ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΧΑΝΙΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ

Πτυχιακή εργασία με τίτλο

ΚΡΥΠΤΟΓΡΑΦΙΑ

Φοιτητής:

Σμυρνάκης Αθανάσιος

Α.Μ. 2014

ΧΑΝΙΑ

ΑΥΓΟΥΣΤΟΣ 2008

ΠΕΡΙΛΗΨΗ

Η πτυχιακή αυτή εργασία αποτελεί μια βασική προσέγγιση στον κόσμο της κρυπτογραφίας. Ξεκινάει με κάποιες βασικές έννοιες και ορολογίες κρυπτογραφίας και συνεχίζει με τα είδη κρυπτογραφίας που υπάρχουν, καθώς και τα πεδία εφαρμογής της. Ακολουθεί μια σύντομη ιστορική αναδρομή σε κάποια σημαντικά γεγονότα - σταθμούς στην ιστορία της κρυπτογραφίας μέχρι τις μέρες μας. Τέλος, στην εργασία αυτή αναφέρονται τεχνικές των πιο γνωστών μέχρι σήμερα αλγορίθμων κρυπτογραφίας καθώς και παραδείγματα αυτών, δίνοντας μια πιο σφαιρική εικόνα του υπό μελέτη θέματος.

This final work is a basic approach to the world of cryptography. It begins with the basic concept and terminology of cryptography and continues with the existing types of cryptography, as well as its fields of application. A short historical retrospection with some important facts (stations) in the history of cryptography up to nowadays is afterwards referred. Finally, techniques of today's algorithms [cryptography] are being mentioned, as well as a few examples of them, thus giving a more spherical approach to the subject.

Πίνακας Περιεχομένων

1. ΕΙΣΑΓΩΓΗ 6
1.1 Βασικές έννοιες 7
1.2 Είδη Κρυπτοσυστημάτων 8
1.3 Εφαρμογές κρυπτογραφίας 9
2. ΙΣΤΟΡΙΑ ΚΡΥΠΤΟΓΡΑΦΙΑΣ 11
2.1 Η κρυπτογραφία κατά τους αρχαίους χρόνους 11
2.2 Η κρυπτογραφία από το μεσαίωνα μέχρι τον 20ο αιώνα 14
2.3 Η κρυπτογραφία τον 20^{ο} αιώνα 19
2.4 Μηχανικοί αλγόριθμοι κρυπτογράφησης 22
3. ΣΥΓΧΡΟΝΗ ΚΡΥΠΤΟΓΡΑΦΙΑ 23
3.1 Τεχνικές αλγορίθμων 23
3.1.1 DATA ENCRYPTION STANDARD (DES) 24
3.1.2 IDEA 42
3.1.3 Blowfish 48
3.1.4 RC5 52
3.1.5 RSA 54
4. ΠΑΡΑΔΕΙΓΜΑΤΑ ΑΛΓΟΡΙΘΜΩΝ 59
5. ΕΠΙΛΟΓΟΣ 87
6. ΒΙΒΛΙΟΓΡΑΦΙΑ 88

Τεχνική Ορολογία

Κρυπτογράφηση (*encryption*) ονομάζεται η διαδικασία μετασχηματισμού ενός μηνύματος σε μία ακατανόητη μορφή με την χρήση κάποιου κρυπτογραφικού αλγορίθμου ούτως ώστε να μην μπορεί να διαβαστεί από κανέναν εκτός του νόμιμου παραλήπτη. Η αντίστροφη διαδικασία όπου από το κρυπτογραφημένο κείμενο παράγεται το αρχικό μήνυμα ονομάζεται **αποκρυπτογράφηση (*decryption*)**.

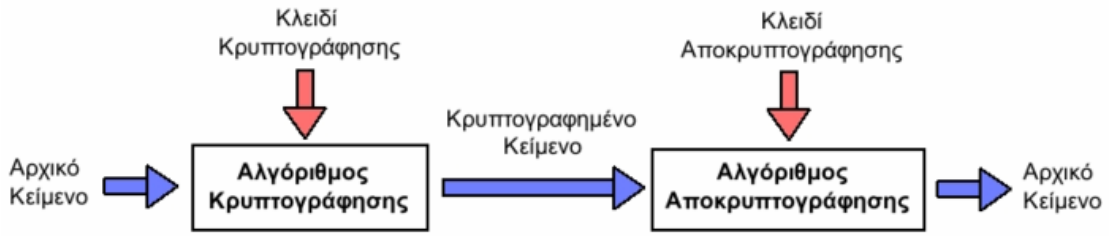
Κρυπτογραφικός αλγόριθμος (cipher) είναι η μέθοδος μετασχηματισμού δεδομένων σε μία μορφή που να μην επιτρέπει την αποκάλυψη των περιεχομένων τους από μη εξουσιοδοτημένα μέρη. Κατά κανόνα ο κρυπτογραφικός αλγόριθμος είναι μία πολύπλοκη μαθηματική συνάρτηση.

Αρχικό κείμενο (*plaintext*) είναι το μήνυμα το οποίο αποτελεί την είσοδο σε μία διεργασία κρυπτογράφησης.

Κλειδί (*key*) είναι ένας αριθμός αρκετών bit που χρησιμοποιείται ως είσοδος στην συνάρτηση κρυπτογράφησης.

Κρυπτογραφημένο κείμενο (*ciphertext*) είναι το αποτέλεσμα της εφαρμογής ενός κρυπτογραφικού αλγόριθμου πάνω στο αρχικό κείμενο.

Κρυπτανάλυση (*cryptanalysis*) είναι μία επιστήμη που ασχολείται με το "σπάσιμο" κάποιας κρυπτογραφικής τεχνικής ούτως ώστε χωρίς να είναι γνωστό το κλειδί της κρυπτογράφησης, το αρχικό κείμενο να μπορεί να αποκωδικοποιηθεί. Η διαδικασία της κρυπτογράφησης και της αποκρυπτογράφησης φαίνεται στο παρακάτω σχήμα.



Σχήμα Ι: Τυπικό 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 στην 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 κρυπτογράφησης.

1. ΕΙΣΑΓΩΓΗ

Η λέξη κρυπτολογία αποτελείται από την ελληνική λέξη κρύπτος – κρυφός και την λέξη λόγος. Είναι ο τομέας που ασχολείται με την μελέτη της ασφαλούς επικοινωνίας. Ο κύριος στόχος είναι να παρέχει μηχανισμούς για 2 ή περισσότερα μέλη να επικοινωνήσουν χωρίς κάποιος άλλος να είναι ικανός να διαβάζει την πληροφορία εκτός από τα μέλη.

Η Κρυπτολογία χωρίζεται σε 2 επιμέρους ενότητες:

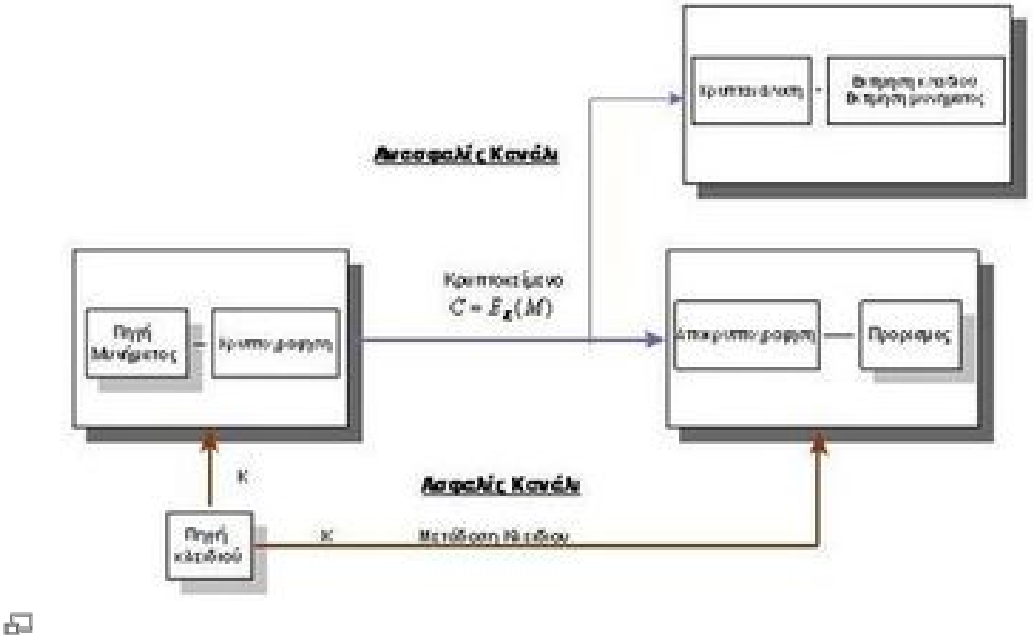
- *Κρυπτογραφία*: η επιστήμη που ασχολείται με τους μαθηματικούς μετασχηματισμούς για την εξασφάλιση της ασφάλειας της πληροφορίας
- *Κρυπτανάλυση*: η επιστήμη που ασχολείται με την ανάλυση και την διάσπαση των Κρυπτοσυστημάτων

Ιστορικά η κρυπτογραφία χρησιμοποιήθηκε για την κρυπτογράφηση μηνυμάτων δηλαδή μετατροπή της πληροφορίας από μια κανονική κατανοητή μορφή σε έναν γρίφο, που χωρίς την γνώση του κρυφού μετασχηματισμού θα παρέμενε ακατανόητος. Κύριο χαρακτηριστικό των παλαιότερων μορφών κρυπτογράφησης ήταν ότι η επεξεργασία γινόταν πάνω στην γλωσσική δομή. Στις νεότερες μορφές η κρυπτογραφία κάνει χρήση του αριθμητικού ισοδύναμου, η έμφαση έχει μεταφερθεί σε διάφορα πεδία των μαθηματικών, όπως διακριτά μαθηματικά, θεωρία αριθμών, θεωρία πληροφορίας, υπολογιστική πολυπλοκότητα, στατιστική και συνδυαστική ανάλυση.

Η κρυπτογραφία παρέχει 4 βασικές λειτουργίες (αντικειμενικοί σκοποί):

- *Εμπιστευτικότητα*: Η πληροφορία προς μετάδοση είναι προσβάσιμη μόνο στα εξουσιοδοτημένα μέλη. Η πληροφορία είναι ακατανόητη σε κάποιον τρίτο.
- *Ακεραιότητα*: Η πληροφορία μπορεί να αλλοιωθεί μόνο από τα εξουσιοδοτημένα μέλη και δεν μπορεί να αλλοιώνεται χωρίς την ανίχνευση της αλλοίωσης.
- *Μη απάρνηση*: Ο αποστολέας ή ο παραλήπτης της πληροφορίας δεν μπορεί να αρνηθεί την αυθεντικότητα της μετάδοσης ή της δημιουργίας της.
- *Πιστοποίηση*: Οι αποστολέας και παραλήπτης μπορούν να εξακριβώνουν τις ταυτότητές τους καθώς και την πηγή και τον προορισμό της πληροφορίας με διαβεβαίωση ότι οι ταυτότητές τους δεν είναι πλαστές.

1.1 Βασικές έννοιες



Σχήμα 1.1: Μοντέλο Τυπικού Κρυπτοσυστήματος

Ο αντικειμενικός στόχος της κρυπτογραφίας είναι να δώσει την δυνατότητα σε 2 πρόσωπα, έστω τον Κώστα και την Βασιλική, 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.

Ένα κρυπτοσύστημα (σύνολο διαδικασιών κρυπτογράφησης - αποκρυπτογράφησης) αποτελείται από μία πεντάδα (P,C,k,E,D):

- Το P είναι ο χώρος όλων των δυνατών μηνυμάτων ή αλλιώς ανοικτών κειμένων
- Το C είναι ο χώρος όλων των δυνατών κρυπτογραφημένων μηνυμάτων ή αλλιώς κρυπτοκειμένων
- Το k είναι ο χώρος όλων των δυνατών κλειδιών ή αλλιώς κλειδοχώρος
- Η E είναι ο κρυπτογραφικός μετασχηματισμός ή κρυπτογραφική συνάρτηση
- Η D είναι η αντίστροφη συνάρτηση ή μετασχηματισμός αποκρυπτογράφησης

Η συνάρτηση κρυπτογράφησης E δέχεται δύο παραμέτρους, μέσα από τον χώρο P και τον χώρο k και παράγει μία ακολουθία που ανήκει στον χώρο C. Η συνάρτηση αποκρυπτογράφησης D δέχεται 2 παραμέτρους, τον χώρο C και τον χώρο k και παράγει μια ακολουθία που ανήκει στον χώρο P.

