTarg,1);
    AddDigits(pArr,4,pTarg,1,pTarg,1);
    AddDigits(pArr,5,pTarg,1,pTarg,1);

    //Fixing digit2
    //pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;
    AddDigits(pArr,0,pTarg,2,pTarg,2);
    AddDigits(pArr,1,pTarg,2,pTarg,2);
    AddDigits(pArr,2,pTarg,2,pTarg,2);
    AddDigits(pArr,3,pTarg,2,pTarg,2);
    AddDigits(pArr,4,pTarg,2,pTarg,2);
    AddDigits(pArr,5,pTarg,2,pTarg,2);

    //Fixing digit3
    //pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
    AddDigits(pArr,0,pTarg,3,pTarg,3);
    AddDigits(pArr,1,pTarg,3,pTarg,3);
    AddDigits(pArr,2,pTarg,3,pTarg,3);
    AddDigits(pArr,3,pTarg,3,pTarg,3);
    AddDigits(pArr,4,pTarg,3,pTarg,3);
```

```
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
AddDigits(pArr,7,pTarg,4,pTarg,4);
AddDigits(pArr,8,pTarg,4,pTarg,4);
AddDigits(pArr,9,pTarg,4,pTarg,4);
AddDigits(pArr,10,pTarg,4,pTarg,4);
AddDigits(pArr,11,pTarg,4,pTarg,4);

//Fixing digit5
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
AddDigits(pArr,6,pTarg,5,pTarg,5);
AddDigits(pArr,9,pTarg,5,pTarg,5);
AddDigits(pArr,10,pTarg,5,pTarg,5);

//Fixing digit6
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);
AddDigits(pArr,9,pTarg,6,pTarg,6);
AddDigits(pArr,10,pTarg,6,pTarg,6);

//Fixing digit7
AddDigits(pArr,7,pTarg,7,pTarg,7);
AddDigits(pArr,9,pTarg,7,pTarg,7);
AddDigits(pArr,11,pTarg,7,pTarg,7);

//Fixing digit8
AddDigits(pArr,12,pTarg,8,pTarg,8);
AddDigits(pArr,13,pTarg,8,pTarg,8);
AddDigits(pArr,14,pTarg,8,pTarg,8);
AddDigits(pArr,15,pTarg,8,pTarg,8);
AddDigits(pArr,16,pTarg,8,pTarg,8);
AddDigits(pArr,17,pTarg,8,pTarg,8);

//Fixing digit9
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
AddDigits(pArr,15,pTarg,9,pTarg,9);
AddDigits(pArr,16,pTarg,9,pTarg,9);

//Fixing digit10
AddDigits(pArr,12,pTarg,10,pTarg,10);
AddDigits(pArr,13,pTarg,10,pTarg,10);
AddDigits(pArr,14,pTarg,10,pTarg,10);
AddDigits(pArr,15,pTarg,10,pTarg,10);
AddDigits(pArr,16,pTarg,10,pTarg,10);
AddDigits(pArr,17,pTarg,10,pTarg,10);

//Fixing digit11
AddDigits(pArr,14,pTarg,11,pTarg,11);
AddDigits(pArr,16,pTarg,11,pTarg,11);
AddDigits(pArr,17,pTarg,11,pTarg,11);

//Fixing digit12
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
```

```
AddDigits(pArr,19,pTarg,12,pTarg,12);
AddDigits(pArr,20,pTarg,12,pTarg,12);
AddDigits(pArr,21,pTarg,12,pTarg,12);
AddDigits(pArr,22,pTarg,12,pTarg,12);
AddDigits(pArr,23,pTarg,12,pTarg,12);

//Fixing digit13
AddDigits(pArr,18,pTarg,13,pTarg,13);
AddDigits(pArr,19,pTarg,13,pTarg,13);
AddDigits(pArr,21,pTarg,13,pTarg,13);
AddDigits(pArr,22,pTarg,13,pTarg,13);
AddDigits(pArr,23,pTarg,13,pTarg,13);

//Fixing digit14
AddDigits(pArr,19,pTarg,14,pTarg,14);
AddDigits(pArr,20,pTarg,14,pTarg,14);
AddDigits(pArr,21,pTarg,14,pTarg,14);
AddDigits(pArr,22,pTarg,14,pTarg,14);

//Fixing digit15
AddDigits(pArr,19,pTarg,15,pTarg,15);
AddDigits(pArr,21,pTarg,15,pTarg,15);
AddDigits(pArr,23,pTarg,15,pTarg,15);

//Fixing digit16
AddDigits(pArr,24,pTarg,16,pTarg,16);
AddDigits(pArr,25,pTarg,16,pTarg,16);
AddDigits(pArr,26,pTarg,16,pTarg,16);
AddDigits(pArr,28,pTarg,16,pTarg,16);

//Fixing digit17
AddDigits(pArr,25,pTarg,17,pTarg,17);
AddDigits(pArr,26,pTarg,17,pTarg,17);
AddDigits(pArr,28,pTarg,17,pTarg,17);
AddDigits(pArr,29,pTarg,17,pTarg,17);

//Fixing digit18
AddDigits(pArr,24,pTarg,18,pTarg,18);
AddDigits(pArr,25,pTarg,18,pTarg,18);
AddDigits(pArr,26,pTarg,18,pTarg,18);
AddDigits(pArr,27,pTarg,18,pTarg,18);
AddDigits(pArr,28,pTarg,18,pTarg,18);

//Fixing digit19
AddDigits(pArr,24,pTarg,19,pTarg,19);
AddDigits(pArr,25,pTarg,19,pTarg,19);
AddDigits(pArr,26,pTarg,19,pTarg,19);
AddDigits(pArr,27,pTarg,19,pTarg,19);
AddDigits(pArr,28,pTarg,19,pTarg,19);

//Fixing digit20
AddDigits(pArr,31,pTarg,20,pTarg,20);
AddDigits(pArr,32,pTarg,20,pTarg,20);
AddDigits(pArr,33,pTarg,20,pTarg,20);
AddDigits(pArr,34,pTarg,20,pTarg,20);
AddDigits(pArr,35,pTarg,20,pTarg,20);
```

```
//Fixing digit21
AddDigits(pArr,30,pTarg,21,pTarg,21);
AddDigits(pArr,31,pTarg,21,pTarg,21);
AddDigits(pArr,33,pTarg,21,pTarg,21);
AddDigits(pArr,34,pTarg,21,pTarg,21);

//Fixing digit22
AddDigits(pArr,31,pTarg,22,pTarg,22);
AddDigits(pArr,32,pTarg,22,pTarg,22);
AddDigits(pArr,35,pTarg,22,pTarg,22);

//Fixing digit23
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
AddDigits(pArr,31,pTarg,23,pTarg,23);
AddDigits(pArr,33,pTarg,23,pTarg,23);
AddDigits(pArr,35,pTarg,23,pTarg,23);

//Fixing digit24
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
AddDigits(pArr,36,pTarg,24,pTarg,24);
AddDigits(pArr,37,pTarg,24,pTarg,24);
AddDigits(pArr,38,pTarg,24,pTarg,24);
AddDigits(pArr,40,pTarg,24,pTarg,24);

//Fixing digit25
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
AddDigits(pArr,36,pTarg,25,pTarg,25);
AddDigits(pArr,37,pTarg,25,pTarg,25);
AddDigits(pArr,39,pTarg,25,pTarg,25);
AddDigits(pArr,40,pTarg,25,pTarg,25);

//Fixing digit26
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
AddDigits(pArr,37,pTarg,26,pTarg,26);
AddDigits(pArr,38,pTarg,26,pTarg,26);
AddDigits(pArr,40,pTarg,26,pTarg,26);
AddDigits(pArr,41,pTarg,26,pTarg,26);

//Fixing digit27
AddDigits(pArr,36,pTarg,27,pTarg,27);
AddDigits(pArr,37,pTarg,27,pTarg,27);
AddDigits(pArr,39,pTarg,27,pTarg,27);
AddDigits(pArr,41,pTarg,27,pTarg,27);

//Fixing digit28
AddDigits(pArr,42,pTarg,28,pTarg,28);
AddDigits(pArr,43,pTarg,28,pTarg,28);
AddDigits(pArr,45,pTarg,28,pTarg,28);
AddDigits(pArr,46,pTarg,28,pTarg,28);

//Fixing digit29
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;
AddDigits(pArr,42,pTarg,29,pTarg,29);
AddDigits(pArr,43,pTarg,29,pTarg,29);
AddDigits(pArr,45,pTarg,29,pTarg,29);
AddDigits(pArr,46,pTarg,29,pTarg,29);
```

```
    //Fixing digit30
    AddDigits(pArr,43,pTarg,30,pTarg,30);
    AddDigits(pArr,44,pTarg,30,pTarg,30);
    AddDigits(pArr,45,pTarg,30,pTarg,30);
    AddDigits(pArr,46,pTarg,30,pTarg,30);
    AddDigits(pArr,47,pTarg,30,pTarg,30);

    //Fixing digit31
    AddDigits(pArr,43,pTarg,31,pTarg,31);
    AddDigits(pArr,44,pTarg,31,pTarg,31);
    AddDigits(pArr,46,pTarg,31,pTarg,31);
    AddDigits(pArr,47,pTarg,31,pTarg,31);

}

void Round5(digit *pArr,digit *pTarg)
{
//Fixing digit0
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,3,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);

//Fixing digit1
AddDigits(pArr,0,pTarg,1,pTarg,1);
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,3,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);

//Fixing digit2
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);

//Fixing digit3
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
AddDigits(pArr,3,pTarg,4,pTarg,4);
AddDigits(pArr,4,pTarg,4,pTarg,4);
AddDigits(pArr,5,pTarg,4,pTarg,4);
AddDigits(pArr,6,pTarg,4,pTarg,4);
AddDigits(pArr,7,pTarg,4,pTarg,4);
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;

//Fixing digit5
AddDigits(pArr,3,pTarg,5,pTarg,5);
AddDigits(pArr,4,pTarg,5,pTarg,5);
```

```
AddDigits(pArr,5,pTarg,5,pTarg,5);
AddDigits(pArr,7,pTarg,5,pTarg,5);
AddDigits(pArr,8,pTarg,5,pTarg,5);
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;

//Fixing digit6
AddDigits(pArr,4,pTarg,6,pTarg,6);
AddDigits(pArr,5,pTarg,6,pTarg,6);
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;

//Fixing digit7
AddDigits(pArr,3,pTarg,7,pTarg,7);
AddDigits(pArr,4,pTarg,7,pTarg,7);
AddDigits(pArr,5,pTarg,7,pTarg,7);
AddDigits(pArr,7,pTarg,7,pTarg,7);

//Fixing digit8
AddDigits(pArr,6,pTarg,8,pTarg,8);
AddDigits(pArr,7,pTarg,8,pTarg,8);
AddDigits(pArr,8,pTarg,8,pTarg,8);
AddDigits(pArr,9,pTarg,8,pTarg,8);
AddDigits(pArr,10,pTarg,8,pTarg,8);
AddDigits(pArr,11,pTarg,8,pTarg,8);
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;

//Fixing digit9
AddDigits(pArr,7,pTarg,9,pTarg,9);
AddDigits(pArr,8,pTarg,9,pTarg,9);
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;

//Fixing digit10
AddDigits(pArr,6,pTarg,10,pTarg,10);
AddDigits(pArr,7,pTarg,10,pTarg,10);
AddDigits(pArr,8,pTarg,10,pTarg,10);
AddDigits(pArr,9,pTarg,10,pTarg,10);
AddDigits(pArr,10,pTarg,10,pTarg,10);
AddDigits(pArr,11,pTarg,10,pTarg,10);
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;

//Fixing digit11
AddDigits(pArr,6,pTarg,11,pTarg,11);
AddDigits(pArr,7,pTarg,11,pTarg,11);
AddDigits(pArr,8,pTarg,11,pTarg,11);
AddDigits(pArr,9,pTarg,11,pTarg,11);
AddDigits(pArr,10,pTarg,11,pTarg,11);

//Fixing digit12
AddDigits(pArr,9,pTarg,12,pTarg,12);
AddDigits(pArr,10,pTarg,12,pTarg,12);
AddDigits(pArr,11,pTarg,12,pTarg,12);
AddDigits(pArr,12,pTarg,12,pTarg,12);
AddDigits(pArr,13,pTarg,12,pTarg,12);
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
```

```
//Fixing digit13
AddDigits(pArr,9,pTarg,13,pTarg,13);
AddDigits(pArr,10,pTarg,13,pTarg,13);
AddDigits(pArr,11,pTarg,13,pTarg,13);
AddDigits(pArr,13,pTarg,13,pTarg,13);
AddDigits(pArr,14,pTarg,13,pTarg,13);
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;

//Fixing digit14
AddDigits(pArr,9,pTarg,14,pTarg,14);
AddDigits(pArr,10,pTarg,14,pTarg,14);
AddDigits(pArr,11,pTarg,14,pTarg,14);
AddDigits(pArr,12,pTarg,14,pTarg,14);
AddDigits(pArr,13,pTarg,14,pTarg,14);
AddDigits(pArr,14,pTarg,14,pTarg,14);

//Fixing digit15
AddDigits(pArr,9,pTarg,15,pTarg,15);
AddDigits(pArr,11,pTarg,15,pTarg,15);
AddDigits(pArr,13,pTarg,15,pTarg,15);
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;

//Fixing digit16
AddDigits(pArr,13,pTarg,16,pTarg,16);
AddDigits(pArr,15,pTarg,16,pTarg,16);
AddDigits(pArr,16,pTarg,16,pTarg,16);
AddDigits(pArr,17,pTarg,16,pTarg,16);
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;

//Fixing digit17
AddDigits(pArr,13,pTarg,17,pTarg,17);
AddDigits(pArr,14,pTarg,17,pTarg,17);
AddDigits(pArr,17,pTarg,17,pTarg,17);

//Fixing digit18
AddDigits(pArr,13,pTarg,18,pTarg,18);
AddDigits(pArr,14,pTarg,18,pTarg,18);
AddDigits(pArr,15,pTarg,18,pTarg,18);
AddDigits(pArr,16,pTarg,18,pTarg,18);
AddDigits(pArr,17,pTarg,18,pTarg,18);
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;

//Fixing digit19
AddDigits(pArr,13,pTarg,19,pTarg,19);
AddDigits(pArr,14,pTarg,19,pTarg,19);
AddDigits(pArr,15,pTarg,19,pTarg,19);
AddDigits(pArr,16,pTarg,19,pTarg,19);
AddDigits(pArr,17,pTarg,19,pTarg,19);
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;

//Fixing digit20
AddDigits(pArr,15,pTarg,20,pTarg,20);
AddDigits(pArr,16,pTarg,20,pTarg,20);
AddDigits(pArr,17,pTarg,20,pTarg,20);
AddDigits(pArr,18,pTarg,20,pTarg,20);
```

```
AddDigits(pArr,19,pTarg,20,pTarg,20);
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;

//Fixing digit21
AddDigits(pArr,16,pTarg,21,pTarg,21);
AddDigits(pArr,17,pTarg,21,pTarg,21);
AddDigits(pArr,19,pTarg,21,pTarg,21);
AddDigits(pArr,20,pTarg,21,pTarg,21);
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;

//Fixing digit22
AddDigits(pArr,15,pTarg,22,pTarg,22);
AddDigits(pArr,16,pTarg,22,pTarg,22);
AddDigits(pArr,17,pTarg,22,pTarg,22);
AddDigits(pArr,20,pTarg,22,pTarg,22);
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;

//Fixing digit23
AddDigits(pArr,15,pTarg,23,pTarg,23);
AddDigits(pArr,17,pTarg,23,pTarg,23);
AddDigits(pArr,19,pTarg,23,pTarg,23);
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;

//Fixing digit24
AddDigits(pArr,19,pTarg,24,pTarg,24);
AddDigits(pArr,21,pTarg,24,pTarg,24);
AddDigits(pArr,22,pTarg,24,pTarg,24);
AddDigits(pArr,23,pTarg,24,pTarg,24);
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;

//Fixing digit25
AddDigits(pArr,19,pTarg,25,pTarg,25);
AddDigits(pArr,20,pTarg,25,pTarg,25);
AddDigits(pArr,22,pTarg,25,pTarg,25);
AddDigits(pArr,23,pTarg,25,pTarg,25);
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;

//Fixing digit26
AddDigits(pArr,19,pTarg,26,pTarg,26);
AddDigits(pArr,20,pTarg,26,pTarg,26);
AddDigits(pArr,21,pTarg,26,pTarg,26);
AddDigits(pArr,22,pTarg,26,pTarg,26);
AddDigits(pArr,23,pTarg,26,pTarg,26);
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;

//Fixing digit27
AddDigits(pArr,21,pTarg,27,pTarg,27);
AddDigits(pArr,22,pTarg,27,pTarg,27);
AddDigits(pArr,23,pTarg,27,pTarg,27);

//Fixing digit28
AddDigits(pArr,21,pTarg,28,pTarg,28);
AddDigits(pArr,22,pTarg,28,pTarg,28);
AddDigits(pArr,23,pTarg,28,pTarg,28);
AddDigits(pArr,24,pTarg,28,pTarg,28);
AddDigits(pArr,25,pTarg,28,pTarg,28);
AddDigits(pArr,26,pTarg,28,pTarg,28);
```

```
//Fixing digit29
AddDigits(pArr,22,pTarg,29,pTarg,29);
AddDigits(pArr,23,pTarg,29,pTarg,29);
AddDigits(pArr,25,pTarg,29,pTarg,29);
AddDigits(pArr,26,pTarg,29,pTarg,29);
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;

//Fixing digit30
AddDigits(pArr,21,pTarg,30,pTarg,30);
AddDigits(pArr,22,pTarg,30,pTarg,30);
AddDigits(pArr,24,pTarg,30,pTarg,30);
AddDigits(pArr,26,pTarg,30,pTarg,30);
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;

//Fixing digit31
AddDigits(pArr,21,pTarg,31,pTarg,31);
AddDigits(pArr,23,pTarg,31,pTarg,31);
AddDigits(pArr,25,pTarg,31,pTarg,31);
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;


}

void Round6(digit *pArr,digit *pTarg)
{
//Fixing digit0
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,3,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;

//Fixing digit1
AddDigits(pArr,0,pTarg,1,pTarg,1);
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,3,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;

//Fixing digit2
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;

//Fixing digit3
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);
```

```
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;

//Fixing digit4
AddDigits(pArr,3,pTarg,4,pTarg,4);
AddDigits(pArr,4,pTarg,4,pTarg,4);
AddDigits(pArr,5,pTarg,4,pTarg,4);
AddDigits(pArr,6,pTarg,4,pTarg,4);
AddDigits(pArr,7,pTarg,4,pTarg,4);

//Fixing digit5
AddDigits(pArr,4,pTarg,5,pTarg,5);
AddDigits(pArr,5,pTarg,5,pTarg,5);
AddDigits(pArr,8,pTarg,5,pTarg,5);
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;

//Fixing digit6
AddDigits(pArr,4,pTarg,6,pTarg,6);
AddDigits(pArr,5,pTarg,6,pTarg,6);
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);

//Fixing digit7
AddDigits(pArr,3,pTarg,7,pTarg,7);
AddDigits(pArr,4,pTarg,7,pTarg,7);
AddDigits(pArr,5,pTarg,7,pTarg,7);
AddDigits(pArr,7,pTarg,7,pTarg,7);

//Fixing digit8
AddDigits(pArr,6,pTarg,8,pTarg,8);
AddDigits(pArr,7,pTarg,8,pTarg,8);
AddDigits(pArr,8,pTarg,8,pTarg,8);
AddDigits(pArr,10,pTarg,8,pTarg,8);
AddDigits(pArr,11,pTarg,8,pTarg,8);

//Fixing digit9
AddDigits(pArr,7,pTarg,9,pTarg,9);
AddDigits(pArr,8,pTarg,9,pTarg,9);

//Fixing digit10
AddDigits(pArr,6,pTarg,10,pTarg,10);
AddDigits(pArr,7,pTarg,10,pTarg,10);
AddDigits(pArr,8,pTarg,10,pTarg,10);
AddDigits(pArr,9,pTarg,10,pTarg,10);
AddDigits(pArr,10,pTarg,10,pTarg,10);
AddDigits(pArr,11,pTarg,10,pTarg,10);
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;

//Fixing digit11
AddDigits(pArr,8,pTarg,11,pTarg,11);
AddDigits(pArr,10,pTarg,11,pTarg,11);
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;

//Fixing digit12
AddDigits(pArr,9,pTarg,12,pTarg,12);
AddDigits(pArr,10,pTarg,12,pTarg,12);
AddDigits(pArr,11,pTarg,12,pTarg,12);
```

```
AddDigits(pArr,12,pTarg,12,pTarg,12);
AddDigits(pArr,13,pTarg,12,pTarg,12);

//Fixing digit13
AddDigits(pArr,9,pTarg,13,pTarg,13);
AddDigits(pArr,10,pTarg,13,pTarg,13);
AddDigits(pArr,11,pTarg,13,pTarg,13);
AddDigits(pArr,13,pTarg,13,pTarg,13);
AddDigits(pArr,14,pTarg,13,pTarg,13);
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;

//Fixing digit14
AddDigits(pArr,10,pTarg,14,pTarg,14);
AddDigits(pArr,11,pTarg,14,pTarg,14);
AddDigits(pArr,12,pTarg,14,pTarg,14);
AddDigits(pArr,13,pTarg,14,pTarg,14);
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;

//Fixing digit15
AddDigits(pArr,9,pTarg,15,pTarg,15);
AddDigits(pArr,11,pTarg,15,pTarg,15);
AddDigits(pArr,13,pTarg,15,pTarg,15);
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;

//Fixing digit16
AddDigits(pArr,13,pTarg,16,pTarg,16);
AddDigits(pArr,15,pTarg,16,pTarg,16);
AddDigits(pArr,16,pTarg,16,pTarg,16);
AddDigits(pArr,17,pTarg,16,pTarg,16);
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;

//Fixing digit17
AddDigits(pArr,12,pTarg,17,pTarg,17);
AddDigits(pArr,13,pTarg,17,pTarg,17);
AddDigits(pArr,15,pTarg,17,pTarg,17);
AddDigits(pArr,16,pTarg,17,pTarg,17);

//Fixing digit18
AddDigits(pArr,13,pTarg,18,pTarg,18);
AddDigits(pArr,14,pTarg,18,pTarg,18);
AddDigits(pArr,15,pTarg,18,pTarg,18);
AddDigits(pArr,16,pTarg,18,pTarg,18);
AddDigits(pArr,17,pTarg,18,pTarg,18);
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;

//Fixing digit19
AddDigits(pArr,13,pTarg,19,pTarg,19);
AddDigits(pArr,14,pTarg,19,pTarg,19);
AddDigits(pArr,15,pTarg,19,pTarg,19);
AddDigits(pArr,16,pTarg,19,pTarg,19);
AddDigits(pArr,17,pTarg,19,pTarg,19);
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;

//Fixing digit20
AddDigits(pArr,15,pTarg,20,pTarg,20);
AddDigits(pArr,16,pTarg,20,pTarg,20);
AddDigits(pArr,17,pTarg,20,pTarg,20);
```

```
AddDigits(pArr,18,pTarg,20,pTarg,20);
AddDigits(pArr,19,pTarg,20,pTarg,20);

//Fixing digit21
AddDigits(pArr,16,pTarg,21,pTarg,21);
AddDigits(pArr,17,pTarg,21,pTarg,21);
AddDigits(pArr,19,pTarg,21,pTarg,21);
AddDigits(pArr,20,pTarg,21,pTarg,21);

//Fixing digit22
AddDigits(pArr,15,pTarg,22,pTarg,22);
AddDigits(pArr,16,pTarg,22,pTarg,22);
AddDigits(pArr,17,pTarg,22,pTarg,22);
AddDigits(pArr,18,pTarg,22,pTarg,22);
AddDigits(pArr,19,pTarg,22,pTarg,22);
AddDigits(pArr,20,pTarg,22,pTarg,22);

//Fixing digit23
AddDigits(pArr,15,pTarg,23,pTarg,23);
AddDigits(pArr,17,pTarg,23,pTarg,23);
AddDigits(pArr,19,pTarg,23,pTarg,23);
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;

//Fixing digit24
AddDigits(pArr,19,pTarg,24,pTarg,24);
AddDigits(pArr,21,pTarg,24,pTarg,24);
AddDigits(pArr,22,pTarg,24,pTarg,24);
AddDigits(pArr,23,pTarg,24,pTarg,24);
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;

//Fixing digit25
AddDigits(pArr,19,pTarg,25,pTarg,25);
AddDigits(pArr,20,pTarg,25,pTarg,25);
AddDigits(pArr,22,pTarg,25,pTarg,25);
AddDigits(pArr,23,pTarg,25,pTarg,25);
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;

//Fixing digit26
AddDigits(pArr,18,pTarg,26,pTarg,26);
AddDigits(pArr,19,pTarg,26,pTarg,26);
AddDigits(pArr,21,pTarg,26,pTarg,26);
AddDigits(pArr,22,pTarg,26,pTarg,26);

//Fixing digit27
AddDigits(pArr,18,pTarg,27,pTarg,27);
AddDigits(pArr,20,pTarg,27,pTarg,27);
AddDigits(pArr,22,pTarg,27,pTarg,27);
AddDigits(pArr,23,pTarg,27,pTarg,27);
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;

//Fixing digit28
AddDigits(pArr,21,pTarg,28,pTarg,28);
AddDigits(pArr,22,pTarg,28,pTarg,28);
AddDigits(pArr,23,pTarg,28,pTarg,28);
AddDigits(pArr,24,pTarg,28,pTarg,28);
AddDigits(pArr,25,pTarg,28,pTarg,28);
AddDigits(pArr,26,pTarg,28,pTarg,28);
```

```
//Fixing digit29
AddDigits(pArr,22,pTarg,29,pTarg,29);
AddDigits(pArr,23,pTarg,29,pTarg,29);
AddDigits(pArr,25,pTarg,29,pTarg,29);
AddDigits(pArr,26,pTarg,29,pTarg,29);
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;

//Fixing digit30
AddDigits(pArr,21,pTarg,30,pTarg,30);
AddDigits(pArr,22,pTarg,30,pTarg,30);
AddDigits(pArr,24,pTarg,30,pTarg,30);
AddDigits(pArr,26,pTarg,30,pTarg,30);
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;

//Fixing digit31
AddDigits(pArr,21,pTarg,31,pTarg,31);
AddDigits(pArr,23,pTarg,31,pTarg,31);
AddDigits(pArr,25,pTarg,31,pTarg,31);
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
}

void Round7(digit *pArr,digit *pTarg)
{
    //Fixing digit0
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,3,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;

//Fixing digit1
AddDigits(pArr,0,pTarg,1,pTarg,1);
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,3,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;

//Fixing digit2
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;

//Fixing digit3
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
```

```
//Fixing digit4
AddDigits(pArr,3,pTarg,4,pTarg,4);
AddDigits(pArr,4,pTarg,4,pTarg,4);
AddDigits(pArr,5,pTarg,4,pTarg,4);
AddDigits(pArr,6,pTarg,4,pTarg,4);
AddDigits(pArr,7,pTarg,4,pTarg,4);

//Fixing digit5
AddDigits(pArr,3,pTarg,5,pTarg,5);
AddDigits(pArr,4,pTarg,5,pTarg,5);
AddDigits(pArr,5,pTarg,5,pTarg,5);
AddDigits(pArr,7,pTarg,5,pTarg,5);
AddDigits(pArr,8,pTarg,5,pTarg,5);

//Fixing digit6
AddDigits(pArr,4,pTarg,6,pTarg,6);
AddDigits(pArr,5,pTarg,6,pTarg,6);
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);

//Fixing digit7
AddDigits(pArr,3,pTarg,7,pTarg,7);
AddDigits(pArr,5,pTarg,7,pTarg,7);
AddDigits(pArr,7,pTarg,7,pTarg,7);

//Fixing digit8
AddDigits(pArr,6,pTarg,8,pTarg,8);
AddDigits(pArr,7,pTarg,8,pTarg,8);
AddDigits(pArr,8,pTarg,8,pTarg,8);
AddDigits(pArr,9,pTarg,8,pTarg,8);
AddDigits(pArr,10,pTarg,8,pTarg,8);
AddDigits(pArr,11,pTarg,8,pTarg,8);

//Fixing digit9
AddDigits(pArr,7,pTarg,9,pTarg,9);
AddDigits(pArr,8,pTarg,9,pTarg,9);

//Fixing digit10
AddDigits(pArr,6,pTarg,10,pTarg,10);
AddDigits(pArr,7,pTarg,10,pTarg,10);
AddDigits(pArr,8,pTarg,10,pTarg,10);
AddDigits(pArr,9,pTarg,10,pTarg,10);
AddDigits(pArr,10,pTarg,10,pTarg,10);
AddDigits(pArr,11,pTarg,10,pTarg,10);
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;

//Fixing digit11
AddDigits(pArr,8,pTarg,11,pTarg,11);
AddDigits(pArr,9,pTarg,11,pTarg,11);
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;

//Fixing digit12
AddDigits(pArr,9,pTarg,12,pTarg,12);
AddDigits(pArr,10,pTarg,12,pTarg,12);
AddDigits(pArr,11,pTarg,12,pTarg,12);
AddDigits(pArr,12,pTarg,12,pTarg,12);
```

```
AddDigits(pArr,13,pTarg,12,pTarg,12);

//Fixing digit13
AddDigits(pArr,9,pTarg,13,pTarg,13);
AddDigits(pArr,10,pTarg,13,pTarg,13);
AddDigits(pArr,11,pTarg,13,pTarg,13);
AddDigits(pArr,13,pTarg,13,pTarg,13);
AddDigits(pArr,14,pTarg,13,pTarg,13);
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;

//Fixing digit14
AddDigits(pArr,9,pTarg,14,pTarg,14);
AddDigits(pArr,12,pTarg,14,pTarg,14);
AddDigits(pArr,13,pTarg,14,pTarg,14);

//Fixing digit15
AddDigits(pArr,9,pTarg,15,pTarg,15);
AddDigits(pArr,11,pTarg,15,pTarg,15);
AddDigits(pArr,13,pTarg,15,pTarg,15);
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;

//Fixing digit16
AddDigits(pArr,13,pTarg,16,pTarg,16);
AddDigits(pArr,15,pTarg,16,pTarg,16);
AddDigits(pArr,16,pTarg,16,pTarg,16);
AddDigits(pArr,17,pTarg,16,pTarg,16);
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;

//Fixing digit17
AddDigits(pArr,13,pTarg,17,pTarg,17);
AddDigits(pArr,14,pTarg,17,pTarg,17);
AddDigits(pArr,17,pTarg,17,pTarg,17);
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;

//Fixing digit18
AddDigits(pArr,13,pTarg,18,pTarg,18);
AddDigits(pArr,14,pTarg,18,pTarg,18);
AddDigits(pArr,15,pTarg,18,pTarg,18);
AddDigits(pArr,16,pTarg,18,pTarg,18);
AddDigits(pArr,17,pTarg,18,pTarg,18);
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;

//Fixing digit19
AddDigits(pArr,13,pTarg,19,pTarg,19);
AddDigits(pArr,14,pTarg,19,pTarg,19);
AddDigits(pArr,15,pTarg,19,pTarg,19);
AddDigits(pArr,16,pTarg,19,pTarg,19);
AddDigits(pArr,17,pTarg,19,pTarg,19);
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;

//Fixing digit20
AddDigits(pArr,15,pTarg,20,pTarg,20);
AddDigits(pArr,16,pTarg,20,pTarg,20);
AddDigits(pArr,17,pTarg,20,pTarg,20);
AddDigits(pArr,18,pTarg,20,pTarg,20);
AddDigits(pArr,19,pTarg,20,pTarg,20);
```

```
//Fixing digit21
AddDigits(pArr,16,pTarg,21,pTarg,21);
AddDigits(pArr,17,pTarg,21,pTarg,21);
AddDigits(pArr,19,pTarg,21,pTarg,21);
AddDigits(pArr,20,pTarg,21,pTarg,21);

//Fixing digit22
AddDigits(pArr,18,pTarg,22,pTarg,22);
AddDigits(pArr,19,pTarg,22,pTarg,22);
AddDigits(pArr,20,pTarg,22,pTarg,22);

//Fixing digit23
AddDigits(pArr,15,pTarg,23,pTarg,23);
AddDigits(pArr,17,pTarg,23,pTarg,23);
AddDigits(pArr,19,pTarg,23,pTarg,23);
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;

//Fixing digit24
AddDigits(pArr,19,pTarg,24,pTarg,24);
AddDigits(pArr,21,pTarg,24,pTarg,24);
AddDigits(pArr,22,pTarg,24,pTarg,24);
AddDigits(pArr,23,pTarg,24,pTarg,24);
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;

//Fixing digit25
AddDigits(pArr,19,pTarg,25,pTarg,25);
AddDigits(pArr,20,pTarg,25,pTarg,25);
AddDigits(pArr,22,pTarg,25,pTarg,25);
AddDigits(pArr,23,pTarg,25,pTarg,25);
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;

//Fixing digit26
AddDigits(pArr,19,pTarg,26,pTarg,26);
AddDigits(pArr,20,pTarg,26,pTarg,26);
AddDigits(pArr,21,pTarg,26,pTarg,26);
AddDigits(pArr,22,pTarg,26,pTarg,26);
AddDigits(pArr,23,pTarg,26,pTarg,26);
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;

//Fixing digit27
AddDigits(pArr,21,pTarg,27,pTarg,27);
AddDigits(pArr,22,pTarg,27,pTarg,27);
AddDigits(pArr,23,pTarg,27,pTarg,27);

//Fixing digit28
AddDigits(pArr,22,pTarg,28,pTarg,28);
AddDigits(pArr,23,pTarg,28,pTarg,28);
AddDigits(pArr,25,pTarg,28,pTarg,28);
AddDigits(pArr,26,pTarg,28,pTarg,28);

//Fixing digit29
AddDigits(pArr,22,pTarg,29,pTarg,29);
AddDigits(pArr,23,pTarg,29,pTarg,29);
AddDigits(pArr,25,pTarg,29,pTarg,29);
AddDigits(pArr,26,pTarg,29,pTarg,29);
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;
```