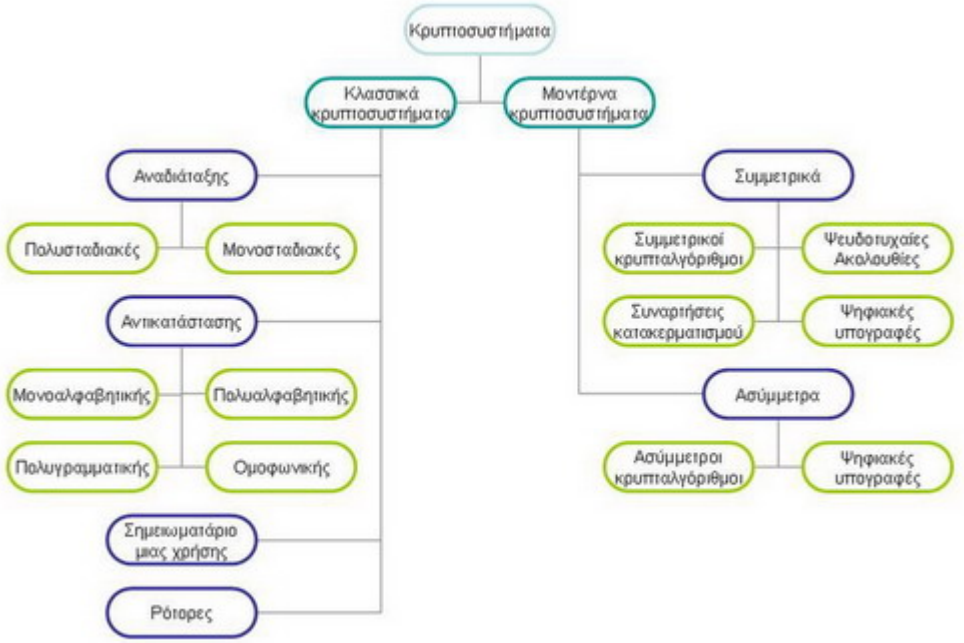
Το Σύστημα του Σχήματος λειτουργεί με τον ακόλουθο τρόπο :

1. Ο αποστολέας επιλέγει ένα κλειδί μήκους *n* από τον χώρο κλειδιών με τυχαίο τρόπο, όπου τα n στοιχεία του K είναι στοιχεία από ένα πεπερασμένο αλφάβητο.
2. Αποστέλλει το κλειδί στον παραλήπτη μέσα από ένα ασφαλές κανάλι.
3. Ο αποστολέας δημιουργεί ένα μήνυμα από τον χώρο μηνυμάτων.
4. Η συνάρτηση κρυπτογράφησης παίρνει τις δυο εισόδους (κλειδί και μήνυμα) και παράγει μια κρυπτοακολουθία συμβόλων (έναν γρίφο) και η ακολουθία αυτή αποστέλλεται διαμέσου ενός μη ασφαλούς καναλιού.
5. Η συνάρτηση αποκρυπτογράφησης παίρνει ως όρισμα τις 2 τιμές (κλειδί και γρίφο) και παράγει την ισοδύναμη ακολουθία μηνύματος.

Ο αντίπαλος παρακολουθεί την επικοινωνία, ενημερώνεται για την κρυπτοακολουθία αλλά δεν έχει γνώση για την κλείδα που χρησιμοποιήθηκε και δεν μπορεί να αναδημιουργήσει το μήνυμα. Αν ο αντίπαλος επιλέξει να παρακολουθεί όλα τα μηνύματα θα προσανατολιστεί στην εξεύρεση του κλειδιού. Αν ο αντίπαλος ενδιαφέρεται μόνο για το υπάρχον μήνυμα θα παράγει μια εκτίμηση για την πληροφορία του μηνύματος.

1.2 Είδη Κρυπτοσυστημάτων

Τα κρυπτοσυστήματα χωρίζονται σε 2 μεγάλες κατηγορίες τα Κλασσικά Κρυπτοσυστήματα και τα Μοντέρνα Κρυπτοσυστήματα.



Σχήμα 1.2: Είδη Κρυπτοσυστημάτων

1.3 Εφαρμογές κρυπτογραφίας

Η εξέλιξη της χρησιμοποίησης της κρυπτογραφίας ολοένα αυξάνεται καθιστώντας πλέον αξιόπιστη την μεταφορά της πληροφορίας για διάφορους λειτουργικούς σκοπούς

1. Ασφάλεια συναλλαγών σε τράπεζες δίκτυα - ATM
2. Κινητή τηλεφωνία (ΤΕΤΡΑ-ΤΕΤΡΑΠΟΛ-GSM)
3. Σταθερή τηλεφωνία (cryptophones)
4. Διασφάλιση Εταιρικών πληροφοριών
5. Στρατιωτικά δίκτυα (Τακτικά συστήματα επικοινωνιών μάχης)
6. Διπλωματικά δίκτυα (Τηλεγραφήματα)
7. Ηλεκτρονικές επιχειρήσεις (πιστωτικές κάρτες, πληρωμές)
8. Ηλεκτρονική ψηφοφορία
9. Ηλεκτρονική δημοπρασία
10. Ηλεκτρονικό γραμματοκιβώτιο
11. Συστήματα συναγερμών

12. Συστήματα βιομετρικής αναγνώρισης
13. Έξυπνες κάρτες
14. Ιδιωτικά δίκτυα (VPN)
15. Word Wide Web
16. Δορυφορικές εφαρμογές (δορυφορική τηλεόραση)
17. Ασύρματα δίκτυα (Hipperlan, bluetooth, 802.11x)
18. Συστήματα ιατρικών δεδομένων και άλλων βάσεων δεδομένων
19. Τηλεσυνδιάσκεψη - Τηλεφωνία μέσω διαδικτύου (VOIP)

2. ΙΣΤΟΡΙΑ ΚΡΥΠΤΟΓΡΑΦΙΑΣ

Η κρυπτογραφία, η επιστήμη της κρυπτογράφησης και αποκρυπτογράφησης πληροφοριών, μπορεί να χαρακτηριστεί σαν ένα από τα αρχαιότερα επαγγέλματα της ανθρωπότητας, έχοντας τις ρίζες της βαθιά στο παρελθόν.

2.1 Η κρυπτογραφία κατά τους αρχαίους χρόνους

Κατά το 1900 π.Χ. στην Αίγυπτο για πρώτη φορά χρησιμοποιήθηκε μια παραλλαγή των τυπικών ιερογλυφικών της εποχής για την επικοινωνία.

Η Κρητική εικονογραφική (ή ιερογλυφική) γραφή, δεν μας έχει αποκαλύψει τον κώδικα της, γνωρίζουμε ωστόσο ότι δεν πρόκειται για γραφή που χρησιμοποιεί εικόνες ως σημεία, αλλά για φωνητική γραφή, η οποία εξαντλείται σε περίπου διακόσιους σφραγιδόλιθους και συνυπήρχε με την γραμμική γραφή Α, τόσο χρονικά όσο και τοπικά, όπως προκύπτει από τις ανασκαφές στο ανάκτορο Μαλίων της Κρήτης. Εμφανίζεται στο Δίσκο της Φαιστού Σχήμα (2.2), που ανακαλύφθηκε το 1908, στην νότια Κρήτη. Πρόκειται για μια κυκλική πινακίδα, που χρονολογείται γύρω στο 1700 π.Χ. και φέρει γραφή με την μορφή δύο σπειρών. Τα σύμβολα δεν είναι χειροποίητα, αλλά έχουν χαραχθεί με την βοήθεια μίας ποικιλίας σφραγίδων, καθιστώντας τον Δίσκο ως το αρχαιότερο δείγμα στοιχειοθεσίας. Δεν υπάρχει άλλο ανάλογο εύρημα, έτσι η αποκρυπτογράφηση στηρίζεται σε πολύ περιορισμένες πληροφορίες. Μέχρι σήμερα, δεν έχει αποκρυπτογραφηθεί και παραμένει η πιο μυστηριώδης αρχαία ευρωπαϊκή γραφή.

Ο Ηρόδοτος περιγράφει πώς μεταφέρονταν κρυπτογραφημένα μηνύματα από τους αγγελιοφόρους. Οι αρχαίοι Εβραίοι κρυπτογραφούσαν κάποιες λέξεις στις περγαμηνές τους.

Μια από τις παλαιότερες αναφορές στην κρυπτογραφία υπάρχει στην Ιλιάδα του Ομήρου, όπου αναφέρεται η αποστολή ενός κρυπτογραφημένου μηνύματος από τον Βελλερεφόντη.

Στην Ελλάδα (Ιλιάδα, ραψωδία στ΄), ο Βελλερεφόντης πάει στη Λυκία μεταφέροντας

γραπτό κρυφό μήνυμα. Στην Ιστορία του Ηροδότου ο Ιστιαίος της Μιλήτου, αιχμάλωτος του Δαρείου, στέλνει μήνυμα στους δικούς του γραμμένο στο κεφάλι εμπίστου σκλάβου. Ο Ηρόδοτος, επίσης, αναφέρει πως ο Δημάρατος, εξόριστος στη Σούσα της Περσίας, προειδοποιεί τους Σπαρτιάτες για το σχέδιο εισβολής του Ξέρξη, στέλνοντας μήνυμα καλυμμένο από κερί. Η Γοργώ, σύζυγος του Λεωνίδα, αποκάλυψε το μήνυμα και ο Ξέρξης έχασε το πλεονέκτημα του αιφνιδιασμού. Τέλος, είναι πολύ γνωστή η ιστορία όπου ο Περίανδρος της Κορίνθου ζήτησε τη γνώμη του Θρασύβουλου της Μιλήτου για το πώς θα κυβερνήσει περισσότερα χρόνια. Ο Θρασύβουλος δεν απάντησε στον αγγελιοφόρο του Περίανδρου, αλλά, καθώς περπατούσε μαζί του μέσα σε ένα λιβάδι σπαρμένο με στάρι, άρχισε να κόβει τα στάρια που φύτρωναν πάνω απ' τ' άλλα.

Ο Πολύβιος ήταν ο πρώτος που χρησιμοποίησε αριθμούς σε μορφή πίνακα για την κωδικοποίηση γραμμάτων.