```
//Fixing digit30
AddDigits(pArr,21,pTarg,30,pTarg,30);
AddDigits(pArr,22,pTarg,30,pTarg,30);
AddDigits(pArr,23,pTarg,30,pTarg,30);
AddDigits(pArr,24,pTarg,30,pTarg,30);
AddDigits(pArr,25,pTarg,30,pTarg,30);

//Fixing digit31
AddDigits(pArr,21,pTarg,31,pTarg,31);
AddDigits(pArr,22,pTarg,31,pTarg,31);
AddDigits(pArr,24,pTarg,31,pTarg,31);
AddDigits(pArr,25,pTarg,31,pTarg,31);
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;

}

void Round8(digit *pArr,digit *pTarg)
{
//Fixing digit0
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,3,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);

//Fixing digit1
AddDigits(pArr,0,pTarg,1,pTarg,1);
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,3,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);

//Fixing digit2
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);

//Fixing digit3
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
AddDigits(pArr,3,pTarg,4,pTarg,4);
AddDigits(pArr,4,pTarg,4,pTarg,4);
AddDigits(pArr,5,pTarg,4,pTarg,4);
AddDigits(pArr,6,pTarg,4,pTarg,4);
AddDigits(pArr,7,pTarg,4,pTarg,4);

//Fixing digit5
AddDigits(pArr,4,pTarg,5,pTarg,5);
AddDigits(pArr,5,pTarg,5,pTarg,5);
```

```
AddDigits(pArr,8,pTarg,5,pTarg,5);

//Fixing digit6
AddDigits(pArr,4,pTarg,6,pTarg,6);
AddDigits(pArr,5,pTarg,6,pTarg,6);
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);

//Fixing digit7
AddDigits(pArr,5,pTarg,7,pTarg,7);
AddDigits(pArr,7,pTarg,7,pTarg,7);
AddDigits(pArr,8,pTarg,7,pTarg,7);
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;

//Fixing digit8
AddDigits(pArr,6,pTarg,8,pTarg,8);
AddDigits(pArr,7,pTarg,8,pTarg,8);
AddDigits(pArr,8,pTarg,8,pTarg,8);
AddDigits(pArr,10,pTarg,8,pTarg,8);
AddDigits(pArr,11,pTarg,8,pTarg,8);
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;

//Fixing digit9
AddDigits(pArr,7,pTarg,9,pTarg,9);
AddDigits(pArr,8,pTarg,9,pTarg,9);

//Fixing digit10
AddDigits(pArr,6,pTarg,10,pTarg,10);
AddDigits(pArr,7,pTarg,10,pTarg,10);
AddDigits(pArr,8,pTarg,10,pTarg,10);
AddDigits(pArr,9,pTarg,10,pTarg,10);
AddDigits(pArr,10,pTarg,10,pTarg,10);
AddDigits(pArr,11,pTarg,10,pTarg,10);
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;

//Fixing digit11
AddDigits(pArr,6,pTarg,11,pTarg,11);
AddDigits(pArr,8,pTarg,11,pTarg,11);
AddDigits(pArr,9,pTarg,11,pTarg,11);
AddDigits(pArr,11,pTarg,11,pTarg,11);

//Fixing digit12
AddDigits(pArr,9,pTarg,12,pTarg,12);
AddDigits(pArr,10,pTarg,12,pTarg,12);
AddDigits(pArr,11,pTarg,12,pTarg,12);
AddDigits(pArr,12,pTarg,12,pTarg,12);
AddDigits(pArr,13,pTarg,12,pTarg,12);
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;

//Fixing digit13
AddDigits(pArr,9,pTarg,13,pTarg,13);
AddDigits(pArr,10,pTarg,13,pTarg,13);
AddDigits(pArr,11,pTarg,13,pTarg,13);
AddDigits(pArr,13,pTarg,13,pTarg,13);
AddDigits(pArr,14,pTarg,13,pTarg,13);
```

```
//Fixing digit14
AddDigits(pArr,9,pTarg,14,pTarg,14);
AddDigits(pArr,12,pTarg,14,pTarg,14);
AddDigits(pArr,13,pTarg,14,pTarg,14);
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;

//Fixing digit15
AddDigits(pArr,9,pTarg,15,pTarg,15);
AddDigits(pArr,11,pTarg,15,pTarg,15);
AddDigits(pArr,13,pTarg,15,pTarg,15);
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;

//Fixing digit16
AddDigits(pArr,13,pTarg,16,pTarg,16);
AddDigits(pArr,15,pTarg,16,pTarg,16);
AddDigits(pArr,16,pTarg,16,pTarg,16);
AddDigits(pArr,17,pTarg,16,pTarg,16);
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;

//Fixing digit17
AddDigits(pArr,12,pTarg,17,pTarg,17);
AddDigits(pArr,13,pTarg,17,pTarg,17);
AddDigits(pArr,15,pTarg,17,pTarg,17);
AddDigits(pArr,16,pTarg,17,pTarg,17);
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;

//Fixing digit18
AddDigits(pArr,13,pTarg,18,pTarg,18);
AddDigits(pArr,14,pTarg,18,pTarg,18);
AddDigits(pArr,15,pTarg,18,pTarg,18);
AddDigits(pArr,16,pTarg,18,pTarg,18);
AddDigits(pArr,17,pTarg,18,pTarg,18);
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;

//Fixing digit19
AddDigits(pArr,13,pTarg,19,pTarg,19);
AddDigits(pArr,14,pTarg,19,pTarg,19);
AddDigits(pArr,15,pTarg,19,pTarg,19);
AddDigits(pArr,16,pTarg,19,pTarg,19);
AddDigits(pArr,17,pTarg,19,pTarg,19);
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;

//Fixing digit20
AddDigits(pArr,15,pTarg,20,pTarg,20);
AddDigits(pArr,16,pTarg,20,pTarg,20);
AddDigits(pArr,17,pTarg,20,pTarg,20);
AddDigits(pArr,18,pTarg,20,pTarg,20);
AddDigits(pArr,19,pTarg,20,pTarg,20);
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;

//Fixing digit21
AddDigits(pArr,16,pTarg,21,pTarg,21);
AddDigits(pArr,17,pTarg,21,pTarg,21);
AddDigits(pArr,19,pTarg,21,pTarg,21);
AddDigits(pArr,20,pTarg,21,pTarg,21);
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
```

```
//Fixing digit22
AddDigits(pArr,15,pTarg,22,pTarg,22);
AddDigits(pArr,16,pTarg,22,pTarg,22);
AddDigits(pArr,17,pTarg,22,pTarg,22);
AddDigits(pArr,18,pTarg,22,pTarg,22);
AddDigits(pArr,20,pTarg,22,pTarg,22);
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;

//Fixing digit23
AddDigits(pArr,15,pTarg,23,pTarg,23);
AddDigits(pArr,17,pTarg,23,pTarg,23);
AddDigits(pArr,19,pTarg,23,pTarg,23);

//Fixing digit24
AddDigits(pArr,19,pTarg,24,pTarg,24);
AddDigits(pArr,21,pTarg,24,pTarg,24);
AddDigits(pArr,22,pTarg,24,pTarg,24);
AddDigits(pArr,23,pTarg,24,pTarg,24);

//Fixing digit25
AddDigits(pArr,19,pTarg,25,pTarg,25);
AddDigits(pArr,20,pTarg,25,pTarg,25);
AddDigits(pArr,22,pTarg,25,pTarg,25);
AddDigits(pArr,23,pTarg,25,pTarg,25);

//Fixing digit26
AddDigits(pArr,18,pTarg,26,pTarg,26);
AddDigits(pArr,19,pTarg,26,pTarg,26);
AddDigits(pArr,21,pTarg,26,pTarg,26);
AddDigits(pArr,22,pTarg,26,pTarg,26);

//Fixing digit27
AddDigits(pArr,18,pTarg,27,pTarg,27);
AddDigits(pArr,20,pTarg,27,pTarg,27);
AddDigits(pArr,22,pTarg,27,pTarg,27);
AddDigits(pArr,23,pTarg,27,pTarg,27);
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;

//Fixing digit28
AddDigits(pArr,21,pTarg,28,pTarg,28);
AddDigits(pArr,23,pTarg,28,pTarg,28);
AddDigits(pArr,25,pTarg,28,pTarg,28);
AddDigits(pArr,26,pTarg,28,pTarg,28);
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;

//Fixing digit29
AddDigits(pArr,22,pTarg,29,pTarg,29);
AddDigits(pArr,23,pTarg,29,pTarg,29);
AddDigits(pArr,25,pTarg,29,pTarg,29);
AddDigits(pArr,26,pTarg,29,pTarg,29);
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;

//Fixing digit30
AddDigits(pArr,22,pTarg,30,pTarg,30);
AddDigits(pArr,24,pTarg,30,pTarg,30);
AddDigits(pArr,25,pTarg,30,pTarg,30);
```

```
AddDigits(pArr,26,pTarg,30,pTarg,30);
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;

//Fixing digit31
AddDigits(pArr,21,pTarg,31,pTarg,31);
AddDigits(pArr,22,pTarg,31,pTarg,31);
AddDigits(pArr,23,pTarg,31,pTarg,31);
AddDigits(pArr,25,pTarg,31,pTarg,31);
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;

}

void Round9(digit *pArr,digit *pTarg)
{
//Fixing digit0
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,3,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;

//Fixing digit1
AddDigits(pArr,0,pTarg,1,pTarg,1);
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,3,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;

//Fixing digit2
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;

//Fixing digit3
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;

//Fixing digit4
AddDigits(pArr,3,pTarg,4,pTarg,4);
AddDigits(pArr,4,pTarg,4,pTarg,4);
AddDigits(pArr,5,pTarg,4,pTarg,4);
AddDigits(pArr,6,pTarg,4,pTarg,4);
AddDigits(pArr,7,pTarg,4,pTarg,4);

//Fixing digit5
AddDigits(pArr,3,pTarg,5,pTarg,5);
AddDigits(pArr,4,pTarg,5,pTarg,5);
```

```
AddDigits(pArr,5,pTarg,5,pTarg,5);
AddDigits(pArr,7,pTarg,5,pTarg,5);
AddDigits(pArr,8,pTarg,5,pTarg,5);

//Fixing digit6
AddDigits(pArr,4,pTarg,6,pTarg,6);
AddDigits(pArr,5,pTarg,6,pTarg,6);
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);

//Fixing digit7
AddDigits(pArr,3,pTarg,7,pTarg,7);
AddDigits(pArr,5,pTarg,7,pTarg,7);
AddDigits(pArr,7,pTarg,7,pTarg,7);
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;

//Fixing digit8
AddDigits(pArr,6,pTarg,8,pTarg,8);
AddDigits(pArr,7,pTarg,8,pTarg,8);
AddDigits(pArr,8,pTarg,8,pTarg,8);
AddDigits(pArr,10,pTarg,8,pTarg,8);
AddDigits(pArr,11,pTarg,8,pTarg,8);

//Fixing digit9
AddDigits(pArr,7,pTarg,9,pTarg,9);
AddDigits(pArr,8,pTarg,9,pTarg,9);

//Fixing digit10
AddDigits(pArr,6,pTarg,10,pTarg,10);
AddDigits(pArr,7,pTarg,10,pTarg,10);
AddDigits(pArr,8,pTarg,10,pTarg,10);
AddDigits(pArr,9,pTarg,10,pTarg,10);
AddDigits(pArr,10,pTarg,10,pTarg,10);
AddDigits(pArr,11,pTarg,10,pTarg,10);
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;

//Fixing digit11
AddDigits(pArr,8,pTarg,11,pTarg,11);
AddDigits(pArr,9,pTarg,11,pTarg,11);

//Fixing digit12
AddDigits(pArr,9,pTarg,12,pTarg,12);
AddDigits(pArr,10,pTarg,12,pTarg,12);
AddDigits(pArr,11,pTarg,12,pTarg,12);
AddDigits(pArr,12,pTarg,12,pTarg,12);
AddDigits(pArr,13,pTarg,12,pTarg,12);
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;

//Fixing digit13
AddDigits(pArr,9,pTarg,13,pTarg,13);
AddDigits(pArr,10,pTarg,13,pTarg,13);
AddDigits(pArr,11,pTarg,13,pTarg,13);
AddDigits(pArr,13,pTarg,13,pTarg,13);
AddDigits(pArr,14,pTarg,13,pTarg,13);

//Fixing digit14
```

```
AddDigits(pArr,10,pTarg,14,pTarg,14);
AddDigits(pArr,11,pTarg,14,pTarg,14);
AddDigits(pArr,12,pTarg,14,pTarg,14);
AddDigits(pArr,13,pTarg,14,pTarg,14);
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;

//Fixing digit15
AddDigits(pArr,9,pTarg,15,pTarg,15);
AddDigits(pArr,11,pTarg,15,pTarg,15);
AddDigits(pArr,13,pTarg,15,pTarg,15);

//Fixing digit16
AddDigits(pArr,13,pTarg,16,pTarg,16);
AddDigits(pArr,15,pTarg,16,pTarg,16);
AddDigits(pArr,16,pTarg,16,pTarg,16);
AddDigits(pArr,17,pTarg,16,pTarg,16);
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;

//Fixing digit17
AddDigits(pArr,13,pTarg,17,pTarg,17);
AddDigits(pArr,14,pTarg,17,pTarg,17);
AddDigits(pArr,17,pTarg,17,pTarg,17);
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;

//Fixing digit18
AddDigits(pArr,13,pTarg,18,pTarg,18);
AddDigits(pArr,14,pTarg,18,pTarg,18);
AddDigits(pArr,15,pTarg,18,pTarg,18);
AddDigits(pArr,16,pTarg,18,pTarg,18);
AddDigits(pArr,17,pTarg,18,pTarg,18);
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;

//Fixing digit19
AddDigits(pArr,13,pTarg,19,pTarg,19);
AddDigits(pArr,14,pTarg,19,pTarg,19);
AddDigits(pArr,15,pTarg,19,pTarg,19);
AddDigits(pArr,16,pTarg,19,pTarg,19);
AddDigits(pArr,17,pTarg,19,pTarg,19);
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;

//Fixing digit20
AddDigits(pArr,15,pTarg,20,pTarg,20);
AddDigits(pArr,16,pTarg,20,pTarg,20);
AddDigits(pArr,17,pTarg,20,pTarg,20);
AddDigits(pArr,18,pTarg,20,pTarg,20);
AddDigits(pArr,19,pTarg,20,pTarg,20);
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;

//Fixing digit21
AddDigits(pArr,16,pTarg,21,pTarg,21);
AddDigits(pArr,17,pTarg,21,pTarg,21);
AddDigits(pArr,19,pTarg,21,pTarg,21);
AddDigits(pArr,20,pTarg,21,pTarg,21);
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;

//Fixing digit22
AddDigits(pArr,18,pTarg,22,pTarg,22);
```

```
AddDigits(pArr,19,pTarg,22,pTarg,22);
AddDigits(pArr,20,pTarg,22,pTarg,22);

//Fixing digit23
AddDigits(pArr,15,pTarg,23,pTarg,23);
AddDigits(pArr,17,pTarg,23,pTarg,23);
AddDigits(pArr,19,pTarg,23,pTarg,23);

//Fixing digit24
AddDigits(pArr,19,pTarg,24,pTarg,24);
AddDigits(pArr,21,pTarg,24,pTarg,24);
AddDigits(pArr,22,pTarg,24,pTarg,24);
AddDigits(pArr,23,pTarg,24,pTarg,24);

//Fixing digit25
AddDigits(pArr,19,pTarg,25,pTarg,25);
AddDigits(pArr,20,pTarg,25,pTarg,25);
AddDigits(pArr,22,pTarg,25,pTarg,25);
AddDigits(pArr,23,pTarg,25,pTarg,25);

//Fixing digit26
AddDigits(pArr,19,pTarg,26,pTarg,26);
AddDigits(pArr,20,pTarg,26,pTarg,26);
AddDigits(pArr,21,pTarg,26,pTarg,26);
AddDigits(pArr,22,pTarg,26,pTarg,26);
AddDigits(pArr,23,pTarg,26,pTarg,26);

//Fixing digit27
AddDigits(pArr,21,pTarg,27,pTarg,27);
AddDigits(pArr,22,pTarg,27,pTarg,27);
AddDigits(pArr,23,pTarg,27,pTarg,27);
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;

//Fixing digit28
AddDigits(pArr,21,pTarg,28,pTarg,28);
AddDigits(pArr,23,pTarg,28,pTarg,28);
AddDigits(pArr,25,pTarg,28,pTarg,28);
AddDigits(pArr,26,pTarg,28,pTarg,28);
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;

//Fixing digit29
AddDigits(pArr,22,pTarg,29,pTarg,29);
AddDigits(pArr,23,pTarg,29,pTarg,29);
AddDigits(pArr,25,pTarg,29,pTarg,29);
AddDigits(pArr,26,pTarg,29,pTarg,29);
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;

//Fixing digit30
AddDigits(pArr,21,pTarg,30,pTarg,30);
AddDigits(pArr,22,pTarg,30,pTarg,30);
AddDigits(pArr,24,pTarg,30,pTarg,30);
AddDigits(pArr,26,pTarg,30,pTarg,30);
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;

//Fixing digit31
AddDigits(pArr,21,pTarg,31,pTarg,31);
AddDigits(pArr,22,pTarg,31,pTarg,31);
```

```
AddDigits(pArr,24,pTarg,31,pTarg,31);
AddDigits(pArr,25,pTarg,31,pTarg,31);
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;

}

void Round10(digit *pArr,digit *pTarg)
{
//Fixing digit0
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,2,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);

//Fixing digit1
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,2,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
AddDigits(pArr,5,pTarg,1,pTarg,1);

//Fixing digit2
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);

//Fixing digit3
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
AddDigits(pArr,7,pTarg,4,pTarg,4);
AddDigits(pArr,8,pTarg,4,pTarg,4);
AddDigits(pArr,9,pTarg,4,pTarg,4);
AddDigits(pArr,10,pTarg,4,pTarg,4);
AddDigits(pArr,11,pTarg,4,pTarg,4);

//Fixing digit5
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
AddDigits(pArr,6,pTarg,5,pTarg,5);
AddDigits(pArr,7,pTarg,5,pTarg,5);
AddDigits(pArr,9,pTarg,5,pTarg,5);
AddDigits(pArr,10,pTarg,5,pTarg,5);
AddDigits(pArr,11,pTarg,5,pTarg,5);
```

```
//Fixing digit6
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);
AddDigits(pArr,9,pTarg,6,pTarg,6);
AddDigits(pArr,10,pTarg,6,pTarg,6);

//Fixing digit7
AddDigits(pArr,7,pTarg,7,pTarg,7);
AddDigits(pArr,9,pTarg,7,pTarg,7);
AddDigits(pArr,10,pTarg,7,pTarg,7);
AddDigits(pArr,11,pTarg,7,pTarg,7);

//Fixing digit8
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
AddDigits(pArr,12,pTarg,8,pTarg,8);
AddDigits(pArr,13,pTarg,8,pTarg,8);
AddDigits(pArr,14,pTarg,8,pTarg,8);
AddDigits(pArr,15,pTarg,8,pTarg,8);
AddDigits(pArr,16,pTarg,8,pTarg,8);
AddDigits(pArr,17,pTarg,8,pTarg,8);

//Fixing digit9
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
AddDigits(pArr,15,pTarg,9,pTarg,9);
AddDigits(pArr,16,pTarg,9,pTarg,9);

//Fixing digit10
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;
AddDigits(pArr,12,pTarg,10,pTarg,10);
AddDigits(pArr,13,pTarg,10,pTarg,10);
AddDigits(pArr,14,pTarg,10,pTarg,10);
AddDigits(pArr,15,pTarg,10,pTarg,10);
AddDigits(pArr,16,pTarg,10,pTarg,10);
AddDigits(pArr,17,pTarg,10,pTarg,10);

//Fixing digit11
AddDigits(pArr,13,pTarg,11,pTarg,11);
AddDigits(pArr,14,pTarg,11,pTarg,11);
AddDigits(pArr,15,pTarg,11,pTarg,11);
AddDigits(pArr,16,pTarg,11,pTarg,11);
AddDigits(pArr,17,pTarg,11,pTarg,11);

//Fixing digit12
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
AddDigits(pArr,19,pTarg,12,pTarg,12);
AddDigits(pArr,20,pTarg,12,pTarg,12);
AddDigits(pArr,21,pTarg,12,pTarg,12);
AddDigits(pArr,22,pTarg,12,pTarg,12);
AddDigits(pArr,23,pTarg,12,pTarg,12);

//Fixing digit13
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
```

```
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
AddDigits(pArr,18,pTarg,13,pTarg,13);
AddDigits(pArr,19,pTarg,13,pTarg,13);
AddDigits(pArr,21,pTarg,13,pTarg,13);
AddDigits(pArr,22,pTarg,13,pTarg,13);
AddDigits(pArr,23,pTarg,13,pTarg,13);

//Fixing digit14
AddDigits(pArr,18,pTarg,14,pTarg,14);
AddDigits(pArr,19,pTarg,14,pTarg,14);
AddDigits(pArr,20,pTarg,14,pTarg,14);
AddDigits(pArr,21,pTarg,14,pTarg,14);
AddDigits(pArr,22,pTarg,14,pTarg,14);
AddDigits(pArr,23,pTarg,14,pTarg,14);

//Fixing digit15
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
AddDigits(pArr,19,pTarg,15,pTarg,15);
AddDigits(pArr,21,pTarg,15,pTarg,15);
AddDigits(pArr,23,pTarg,15,pTarg,15);

//Fixing digit16
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;
AddDigits(pArr,24,pTarg,16,pTarg,16);
AddDigits(pArr,25,pTarg,16,pTarg,16);
AddDigits(pArr,26,pTarg,16,pTarg,16);
AddDigits(pArr,28,pTarg,16,pTarg,16);

//Fixing digit17
AddDigits(pArr,24,pTarg,17,pTarg,17);
AddDigits(pArr,27,pTarg,17,pTarg,17);
AddDigits(pArr,28,pTarg,17,pTarg,17);

//Fixing digit18
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;
AddDigits(pArr,24,pTarg,18,pTarg,18);
AddDigits(pArr,25,pTarg,18,pTarg,18);
AddDigits(pArr,26,pTarg,18,pTarg,18);
AddDigits(pArr,27,pTarg,18,pTarg,18);
AddDigits(pArr,28,pTarg,18,pTarg,18);

//Fixing digit19
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;
AddDigits(pArr,24,pTarg,19,pTarg,19);
AddDigits(pArr,25,pTarg,19,pTarg,19);
AddDigits(pArr,26,pTarg,19,pTarg,19);
AddDigits(pArr,27,pTarg,19,pTarg,19);
AddDigits(pArr,28,pTarg,19,pTarg,19);

//Fixing digit20
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;
AddDigits(pArr,31,pTarg,20,pTarg,20);
AddDigits(pArr,32,pTarg,20,pTarg,20);
AddDigits(pArr,33,pTarg,20,pTarg,20);
AddDigits(pArr,34,pTarg,20,pTarg,20);
AddDigits(pArr,35,pTarg,20,pTarg,20);
```

```
//Fixing digit21
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
AddDigits(pArr,30,pTarg,21,pTarg,21);
AddDigits(pArr,31,pTarg,21,pTarg,21);
AddDigits(pArr,33,pTarg,21,pTarg,21);
AddDigits(pArr,34,pTarg,21,pTarg,21);

//Fixing digit22
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
AddDigits(pArr,30,pTarg,22,pTarg,22);
AddDigits(pArr,33,pTarg,22,pTarg,22);
AddDigits(pArr,34,pTarg,22,pTarg,22);
AddDigits(pArr,35,pTarg,22,pTarg,22);

//Fixing digit23
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
AddDigits(pArr,31,pTarg,23,pTarg,23);
AddDigits(pArr,33,pTarg,23,pTarg,23);
AddDigits(pArr,35,pTarg,23,pTarg,23);

//Fixing digit24
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
AddDigits(pArr,36,pTarg,24,pTarg,24);
AddDigits(pArr,37,pTarg,24,pTarg,24);
AddDigits(pArr,38,pTarg,24,pTarg,24);
AddDigits(pArr,40,pTarg,24,pTarg,24);

//Fixing digit25
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
AddDigits(pArr,36,pTarg,25,pTarg,25);
AddDigits(pArr,37,pTarg,25,pTarg,25);
AddDigits(pArr,39,pTarg,25,pTarg,25);
AddDigits(pArr,40,pTarg,25,pTarg,25);

//Fixing digit26
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
AddDigits(pArr,36,pTarg,26,pTarg,26);
AddDigits(pArr,37,pTarg,26,pTarg,26);
AddDigits(pArr,38,pTarg,26,pTarg,26);
AddDigits(pArr,39,pTarg,26,pTarg,26);
AddDigits(pArr,40,pTarg,26,pTarg,26);

//Fixing digit27
AddDigits(pArr,36,pTarg,27,pTarg,27);
AddDigits(pArr,37,pTarg,27,pTarg,27);
AddDigits(pArr,38,pTarg,27,pTarg,27);

//Fixing digit28
AddDigits(pArr,42,pTarg,28,pTarg,28);
AddDigits(pArr,43,pTarg,28,pTarg,28);
AddDigits(pArr,44,pTarg,28,pTarg,28);
AddDigits(pArr,45,pTarg,28,pTarg,28);
AddDigits(pArr,46,pTarg,28,pTarg,28);
```

```
AddDigits(pArr,47,pTarg,28,pTarg,28);

//Fixing digit29
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;
AddDigits(pArr,42,pTarg,29,pTarg,29);
AddDigits(pArr,43,pTarg,29,pTarg,29);
AddDigits(pArr,45,pTarg,29,pTarg,29);
AddDigits(pArr,46,pTarg,29,pTarg,29);

//Fixing digit30
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
AddDigits(pArr,42,pTarg,30,pTarg,30);
AddDigits(pArr,44,pTarg,30,pTarg,30);
AddDigits(pArr,46,pTarg,30,pTarg,30);
AddDigits(pArr,47,pTarg,30,pTarg,30);

//Fixing digit31
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
AddDigits(pArr,43,pTarg,31,pTarg,31);
AddDigits(pArr,45,pTarg,31,pTarg,31);
AddDigits(pArr,47,pTarg,31,pTarg,31);


}

void Round11(digit *pArr,digit *pTarg)
{
//Fixing digit0
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,2,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);

//Fixing digit1
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,2,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
AddDigits(pArr,5,pTarg,1,pTarg,1);

//Fixing digit2
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);

//Fixing digit3
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
```

```
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
AddDigits(pArr,7,pTarg,4,pTarg,4);
AddDigits(pArr,8,pTarg,4,pTarg,4);
AddDigits(pArr,9,pTarg,4,pTarg,4);
AddDigits(pArr,10,pTarg,4,pTarg,4);
AddDigits(pArr,11,pTarg,4,pTarg,4);

//Fixing digit5
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
AddDigits(pArr,6,pTarg,5,pTarg,5);
AddDigits(pArr,7,pTarg,5,pTarg,5);
AddDigits(pArr,9,pTarg,5,pTarg,5);
AddDigits(pArr,10,pTarg,5,pTarg,5);
AddDigits(pArr,11,pTarg,5,pTarg,5);

//Fixing digit6
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);
AddDigits(pArr,9,pTarg,6,pTarg,6);
AddDigits(pArr,10,pTarg,6,pTarg,6);

//Fixing digit7
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;
AddDigits(pArr,7,pTarg,7,pTarg,7);
AddDigits(pArr,9,pTarg,7,pTarg,7);
AddDigits(pArr,11,pTarg,7,pTarg,7);

//Fixing digit8
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
AddDigits(pArr,12,pTarg,8,pTarg,8);
AddDigits(pArr,13,pTarg,8,pTarg,8);
AddDigits(pArr,14,pTarg,8,pTarg,8);
AddDigits(pArr,15,pTarg,8,pTarg,8);
AddDigits(pArr,16,pTarg,8,pTarg,8);
AddDigits(pArr,17,pTarg,8,pTarg,8);

//Fixing digit9
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
AddDigits(pArr,15,pTarg,9,pTarg,9);
AddDigits(pArr,16,pTarg,9,pTarg,9);
```

```
//Fixing digit10
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;
AddDigits(pArr,12,pTarg,10,pTarg,10);
AddDigits(pArr,13,pTarg,10,pTarg,10);
AddDigits(pArr,14,pTarg,10,pTarg,10);
AddDigits(pArr,15,pTarg,10,pTarg,10);
AddDigits(pArr,16,pTarg,10,pTarg,10);
AddDigits(pArr,17,pTarg,10,pTarg,10);

//Fixing digit11
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
AddDigits(pArr,14,pTarg,11,pTarg,11);
AddDigits(pArr,15,pTarg,11,pTarg,11);

//Fixing digit12
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
AddDigits(pArr,19,pTarg,12,pTarg,12);
AddDigits(pArr,20,pTarg,12,pTarg,12);
AddDigits(pArr,21,pTarg,12,pTarg,12);
AddDigits(pArr,22,pTarg,12,pTarg,12);
AddDigits(pArr,23,pTarg,12,pTarg,12);

//Fixing digit13
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
AddDigits(pArr,18,pTarg,13,pTarg,13);
AddDigits(pArr,19,pTarg,13,pTarg,13);
AddDigits(pArr,21,pTarg,13,pTarg,13);
AddDigits(pArr,22,pTarg,13,pTarg,13);
AddDigits(pArr,23,pTarg,13,pTarg,13);

//Fixing digit14
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;
AddDigits(pArr,19,pTarg,14,pTarg,14);
AddDigits(pArr,20,pTarg,14,pTarg,14);
AddDigits(pArr,23,pTarg,14,pTarg,14);

//Fixing digit15
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
AddDigits(pArr,19,pTarg,15,pTarg,15);
AddDigits(pArr,21,pTarg,15,pTarg,15);
AddDigits(pArr,23,pTarg,15,pTarg,15);

//Fixing digit16
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;
AddDigits(pArr,24,pTarg,16,pTarg,16);
AddDigits(pArr,25,pTarg,16,pTarg,16);
AddDigits(pArr,26,pTarg,16,pTarg,16);
AddDigits(pArr,28,pTarg,16,pTarg,16);

//Fixing digit17
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;
AddDigits(pArr,24,pTarg,17,pTarg,17);
AddDigits(pArr,27,pTarg,17,pTarg,17);
```

```
AddDigits(pArr,28,pTarg,17,pTarg,17);

//Fixing digit18
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;
AddDigits(pArr,24,pTarg,18,pTarg,18);
AddDigits(pArr,25,pTarg,18,pTarg,18);
AddDigits(pArr,26,pTarg,18,pTarg,18);
AddDigits(pArr,27,pTarg,18,pTarg,18);
AddDigits(pArr,28,pTarg,18,pTarg,18);

//Fixing digit19
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;
AddDigits(pArr,24,pTarg,19,pTarg,19);
AddDigits(pArr,25,pTarg,19,pTarg,19);
AddDigits(pArr,26,pTarg,19,pTarg,19);
AddDigits(pArr,27,pTarg,19,pTarg,19);
AddDigits(pArr,28,pTarg,19,pTarg,19);

//Fixing digit20
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;
AddDigits(pArr,31,pTarg,20,pTarg,20);
AddDigits(pArr,32,pTarg,20,pTarg,20);
AddDigits(pArr,33,pTarg,20,pTarg,20);
AddDigits(pArr,34,pTarg,20,pTarg,20);
AddDigits(pArr,35,pTarg,20,pTarg,20);

//Fixing digit21
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
AddDigits(pArr,30,pTarg,21,pTarg,21);
AddDigits(pArr,31,pTarg,21,pTarg,21);
AddDigits(pArr,33,pTarg,21,pTarg,21);
AddDigits(pArr,34,pTarg,21,pTarg,21);

//Fixing digit22
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
AddDigits(pArr,30,pTarg,22,pTarg,22);
AddDigits(pArr,31,pTarg,22,pTarg,22);
AddDigits(pArr,32,pTarg,22,pTarg,22);

//Fixing digit23
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
AddDigits(pArr,31,pTarg,23,pTarg,23);
AddDigits(pArr,33,pTarg,23,pTarg,23);
AddDigits(pArr,35,pTarg,23,pTarg,23);

//Fixing digit24
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
AddDigits(pArr,36,pTarg,24,pTarg,24);
AddDigits(pArr,37,pTarg,24,pTarg,24);
AddDigits(pArr,38,pTarg,24,pTarg,24);
AddDigits(pArr,40,pTarg,24,pTarg,24);

//Fixing digit25
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
```

```
AddDigits(pArr,36,pTarg,25,pTarg,25);
AddDigits(pArr,37,pTarg,25,pTarg,25);
AddDigits(pArr,39,pTarg,25,pTarg,25);
AddDigits(pArr,40,pTarg,25,pTarg,25);

//Fixing digit26
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
AddDigits(pArr,36,pTarg,26,pTarg,26);
AddDigits(pArr,37,pTarg,26,pTarg,26);
AddDigits(pArr,38,pTarg,26,pTarg,26);
AddDigits(pArr,39,pTarg,26,pTarg,26);
AddDigits(pArr,40,pTarg,26,pTarg,26);

//Fixing digit27
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;
AddDigits(pArr,36,pTarg,27,pTarg,27);
AddDigits(pArr,37,pTarg,27,pTarg,27);
AddDigits(pArr,38,pTarg,27,pTarg,27);

//Fixing digit28
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;
AddDigits(pArr,42,pTarg,28,pTarg,28);
AddDigits(pArr,43,pTarg,28,pTarg,28);
AddDigits(pArr,45,pTarg,28,pTarg,28);
AddDigits(pArr,46,pTarg,28,pTarg,28);

//Fixing digit29
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;
AddDigits(pArr,42,pTarg,29,pTarg,29);
AddDigits(pArr,43,pTarg,29,pTarg,29);
AddDigits(pArr,45,pTarg,29,pTarg,29);
AddDigits(pArr,46,pTarg,29,pTarg,29);

//Fixing digit30
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
AddDigits(pArr,43,pTarg,30,pTarg,30);
AddDigits(pArr,44,pTarg,30,pTarg,30);
AddDigits(pArr,45,pTarg,30,pTarg,30);
AddDigits(pArr,46,pTarg,30,pTarg,30);
AddDigits(pArr,47,pTarg,30,pTarg,30);