Η Σπαρτιατική σκυτάλη ανάγεται στον 5ο πΧ. αιώνα. Πρόκειται για ένα ξύλινο ραβδί γύρω από το οποίο τυλίγεται μια λωρίδα από δέρμα ή περγαμηνή. Ο αποστολέας γράφει το μήνυμα κατά μήκος της σκυτάλης και μετά ξετυλίγει τη λωρίδα η οποία φαίνεται να περιέχει μια σειρά γράμματα χωρίς νόημα˙ το μήνυμα έχει αναδιαταχθεί. Για να διαβάσει το μήνυμα ο παραλήπτης, απλά, τυλίγει τη λωρίδα γύρω από μία σκυτάλη ιδίας διαμέτρου με αυτή που χρησιμοποίησε ο αποστολέας. Το 404 πΧ. ο Λύσανδρος απέκρουσε επιτυχώς μια επίθεση του Πέρση σατράπη Φαρνάβαζου χάρη στη σκυτάλη που του έφερε ένας από τους πέντε, αρχικώς, αγγελιοφόρους.
Τον 4ο πΧ. αιώνα είχαμε το «Περί της άμυνας των οχυρών», ένα από τα πρώτα βιβλία κρυπτανάλυσης. Το βιβλίο αυτό, γραμμένο από τον Αινεία τον Τακτικό από την Μεγαλόπολη, περιέχει ένα κεφάλαιο Κρυπτολογίας. Η πολύ κρυφή μέθοδός του με τρύπες σε αστραγάλους και κλωστές, είναι απογοητευτική. Όμως το κείμενο περιέχει την πολύ έξυπνη μέθοδο, όπου μία σελίδα βιβλίου καλύπτεται από μία λευκή σελίδα με διατρήσεις. Τότε, στα παράθυρα, διαβάζουμε το μυστικό κείμενο. Παραλλαγή της μεθόδου αυτής χρησιμοποιήθηκε από τους Γερμανούς στον 2ο παγκόσμιο πόλεμο. Ο Πολύβιος (2ος αιώνας πΧ.) επινόησε ένα κρυπτοσύστημα όπου τα γράμματα του απλού κειμένου αντικαθίστανται από ζεύγη συμβόλων (αριθμών), όπως φαίνεται στον παρακάτω πίνακα

	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	IJ	K
3	L	M	N	O	P
4	Q	R	S	T	U
5	V	W	X	Y	Z

Έτσι, το απλό κείμενο:

"LET NONE ENTER IGNORANT OF GEOMETRY"
("ΜΗΔΕΙΣ ΑΓΕΩΜΕΤΡΗΤΟΣ ΕΙΣΗΤΩ" – επιγραφή στην είσοδο της Ακαδημίας του Πλάτωνα), δίνει το κρυπτοκείμενο

"31 15 44 33 34 33 15 15 33 44 15 42 24 22 33
34 42 11 33 44 34 21 22 15 34 32 15 44 42 54"

Οι Φαραώ συνήθιζαν να γράφουν τα μηνύματά τους στο ξυρισμένο κεφάλι κάποιου σκλάβου και να τον στέλνουν στον παραλήπτη όταν τα μαλλιά του είχαν ξαναμεγαλώσει. Αυτός δεν είχε παρά να ξυρίσει το σκλάβο για να διαβάσει το μήνυμα. Μερικές φορές απλοποιούταν η αποστολή στέλνοντας μόνο το κεφάλι. Η μέθοδος αυτή είχε προφανή προβλήματα και η αξιοπιστία της επιβαρυνόταν ακόμα περισσότερο από τη συνήθεια των σκλάβων να προσπαθούν να απελευθερωθούν από τα αφεντικά τους.

Οι Αιγύπτιοι ιερείς χρησιμοποιούσαν μέθοδο αντίστοιχη με τη σκυτάλη των Σπαρτιατών, την οποία χρησιμοποίησε και ο Ιούλιος Καίσαρας. Επίσης μεθόδους κρυπτογραφίας είχαν αναπτύξει και οι Αριστοτέλης, Πυθαγόρας και Νέρωνας.

Ο Ιούλιος Καίσαρας (100 – 44 π.Χ.) χρησιμοποίησε μια απλή αντικατάσταση στο κανονικό αλφάβητο (μετακίνηση – shift των γραμμάτων κατά μια προκαθορισμένη ποσότητα – τρία γράμματα) στις "κυβερνητικές" επικοινωνίες (Caesar cipher). ο Βαλέριος Πρόβος, έγραψε ολόκληρη πραγματεία για τα κρυπτοσυστήματά του που, δυστυχώς δεν σώθηκε. Ωστόσο, χάρη στους «Βίους των δώδεκα καισάρων» του Σουητώνιου (2ος μΧ.αιώνας), έχουμε λεπτομερή περιγραφή μιας αντικατάστασης που χρησιμοποιούσε ο Καίσαρας. Αυτή μετατόπιζε τρεις θέσεις προς τα δεξιά τα γράμματα του λατινικού αλφαβήτου όπου τα τρία τελευταία γινόντουσαν αντίστοιχα A, B, C.

Το απλό κείμενο

"BOUDICCA HAS BURNED LONDINIOUM"

γίνεται

"ERXGLFFD KDV EXUQHG ORQGLQLXP"

αντίστοιχα.

Ο Αύγουστος Καίσαρας χρησιμοποίησε την ίδια μέθοδο μετακινώντας κατά ένα γράμμα

2.2 Η κρυπτογραφία από το μεσαίωνα μέχρι τον 20ο αιώνα

Η περίοδος στην Ευρώπη από το 400 – 1200 μΧ. είναι γνωστή σαν μεσαίωνας. Οι επιστήμες, μεταξύ αυτών και η Κρυπτογραφία, ήταν σε παρακμή. Το 529 μΧ., μετά από ζωή 9 αιώνων, έκλεισε η Ακαδημία του Πλάτωνα. Σχεδόν αποκλειστικά τα μόνα εκπαιδευτικά ιδρύματα που λειτουργούσαν στη Δ. Ευρώπη, την εποχή αυτή, ήταν τα μοναστήρια, κυρίως των Βενεδεκτίνων. Την περίοδο αυτή η Κρυπτογραφία αναπτύχθηκε στην Ινδία και στις ισλαμικές χώρες.

Οι πρώτοι που κατάλαβαν καλά τις αρχές της κρυπτογραφίας και της κρυπτανάλυσης ήταν οι Άραβες. Κατασκεύασαν και χρησιμοποίησαν αλγόριθμους αντικατάστασης και μετατόπισης και ανακάλυψαν τη χρήση της συχνότητας των χαρακτήρων και των πιθανοτήτων στην κρυπτανάλυση. Έτσι το 1412 ο Al-Kalka-Shandi συμπεριέλαβε την περιγραφή αρκετών κρυπτογραφικών συστημάτων στην εγκυκλοπαίδεια Subh al-a'sha και έδωσε σαφείς οδηγίες και παραδείγματα για την κρυπτανάλυση κρυπτογραφημένων κειμένων χρησιμοποιώντας τη συχνότητα των χαρακτήρων.

Ο βραχμάνος λόγιος Βατσιγιάνα έγραψε τον 4ο μΧ. αιώνα το περίφημο «Κάμα Σούτρα».Τα «Κάμα Σούτρα» συνιστούν στις γυναίκες να μελετούν 64 τέχνες όπως, μαγειρική, ενδυματολογία, αρωματοποιία κλπ. Η 45η τέχνη του καταλόγου είναι η μιλεχίτα βικάλπα, δηλ., η τέχνη της μυστικής γραφής (σύσταση που αφορά την απόκρυψη των ερωτικών τους περιπετειών). Μία προτεινόμενη τεχνική είναι το

ζευγάρωμα των γραμμάτων του αλφαβήτου τυχαία και ακολούθως κάθε γράμμα αντικαθίσταται με το ταίρι του.

Παράδειγμα: αν ζευγαρώσω τα γράμματα του ελληνικού αλφαβήτου πΧ. όπως παρακάτω

Α, Δ, Η, Ι, Κ, Μ, Ο, Ρ, Σ, Θ, Υ, Ζ
↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕
Ω, Χ, Β, Γ, Ζ, Ψ, Λ, Ν, Ε, Φ, Π, Τ
τότε το

"ΣΥΝΑΝΤΗΣΗ ΤΑ ΜΕΣΑΝΥΧΤΑ"
κωδικοποιείται σαν

"ΕΠΡΩΡΖΒΕΒ ΖΩ ΨΣΕΩΡΠΑΖΩ"

Ο σημαντικότερος εκπρόσωπος των Αράβων κρυπτολόγων είναι ο πανεπιστήμων του 9^{ου} αιώνα Αλ Κιντί. έγραψε πάνω από 290 βιβλία Μαθηματικών – Γλωσσολογίας – Αστρολογίας– Ιατρικής και Μουσικής. Το 1987 στα Οθωμανικά Αρχεία Σουλεϊμανιγιέ της Κων/πολης, ανακαλύφθηκε η πραγματεία του "Περί αποκρυπτογράφησης κρυπτογραφημένων μηνυμάτων" και λέει τα εξής: Ένας τρόπος να διαβάσουμε ένα κρυπτογραφημένο κείμενο αν γνωρίζουμε τη γλώσσα του, είναι να βρούμε ένα διαφορετικό ακρυπτογράφητο. στην ίδια γλώσσα που να καλύπτει, περίπου ένα φύλλο, και μετά να μετρήσουμε τη συχνότητα εμφάνισης κάθε γράμματος. Το συχνότερα εμφανιζόμενο γράμμα ονομάζεται *πρώτο*, το αμέσως επόμενο *δεύτερο*, μέχρι να χαρακτηρίσουμε όλα τα γράμματα του α.κ. Στη συνέχεια πιάνουμε το κ.κ. που θέλαμε να αποκρυπτογραφήσουμε και ταξινομούμε με τον ίδιο τρόπο τα σύμβολά του. Βρίσκουμε το συχνότερα εμφανιζόμενο σύμβολο και το αντικαθιστούμε με το *πρώτο* γράμμα, το *δεύτερο* στη συχνότητα σύμβολο με το *δεύτερο* γράμμα, το *τρίτο* με το *τρίτο* γράμμα κοκ.

Η Ευρωπαϊκή κρυπτολογία έχει τις ρίζες της το μεσαίωνα, που αναπτύχθηκε από τους Πάπα και τις Ιταλικές πόλεις κράτη, αλλά τα περισσότερα συστήματα βασίζονταν στην απλή αντικατάσταση γραμμάτων της αλφαβήτου (όπως στον αλγόριθμο του Καίσαρα). Οι πρώτοι αλγόριθμοι βασίζονταν στην αντικατάσταση των φωνηέντων. Το πρώτο Ευρωπαϊκό εγχειρίδιο κρυπτογραφίας (1379) ήταν μια συλλογή αλγορίθμων από τον Gabriele de Lavinde of Parma, για τον Πάπα. Το 1470 ο Leon Battista Alberti εξέδωσε το "Trattati in cifra", όπου περιγράφεται ο πρώτος δίσκος κρυπτογράφησης (τον οποίο είχε κατασκευάσει το 1460), χρησιμοποιώντας

και την έννοια της χρήσης πολλαπλών αλφαβήτων. Επίσης στο βιβλίο αυτό περιέγραφε και τις αρχές της ανάλυσης συχνότητας των γραμμάτων.

Ο Φλωρεντινός Λέων Μπατίστα Αλμπέρτι (1404 μΧ.) είναι ο συγγραφέας του «De re aedificatoria» δηλ. «Περί οικοδομικής τέχνης» (1467 μΧ.), του πρώτου τυπωμένου βιβλίου αρχιτεκτονικής. Είναι γνωστότερος σαν ο σχεδιαστής της πρώτης Φοντάνα ντι Τρέβι στη Ρώμη. Στην κρυπτανάλυση ήταν ο πρώτος που σκέφθηκε ένα πολυαλφαβητικό σύστημα, δηλ. ένα σύστημα με περισσότερα από ένα κρυπτογραφικά αλφάβητα, γεγονός που κάνει την κρυπτανάλυση δυσκολότερη αφού δεν διατηρεί τις συχνότητες των γραμμάτων. Επίσης επινόησε και την πρώτη μετά τη σκυτάλη κρυπτογραφική μηχανή, τους λεγόμενους *δίσκους του Alberti.* Πήρε δύο χάλκινους δίσκους διαφορετικής διαμέτρου, τους έκανε ομόκεντρους και χάραξε ένα αλφάβητο στην περιφέρεια του κάθε δίσκου. Οι δύο δίσκοι μπορούν ναπεριστρέφονται ανεξάρτητα. Ο έξω δίσκος αναφέρεται στο α.κ. και οι αντίστοιχες θέσεις του μέσα δίσκου μας δίνουν το κ.κ.. Για περισσότερη δυσκολία ένα μέρος του μηνύματος κρυπτογραφείται σε μια θέση του εσωτερικού δίσκου και ένα άλλο μέρος σε άλλη θέση του δίσκου (πολυαλφαβητικό)
Το 1499 μΧ. ο Γερμανός abbas Johanes Trithemius έγραψε τη Στενογραφία, ένα βιβλίο επικοινωνίας με τα πνεύματα. Το βιβλίο του κατέληξε στο Index Librorum Prohibitorum, αλλά και στην ανάλογη λίστα των Προτεσταντών. Ο τρίτος τόμος του περιείχε πίνακες αριθμών που δήθεν αντιπροσώπευαν κώδικες επικοινωνίας με τα πνεύματα. 150 χρόνια αργότερα, το 1676 μΧ., ο Heidel ένας δικηγόρος από το Mainz , ισχυρίστηκε ότι έσπασε τον κώδικα του Trithemius, αλλά έγραψε τη λύση του σε κώδικα. Μόλις το 1996 μΧ. σπάσανε οι κώδικες του Trithemius και του Heidel που, όμως δυστυχώς ήσαν ανόητα ξόρκια. Πολύ πιο ενδιαφέρον είχε η Πολυγραφία (1518 μΧ.) του Trithemius, το πρώτο τυπωμένο βιβλίο Κρυπτογραφίας.