//Fixing digit31
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
AddDigits(pArr,43,pTarg,31,pTarg,31);
AddDigits(pArr,44,pTarg,31,pTarg,31);
AddDigits(pArr,46,pTarg,31,pTarg,31);
AddDigits(pArr,47,pTarg,31,pTarg,31);
}

void Round12(digit *pArr,digit *pTarg)
{
//Fixing digit0
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;
```

```
pTarg[0].variable[0]=pTarg[0].variable[0]^Ace;
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,2,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);

//Fixing digit1
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;
pTarg[1].variable[0]=pTarg[1].variable[0]^Ace;
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,2,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
AddDigits(pArr,5,pTarg,1,pTarg,1);

//Fixing digit2
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;
pTarg[2].variable[0]=pTarg[2].variable[0]^Ace;
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);

//Fixing digit3
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
pTarg[3].variable[0]=pTarg[3].variable[0]^Ace;
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
AddDigits(pArr,7,pTarg,4,pTarg,4);
AddDigits(pArr,8,pTarg,4,pTarg,4);
AddDigits(pArr,9,pTarg,4,pTarg,4);
AddDigits(pArr,10,pTarg,4,pTarg,4);
AddDigits(pArr,11,pTarg,4,pTarg,4);

//Fixing digit5
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
AddDigits(pArr,6,pTarg,5,pTarg,5);
AddDigits(pArr,7,pTarg,5,pTarg,5);
AddDigits(pArr,9,pTarg,5,pTarg,5);
AddDigits(pArr,10,pTarg,5,pTarg,5);
AddDigits(pArr,11,pTarg,5,pTarg,5);

//Fixing digit6
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
AddDigits(pArr,6,pTarg,6,pTarg,6);
```

```
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);
AddDigits(pArr,9,pTarg,6,pTarg,6);
AddDigits(pArr,10,pTarg,6,pTarg,6);

//Fixing digit7
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;
AddDigits(pArr,7,pTarg,7,pTarg,7);
AddDigits(pArr,9,pTarg,7,pTarg,7);
AddDigits(pArr,11,pTarg,7,pTarg,7);

//Fixing digit8
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
AddDigits(pArr,12,pTarg,8,pTarg,8);
AddDigits(pArr,13,pTarg,8,pTarg,8);
AddDigits(pArr,15,pTarg,8,pTarg,8);
AddDigits(pArr,16,pTarg,8,pTarg,8);
AddDigits(pArr,17,pTarg,8,pTarg,8);

//Fixing digit9
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
AddDigits(pArr,15,pTarg,9,pTarg,9);
AddDigits(pArr,16,pTarg,9,pTarg,9);

//Fixing digit10
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;
AddDigits(pArr,12,pTarg,10,pTarg,10);
AddDigits(pArr,13,pTarg,10,pTarg,10);
AddDigits(pArr,14,pTarg,10,pTarg,10);
AddDigits(pArr,15,pTarg,10,pTarg,10);
AddDigits(pArr,16,pTarg,10,pTarg,10);
AddDigits(pArr,17,pTarg,10,pTarg,10);

//Fixing digit11
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
AddDigits(pArr,14,pTarg,11,pTarg,11);
AddDigits(pArr,15,pTarg,11,pTarg,11);

//Fixing digit12
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
AddDigits(pArr,19,pTarg,12,pTarg,12);
AddDigits(pArr,20,pTarg,12,pTarg,12);
AddDigits(pArr,21,pTarg,12,pTarg,12);
AddDigits(pArr,22,pTarg,12,pTarg,12);
AddDigits(pArr,23,pTarg,12,pTarg,12);

//Fixing digit13
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
AddDigits(pArr,18,pTarg,13,pTarg,13);
AddDigits(pArr,19,pTarg,13,pTarg,13);
AddDigits(pArr,21,pTarg,13,pTarg,13);
AddDigits(pArr,22,pTarg,13,pTarg,13);
AddDigits(pArr,23,pTarg,13,pTarg,13);
```

```
//Fixing digit14
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;
AddDigits(pArr,19,pTarg,14,pTarg,14);
AddDigits(pArr,20,pTarg,14,pTarg,14);
AddDigits(pArr,21,pTarg,14,pTarg,14);
AddDigits(pArr,22,pTarg,14,pTarg,14);

//Fixing digit15
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
AddDigits(pArr,19,pTarg,15,pTarg,15);
AddDigits(pArr,21,pTarg,15,pTarg,15);
AddDigits(pArr,23,pTarg,15,pTarg,15);

//Fixing digit16
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;
AddDigits(pArr,24,pTarg,16,pTarg,16);
AddDigits(pArr,25,pTarg,16,pTarg,16);
AddDigits(pArr,26,pTarg,16,pTarg,16);
AddDigits(pArr,28,pTarg,16,pTarg,16);

//Fixing digit17
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;
AddDigits(pArr,24,pTarg,17,pTarg,17);
AddDigits(pArr,27,pTarg,17,pTarg,17);
AddDigits(pArr,28,pTarg,17,pTarg,17);

//Fixing digit18
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;
AddDigits(pArr,24,pTarg,18,pTarg,18);
AddDigits(pArr,25,pTarg,18,pTarg,18);
AddDigits(pArr,26,pTarg,18,pTarg,18);
AddDigits(pArr,27,pTarg,18,pTarg,18);
AddDigits(pArr,28,pTarg,18,pTarg,18);

//Fixing digit19
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;
AddDigits(pArr,24,pTarg,19,pTarg,19);
AddDigits(pArr,25,pTarg,19,pTarg,19);
AddDigits(pArr,26,pTarg,19,pTarg,19);
AddDigits(pArr,27,pTarg,19,pTarg,19);
AddDigits(pArr,28,pTarg,19,pTarg,19);

//Fixing digit20
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;
AddDigits(pArr,31,pTarg,20,pTarg,20);
AddDigits(pArr,32,pTarg,20,pTarg,20);
AddDigits(pArr,33,pTarg,20,pTarg,20);
AddDigits(pArr,34,pTarg,20,pTarg,20);
AddDigits(pArr,35,pTarg,20,pTarg,20);

//Fixing digit21
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
AddDigits(pArr,30,pTarg,21,pTarg,21);
AddDigits(pArr,31,pTarg,21,pTarg,21);
AddDigits(pArr,33,pTarg,21,pTarg,21);
```

```
AddDigits(pArr,34,pTarg,21,pTarg,21);

//Fixing digit22
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
AddDigits(pArr,30,pTarg,22,pTarg,22);
AddDigits(pArr,31,pTarg,22,pTarg,22);
AddDigits(pArr,32,pTarg,22,pTarg,22);

//Fixing digit23
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
AddDigits(pArr,31,pTarg,23,pTarg,23);
AddDigits(pArr,33,pTarg,23,pTarg,23);
AddDigits(pArr,35,pTarg,23,pTarg,23);

//Fixing digit24
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
AddDigits(pArr,36,pTarg,24,pTarg,24);
AddDigits(pArr,37,pTarg,24,pTarg,24);
AddDigits(pArr,38,pTarg,24,pTarg,24);
AddDigits(pArr,40,pTarg,24,pTarg,24);

//Fixing digit25
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
AddDigits(pArr,36,pTarg,25,pTarg,25);
AddDigits(pArr,37,pTarg,25,pTarg,25);
AddDigits(pArr,39,pTarg,25,pTarg,25);
AddDigits(pArr,40,pTarg,25,pTarg,25);

//Fixing digit26
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
AddDigits(pArr,36,pTarg,26,pTarg,26);
AddDigits(pArr,37,pTarg,26,pTarg,26);
AddDigits(pArr,38,pTarg,26,pTarg,26);
AddDigits(pArr,39,pTarg,26,pTarg,26);
AddDigits(pArr,40,pTarg,26,pTarg,26);

//Fixing digit27
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;
AddDigits(pArr,36,pTarg,27,pTarg,27);
AddDigits(pArr,37,pTarg,27,pTarg,27);
AddDigits(pArr,38,pTarg,27,pTarg,27);

//Fixing digit28
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;
AddDigits(pArr,42,pTarg,28,pTarg,28);
AddDigits(pArr,43,pTarg,28,pTarg,28);
AddDigits(pArr,45,pTarg,28,pTarg,28);
AddDigits(pArr,47,pTarg,28,pTarg,28);

//Fixing digit29
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;
AddDigits(pArr,42,pTarg,29,pTarg,29);
```

```
AddDigits(pArr,43,pTarg,29,pTarg,29);
AddDigits(pArr,45,pTarg,29,pTarg,29);
AddDigits(pArr,46,pTarg,29,pTarg,29);

//Fixing digit30
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
AddDigits(pArr,42,pTarg,30,pTarg,30);
AddDigits(pArr,44,pTarg,30,pTarg,30);
AddDigits(pArr,46,pTarg,30,pTarg,30);
AddDigits(pArr,47,pTarg,30,pTarg,30);

//Fixing digit31
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
AddDigits(pArr,43,pTarg,31,pTarg,31);
AddDigits(pArr,44,pTarg,31,pTarg,31);
AddDigits(pArr,46,pTarg,31,pTarg,31);
AddDigits(pArr,47,pTarg,31,pTarg,31);
}

void Round13(digit *pArr,digit *pTarg)
{
//Fixing digit0
AddDigits(pArr,0,pTarg,0,pTarg,0);
AddDigits(pArr,1,pTarg,0,pTarg,0);
AddDigits(pArr,2,pTarg,0,pTarg,0);
AddDigits(pArr,4,pTarg,0,pTarg,0);
AddDigits(pArr,5,pTarg,0,pTarg,0);

//Fixing digit1
AddDigits(pArr,1,pTarg,1,pTarg,1);
AddDigits(pArr,2,pTarg,1,pTarg,1);
AddDigits(pArr,4,pTarg,1,pTarg,1);
AddDigits(pArr,5,pTarg,1,pTarg,1);

//Fixing digit2
AddDigits(pArr,0,pTarg,2,pTarg,2);
AddDigits(pArr,1,pTarg,2,pTarg,2);
AddDigits(pArr,2,pTarg,2,pTarg,2);
AddDigits(pArr,3,pTarg,2,pTarg,2);
AddDigits(pArr,4,pTarg,2,pTarg,2);
AddDigits(pArr,5,pTarg,2,pTarg,2);

//Fixing digit3
AddDigits(pArr,0,pTarg,3,pTarg,3);
AddDigits(pArr,1,pTarg,3,pTarg,3);
AddDigits(pArr,2,pTarg,3,pTarg,3);
AddDigits(pArr,3,pTarg,3,pTarg,3);
AddDigits(pArr,4,pTarg,3,pTarg,3);
AddDigits(pArr,5,pTarg,3,pTarg,3);

//Fixing digit4
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
pTarg[4].variable[0]=pTarg[4].variable[0]^Ace;
AddDigits(pArr,7,pTarg,4,pTarg,4);
AddDigits(pArr,8,pTarg,4,pTarg,4);
AddDigits(pArr,9,pTarg,4,pTarg,4);
```

```
AddDigits(pArr,10,pTarg,4,pTarg,4);
AddDigits(pArr,11,pTarg,4,pTarg,4);

//Fixing digit5
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
pTarg[5].variable[0]=pTarg[5].variable[0]^Ace;
AddDigits(pArr,6,pTarg,5,pTarg,5);
AddDigits(pArr,9,pTarg,5,pTarg,5);
AddDigits(pArr,10,pTarg,5,pTarg,5);

//Fixing digit6
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
pTarg[6].variable[0]=pTarg[6].variable[0]^Ace;
AddDigits(pArr,6,pTarg,6,pTarg,6);
AddDigits(pArr,7,pTarg,6,pTarg,6);
AddDigits(pArr,8,pTarg,6,pTarg,6);
AddDigits(pArr,9,pTarg,6,pTarg,6);
AddDigits(pArr,10,pTarg,6,pTarg,6);

//Fixing digit7
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;
pTarg[7].variable[0]=pTarg[7].variable[0]^Ace;
AddDigits(pArr,7,pTarg,7,pTarg,7);
AddDigits(pArr,9,pTarg,7,pTarg,7);
AddDigits(pArr,10,pTarg,7,pTarg,7);
AddDigits(pArr,11,pTarg,7,pTarg,7);

//Fixing digit8
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
pTarg[8].variable[0]=pTarg[8].variable[0]^Ace;
AddDigits(pArr,12,pTarg,8,pTarg,8);
AddDigits(pArr,13,pTarg,8,pTarg,8);
AddDigits(pArr,15,pTarg,8,pTarg,8);
AddDigits(pArr,16,pTarg,8,pTarg,8);
AddDigits(pArr,17,pTarg,8,pTarg,8);

//Fixing digit9
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
pTarg[9].variable[0]=pTarg[9].variable[0]^Ace;
AddDigits(pArr,15,pTarg,9,pTarg,9);
AddDigits(pArr,16,pTarg,9,pTarg,9);

//Fixing digit10
pTarg[10].variable[0]=pTarg[10].variable[0]^Ace;
AddDigits(pArr,12,pTarg,10,pTarg,10);
AddDigits(pArr,13,pTarg,10,pTarg,10);
AddDigits(pArr,14,pTarg,10,pTarg,10);
AddDigits(pArr,15,pTarg,10,pTarg,10);
AddDigits(pArr,16,pTarg,10,pTarg,10);
AddDigits(pArr,17,pTarg,10,pTarg,10);

//Fixing digit11
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
pTarg[11].variable[0]=pTarg[11].variable[0]^Ace;
AddDigits(pArr,13,pTarg,11,pTarg,11);
```

```
AddDigits(pArr,15,pTarg,11,pTarg,11);

//Fixing digit12
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
pTarg[12].variable[0]=pTarg[12].variable[0]^Ace;
AddDigits(pArr,19,pTarg,12,pTarg,12);
AddDigits(pArr,20,pTarg,12,pTarg,12);
AddDigits(pArr,21,pTarg,12,pTarg,12);
AddDigits(pArr,22,pTarg,12,pTarg,12);
AddDigits(pArr,23,pTarg,12,pTarg,12);

//Fixing digit13
pTarg[13].variable[0]=pTarg[13].variable[0]^Ace;
AddDigits(pArr,18,pTarg,13,pTarg,13);
AddDigits(pArr,19,pTarg,13,pTarg,13);
AddDigits(pArr,21,pTarg,13,pTarg,13);
AddDigits(pArr,22,pTarg,13,pTarg,13);
AddDigits(pArr,23,pTarg,13,pTarg,13);

//Fixing digit14
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;
pTarg[14].variable[0]=pTarg[14].variable[0]^Ace;
AddDigits(pArr,19,pTarg,14,pTarg,14);
AddDigits(pArr,20,pTarg,14,pTarg,14);
AddDigits(pArr,21,pTarg,14,pTarg,14);
AddDigits(pArr,22,pTarg,14,pTarg,14);

//Fixing digit15
pTarg[15].variable[0]=pTarg[15].variable[0]^Ace;
AddDigits(pArr,19,pTarg,15,pTarg,15);
AddDigits(pArr,21,pTarg,15,pTarg,15);
AddDigits(pArr,23,pTarg,15,pTarg,15);

//Fixing digit16
pTarg[16].variable[0]=pTarg[16].variable[0]^Ace;
AddDigits(pArr,24,pTarg,16,pTarg,16);
AddDigits(pArr,25,pTarg,16,pTarg,16);
AddDigits(pArr,26,pTarg,16,pTarg,16);
AddDigits(pArr,28,pTarg,16,pTarg,16);

//Fixing digit17
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;
pTarg[17].variable[0]=pTarg[17].variable[0]^Ace;
AddDigits(pArr,25,pTarg,17,pTarg,17);
AddDigits(pArr,26,pTarg,17,pTarg,17);
AddDigits(pArr,28,pTarg,17,pTarg,17);
AddDigits(pArr,29,pTarg,17,pTarg,17);

//Fixing digit18
pTarg[18].variable[0]=pTarg[18].variable[0]^Ace;
AddDigits(pArr,24,pTarg,18,pTarg,18);
AddDigits(pArr,25,pTarg,18,pTarg,18);
AddDigits(pArr,26,pTarg,18,pTarg,18);
AddDigits(pArr,27,pTarg,18,pTarg,18);
AddDigits(pArr,28,pTarg,18,pTarg,18);
```

```
//Fixing digit19
pTarg[19].variable[0]=pTarg[19].variable[0]^Ace;
AddDigits(pArr,24,pTarg,19,pTarg,19);
AddDigits(pArr,25,pTarg,19,pTarg,19);
AddDigits(pArr,26,pTarg,19,pTarg,19);
AddDigits(pArr,27,pTarg,19,pTarg,19);
AddDigits(pArr,28,pTarg,19,pTarg,19);

//Fixing digit20
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;
pTarg[20].variable[0]=pTarg[20].variable[0]^Ace;
AddDigits(pArr,31,pTarg,20,pTarg,20);
AddDigits(pArr,32,pTarg,20,pTarg,20);
AddDigits(pArr,33,pTarg,20,pTarg,20);
AddDigits(pArr,34,pTarg,20,pTarg,20);
AddDigits(pArr,35,pTarg,20,pTarg,20);

//Fixing digit21
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
pTarg[21].variable[0]=pTarg[21].variable[0]^Ace;
AddDigits(pArr,30,pTarg,21,pTarg,21);
AddDigits(pArr,31,pTarg,21,pTarg,21);
AddDigits(pArr,33,pTarg,21,pTarg,21);
AddDigits(pArr,34,pTarg,21,pTarg,21);

//Fixing digit22
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
pTarg[22].variable[0]=pTarg[22].variable[0]^Ace;
AddDigits(pArr,30,pTarg,22,pTarg,22);
AddDigits(pArr,31,pTarg,22,pTarg,22);
AddDigits(pArr,32,pTarg,22,pTarg,22);
AddDigits(pArr,33,pTarg,22,pTarg,22);
AddDigits(pArr,34,pTarg,22,pTarg,22);
AddDigits(pArr,35,pTarg,22,pTarg,22);

//Fixing digit23
pTarg[23].variable[0]=pTarg[23].variable[0]^Ace;
AddDigits(pArr,31,pTarg,23,pTarg,23);
AddDigits(pArr,33,pTarg,23,pTarg,23);
AddDigits(pArr,35,pTarg,23,pTarg,23);

//Fixing digit24
pTarg[24].variable[0]=pTarg[24].variable[0]^Ace;
AddDigits(pArr,36,pTarg,24,pTarg,24);
AddDigits(pArr,37,pTarg,24,pTarg,24);
AddDigits(pArr,38,pTarg,24,pTarg,24);
AddDigits(pArr,40,pTarg,24,pTarg,24);

//Fixing digit25
pTarg[25].variable[0]=pTarg[25].variable[0]^Ace;
AddDigits(pArr,36,pTarg,25,pTarg,25);
AddDigits(pArr,37,pTarg,25,pTarg,25);
AddDigits(pArr,39,pTarg,25,pTarg,25);
AddDigits(pArr,40,pTarg,25,pTarg,25);

//Fixing digit26
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
```

```
pTarg[26].variable[0]=pTarg[26].variable[0]^Ace;
AddDigits(pArr,37,pTarg,26,pTarg,26);
AddDigits(pArr,38,pTarg,26,pTarg,26);
AddDigits(pArr,40,pTarg,26,pTarg,26);
AddDigits(pArr,41,pTarg,26,pTarg,26);

//Fixing digit27
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;
pTarg[27].variable[0]=pTarg[27].variable[0]^Ace;
AddDigits(pArr,36,pTarg,27,pTarg,27);
AddDigits(pArr,37,pTarg,27,pTarg,27);
AddDigits(pArr,39,pTarg,27,pTarg,27);
AddDigits(pArr,41,pTarg,27,pTarg,27);

//Fixing digit28
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;
pTarg[28].variable[0]=pTarg[28].variable[0]^Ace;
AddDigits(pArr,42,pTarg,28,pTarg,28);
AddDigits(pArr,43,pTarg,28,pTarg,28);
AddDigits(pArr,44,pTarg,28,pTarg,28);
AddDigits(pArr,45,pTarg,28,pTarg,28);
AddDigits(pArr,46,pTarg,28,pTarg,28);
AddDigits(pArr,47,pTarg,28,pTarg,28);

//Fixing digit29
pTarg[29].variable[0]=pTarg[29].variable[0]^Ace;
AddDigits(pArr,42,pTarg,29,pTarg,29);
AddDigits(pArr,43,pTarg,29,pTarg,29);
AddDigits(pArr,45,pTarg,29,pTarg,29);
AddDigits(pArr,46,pTarg,29,pTarg,29);

//Fixing digit30
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
pTarg[30].variable[0]=pTarg[30].variable[0]^Ace;
AddDigits(pArr,42,pTarg,30,pTarg,30);
AddDigits(pArr,44,pTarg,30,pTarg,30);
AddDigits(pArr,46,pTarg,30,pTarg,30);
AddDigits(pArr,47,pTarg,30,pTarg,30);

//Fixing digit31
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
pTarg[31].variable[0]=pTarg[31].variable[0]^Ace;
AddDigits(pArr,43,pTarg,31,pTarg,31);
AddDigits(pArr,45,pTarg,31,pTarg,31);
AddDigits(pArr,47,pTarg,31,pTarg,31);
}

void find_digits(digit *pArr,digit *pTarg)
{
    int i,j;

    //Dump all digits
    for(i=0;i<32;i++)
      for(j=0;j<65;j++)
```

```
            pTarg[i].variable[j]=Zero;
        //call whichever round of equations you want
        Round10(pArr,pTarg);
        for(i=0;i<4;i++)
            for(j=1;j<65;j++)
                pTarg[i].variable[j]=pTarg[i].variable[j]^Ace;

}
```

system_creator.h

```
/***************************************************************************
*****
                          system_creator.h  -  description
                             -------------------
    begin                : Tue Feb 24 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

***************************************************************************
****/

/***************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************
****/

extern void systemcreator(bool *text2encr,digit *CipherText,bool
*known_bits,bool *bits2use);
extern void systemcreator_decr(bool *ciphertext,digit *Plaintext,bool
*known_bits,bool *bits2use);
```

system_creator.cpp

```
/***************************************************************************
*****
                          system_creator.cpp  -  description
                             -------------------
    begin                : Tue Feb 24 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

***************************************************************************
****/

/***************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************
****/

#define Ace true
#define Zero false

#include <arbitrary.h>
#include <standardDES.h>

using namespace std;
//tries to implement encryption with an arbitrary key
void systemcreator(bool *text2encr,digit *CipherText,bool
*known_bits,bool *bits2use)
{
    digit Knew[56];
    digit text2encrNew[64];
    digit Ci[28],Di[28];
    digit oldLeft[32],oldRight[32],tmpLeft[32],tmpRight[32];
    digit EA[48];
    digit tmpK[56],tmp[48];
    digit strfinal[32],finalstrSboxed[32];
    digit origK[64];
    int i,r,k;

    //set the original arbitrary key

    for(r=0;r<64;r++)
```

```
    {
        text2encrNew[r].variable[0]=text2encr[r];
        for(i=0;i<65;i++)
        {
          if(r+1==i && r%8!=7) origK[r].variable[i]=Ace;
          else origK[r].variable[i]=Zero ;
          if(i>0) text2encrNew[r].variable[i]=Zero;
        }
    }
    //now make the substitution of the known bits
    //with their known values dump the useless bits
    //these are always the last in each octet so as I count 0...7 its
always the 7th
    for(r=0;r<64;r++)
    {
        if(bits2use[r]==Ace || r%8==7)
        {
            origK[r].variable[r+1]=Zero;
            origK[r].variable[0]=known_bits[r];
        }
    }

    IP_arb(text2encrNew);
    PC1_arb(origK,Knew);
    L28_arb(Knew,Ci);
    R28_arb(Knew,Di);
    L_arb(text2encrNew,oldLeft);
    R_arb(text2encrNew,oldRight);

    for (i=1;i<17;i++)
    {
        //tmpLeft=oldRight
        for(r=0;r<32;r++)
            for(k=0;k<65;k++)
                tmpLeft[r].variable[k]=oldRight[r].variable[k];

        ExpDES_arb(oldRight,EA);
        LS1_arb(Ci);
        LS1_arb(Di);

        if(i==1||i==2||i==9||i==16)
        {
          LS1_arb(Ci);
          LS1_arb(Di);
        }

        JoinTables_arb(Ci,Di,tmpK,28);

        PC2_arb(tmpK,Knew);
        AddTables_arb(EA,Knew,tmp,48);

        //emulate sboxes
        find_digits(tmp,finalstrSboxed);
        Perm_arb(finalstrSboxed,strfinal);

        AddTables_arb(oldLeft,strfinal, tmpRight,32);
        for(r=0;r<32;r++)
```

```
            for(k=0;k<65;k++)
            {
                oldRight[r].variable[k] = tmpRight[r].variable[k];
                oldLeft[r].variable[k]  = tmpLeft[r].variable[k];
            }

    }//end i loop

    JoinTables_arb(tmpRight,tmpLeft,text2encrNew,32);
    IPinv_arb(text2encrNew);

    for(r=0;r<64;r++)
        CipherText[r]=text2encrNew[r];

}//end systemcreator

//tries to implement decreption with an arbitrary key
void systemcreator_decr(bool *ciphertext,digit *Plaintext,bool
*known_bits,bool *bits2use)
{
    digit Knew[56];
    digit ciphertextNew[64];
    digit Ci[28],Di[28];
    digit oldLeft[32],oldRight[32],tmpLeft[32],tmpRight[32];
    digit EA[48];
    digit tmpK[56],tmp[48];
    digit strfinal[32],finalstrSboxed[32];
    digit origK[64];
    digit tempKey[16][48];
    int i,r,k;

    //set the original arbitrary key
    for(r=0;r<64;r++)
    {
        ciphertextNew[r].variable[0]=ciphertext[r];
        for(i=0;i<65;i++)
        {
          if(r+1==i && r%8!=7) origK[r].variable[i]=Ace;
          else origK[r].variable[i]=Zero ;
          if(i>0) ciphertextNew[r].variable[i]=Zero;
        }
    }

    for(r=0;r<64;r++)
    {
        if(bits2use[r]==Ace || r%8==7)
        {
            origK[r].variable[r+1]=Zero;
            origK[r].variable[0]=known_bits[r];
        }
    }

    //create the keys
    PC1_arb(origK,Knew);
    L28_arb(Knew,Ci);
    R28_arb(Knew,Di);
    for (i=1;i<17;i++)
```

```
    {
        LS1_arb(Ci);
        LS1_arb(Di);
        if(i==1||i==2||i==9||i==16)
        {
          LS1_arb(Ci);
          LS1_arb(Di);
        }

        JoinTables_arb(Ci,Di,tmpK,28);
        PC2_arb(tmpK,Knew);

        for(k=0;k<48;k++)
            for(r=0;r<65;r++)
                tempKey[i][k].variable[r]=Knew[k].variable[r];
    }

    IP_arb(ciphertextNew);
    L_arb(ciphertextNew,oldLeft);
    R_arb(ciphertextNew,oldRight);

    for (i=1;i<17;i++)
    {
        //tmpLeft=oldRight
        for(r=0;r<32;r++)
            for(k=0;k<65;k++)
                tmpLeft[r].variable[k]=oldRight[r].variable[k];

        ExpDES_arb(oldRight,EA);

        for(k=0;k<48;k++)
            for(r=0;r<65;r++)
                Knew[k].variable[r]=tempKey[16-i][k].variable[r];

        AddTables_arb(EA,Knew,tmp,48);

        //emulate sboxes
        find_digits(tmp,finalstrSboxed);
        Perm_arb(finalstrSboxed,strfinal);

        AddTables_arb(oldLeft,strfinal, tmpRight,32);
        for(r=0;r<32;r++)
            for(k=0;k<65;k++)
            {
                oldRight[r].variable[k] = tmpRight[r].variable[k];
                oldLeft[r].variable[k]  = tmpLeft[r].variable[k];
            }

    }//end i loop

    JoinTables_arb(tmpRight,tmpLeft,ciphertextNew,32);
    IPinv_arb(ciphertextNew);

    for(r=0;r<64;r++)
        Plaintext[r]=ciphertextNew[r];

}//end systemcreator_decr
```

standardDES.h

```
/***************************************************************************
*****
                          standardDES.h  -  description
                             -------------------
    begin                : Tue Feb 24 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

***************************************************************************
****/


extern void conv2binary(int n,bool *TargArray);
extern void loadSboxes(void);
extern void EncrDes(bool *text2encr,bool *origK,bool *CipherText);
extern void DecrDes(bool *text2encr,bool *origK,bool *CipherText);
```

standardDES.cpp

```
/*************************************************************************
*****
                          standardDES.cpp  -  description

    Here are all the functions that are needed in order to encrypt
using DES.
    The only so far supported mode is ECB.
                             -------------------
    begin                : Tue Feb 24 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

*************************************************************************
****/

/*************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

*************************************************************************
****/



#include <fstream>
//needed for random number generation
#include <ctime>
//local used libraries
#include <arbitrary.h>
#include <stdlib.h>
#include <iostream>

#define WIDTH 64 //64=16*4 In each Sbox we have 16 elements, each of
these takes 4bits, Totally 4*16
#define HEIGHT 32 //We have 32 lines totally for all S-boxes
#define maxlen_words 512   //512

#define Zero false
#define Ace true

using namespace std;
```

```
//The table below contains all the Sbox values
int Sbox[HEIGHT][WIDTH] ;

void conv2binary(int n,bool *TargArray)
{

    TargArray[0]=n/128;     n=n%128;    TargArray[1]=n/64;  n=n%64;
    TargArray[2]=n/32;      n=n%32;     TargArray[3]=n/16;  n=n%16;
    TargArray[4]=n/8;       n=n%8;      TargArray[5]=n/4;   n=n%4;
    TargArray[6]=n/2;       n=n%2;      TargArray[7]=n;


}//end of conv2binary function

void IP(bool *pArray)
{

    bool pTargArray[64];
    int i;

    pTargArray[0]=pArray[57];       pTargArray[1]=pArray[49];
pTargArray[2]=pArray[41];
    pTargArray[3]=pArray[33];       pTargArray[4]=pArray[25];
pTargArray[5]=pArray[17];
    pTargArray[6]=pArray[9];        pTargArray[7]=pArray[1];
pTargArray[8]=pArray[59];
    pTargArray[9]=pArray[51];       pTargArray[10]=pArray[43];
pTargArray[11]=pArray[35];
    pTargArray[12]=pArray[27];      pTargArray[13]=pArray[19];
pTargArray[14]=pArray[11];
    pTargArray[15]=pArray[3];       pTargArray[16]=pArray[61];
pTargArray[17]=pArray[53];
    pTargArray[18]=pArray[45];      pTargArray[19]=pArray[37];
pTargArray[20]=pArray[29];
    pTargArray[21]=pArray[21];      pTargArray[22]=pArray[13];
pTargArray[23]=pArray[5];
    pTargArray[24]=pArray[63];      pTargArray[25]=pArray[55];
pTargArray[26]=pArray[47];
    pTargArray[27]=pArray[39];      pTargArray[28]=pArray[31];
pTargArray[29]=pArray[23];
    pTargArray[30]=pArray[15];      pTargArray[31]=pArray[7];
pTargArray[32]=pArray[56];
    pTargArray[33]=pArray[48];      pTargArray[34]=pArray[40];
pTargArray[35]=pArray[32];
    pTargArray[36]=pArray[24];      pTargArray[37]=pArray[16];
pTargArray[38]=pArray[8];
    pTargArray[39]=pArray[0];       pTargArray[40]=pArray[58];
pTargArray[41]=pArray[50];
    pTargArray[42]=pArray[42];      pTargArray[43]=pArray[34];
pTargArray[44]=pArray[26];
    pTargArray[45]=pArray[18];      pTargArray[46]=pArray[10];
pTargArray[47]=pArray[2];
    pTargArray[48]=pArray[60];      pTargArray[49]=pArray[52];
pTargArray[50]=pArray[44];
    pTargArray[51]=pArray[36];      pTargArray[52]=pArray[28];
pTargArray[53]=pArray[20];
```

```
    pTargArray[54]=pArray[12];      pTargArray[55]=pArray[4];
pTargArray[56]=pArray[62];
    pTargArray[57]=pArray[54];      pTargArray[58]=pArray[46];
pTargArray[59]=pArray[38];
    pTargArray[60]=pArray[30];      pTargArray[61]=pArray[22];
pTargArray[62]=pArray[14];
    pTargArray[63]=pArray[6];

    for(i=0;i<64;i++)
        pArray[i]=pTargArray[i];
}


void IPinv(bool *pArray)
{
    bool pTargArray[64];
    int i;
    pTargArray[0]=pArray[39];       pTargArray[1]=pArray[7];
pTargArray[2]=pArray[47];
    pTargArray[3]=pArray[15];       pTargArray[4]=pArray[55];
pTargArray[5]=pArray[23];
    pTargArray[6]=pArray[63];       pTargArray[7]=pArray[31];
pTargArray[8]=pArray[38];
    pTargArray[9]=pArray[6];        pTargArray[10]=pArray[46];
pTargArray[11]=pArray[14];
    pTargArray[12]=pArray[54];      pTargArray[13]=pArray[22];
pTargArray[14]=pArray[62];
    pTargArray[15]=pArray[30];      pTargArray[16]=pArray[37];
pTargArray[17]=pArray[5];
    pTargArray[18]=pArray[45];      pTargArray[19]=pArray[13];
pTargArray[20]=pArray[53];
    pTargArray[21]=pArray[21];      pTargArray[22]=pArray[61];
pTargArray[23]=pArray[29];
    pTargArray[24]=pArray[36];      pTargArray[25]=pArray[4];
pTargArray[26]=pArray[44];
    pTargArray[27]=pArray[12];      pTargArray[28]=pArray[52];
pTargArray[29]=pArray[20];
    pTargArray[30]=pArray[60];      pTargArray[31]=pArray[28];
pTargArray[32]=pArray[35];
    pTargArray[33]=pArray[3];       pTargArray[34]=pArray[43];
pTargArray[35]=pArray[11];
    pTargArray[36]=pArray[51];      pTargArray[37]=pArray[19];
pTargArray[38]=pArray[59];
    pTargArray[39]=pArray[27];      pTargArray[40]=pArray[34];
pTargArray[41]=pArray[2];
    pTargArray[42]=pArray[42];      pTargArray[43]=pArray[10];
pTargArray[44]=pArray[50];
    pTargArray[45]=pArray[18];      pTargArray[46]=pArray[58];
pTargArray[47]=pArray[26];
    pTargArray[48]=pArray[33];      pTargArray[49]=pArray[1];
pTargArray[50]=pArray[41];
    pTargArray[51]=pArray[9];       pTargArray[52]=pArray[49];
pTargArray[53]=pArray[17];
    pTargArray[54]=pArray[57];      pTargArray[55]=pArray[25];
pTargArray[56]=pArray[32];
    pTargArray[57]=pArray[0];       pTargArray[58]=pArray[40];
pTargArray[59]=pArray[8];
```