Τούτο περιείχε τετράγωνα γραμμάτων όπου η πρώτη γραμμή περιέχει τα 26 γράμματα του λατινικού αλφαβήτου και οι υπόλοιπες 26 γραμμές είναι κυκλικές μεταθέσεις, της πρώτης γραμμής κατά 0, 1, 2, …, 25 θέσεις. Οι στήλες αντιστοιχούν στο **ακρυπτογράφητο κείμενο**, το **κρυπτογραφημένο κείμενο** βρίσκεται στην τομή γραμμής και στήλης. Έστω το ακρυπτογράφητο .κείμενο .

DEUS

γίνεται

DFWV

Η 1η γραμμή έχει κάτω από το γράμμα **D** της 1ης γραμμής, πάλι το γράμμα **D**
Η 3η γραμμή έχει κάτω από το **E** της 1ης γραμμής το γράμμα **F**
Η 4η » » » **U** » » » **W**
Η 5η » » » **S** » » » **V**
Άρα

DEUS → DFWV

Ο Sir Francis Bacon το 1563 περιέγραψε έναν αλγόριθμο που σήμερα φέρει το όνομά του. Ήταν ένας αλγόριθμος που χρησιμοποιούσε κωδικοποίηση 5 bits. Τον αλγόριθμο αυτό τον εξέλιξε σαν μια μέθοδο στεγανογραφίας, χρησιμοποιώντας μία μεταβολή στη μορφή των χαρακτήρων μετέφερε κάθε bit της κωδικοποίησης. Ο Blaise de Vigenere δημοσίευσε ένα βιβλίο πάνω στην κρυπτολογία το 1585, που περιέγραφε τον αλγόριθμο της πολυαλφαβητικής αντικατάστασης. Ακολούθησαν και άλλα βιβλία πάνω στην κρυπτογραφία με εξελίξεις των αλγορίθμων.

Τα μυστικά της κρυπτολογίας φυλάσσονταν στα μοναστήρια ή στα μυστικά αρχεία των βασιλιάδων και λίγες μέθοδοι γίνονταν ευρέως γνωστές.

Κατά την αναγέννηση η κρυπτολογία έγινε χωριστή επιστήμη και ταυτόχρονα οι εφαρμοστές της αναζητούσαν μια γενική γλώσσα.

Το 1600 ο Καρδινάλιος Ρισελιέ χρησιμοποιούσε μια κάρτα με τρύπες για να γράψει το μήνυμά του. Όταν τελείωνε γέμιζε τα κενά με λέξεις ώστε να μοιάζει με ένα κανονικό γράμμα. Για την αποκωδικοποίηση χρειαζόταν η κάρτα με την οποία είχε γραφεί το γράμμα.

Το 1641 μΧ. ο πρώτος γραμματέας της Βασιλικής Εταιρείας του Λονδίνου John Wilkins εισήγαγε τις λέξεις Κρυπτογραφία και Κρυπτολογία στην αγγλική γλώσσα.
Το 1586 μΧ. κυκλοφόρησε το βιβλίο «Πραγματεία περί αριθμών» («Traité des chiffres») του Γάλλου διπλωμάτη και κρυπταναλυτή Blaise de Visenére. Το βιβλίο περιέχει πολλά πολυαλφαβητικά συστήματα, όπως τα τετράγωνα του Trithemius και του Belaso, τα οποία σήμερα είναι γνωστά σαν τετράγωνα ή πίνακες Visenére.

Σημαντικό είναι το κρυπτοσύστημα που προτείνει ο Visenére όπου το ακρυπτογράφητο κείμενο ή το κρυπτογραφημένο κείμενο είναι το κλειδί. Έστω, για παράδειγμα, ότι πήραμε το κρύπτογραφημενο κείμενο

CWRQ PAFV QABRC

και έστω ότι επίσης γνωρίζουμε το πρώτο γράμμα του κλειδιού **K** και του ακρυπτογράφητου κειμένου **S** (πληροφορία που δεν χρειάζεται αφού στη γραμμή του **K** η πρώτη εμφάνιση του **C** είναι στην στήλη **S**). Το δεύτερο γράμμα του κλειδιού είναι, λοιπόν, το **S**. Στη γραμμή **S** η πρώτη εμφάνιση του **W** είναι στη στήλη **E**. Το τρίτο γράμμα του κλειδιού είναι, λοιπόν, το **E**. Στη γραμμή **E** η πρώτη εμφάνιση του **R** είναι στη στήλη **N**. Άρα το 4ο γράμμα του κλειδιού είναι το **N**. Στη γραμμή **N** η πρώτη εμφάνιση του **Q** είναι στη στήλη **D**, κοκ. Το απλό κείμενο, λοιπόν, είναι :

SEND MORE MONEY.

Το 1776 ο Αμερικάνος Arthur Lee ανέπτυξε ένα κώδικα με βιβλίο τον οποίο σύντομα υιοθέτησε ο στρατός.

Ο υπουργός προεδρίας του G. Washington και 3ος πρόεδρος των ΗΠΑ Thomas Jefferson επινόησε το περιστροφικό κρυπτοσύστημα. Ένας κύλινδρος μήκους 15 πόντων περίπου αποτελείται από 36 κυκλικούς δίσκους παραλλήλους προς τη βάση του. Στις περιφέρειες των 36 δίσκων χαράσσονται 36 αλφάβητα ένα στο ύψος του κάθε δίσκου με τυχαία διάταξη. Για να κρυπτογραφήσουμε ένα μήνυμα μέχρι 36 γραμμάτων περιστρέφουμε τους δίσκους έτσι ώστε να γραφτεί το μήνυμα σε μια οριζόντια γραμμή-γενέτειρα της κυλινδρικής επιφάνειας. Μία από τις υπόλοιπες 25 γραμμές-γενέτειρες αποτελεί το κ.κ.. Το σύστημα αυτό εφαρμόστηκε από τον αμερικανικό στρατό πρώτη φορά στην εξερεύνηση της Louisiana-Territory από τους Lewis-Clark και Sacajawea το 1802-1804 μέχρι το 1922, ενώ το ναυτικό των ΗΠΑ το χρησιμοποιούσε μέχρι το 1960.

Ένας άλλος διάσημος κώδικας είναι ο κώδικας Μορς, που αναπτύχθηκε από τον Samuel Morse το 1832, και απλώς περιγράφει τον τρόπο κωδικοποίησης του αλφαβήτου σε μακρείς και σύντομους ήχους. Με την ταυτόχρονη ανακάλυψη του τηλέγραφου ο κώδικας Μορς βοήθησε στην επικοινωνία των ανθρώπων σε μεγάλες αποστάσεις.

Το 1860 οι μεγάλοι κώδικες χρησιμοποιούνταν συχνά στις διπλωματικές επικοινωνίες. Στη διπλωματία και κατά τις περιόδους πολέμου υπήρχε αυξημένη χρήση της κρυπτογραφίας, χαρακτηριστικό παράδειγμα είναι τα one-time pads που χρησιμοποιούνταν ευρέως.

Στα πρώτα χρόνια της Αμερικάνικης ιστορίας έχουμε την ευρεία χρήση κωδίκων σε βιβλία. Κατά τον εμφύλιο πόλεμο έγινε εκτεταμένη χρήση αλγορίθμων μετάθεσης από το ένα μέρος και του αλγορίθμου Vigenere από το άλλο μέρος. Στην προσπάθεια αποκρυπτογράφησης των εχθρικών μηνυμάτων χρησιμοποιήθηκαν μέχρι και δημοσιεύσεις κωδικοποιημένων μηνυμάτων στις εφημερίδες, ζητώντας τη βοήθεια των αναγνωστών.

2.3 Η κρυπτογραφία τον 20^{ο} αιώνα

Αν και η κρυπτογραφία χρησιμοποιήθηκε κατά τον 1^{ο} Παγκόσμιο Πόλεμο, δύο από τις πιο αξιοπρόσεκτες μηχανές εμφανίστηκαν κατά τον 2^{ο} Παγκόσμιο Πόλεμο: οι Γερμανοί χρησιμοποίησαν την Enigma machine που αναπτύχθηκε από τον Arthur Scherbius και οι Γιαπωνέζοι την Purple Machine που αναπτύχθηκε χρησιμοποιώντας τεχνικές που ανακαλύφτηκαν από τον Herbert O. Yardley.

Το 1918 ο Γερμανός εφευρέτης Άρθουρ Σέρμπιους ανέπτυξε ένα μηχανικό κρυπτογραφικό σύστημα που θα μπορούσε να θεωρηθεί σαν η ηλεκτρική παραλλαγή των δίσκων του Alberti. Η εφεύρεση του έγινε γνωστή σαν **Αίνιγμα**. Η βασική μορφή του Αινίγματος αποτελείται

i) από ένα πληκτρολόγιο για την εισαγωγή του ακρυπτογράφητου κειμένου

ii) από μία αναδιατακτική μονάδα που κρυπτογραφεί κάθε γράμμα του ακρυπτογράφητου κειμένου σ' ένα αντίστοιχογράμμα του κρυπτογραφημένου κειμένου.

iii) σ' έναν ηλεκτρικό πίνακα που δείχνει το αντίστοιχο γράμμα του κρυπτογραφημένου κειμένου

Η διαδικασία αυτή, σύντομα, έγινε πιο πολύπλοκη με την προσθήκη περισσότερων αναδιατακτών, βυσμάτων, ανακλαστών και έφθασε να έχει πάνω από 10 τετράκις εκατομμύρια πιθανά κλειδιά για να σπάσει το κ.κ. Ο γερμανικός στρατός αγόρασε πάνω από 30.000 συσκευές Αίνιγμα (αν και ο Σέρμπιους σκοτώθηκε σε αυτοκινητιστικό ατύχημα το 1929) και φαινόταν ότι η Γερμανία διαθέτει τις ασφαλέστερες επικοινωνίες στον κόσμο. Όμως η φιλοχρηματία κάποιου Χανς-Τίλο Σμιτ, η αγκύλωση των χιτλερικών χειριστών να αρχίζουν όλα τα ημερήσια μηνύματα με το μήνυμα-κλειδί της ημέρας, μήκους 3 που κρυπτογραφείτο δυο φορές, και η ικανότητα του Πολωνού μαθηματικού Μάριαν Ρεζέφσκι, επέτρεψαν στην ομάδα των Πολωνών κρυπταναλυτών να σπάσουν το αίνιγμα, στην σχετικά απλή μορφή που είχε το 1938. Μάλιστα τα σημαντικά κρυπταναλυτικά αποτελέσματα των Πολωνών κρυπταναλυτών 2 βδομάδες πριν τη χιτλερική εισβολή στην Πολωνία, η πολωνική κυβέρνηση τα έδωσε στους συμμάχους της Άγγλους.
Η πολεμική μορφή που πήρε η συσκευή Αίνιγμα εθεωρείτο από τους συμμάχους άθραυστη. Όμως οι πολωνικές αποκαλύψεις απέδειξαν στους συμμάχους την αξία της πρόσληψης μαθηματικών για το σπάσιμο των κωδίκων. Μια μεγάλη ομάδα κρυπταναλυτών μελέτησε στο Μπλίτσλεϋ Παρκ του Μπάκιμχαμσάϊρ και κατάφεραν να σπάσουν το Αίνιγμα. Οι πληροφορίες που έπεσαν στα χέρια των Συμμάχων, έτσι, ήταν εξαιρετικά σημαντικές και, όπως ισχυρίζεται ο επίσημος ιστορικός της Βρετανικής Υπηρεσίας Πληροφοριών συντόμεψαν τη διάρκεια του πολέμου κατά 3 έτη. Στον ελλαδικό χώρο το Μπλίτσλεϋ ειδοποίησε τους συμμάχους για τη γερμανική εισβολή στην Ελλάδα το ’41, πράγμα που πιθανώς να βοήθησε τα βρετανικά στρατεύματα να αποχωρήσουν χωρίς βαριές απώλειες.