```
    pTargArray[60]=pArray[48];      pTargArray[61]=pArray[16];
pTargArray[62]=pArray[56];
    pTargArray[63]=pArray[24];

    for(i=0;i<65;i++)
        pArray[i]=pTargArray[i];

}

void ExpDES(bool *pArray,bool *pTargArray)
{

    pTargArray[0]=pArray[31];   pTargArray[1]=pArray[0];
pTargArray[2]=pArray[1];
    pTargArray[3]=pArray[2];    pTargArray[4]=pArray[3];
pTargArray[5]=pArray[4];
    pTargArray[6]=pArray[3];    pTargArray[7]=pArray[4];
pTargArray[8]=pArray[5];
    pTargArray[9]=pArray[6];    pTargArray[10]=pArray[7];
pTargArray[11]=pArray[8];
    pTargArray[12]=pArray[7];   pTargArray[13]=pArray[8];
pTargArray[14]=pArray[9];
    pTargArray[15]=pArray[10];  pTargArray[16]=pArray[11];
pTargArray[17]=pArray[12];
    pTargArray[18]=pArray[11];  pTargArray[19]=pArray[12];
pTargArray[20]=pArray[13];
    pTargArray[21]=pArray[14];  pTargArray[22]=pArray[15];
pTargArray[23]=pArray[16];
    pTargArray[24]=pArray[15];  pTargArray[25]=pArray[16];
pTargArray[26]=pArray[17];
    pTargArray[27]=pArray[18];  pTargArray[28]=pArray[19];
pTargArray[29]=pArray[20];
    pTargArray[30]=pArray[19];  pTargArray[31]=pArray[20];
pTargArray[32]=pArray[21];
    pTargArray[33]=pArray[22];  pTargArray[34]=pArray[23];
pTargArray[35]=pArray[24];
    pTargArray[36]=pArray[23];  pTargArray[37]=pArray[24];
pTargArray[38]=pArray[25];
    pTargArray[39]=pArray[26];  pTargArray[40]=pArray[27];
pTargArray[41]=pArray[28];
    pTargArray[42]=pArray[27];  pTargArray[43]=pArray[28];
pTargArray[44]=pArray[29];
    pTargArray[45]=pArray[30];  pTargArray[46]=pArray[31];
pTargArray[47]=pArray[0];
}

void PC1(bool *pArray,bool *pTargArray)
{

     //input 64bits
     //output 56bits

    pTargArray[0]=pArray[56];       pTargArray[1]=pArray[48];
pTargArray[2]=pArray[40];
    pTargArray[3]=pArray[32];       pTargArray[4]=pArray[24];
pTargArray[5]=pArray[16];
```

```
    pTargArray[6]=pArray[8];        pTargArray[7]=pArray[0];
pTargArray[8]=pArray[57];
    pTargArray[9]=pArray[49];       pTargArray[10]=pArray[41];
pTargArray[11]=pArray[33];
    pTargArray[12]=pArray[25];      pTargArray[13]=pArray[17];
pTargArray[14]=pArray[9];
    pTargArray[15]=pArray[1];       pTargArray[16]=pArray[58];
pTargArray[17]=pArray[50];
    pTargArray[18]=pArray[42];      pTargArray[19]=pArray[34];
pTargArray[20]=pArray[26];
    pTargArray[21]=pArray[18];      pTargArray[22]=pArray[10];
pTargArray[23]=pArray[2];
    pTargArray[24]=pArray[59];      pTargArray[25]=pArray[51];
pTargArray[26]=pArray[43];
    pTargArray[27]=pArray[35];      pTargArray[28]=pArray[62];
pTargArray[29]=pArray[54];
    pTargArray[30]=pArray[46];      pTargArray[31]=pArray[38];
pTargArray[32]=pArray[30];
    pTargArray[33]=pArray[22];      pTargArray[34]=pArray[14];
pTargArray[35]=pArray[6];
    pTargArray[36]=pArray[61];      pTargArray[37]=pArray[53];
pTargArray[38]=pArray[45];
    pTargArray[39]=pArray[37];      pTargArray[40]=pArray[29];
pTargArray[41]=pArray[21];
    pTargArray[42]=pArray[13];      pTargArray[43]=pArray[5];
pTargArray[44]=pArray[60];
    pTargArray[45]=pArray[52];      pTargArray[46]=pArray[44];
pTargArray[47]=pArray[36];
    pTargArray[48]=pArray[28];      pTargArray[49]=pArray[20];
pTargArray[50]=pArray[12];
    pTargArray[51]=pArray[4];       pTargArray[52]=pArray[27];
pTargArray[53]=pArray[19];
    pTargArray[54]=pArray[11];      pTargArray[55]=pArray[3];

}

void PC2(bool *pArray,bool *pTargArray)
{

     //input 56bits
     //output 48bits

    pTargArray[0]=pArray[13];       pTargArray[1]=pArray[16];
pTargArray[2]=pArray[10];
    pTargArray[3]=pArray[23];       pTargArray[4]=pArray[0];
pTargArray[5]=pArray[4];
    pTargArray[6]=pArray[2];        pTargArray[7]=pArray[27];
pTargArray[8]=pArray[14];
    pTargArray[9]=pArray[5];        pTargArray[10]=pArray[20];
pTargArray[11]=pArray[9];
    pTargArray[12]=pArray[22];      pTargArray[13]=pArray[18];
pTargArray[14]=pArray[11];
    pTargArray[15]=pArray[3];       pTargArray[16]=pArray[25];
pTargArray[17]=pArray[7];
    pTargArray[18]=pArray[15];      pTargArray[19]=pArray[6];
pTargArray[20]=pArray[26];
```

```
    pTargArray[21]=pArray[19];      pTargArray[22]=pArray[12];
pTargArray[23]=pArray[1];
    pTargArray[24]=pArray[40];      pTargArray[25]=pArray[51];
pTargArray[26]=pArray[30];
    pTargArray[27]=pArray[36];      pTargArray[28]=pArray[46];
pTargArray[29]=pArray[54];
    pTargArray[30]=pArray[29];      pTargArray[31]=pArray[39];
pTargArray[32]=pArray[50];
    pTargArray[33]=pArray[44];      pTargArray[34]=pArray[32];
pTargArray[35]=pArray[47];
    pTargArray[36]=pArray[43];      pTargArray[37]=pArray[48];
pTargArray[38]=pArray[38];
    pTargArray[39]=pArray[55];      pTargArray[40]=pArray[33];
pTargArray[41]=pArray[52];
    pTargArray[42]=pArray[45];      pTargArray[43]=pArray[41];
pTargArray[44]=pArray[49];
    pTargArray[45]=pArray[35];      pTargArray[46]=pArray[28];
pTargArray[47]=pArray[31];

}

void LS1(bool *pArray)
{
    bool pTargArray[28];
    int i;
    for(i=0;i<27;i++)
        pTargArray[i]=pArray[i+1];
    pTargArray[27]=pArray[0];

    for(i=0;i<28;i++)
        pArray[i]=pTargArray[i];
 }



void L(bool *pArray,bool *pTargArray)
{
//input 64bits
//output 32bits

    int i;

    for(i=0;i<32;i++)
        pTargArray[i]=pArray[i];
}

void R(bool *pArray,bool *pTargArray)
{
//input 64bits
//output 32bits

    int i;

    for(i=0;i<32;i++)
        pTargArray[i]=pArray[i+32];

}
```

```
void Perm(bool *pArray,bool *pTargArray)
{

    int i;

    pTargArray[0]=pArray[15];       pTargArray[1]=pArray[6];
pTargArray[2]=pArray[19];
    pTargArray[3]=pArray[20];       pTargArray[4]=pArray[28];
pTargArray[5]=pArray[11];
    pTargArray[6]=pArray[27];       pTargArray[7]=pArray[16];
pTargArray[8]=pArray[0];
    pTargArray[9]=pArray[14];       pTargArray[10]=pArray[22];
pTargArray[11]=pArray[25];
    pTargArray[12]=pArray[4];       pTargArray[13]=pArray[17];
pTargArray[14]=pArray[30];
    pTargArray[15]=pArray[9];       pTargArray[16]=pArray[1];
pTargArray[17]=pArray[7];
    pTargArray[18]=pArray[23];      pTargArray[19]=pArray[13];
pTargArray[20]=pArray[31];
    pTargArray[21]=pArray[26];      pTargArray[22]=pArray[2];
pTargArray[23]=pArray[8];
    pTargArray[24]=pArray[18];      pTargArray[25]=pArray[12];
pTargArray[26]=pArray[29];
    pTargArray[27]=pArray[5];       pTargArray[28]=pArray[21];
pTargArray[29]=pArray[10];
    pTargArray[30]=pArray[3];       pTargArray[31]=pArray[24];

    for(i=0;i<32;i++)
        pArray[i]=pTargArray[i];
}

void L28(bool *pArray,bool *pTargArray)
{
    int i;
    for(i=0;i<28;i++)
        pTargArray[i]=pArray[i];
}

void R28(bool *pArray,bool *pTargArray)
{
    int i;
    for(i=0;i<28;i++)
        pTargArray[i]=pArray[i+28];
}

// Joins two tables of the same length
// and the output is stored in the target table.
// Have to parse length of the original tables
void JoinTables(bool *table1,bool *table2,bool *target_table,int count)
{
    int i;
    for(i=0;i<count;i++)
    {
        target_table[i]=table1[i];
        target_table[i+count]=table2[i];
    }
```

```
}

//This function adds two tables modulo a number
void AddTables(bool *table1,bool *table2,bool *target_table,int count)
{
    int i;
    for(i=0;i<count;i++)
            target_table[i]=table1[i]^table2[i];
}

void findSboxValue(bool *digits,bool *targetArray,int k)
{
    //The first 3 binary digits tell me which sbox to refer to.
    int target_row,target_col;

    target_row=4*k+2*digits[0]+digits[5];

target_col=(8*digits[1]+4*digits[2]+2*digits[3]+digits[4])*4;//every
entry takes 4bits

    targetArray[0]=Sbox[target_row][target_col];
    targetArray[1]=Sbox[target_row][target_col+1];
    targetArray[2]=Sbox[target_row][target_col+2];
    targetArray[3]=Sbox[target_row][target_col+3];

}//end of findSboxValue

void loadSboxes(void)
{
    ifstream sbox_File;
    int i,j;
    char char_read;
    //open the file that contains the Sboxes
    sbox_File.open("./sboxesbin.txt", ios_base::in );

    i=j=0;

    //Assign the values of the Sboxes to the table
    while (! sbox_File.eof() )
    {
        i=i%HEIGHT;
        j=j%WIDTH;
        sbox_File.get(char_read);
        if (char_read=='0' || char_read=='1')
        {
          Sbox[i][j]=(char_read=='1');
          j++;
        }
        if (j==WIDTH) i++;
    }
    sbox_File.close();
}   //end of loadSboxes

void EncrDes(bool *text2encr,bool *origK,bool *CipherText)
{
    bool Knew[56];
    bool text2encrNew[64];
```

```
bool Ci[28],Di[28];
bool oldLeft[32],oldRight[32],tmpLeft[32],tmpRight[32];
bool EA[48];
bool tmpK[56],tmp[48],SboxParse[6];
int i,k,r;
bool strSboxed[4];
bool strfinal[32],finalstrSboxed[32];

for(r=0;r<64;r++)
    text2encrNew[r]=text2encr[r];

IP(text2encrNew);
PC1(origK,Knew);
L28(Knew,Ci);
R28(Knew,Di);
L(text2encrNew,oldLeft);
R(text2encrNew,oldRight);

for (i=1;i<17;i++)
{
    //tmpLeft=oldRight
    for(r=0;r<32;r++)
        tmpLeft[r]=oldRight[r];
    ExpDES(oldRight,EA);
    LS1(Ci);
    LS1(Di);
    if(i==1||i==2||i==9||i==16)
    {
      LS1(Ci);
      LS1(Di);
    }

    JoinTables(Ci,Di,tmpK,28);
    PC2(tmpK,Knew);
    AddTables(EA,Knew,tmp,48);
    //for each 6 digits of tmp get their sbox value
    for(k=0;k<8;k++)
    {
        for(r=0;r<6;r++) //parse the values to the proper SBOX
            SboxParse[r]=tmp[6*k+r];
        findSboxValue(SboxParse,strSboxed,k);
        for(r=0;r<4;r++)
            finalstrSboxed[4*k+r]=strSboxed[r];
    }
    Perm(finalstrSboxed,strfinal);
    AddTables(oldLeft,strfinal,tmpRight,32);

    for(r=0;r<32;r++)
    {
        oldRight[r]=tmpRight[r];
        oldLeft[r]=tmpLeft[r];
    }
}//end i loop
JoinTables(tmpRight,tmpLeft,text2encrNew,32);

IPinv(text2encrNew);
```

```
    for(r=0;r<64;r++)
        CipherText[r]=text2encrNew[r];

}

void DecrDes(bool *text2decr,bool *origK,bool *PlainText)
{
    bool Knew[56];
    bool text2decrNew[64];
    bool Ci[28],Di[28];
    bool oldLeft[32],oldRight[32],tmpLeft[32],tmpRight[32];
    bool EA[48];
    bool tmpK[56],tmp[56],SboxParse[6];
    int i,k,r;
    bool strSboxed[4];
    bool strfinal[32],finalstrSboxed[32];
    bool tmpKey[17][48];


    for(r=0;r<64;r++)
        text2decrNew[r]=text2decr[r];

    IP(text2decrNew);
    PC1(origK,Knew);
    L28(Knew,Ci);
    R28(Knew,Di);
    L(text2decrNew,oldLeft);
    R(text2decrNew,oldRight);
    //construct the keys
    for (i=1;i<17;i++)
    {
        LS1(Ci);
        LS1(Di);
        if(i==1||i==2||i==9||i==16)
        {
          LS1(Ci);
          LS1(Di);
        }

        JoinTables(Ci,Di,tmpK,28);
        PC2(tmpK,Knew);
        for(r=0;r<48;r++)
            tmpKey[i][r]=Knew[r];
    }

    for (i=1;i<17;i++)
    {
        //tmpLeft=oldRight
        for(r=0;r<32;r++)
            tmpLeft[r]=oldRight[r];
        ExpDES(oldRight,EA);

        for(r=0;r<48;r++)
            Knew[r]=tmpKey[17-i][r];
        AddTables(EA,Knew,tmp,48);
        //this is what the original DES is doing whith its SBOXES
        //for each 6 digits of tmp get their sbox value
```

```
        for(k=0;k<8;k++)
        {
            for(r=0;r<6;r++) //parse the values to the proper SBOX
                SboxParse[r]=tmp[6*k+r];
            findSboxValue(SboxParse,strSboxed,k);
            for(r=0;r<4;r++)
                finalstrSboxed[4*k+r]=strSboxed[r];
        }
        Perm(finalstrSboxed,strfinal);
        AddTables(oldLeft,strfinal,tmpRight,32);

        for(r=0;r<32;r++)
        {
            oldRight[r]=tmpRight[r];
            oldLeft[r]=tmpLeft[r];
        }
    }//end i loop
    JoinTables(tmpRight,tmpLeft,text2decrNew,32);
    IPinv(text2decrNew);

    for(r=0;r<64;r++)
        PlainText[r]=text2decrNew[r];
}
```

linear.h

```
/***************************************************************************
*****
                          linear.h  -  description
                             -------------------
    begin                : Sun Feb 22 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

    Here are all the functions that are needed to solve the linear
system
    the linear system is parsed as an array of 64 rows
    and 65 columns plus an extra column which is from the ciphertext.
    The first column as well as the one from the ciphertext are added
together
    modulo 2 in order to have the stable coefficients.


***************************************************************************
****/

/***************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************
****/

#define Rows 64
#define Cols 64

extern void l_solve(digit *CipherTextArb,bool *CipherText,bool
*probKey,bool *digits2Bused);
```

linear.cpp

```
/***************************************************************************
*****
                          linear.cpp  -  description
 Here is all the code that is used in order to solve the linear system
that is created.
                             -------------------
    begin                : Sun Feb 22 2004
    copyright            : (C) 2004 by C.Patsakis
    email                : kpatsak@unipi.gr

    Here are all the functions that are needed to solve the linear
system
    the linear system is parsed as an array of 64 rows
    and 65 columns plus an extra column which is from the ciphertext.
    The first column as well as the one from the ciphertext are added
together
    modulo 2 in order to have the stable coefficients.