Από όλες τις ιστορικές προσωπικότητες που συνέβαλαν στην ανάπτυξη της κρυπτογραφίας ο William Frederick Friedman, ιδρυτής των Riverbank Laboratories, κρυπτολόγος της Αμερικανικής κυβέρνησης και οδηγός του σπασίματος του κώδικα της Ιαπωνικής Purple Machine κατά τον 2^{ο} Παγκόσμιο Πόλεμο, θεωρείται ο πατέρας της Αμερικάνικης κρυπτανάλυσης. Το 1918 έγραψε το βιβλίο «The Index of Coincidence and Its Applications in Cryptography» που ακόμα θεωρείται από αρκετούς σαν το σημαντικότερο σύγγραμμα πάνω στην κρυπτογραφία κατά τον 20^{ο} αιώνα.

Τα μεταπολεμικά χρόνια η Κρυπτογραφία γνώρισε μεγάλη ανάπτυξη χάρις, κυρίως, στην τεχνολογία των υπολογιστών. Ο **Κολοσσός** (1943) με τις 1500 ηλεκτρονικές λυχνίες του και το γεγονός ότι ήταν προγραμματισμένος, ο **ENIAC** (1945) με τις

18.000 ηλεκτρονικές λυχνίες που μπορούσε να εκτελεί 5.000 υπολογισμούς ανά sec κοκ.

Τη δεκαετία του 1970 ο Dr. Horst Feistel δημιούργησε τον πρόγονο του σημερινού Data Encryption Standard (DES) με την οικογένεια ciphers, που ονομάστηκε 'Feistel ciphers', δουλεύοντας στο Watson Research Laboratory της IBM. Το 1976 η National Security Agency (NSA) σε συνεργασία με τον Feistel δημιούργησε τον αλγόριθμο FIPS PUB-46, γνωστό σήμερα σαν DES. Σήμερα, η εξέλιξή του σε triple-DES είναι το πρότυπο ασφαλείας που χρησιμοποιείται από τους οικονομικούς οργανισμούς των Ηνωμένων Πολιτειών. Επίσης το 1976 δύο συνεργάτες του Feistel, ο Whitfield Diffie και ο Martin Hellman, εισήγαγαν για πρώτη φορά την ιδέα της κρυπτογραφίας δημοσίου κλειδιού στο άρθρο "New Directions in Cryptography". Η κρυπτογραφία δημοσίου κλειδιού είναι αυτό που χρησιμοποιεί το ευρέως χρησιμοποιούμενο σήμερα PGP.

Στα μέσα της δεκαετίας του 1980 ο αλγόριθμος ROT13 χρησιμοποιήθηκε από χρήστες του USENET "για να μη βλέπουν τα μηνύματά με επιλήψιμο περιεχόμενο αθώα μάτια" και λίγο αργότερα το 1990 μια ανακάλυψη από τους Xuejia Lai και James Massey οδήγησε σε ένα δυνατότερο, 128-bit key cipher με σκοπό να αντικαταστήσει το γερασμένο DES standard. Ο αλγόριθμος IDEA (International Data Encryption Algorithm) που σχεδιάστηκε από αυτούς είχε σκοπό να είναι πιο αποδοτικός με γενικής χρήσης υπολογιστές όπως αυτούς που χρησιμοποιούνται στις επιχειρήσεις και στα νοικοκυριά.

Το FBI ανησυχώντας από την εξάπλωση της κρυπτογραφίας ανανέωσε την προσπάθειά του να έχει πρόσβαση στα μηνύματα κειμένου των Αμερικανών πολιτών. Σε απάντηση ο Phil Zimmerman εξέδωσε την πρώτη έκδοση του Pretty Good Privacy (PGP) το 1991 σαν ένα προϊόν ελεύθερα διαθέσιμο, που χρησιμοποιεί τον αλγόριθμο IDEA. Το PGP, ένα δωρεάν πρόγραμμα του παρέχει στρατιωτικού επιπέδου αλγορίθμους ασφαλείας στην κοινότητα του Internet, έχει εξελιχθεί σε πρότυπο κρυπτογραφίας λόγω της ευρείας διάδοσής του.

Τελευταία, το 1994, ο καθηγητής Ron Rivest, που βοήθησε στην ανάπτυξη του RSA, δημοσίευσε ένα νέο αλγόριθμο, το RC5.

2.4 Μηχανικοί αλγόριθμοι κρυπτογράφησης

Μια ξεχωριστή κατηγορία στο χώρο της κρυπτογραφίας αποτελούν οι συσκευές που χρησιμοποιούσαν κάποιο μηχανικό τρόπο για την κρυπτογράφηση και αποκρυπτογράφηση των μηνυμάτων. Παρακάτω αναφέρουμε μερικούς από αυτούς.

Jefferson cylinder: αναπτύχθηκε το 1790 και αποτελούνταν από 36 δίσκους. Ο κάθε δίσκος είχε ένα τυχαίο αλφάβητο και η σειρά των δίσκων ήταν το κλειδί αποκρυπτογράφησης.

Wheatstone disc: ανακαλύφτηκε από τον Wadsworth το 1817 και αναπτύχθηκε από τον Wheatstone το 1860. Αποτελούνταν από δύο τροχούς που χρησιμοποιούνταν για τη δημιουργία ενός πολυαλφαβητικού αλγορίθμου.

Hagelin machine: μια πραγματικά πρωτοποριακή μηχανή.

Enigma Rotor machine: μια από τις πολύ σημαντικές κατηγορίες μηχανών κρυπτογράφησης. Χρησιμοποιήθηκε πολύ κατά τον $2^{ο}$ Παγκόσμιο Πόλεμο. Αποτελούνταν από μια σειρά περιστρεφόμενους τροχούς με εσωτερικές διασυνδέσεις που παρείχαν την αντικατάσταση χρησιμοποιώντας ένα αλφάβητο που συνεχώς άλλαζε. Ήταν βασισμένη σε ένα σχέδιο που αναπτύχθηκε από τον Arthur Scherbius το 1910. Αποτελείται από τρία μέρη που συνδέονταν με σύρματα. Ένα πληκτρολόγιο εισόδου, τη μονάδα κρυπτογράφησης και ενδεικτικές λυχνίες. Η μονάδα κρυπτογράφησης περιστρεφόταν κατά μια ορισμένη γωνία κάθε φορά που ένα γράμμα κρυπτογραφούνταν. Η μηχανή που χρησιμοποιήθηκε κατά τον $2^{ο}$ Παγκόσμιο Πόλεμο είχε τρεις μονάδες κρυπτογράφησης.

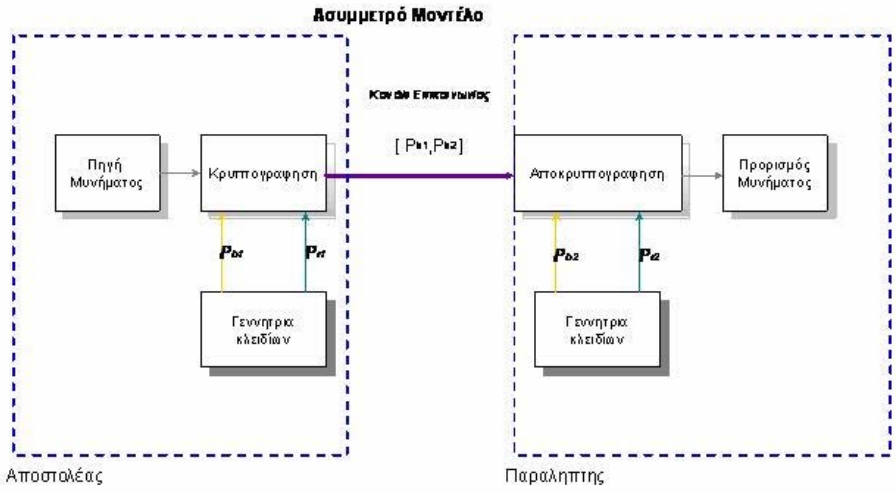
Η αποκρυπτογράφηση της μηχανής Enigma έγινε αφού ένας Γερμανός (ο Hans-Thilo Schmidt) έδωσε κάποια βιβλία κωδικών σε ένα Γάλλο που με τη σειρά του τα έδωσε στον Poles. Βασικές τεχνικές αναπτύχθηκαν από τη Marian Rejewski και επεκτάθηκαν από την Αγγλική αντικατασκοπία με αποτέλεσμα το σπάσιμο του κώδικα. Μετά την πρώτη αποκρυπτογράφηση του κώδικα οι Γερμανοί άλλαξαν τον κώδικα, αλλά δεύτερη διαρροή και συντονισμένες προσπάθειες οδήγησαν ξανά στο σπάσιμό του.

3. ΣΥΓΧΡΟΝΗ ΚΡΥΠΤΟΓΡΑΦΙΑ

3.1 Τεχνικές αλγορίθμων

Υπάρχουν διάφορες τεχνικές αλγορίθμων στις οποίες βασίζονται και οι αντίστοιχοι κώδικες κρυπτογραφίας, εδώ θα αναφέρω τις δυο βασικότερες κατηγορίες τεχνικών αλγορίθμων οι οποίες είναι: 1) Συμμετρικοί αλγόριθμοι 2) Ασύμμετροι αλγόριθμοι. Στους συμμετρικούς αλγόριθμους συγκαταλέγονται οι **DES, IDEA, RCS, BLOWFISH** ενώ στην κατηγορία των ασύμμετρων ένα τυπικό δείγμα είναι ο **RSA**.

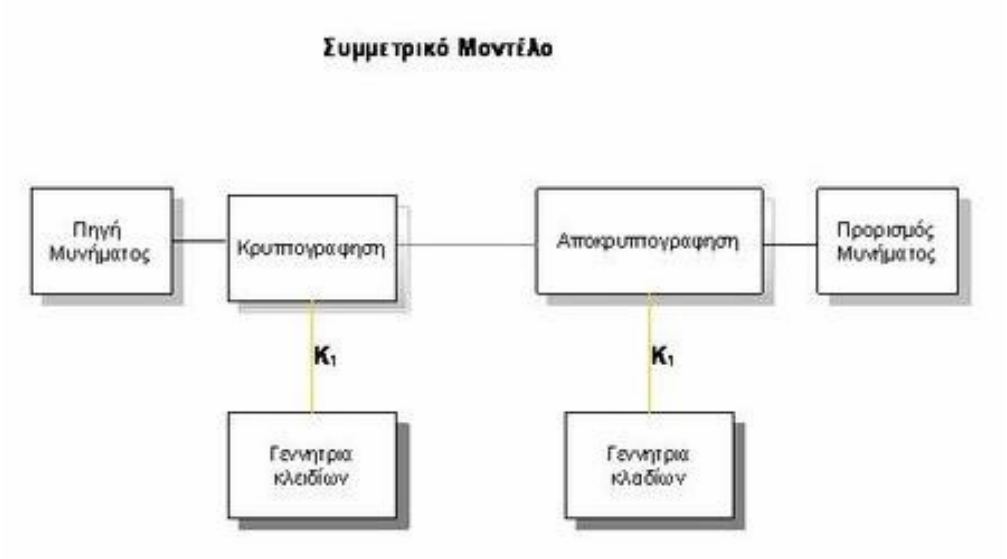
Το ασύμμετρο κρυπτοσύστημα ή κρυπτοσύστημα δημοσίου κλειδιού δημιουργήθηκε για να καλύψει την αδυναμία μεταφοράς κλειδιών που παρουσίαζαν τα συμμετρικά συστήματα. Χαρακτηριστικό του είναι ότι έχει δυο είδη κλειδιών, ένα ιδιωτικό και ένα δημόσιο. Το δημόσιο είναι διαθέσιμο σε όλους ενώ το ιδιωτικό είναι μυστικό. Η βασική σχέση μεταξύ τους είναι : ό,τι κρυπτογραφεί το ένα, μπορεί να το αποκρυπτογραφήσει μόνο το άλλο



Σχήμα 3.1: Ασύμμετρο Κρυπτοσύστημα

Συμμετρικό κρυπτοσύστημα είναι το σύστημα εκείνο το οποίο χρησιμοποιεί κατά την διαδικασία της κρυπτογράφησης αποκρυπτογράφησης ένα κοινό κλειδί (Σχ 1.3). Η ασφάλεια αυτών των αλγορίθμων βασίζεται στην μυστικότητα του κλειδιού. Τα

συμμετρικά κρυπτοσυστήματα προϋποθέτουν την ανταλλαγή του κλειδιού μέσα από ένα ασφαλές κανάλι επικοινωνίας ή μέσα από την φυσική παρουσία των προσώπων. Αυτό το χαρακτηριστικό καθιστά δύσκολη την επικοινωνία μεταξύ απομακρυσμένων ατόμων.



Σχήμα 3.2: Συμμετρικό Κρυπτοσύστημα

Στα παρακάτω υποκεφάλαια θα προσπαθήσω να δώσω μια σχετικά γενική εικόνα των παραπάνω αλγορίθμων για μια καλύτερη κατανόηση τους.

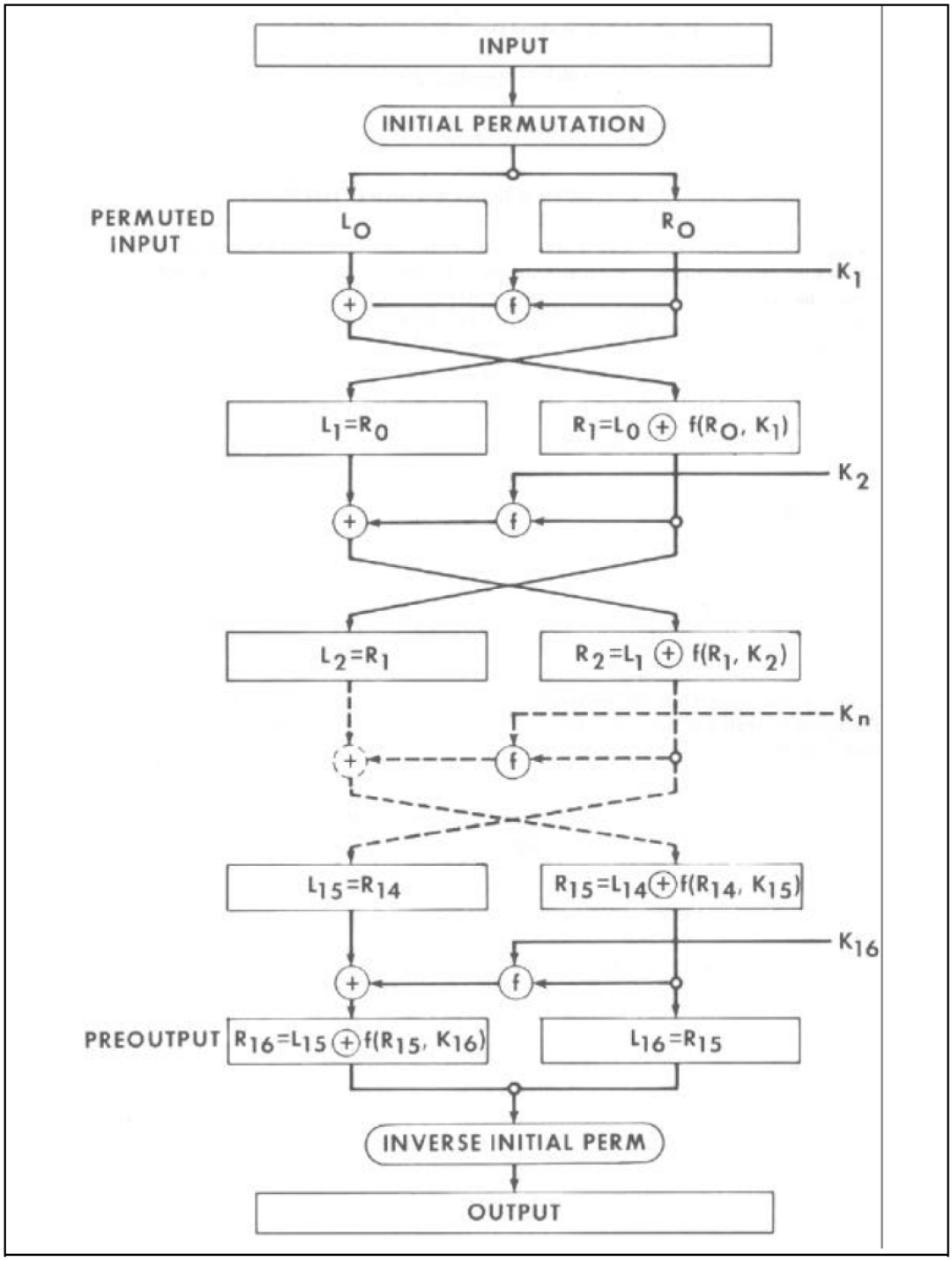
3.1.1 DATA ENCRYPTION STANDARD (DES)

Τα πρότυπα κρυπτογράφησης στοιχείων (DES), γνωστό ως αλγόριθμος κρυπτογράφησης στοιχείων (DEA) από το Ansi και το dea-1 από το ISO, είναι παγκόσμια πρότυπα για 20 έτη. Αν και παρουσιάζει σημάδια των γηρατειών, έχει κρατήσει ψηλά εντυπωσιακά καλά ενάντια στα έτη κρυπτολογικής ανάλυσης και είναι ακόμα ασφαλές ενάντια σε κάθε άλλο παρά ενδεχομένως τον ισχυρότερο των αντιπάλων.

Ο αλγόριθμος έχει ως σκοπό να υπολογίσει και να αποκρυπτογραφήσει τα blocks των στοιχείων που αποτελούνται από 64 μπιτ υπό έλεγχο εξηντατετράμπιτο κλειδί. Η αποκρυπτογράφηση πρέπει να ολοκληρωθεί με τη χρησιμοποίηση του ίδιου κλειδιού όπως για τον υπολογισμό, αλλά με το πρόγραμμα της διευθισιοδότισης τα bit του

κλειδίου άλλαξαν έτσι ώστε η διαδικασία αποκρυπτογράφησης είναι η αντιστροφή της διαδικασίας υπολογισμού. Ένα block που υπολογίζεται υποβάλλεται σε μια αρχική μεταλλαγή IP, κατόπιν σε έναν σύνθετο βασικός-εξαρτώμενο υπολογισμό και τελικά σε μια μεταλλαγή που είναι το αντίστροφο της αρχικής μεταλλαγής της IP. Ο βασικός-εξαρτώμενος υπολογισμός μπορεί να καθοριστεί απλά από μια λειτουργία f, αποκαλούμενη τη cipher λειτουργία, και λειτουργία KS, αποκαλούμενη το πρόγραμμα κλειδί. Μια περιγραφή του υπολογισμού δίνεται πρώτα, μαζί με τις λεπτομέρειες ως προς τον τρόπο με τον οποίο ο αλγόριθμος χρησιμοποιείται για encipherment. Έπειτα, η χρήση του αλγορίθμου για decipherment περιγράφεται. Τέλος, σε έναν καθορισμό της cipher λειτουργίας f δίνονται οι όροι των βασικών λειτουργιών που αποκαλούνται οι επιλεγμένες λειτουργίες Si και οι λειτουργίες μεταλλαγής P.

Πίνακας 3.1


INPUT
INITIAL PERMUTATION
PERMUTED INPUT
L_0
R_0
K_1
f
$L_1=R_0$
$R_1=L_0 \oplus f(R_0, K_1)$
K_2
$L_2=R_1$
$R_2=L_1 \oplus f(R_1, K_2)$
K_n
$L_{15}=R_{14}$
$R_{15}=L_{14} \oplus f(R_{14}, K_{15})$
K_{16}
PREOUTPUT
$R_{16}=L_{15} \oplus f(R_{15}, K_{16})$
$L_{16}=R_{15}$
INVERSE INITIAL PERM
OUTPUT

Η ακόλουθη σημείωση είναι επεξηγηματική : Δίνονται δύο block από bits, L και R. Η LR δείχνει το block που αποτελείται από τα κομμάτια του L που ακολουθούνται από τα κομμάτια του R. Δεδομένου ότι η αλληλουχία είναι συνειρμική,B1B2… B8, παραδείγματος χάριν, δείχνει το block που αποτελείται από τα bits του B1 που ακολουθούνται από τα bits του B2… ακολουθούμενα από τα bits του B8.

Υπολογισμός:

Ένα σκίτσο του υπολογισμού που δίνεται στον πίνακα 3.1:

Τα 64 μπιτ του block εισαγωγής που υπολογίζεται, υποβάλλονται αρχικά στην ακόλουθη μεταλλαγή, την αποκαλούμενη αρχική μεταλλαγή της IP:

IP

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

Αυτή η μεταλλαγμένη εισαγωγή έχει σαν πρώτο bit τα πρώτα 58 bit,σαν δεύτερο bit τα επόμενα 50 και τα λοιπά, με το bit 7 ως τελευταίο bit του. Το μεταλλαγμένο block εισαγωγής δεδομένων είναι η έπειτα εισαγωγή σε ένα σύνθετο κλειδί-εξαρτώμενου υπολογισμού που περιγράφεται παρακάτω. Η παραγωγή εκείνου του υπολογισμού, αποκαλούμενη preoutput, υποβάλλεται έπειτα στην ακόλουθη μεταλλαγή που είναι το αντίστροφο της αρχικής μεταλλαγής:

$\underline{IP^{-1}}$

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

Δηλαδή η εξαγωγή δεδομένων του αλγορίθμου έχει 40 bit του preoutbut block για τα πρώτα bit, τα επόμενα 8 bit για δευτερεύον bit, και τα λοιπά, μέχρι τα 25 bit του preoutput block να είναι τα τελευταία bit της εξαγωγής δεδομένων.

Ο υπολογισμός που χρησιμοποιεί το μεταλλαγμένο block εισαγωγής δεδομένων ως εισαγωγή του για να παραγάγει το block preoutput αποτελείται, απο μια τελική ανταλλαγή των block δεδομένων, 16 επαναλήψεων ενός υπολογισμού που περιγράφεται παρακάτω μέσω της cipher λειτουργία f που λειτουργεί σε δύο block. Ο ένας 32 bits και ο άλλος 48 bits, και παράγει έναν φραγμό 32 μπιτ.

Αφήστε τα 64 bit του block εισαγωγής σε μια επανάληψη να αποτελεσθούν από έναν τριανταδυάμπιτο block L που ακολουθείται από έναν τριανταδυάμπιτο φραγμό R. Χρησιμοποιώντας τη σημείωση που καθορίζεται στην εισαγωγή, το block εισαγωγής είναι έπειτα LR.

Υποθέστε ότι το K είναι ένα block 48 bits που επιλέγεται από το εξηντατετράμπιτο κλειδί. Κατόπιν η παραγωγή L'R μιας επανάληψης με την εισαγωγή LR καθορίζεται από:

(1)
$$L' = R$$
$$R' = L \oplus f(R,K)$$

όπου δείχνω bit ανά bit την προσθήκη modulo 2.

Όπως παρατηρήθηκε πριν, η εισαγωγή της πρώτης επανάληψης του υπολογισμού είναι το μεταλλαγμένο block εισαγωγής.

Εάν L'R είναι η παραγωγή της 16ης επανάληψης τότε το R'L είναι ο φραγμός preoutput. Σε κάθε επανάληψη ένα διαφορετικό block K από τα bits του κλειδιού επιλέγεται από ένα εξηντατετράμπιτο κλειδί που υποδεικνύεται από το ΚΛΕΙΔΊ.