***************************************************************************
****/

/***************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************
****/

#define Rows 64
#define Cols 64

#define Ace true
#define Zero false

#include <standardDES.h>

using namespace std;

struct digit
{
    bool variable[Cols+1];
```

```
    //first position for the stable coefficient and the rest for the
coefficients of the variables
};

bool systemArray[Rows][Cols+1];

//this function adds SourceRow to TargetRow  modulo 2
void addrows(int s, int t)//s-ource t-arget
{
    int i;
    for(i=0;i<Cols+1;i++)
        systemArray[t][i]=systemArray[t][i] ^ systemArray[s][i];
}

//swaps Row1 and Row2
void swaprows(int Row1 ,int Row2)
{
    int i;
    bool tmp[Cols+1];
    if (Row1!=Row2)
    {
        for(i=0;i<Cols+1;i++)
        {
            tmp[i]=systemArray[Row1][i];
            systemArray[Row1][i]=systemArray[Row2][i];
            systemArray[Row2][i]=tmp[i];
        }
    }
}

void l_solve(struct digit *CipherTextArb,bool *CipherText,bool
*probKey,bool *digits2Bused)
{
    //the linear system is supposed to be stored in the 64*65 integer
array
    int currRow=0;
    int currCol;
    int sortedRow=0;
    int i,j;
    int countAces,AcePosition;
    bool found;

    //fill the lines of the system carefull variable j is stored now
    //in j-1 position as the coefficient is placed last not first.
    for(i=0;i<Rows;i++)
    {
      for(j=1;j<Cols+1;j++)
        systemArray[i][j-1]=CipherTextArb[i].variable[j];
      //fix the last Column in order to be the sum
      //of the ciphertext and of the stable part.
      systemArray[i][Cols]=CipherTextArb[i].variable[0] ^
CipherText[i];
    }

   //this is the main linear solver
    for(currCol=0;currCol<Cols;currCol++)
    {
```

```
    //with this loop I find the first row that contains the first
    //non Zero element in the specified column
    found=false;
    currRow=sortedRow;
    while(currRow<Rows && found==false)
    {
        if (systemArray[currRow][currCol]==Ace) found=true;
        else currRow++;
    }
    if(found==true )
    {
      swaprows(currRow,sortedRow);

      for(i=0;i<Rows;i++)
      {
        if(i!=sortedRow && systemArray[i][currCol]==Ace)
          addrows(sortedRow,i);
      }
      sortedRow++;
    }
    //empty probKey, digits2Bused
    probKey[currCol]=Zero;
    digits2Bused[i]=false;
}
//Now I have to change some rows
//those rows have the form 00000(64times)1
//they will end up looking    00000(65times)
for(currRow=0;currRow<Rows;currRow++)
{
    currCol=0;
    found=false;
    while(currCol<Cols && found==false)
    {
      if (systemArray[currRow][currCol]==Ace ) found=true;
      currCol++;
    }
    if (found==false && systemArray[currRow][Cols]==Ace)
        systemArray[currRow][Cols]=Zero;
}

//create now the solution to be parsed
for(currRow=0;currRow<Rows;currRow++)
{
    countAces=0;
    AcePosition=-1;
    currCol=0;
    while (currCol<Cols && countAces<2)
    {
        if (systemArray[currRow][currCol]==Ace)
        {
          AcePosition=currCol;
          countAces++;
        }
        currCol++;
    }

    if (countAces==1)
```

```
        {
            probKey[AcePosition]=systemArray[currRow][Cols];
            digits2Bused[AcePosition]=true;
        }
    }
    //and here it ends...
}
```

Κώδικας για προσέγγιση S-Box

Sbox.cpp

```
/***************************************************************************
*****
                          main.cpp  -  description
      This program tries to find linear equations between the sboxes
                             -------------------
    begin                : Tue Dec  2 12:43:17 CET 2003
    copyright            : (C) 2003 by C.Patsakis
    email                : patsakis@unipi.gr

***************************************************************************
****/

/***************************************************************************
*****
 *
*
 *   This program is free software; you can redistribute it and/or
modify  *
 *   it under the terms of the GNU General Public License as published
by  *
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

***************************************************************************
****/


#include <iostream>
#include <stdlib.h>
#include <fstream>

#define ACE true
#define ZERO false

#define WIDTH 64 //64=16*4 In each Sbox we have 16 elements, each of
these takes 4bits, Totally 4*16
#define HEIGHT 32 //We have 32 lines totally for all S-boxes
#define lenwords 262144

//The table below contains all the Sbox values
bool Sbox[HEIGHT][WIDTH] ;

bool Binaries[262144][18];

using namespace std;
```

```
//This function loads the sboxes from a binary file and stores them in
memory can be used for other sboxes too.
void conv2binary(int n)
{     int k;
      k=n;
      Binaries[k][0]=n/131072;      n=n%131072;
      Binaries[k][1]=n/65536;       n=n%65536;
      Binaries[k][2]=n/32768;       n=n%32768;
      Binaries[k][3]=n/16384;       n=n%16384;
      Binaries[k][4]=n/8192;        n=n%8192;
      Binaries[k][5]=n/4096;        n=n%4096;
      Binaries[k][6]=n/2048;        n=n%2048;
      Binaries[k][7]=n/1024;        n=n%1024;
      Binaries[k][8]=n/512;         n=n%512;
      Binaries[k][9]=n/256;         n=n%256;
      Binaries[k][10]=n/128;        n=n%128;
      Binaries[k][11]=n/64;         n=n%64;
      Binaries[k][12]=n/32;         n=n%32;
      Binaries[k][13]=n/16;         n=n%16;
      Binaries[k][14]=n/8;          n=n%8;
      Binaries[k][15]=n/4;          n=n%4;
      Binaries[k][16]=n/2;          n=n%2;
      Binaries[k][17]=n;

}//end of conv2binary function

void loadBinaryRepr(void)
{
      long i;
      for(i=0;i<lenwords;i++)
            conv2binary(i);
      cout<<"Binary representations have been loaded.\n";
}
void loadSboxes(void)
{
    ifstream inFile;
    int i,j;
    char tmp_read;
    //open the file that contains the Sboxes
    inFile.open("./sboxesbin.txt", ios_base::in );

    i=j=0;

    //Assign the values of the Sboxes to the table
    while (! inFile.eof() )
   {
        i=i%HEIGHT;
        j=j%WIDTH;
        inFile.get(tmp_read);
        if (tmp_read=='0' || tmp_read=='1')
            {
                Sbox[i][j]=(tmp_read=='1');
                j++;
            }
        if (j==WIDTH) i++;
```

```
   }

 //Close the open file
    inFile.close();

    cout << "Sboxes are read \n";
}//end of loadSboxes

/*
This function takes an integer and returns a string which is its binary
representation.
All the empty cells are filled with zeroes. Since the biggest integer
that is going to be used is
512-1 the most efficient way to do it is the following.
*/


/*
This funtion takes as argument the sbox and the text to be parsed to
it.
Returns the output of the sbox.
Remember that all these values are stored in memory on an array.
Change according to the needs of the algorithm.
*/
void findSboxValue(bool *digits, bool *target)
{
    //The first 3 binary digits tell me which sbox to refer to.
    int target_row,target_col;
    int s;

    s=digits[2]*4+digits[1]*2+digits[0];

    target_row=s+2*digits[0]+digits[5];

target_col=(8*digits[1]+4*digits[2]+2*digits[3]+digits[4])*4;//every
entry takes 4bits

    target[0]=Sbox[target_row][target_col];
    target[1]=Sbox[target_row][target_col+1];
    target[2]=Sbox[target_row][target_col+2];
    target[3]=Sbox[target_row][target_col+3];

}//end of findSboxValue

//main program now starts
int main(void )/*int argc, char *argv[]*/
{
    /*
    We have found the proper equations from the previous program.
    We now have to find a better approximation, if possible.
    The better approximation will have the folowing form.
    A_i+x_k*word
    where A_i is the equations we have already calculated,
    x_k is a variable
    and word a random word of (Z_2)^9
    */
```

```
    //These counters count the hits of each equation for each bit.
      long tmpcnt[4];
      long i,j,t,k,r;
      bool tmp;

      bool tmpEquation[18],tmpresults[4];
      bool tmpstr[18];

    /*The table below contains the most propable equations
    first line are the equations, second are the least hits
    third line are the equations, fourth are the most hits
    The columns are the cipher-bits*/

      long results[4][4];

      loadSboxes();
      loadBinaryRepr();

      //assign starting values to the results array
      results[0][0]=results[0][1]=results[0][2]=results[0][3]=0;
      results[1][0]=results[1][1]=results[1][2]=results[1][3]=lenwords;
      results[2][0]=results[2][1]=results[2][2]=results[2][3]=0;
      results[3][0]=results[3][1]=results[3][2]=results[3][3]=0;

      /*
      This loop does the following takes one integer k and represents
it as binary
      for us this means that if  k=011101010...
      then the equation is   x2+x3+x4+x6+x8+...
      */

      //with this loop we take all the possible equations
      for (i=0;i<lenwords;i++ )    //512=2^9
      {
      //clear all the countrers that I store the hits
            tmpcnt[0]=tmpcnt[1]=tmpcnt[2]=tmpcnt[3]=0;

            //take the binary representation of the equation
            for(j=0;j<18;j++)
                  tmpEquation[j]=Binaries[i][j];
            //take now each different word
            for (j=0;j<lenwords;j++)
            {
                  //take the binary representation of the word
                  for(t=0;t<18;t++)
                        tmpstr[t]=Binaries[j][t];
                  //do nor ler the strings to be send to the same
sbox!!!
                  if(!(tmpstr[0]==tmpstr[9] && tmpstr[1]==tmpstr[10] &&
tmpstr[2]==tmpstr[11]))
                  {
                        tmp=ZERO;
                        //look what we have in the new equation and add
its value
                        for (k=0;k<18;k++)
                        {
```

```
                              if (tmpEquation[k]==ACE)
      tmp=tmp^tmpstr[k];
                        }//end k loop

                        findSboxValue(tmpstr,tmpresults);//find the
sbox value for these specific arguments

                        //check each c_i if there is any hit
                        for(r=0;r<4;r++)
                        {//if yes store a hit to the temporary array
for the specific entry
                              if (tmpresults[r]==tmp) tmpcnt[r]++;
                        }//end r loop
                  }//endif

            }//end j loop
            //check whether the hits of this equation should be stored
            for (r=0;r<4;r++)
            {
                  //check if more hits have been found
                  if (tmpcnt[r]>results[3][r])
                  {
                        results[3][r]=tmpcnt[r];            //store
the hits
                        results[2][r]=i;
//store the equation
                  }
                  //check now if less hits have been found
                  if (tmpcnt[r]<results[1][r])
                  {
                        results[1][r]=tmpcnt[r];      //store the hits
                        results[0][r]=i;        //store the equation
                  }
            }//end r loop
      }//end i loop

      //print out the results
      for(i=0;i<4;i++)
      {
            for(j=0;j<4;j++)
            {
                  cout << results[i][j] <<"   ";

            }
            cout <<"\n";
      }//end i loop
}
//end of main program
```

sboxcrasher.cpp

```
#include <iostream>
#include <stdlib.h>
#include <fstream>

#define ACE true
#define ZERO false

#define WIDTH 64 //64=16*4 In each Sbox we have 16 elements, each of
these takes 4bits, Totally 4*16
#define HEIGHT 32 //We have 32 lines totally for all S-boxes
#define lenwords 64
#define length 9
#define totalEquations 512

//The table below contains all the Sbox values
bool Sbox[HEIGHT][WIDTH] ;

bool Binaries[totalEquations][length];

using namespace std;

//This function loads the sboxes from a binary file and stores them in
memory can be used for other sboxes too.
void conv2binary(int n)
{
      int k;
      k=n;
      Binaries[k][0]=n/256;         n=n%256;
      Binaries[k][1]=n/128;         n=n%128;
      Binaries[k][2]=n/64;          n=n%64;
      Binaries[k][3]=n/32;          n=n%32;
      Binaries[k][4]=n/16;          n=n%16;
      Binaries[k][5]=n/8;           n=n%8;
      Binaries[k][6]=n/4;           n=n%4;
      Binaries[k][7]=n/2;           n=n%2;
      Binaries[k][8]=n;

}//end of conv2binary function

void loadBinaryRepr(void)
{
      long i;
      for(i=0;i<totalEquations;i++)
            conv2binary(i);
      cout<<"Binary representations have been loaded.\n";
}
void loadSboxes(void)
{
    ifstream inFile;
    int i,j;
    char tmp_read;
    //open the file that contains the Sboxes
    inFile.open("./sboxesbin.txt", ios_base::in );
```

```
    i=j=0;

    //Assign the values of the Sboxes to the table
    while (! inFile.eof() )
   {
        i=i%HEIGHT;
        j=j%WIDTH;
        inFile.get(tmp_read);
        if (tmp_read=='0' || tmp_read=='1')
            {
                Sbox[i][j]=(tmp_read=='1');
                j++;
            }
        if (j==WIDTH) i++;

   }

 //Close the open file
    inFile.close();

}//end of loadSboxes

void findSboxValue(bool *digits, bool *target)
{
    //The first 3 binary digits tell me which sbox to refer to.
    int target_row,target_col;
    int s;

    s=digits[2]*4+digits[1]*2+digits[0];

    target_row=s+2*digits[0]+digits[5];

target_col=(8*digits[1]+4*digits[2]+2*digits[3]+digits[4])*4;//every
entry takes 4bits

    target[0]=Sbox[target_row][target_col];
    target[1]=Sbox[target_row][target_col+1];
    target[2]=Sbox[target_row][target_col+2];
    target[3]=Sbox[target_row][target_col+3];

}//end of findSboxValue

//main program now starts
int main(void )/*int argc, char *argv[]*/
{
      /*
    We have found the proper equations from the previous program.
    We now have to find a better approximation, if possible.
    The better approximation will have the folowing form.
    A_i+x_k*word
    where A_i is the equations we have already calculated,
    x_k is a variable
    and word a random word of (Z_2)^9
    */

      //These counters count the hits of each equation for each bit.
```

```
	long tmpcnt[4];
	long i,j,t,k,r,h;
	bool tmp;
	int cnt;
	bool tmpEquation[length],tmpresults[4];
	bool tmpstr[length];

	long results[4][4];

	loadSboxes();
	loadBinaryRepr();
	//variable h is used here to declare which sbox we are testing
	for(h=0;h<8;h++)
	{
		cnt=0;
		//assign starting values to the results array
		results[0][0]=results[0][1]=results[0][2]=results[0][3]=-1;

	results[1][0]=results[1][1]=results[1][2]=results[1][3]=lenwords+
1;
		results[2][0]=results[2][1]=results[2][2]=results[2][3]=-1;
		results[3][0]=results[3][1]=results[3][2]=results[3][3]=-1;

		//with this loop we take all the possible equations
		//I have 9 bits instead of 6
		for (i=0;i<totalEquations;i++ )    //totalEquations=2^9
		{
		//clear all the countrers that I store the hits
			tmpcnt[0]=tmpcnt[1]=tmpcnt[2]=tmpcnt[3]=0;

			//take the binary representation of equation i
			for(t=0;t<length;t++)
	tmpEquation[t]=Binaries[i][t];
			//take now each different word
			for (j=64*h;j<64*(h+1);j++)
			{
				//take the binary representation of word j
				for(t=0;t<length;t++)
	tmpstr[t]=Binaries[j][t];

				tmp=ZERO;
				//look what we have in the new equation and add
its value
				for (k=0;k<length;k++)
				{
					if (tmpEquation[k]==ACE)
	tmp=tmp^tmpstr[k];
				}//end k loop

				findSboxValue(tmpstr,tmpresults);//find the
sbox value for these specific arguments

				//check each c_i if there is any hit
				for(r=0;r<4;r++)
				{//if yes store a hit to the temporary array
for the specific entry
					if (tmpresults[r]==tmp) tmpcnt[r]++;
```

```
                        }//end r loop
                        cnt++;

                  }//end j loop
                  //check whether the hits of this equation should be
stored
                  for (r=0;r<4;r++)
                  {
                        //check if more hits have been found
                        if (tmpcnt[r]>results[3][r])
                        {
                              results[3][r]=tmpcnt[r];
//store the hits
                              results[2][r]=i;
//store the equation
                        }
                        //check now if less hits have been found
                        if (tmpcnt[r]<results[1][r])
                        {
                              results[1][r]=tmpcnt[r];      //store the
hits
                              results[0][r]=i;        //store the
equation
                        }
                  }//end r loop
            }//end i loop

            //print out the results
            printf("testing sbox %d\n",h+1);
            for(i=0;i<4;i++)
            {
                  if(i==0)
                  {
                        cout <<"          Least hits\n";
                        cout <<"-------------------------------------
------------------\n";
                  }
                  if(i==2)
                  {
                        cout <<"          Most hits\n";
                        cout <<"-------------------------------------
------------------\n";
                  }
                  for(j=0;j<4;j++)
                  {
                        if(i==1||i==3)
                              printf("%.2f%%
     ",(1.0*results[i][j])/64);//show the percemtage
                        else
                              cout << results[i][j] <<"     ";

                  }
                  if(i==0||i==2) cout<<"  equation\n";
                  if(i==1||i==3) cout<<"  correct\n";
            }//end i loop
            cnt=0;
      }
```

```
        //end of main program
}
```