Με περισσότερη προσοχή μπορούμε να περιγράψουμε τις επαναλήψεις του υπολογισμού λεπτομερέστερα. Υποθέτουμε ότι το KS είναι μια λειτουργία που παίρνει έναν ακέραιο αριθμό n στη σειρά από 1 έως 16 και ένα εξηντατετράμπιτο block ΚΛΕΙΔΙΟΥ ως εισαγωγή και παράγει ως έξοδο έναν block 48 bit που είναι μια μεταλλαγμένη επιλογή των bits από το ΚΛΕΙΔΊ.

(2) $$K_n = KS(n,KEY)$$

Αυτό είναι το Kn που καθορίζεται από τα bits σε 48 ευδιάκριτες θέσεις των bits του ΚΛΕΙΔΙΟΥ. KS καλείται βασικό πρόγραμμα επειδή ο φραγμός K που χρησιμοποιείται στην επανάληψη n'th της (1) είναι ο φραγμός Kn που καθορίζεται από την (2).

Όπως πριν, αφήστε το μεταλλαγμένο block εισαγωγής να είναι LR. Τέλος, αφήστε το ***L()***
και ***R()*** να είναι αντίστοιχα L και R και να αφήσουμε Ln και το Rn να είναι αντίστοιχα L' και R' (1) όταν το L και το R είναι αντίστοιχα ln-1 και Rn-1 και το K είναι Kn δηλαδή όταν το n είναι στη σειρά από 1 έως 16,

(3) $$L_n = R_{n-1}$$
$$R_n = L_{n-1} \oplus f(R_{n-1},K_n)$$

Το block preoutput είναι έπειτα R16L16.

Το πρόγραμμα του κλειδιού παράγει τα 16 Kn που απαιτούνται για τον αλγόριθμο.

Αποκρυπτογράφηση

Η μεταλλαγή ***IP-1*** που εφαρμόζεται στο block preoutput είναι το αντίστροφο της αρχικής μεταλλαγής IP που εφαρμόζεται στην εισαγωγή. Περαιτέρω, από (1),ακολουθεί ότι:

(4)
$$R = L'$$
$$L = R' \oplus f(L',K)$$

Συνεπώς, για να αποκρυπτογραφήσει είναι απαραίτητο να εφαρμοστεί ο ίδιος αλγόριθμος σε έναν υπολογισμένο block μηνυμάτων, και ότι θα προσέχει κάθε επανάληψη του υπολογισμού στο ίδιο block των bit του κλειδιού K που χρησιμοποιείται κατά τη διάρκεια decipherment όπως χρησιμοποιήθηκε κατά τη διάρκεια encipherment του block.Χρησιμοποιώντας τη σημείωση του προηγούμενης παραγράφου, αυτό μπορεί να εκφραστεί από τις εξισώσεις:

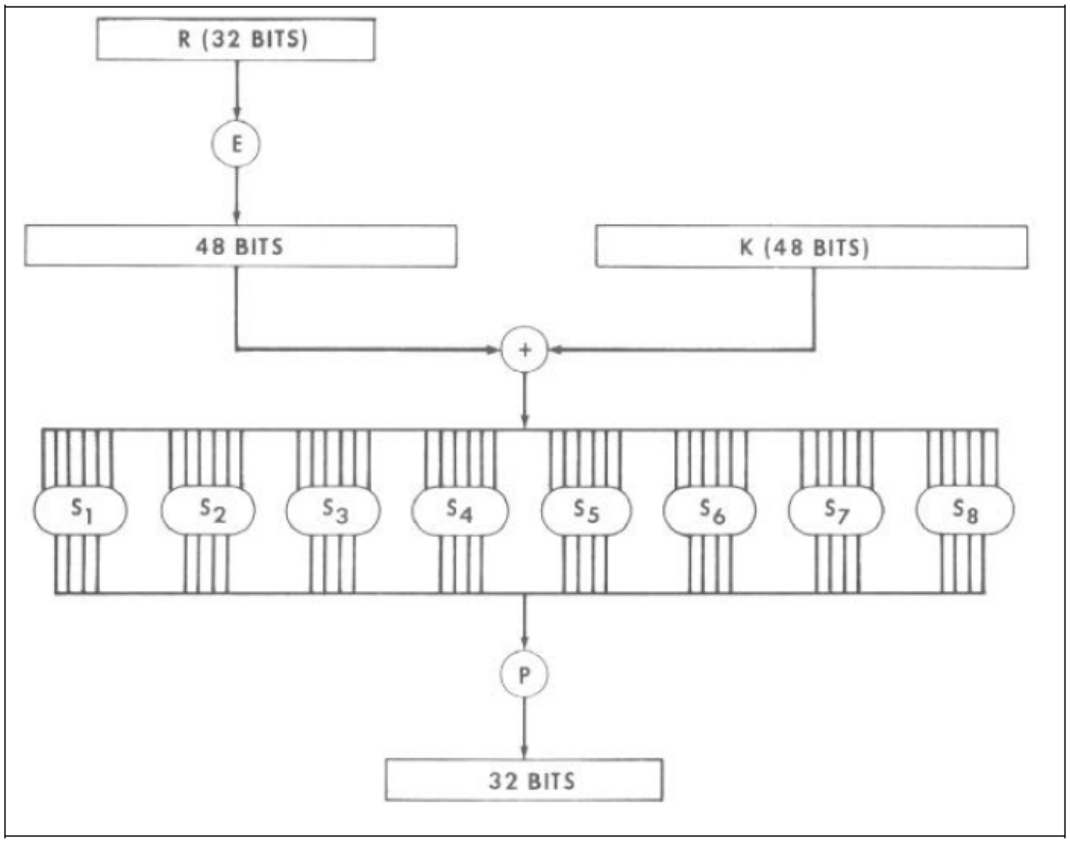
(5)
$$R_{n-1} = L_n$$
$$L_{n-1} = R_n \oplus f(L_n,K_n)$$

όπου τώρα R16L16 είναι το μεταλλαγμένο block εισαγωγής για τον υπολογισμό και το ***L0R0*** αποκρυπτογράφησης είναι το block preoutput. Δηλαδή για το decipherment υπολογισμό με R16L16 ως μεταλλαγμένη εισαγωγή το K16 χρησιμοποιείται στην πρώτη επανάληψη το K15 στη δεύτερη και τα λοιπά, με K1 χρησιμοποιημένο στη 16η επανάληψη.

Η Cipher λειτουργία f

Ένα σκίτσο του υπολογισμού του f (R, K) δίνεται στον πίνακα 3.2.

Πίνακας 3.2



Αν αφήσουμε το Ε να προσδιορίσει μια λειτουργία η οποία παίρνει ένα block 32 bit ως εισαγωγή δεδομένων και παράγει ένα block 48 bit ως εξαγωγή δεδομένων. Υποθέτουμε ότι το Ε είναι έτσι ώστε τα 48 bit της εξόδους του, γράφονται ως 8 blocks των 6 μπιτ ο κάθε ένας, λαμβάνοντας με την επιλογή των bits στην εισαγωγή δεδομένων του την διάταξη σύμφωνα με τον ακόλουθο πίνακα:

E BIT-SELECTION TABLE

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

Κατά συνέπεια τα πρώτα τρία μπιτ του E (R) είναι τα bits στις θέσεις 32, 1 και 2 του R ενώ τα τελευταία 2 bit του E (R) είναι τα bits στις θέσεις 32 και 1.

Κάθε μια από τη μοναδική επιλογή λειτουργείας S1, S2,…, S8, παίρνει ένα block 6 bit ως εισαγωγή και εξάγει έναν block 4 bit ως εξαγωγή και παρουσιάζεται με τη χρησιμοποίηση ενός πίνακα που περιέχει το S1:

***S*$_1$**

Column Number

Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Εάν S1 είναι η λειτουργία που καθορίζεται σε αυτόν τον πίνακα και το B είναι ένα block των 6 bit, τοτε το S1 (B) καθορίζεται ακολουθούθος: Τα πρώτα και τελευταία bit του B αντιπροσωπεύουν στη βάση 2 έναν αριθμό από 0 έως 3. Υποθέστε ότι είναι ο αριθμός i. Τα μεσαία 4 μπιτ του B αντιπροσωπεύουν στη βάση 2 έναν αριθμό από 0 έως 15. Υποθέστε ότι

ο αριθμός είναι ο j. Κοιτάξτε πάνω στον πίνακα τον αριθμό στη σειρά i και τη στήλη j. Είναι ένας αριθμός από το 0 έως 15 και αντιπροσωπεύεται μοναδικά από ένα block των 4 μπιτ. Αυτό το block είναι η εξαγωγή S1 (B) από το S1 από την εισαγωγή δεδομένων του B. Παραδείγματος χάριν, για την εισαγωγή 011011 η σειρά είναι 01, αυτή είναι σειρά 1, και η στήλη καθορίζεται από 1101, η οποία είναι η στήλη 13. Στη σειρά 1 στήλη 13 εμφανίζεται το 5 έτσι ώστε η εξαγωγή είναι το 0101.

Η λειτουργία P μεταλλαγής παράγει μια τριανταδυάμπιτη εξαγωγή από μια τριανταδυάμπιτη εισαγωγή δεδομένων με τη μεταλλαγή των bit του block εισαγωγής. Μια τέτοια λειτουργία καθορίζεται από τον ακόλουθο πίνακα:

P

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

Η εξαγωγή δεδομένων P (L) για τη λειτουργία P που καθορίζεται από αυτόν τον πίνακα λαμβάνεται από την εισαγωγή L με τη λήψη του 16ου bit του L ως το πρώτο bit του P (L), το 7ο bit ως το δεύτερο bit του P (L), και τα λοιπά έως ότου λαμβάνεται το 25ο bit του L ως 32ο bit του P (L).

Υποθέστε ότι S1,…, S8 είναι οκτώ αυστηρές λειτουργίες επιλογής, υποθέστε επίσεις το P να είναι η λειτουργία μεταλλαγής και το E να είναι η λειτουργία που καθορίζεται παραπάνω.

Για να καθορίσουμε το f (P, K) καθορίζουμε αρχικά τα B1,…, B8 για να είναι blocks των 6 bit το κάθε ένα από αυτά.

(6) $$B_1B_2...B_8 = K \oplus E(R)$$

Ο φραγμός f (P, K) καθορίζεται έπειτα να είναι

(7) $$P(S_1(B_1)S_2(B_2)...S_8(B_8))$$

Κατά συνέπεια το ***K E(R)*** διαιρείται πρώτα σε 8 blocks όπως υποδεικνύεται στο (6). Κατόπιν κάθε Bi λαμβάνεται ως εισαγωγή στο Si και τα 8 blocks S1 (B1), S2 (B2),…, S8 (B8) των 4 bit γινονται ένα ενιαίο Block των 32 bit που διαμορφώνει την εισαγωγή στο P. Η εξαγωγή του (7) είναι έπειτα η εξαγωγή της λειτουργίας f για τις εισαγωγές δεδομένων P και K.

ΤΡΙΠΛΟΣ ΑΛΓΟΡΙΘΜΟΣ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ ΔΕΔΟΜΕΝΩΝ

Υποθέστε ότι το EK(I) και το DK(I) να αντιπροσωπεύσουν την DES κρυπτογράφηση και αποκρυπτογράφηση του I χρησιμοποιώντας αντίστοιχα DES κλειδί K αντίστοιχα. Κάθε λειτουργία κρυπτογράφησης / αποκρυπτογράφησης TDEA (όπως διευκρινίζεται στο Ansi X9.52) είναι μια σύνθετη λειτουργία της διαδικασίας DES κρυπτογράφησης και αποκρυπτογράφησης . Οι ακόλουθες διαδικασίες χρησιμοποιούνται:

1. Λειτουργία κρυπτογράφησης TDEA: ο μετασχηματισμός ενός εξηντατετράμπιτου block I σε ένα εξηντατετράμπιτο block O που καθορίζεται ως εξής:

O = EK3(DK2(EK1(I))).

2. Λειτουργία κρυπτογράφησης TDEA: ο μετασχηματισμός ενός εξηντατετράμπιτου block I σε ένα εξηντατετράμπιτο block O που καθορίζεται ως εξής:

O = DK1(EK2(DK3(I)))

Τα πρότυπα διευκρινίζουν τις ακόλουθες επιλογές διαμόρφωσης για τα (K1, K2, K3)

1. Επιλογή διαμόρφωσης 1 : K1, K2 και K3 είναι ανεξάρτητα κλειδιά
2. Επιλογή διαμόρφωσης 2 : K1 και K2 είναι ανεξάρτητα κλειδιά και K3 = K1
3. Επιλογή διαμόρφωσης 3 : K1 = K2 = K3.

Ένας τρόπος λειτουργίας του TDEA λειτουργίας είναι ο οπίσθιος ο οποίος είναι συμβατός με το σύστημα counterpart του DES εάν είναι συμβατό με τις επιλογές διαμόρφωσης για τη λειτουργία TDEA,.

1. Ένα κρυπτογραφημένο plaintext κείμενο με τον DES μπορεί να αποκρυπτογραφηθεί σωστά από έναν αντίστοιχο τρόπο TDEA

2. Ένα κρυπτογραφημένο plaintext κείμενο με τον TDEA μπορεί να αποκρυπτογραφηθεί σωστά από έναν αντίστοιχο τρόπο DES

Κατά χρησιμοποίηση της επιλογής 3 διαμόρφωσης (K1 = K2 = K3), οι τρόποι TECB, TCBC, TCFB και TOFB είναι προς τα πίσω συμβατοί με τον τρόπο της DES λειτουργίας, CBC, CFB, OFB αντίστοιχα.

ΒΑΣΙΚΕΣ ΛΕΙΤΟΥΡΓΙΕΣ ΓΙΑ ΤΟΝ ΑΛΓΟΡΙΘΜΟ ΚΡΥΠΤΟΓΡΑΦΗΣΗΣ ΔΕΔΟΜΕΝΩΝ

Η επιλογή των βασικών λειτουργιών KS, S1,…, S8 και P είναι κρίσιμη για την δυναμική της κρυπτογράφησης ανάλογα τον αλγόριθμο. Διευκρινισμένο κατωτέρω είναι ενα συνιστώμενο σύνολο λειτουργιών, το οποιο περιγράφει τα S1,…, S8 και το P με τον ίδιο τρόπο που περιγράφονται στον αλγόριθμο

Οι βασικές λειτουργίες των S1,…, S8 είναι:

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

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

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

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

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

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

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

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

Η βασική λειτουργία του P είναι:

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

Έχουμε ότι για Kn, 1□*n*□16, είναι το block των 48 bit βλέπε(2) του αλγορίθμου. Ως εκ τούτου, για να περιγράψει το KS, είναι απαραίτητο να περιγράφει ο υπολογισμός Kn από το ΚΛΕΙΔΊ για v = 1, 2,…, 16.

Για να ολοκληρώθει ο καθορισμός του KS είναι σημαντικό να περιγραφούν οι δύο μεταλλαγμένες επιλογές, καθώς επίσης και το πρόγραμμα των αριστερών μετατοπίσεων. Κάθε bit σε κάθε 8 bytes του ΚΛΕΙΔΙΟΥ μπορεί να χρησιμοποιηθεί για την ανίχνευση λάθους στην δημιουργία κλειδιού , τη διανομή και την αποθήκευση. Τα bits 8, 16,…, 64 είναι για τη χρήση στη βεβαίωση ότι κάθε byte είναι σωστά κατανεμημένο.

Η μεταλλαγμένη επιλογή 1 καθορίζεται από τον ακόλουθο πίνακα:

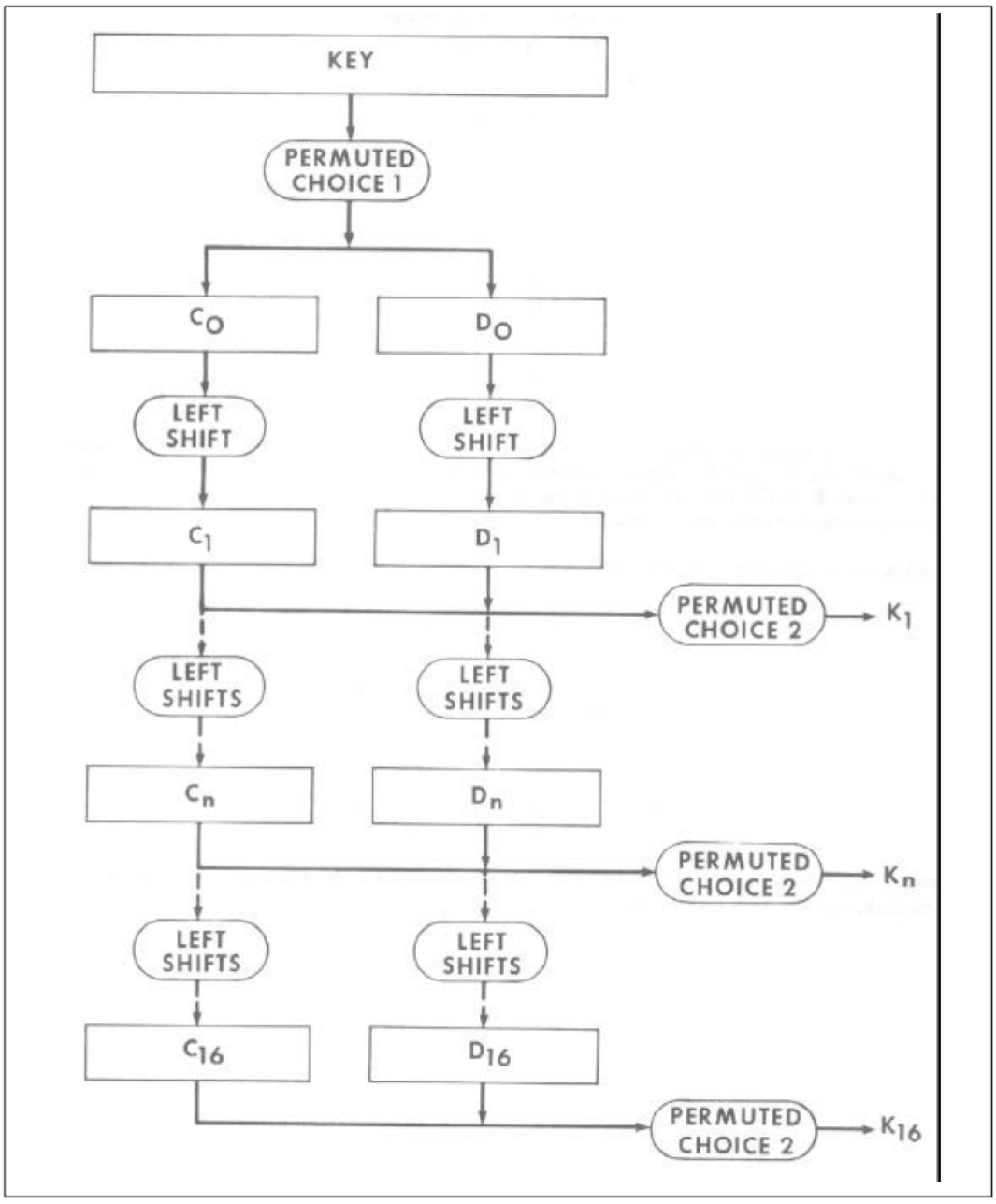
***PC*-1**

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

Ο πίνακας έχει διαιρεθεί σε δύο μέρη, με το πρώτο μέρος να καθορίζει πώς τα bit του ***C( )*** επιλέγονται, και το δεύτερο μέρος που καθορίζει πώς τα bit του ***D( )***επιλέγονται. Τα bits του ΚΛΕΙΔΙΟΥ είναι αριθμημένα από το 1 μέχρι το 64. Τα κομμάτια του ***C( )*** είναι τα αντίστοιχα bits 57, 49, 41,…, 44 και 36 του ΚΛΕΙΔΙΟΎ, με τα bits του ***D()***να είναι bits 63, 55, 47,…, 12 και 4 του ΚΛΕΙΔΙΟΎ.

Με το ***C( )*** και το ***D***( ) προσδιορισμένα, μπορουμε να προσδιορισουμε τώρα πώς τα blocks ***Cn*** και ***Dn*** λαμβάνονται από τα blocks ***Cn-1*** και ***Dn-1***, αντίστοιχα, για n = 1, 2,…, 16. Αυτό γινεται με το ακόλουθο πρόγραμμα των αριστερών μετατοπίσεων των συγκεκριμένων blocks:

Πίνακας 3.3


KEY
PERMUTED CHOICE 1
C_O
D_O
LEFT SHIFT
LEFT SHIFT
C_1
D_1
PERMUTED CHOICE 2
K_1
LEFT SHIFTS
LEFT SHIFTS
C_n
D_n
PERMUTED CHOICE 2
K_n
LEFT SHIFTS
LEFT SHIFTS
C_{16}
D_{16}
PERMUTED CHOICE 2
K_{16}

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

Παραδείγματος χάριν, τα C3 και D3 λαμβάνονται από τα C2 και D2, αντίστοιχα, από δύο αριστερές μετατοπίσεις, και τα C16 και D16 λαμβάνονται από τα C15 και D15, αντίστοιχα, από μια αριστερή μετατόπιση. Σε όλες τις περιπτώσεις, από μια ενιαία αριστερή μετατόπιση η οποία και σημαίνεται από μια περιστροφή των bits μια θέση στα αριστερά, έτσι ώστε μια αριστερή μετατόπιση των bits στις 28 αυτές θέσεις να είναι τα bits που ήταν προηγουμένως στις θέσεις 2, 3,…, 28, 1.

Η μεταλλαγμένη επιλογή 2 καθορίζεται από τον ακόλουθο πίνακα:

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Επομένως, το πρώτο bit του Kn είναι το 14ο bit του CnDn, το δεύτερο bit το 17ο, και τα λοιπά με το 47^{ο} bit, το 29ο, το 48ο και το 32^{ο} bit.

ΤΡΙΠΛΟ ΔΙΆΓΡΑΜΜΑ BLOCK DES

Λειτουργία κρυπτογράφησης TDEA:

$$I \rightarrow \boxed{DES\ E_{K1}} \rightarrow \boxed{DES\ D_{K2}} \rightarrow \boxed{DES\ E_{K3}} \rightarrow O$$

Λειτουργία αποκρυπτογράφησης TDEA:

$$I \rightarrow \boxed{DES\ D_{K3}} \rightarrow \boxed{DES\ E_{K2}} \rightarrow \boxed{DES\ D_{K1}} \rightarrow O$$

3.1.2 IDEA

- **Πρόλογος**

Το IDEA είναι block cipher που λειτουργεί με εξηντατετράμπιτα block plaintext. Το κλειδί είναι 128 bit . Ο ίδιος αλγόριθμος χρησιμοποιείται και για την κρυπτογράφηση και για την αποκρυπτογράφηση.Όπως με όλα άλλα block ciphers που έχουμε δει, το IDEA χρησιμοποιεί και τη σύγχυση και τη επίλυση .Η φιλοσοφία σχεδίου πίσω από τον αλγόριθμο είναι μιας φιλοσοφίας από «μίξης των διαδικασιών από τις διαφορετικές αλγεβρικές ομάδες.» Τρεις αλγεβρικές ομάδες αναμιγνύονται, και όλες εφαρμόζονται εύκολα και στο υλικό και στο λογισμικό:

— XOR
— Addition modulo 2^{16}
— Multiplication modulo $2^{16} + 1$.
IDEA's S-box.)

Όλες αυτές οι διαδικασίες (και αυτές είναι οι μόνες διαδικασίες στον αλγόριθμο είναι no bit-level μεταλλαγή) λειτουργούν στα δεκαεξάμπιτα υπο-block. Αυτός ο αλγόριθμος είναι ακόμα αποδοτικός και στους δεκαεξάμπιτους επεξεργαστές.

- **Περιγραφή του IDEA**

ΤΟ εξηντατετράμπιτο block δεδομένων διαιρείται σε τέσσερα υπο—block των 16-μπιτ: X1, X2, X3, και X4. Αυτά τα τέσσερα υπο—block γίνονται η εισαγωγή στον πρώτο κύκλο του αλγορίθμου. Υπάρχει ένα σύνολο οκτώ κύκλων. Σε κάθε κύκλο τα τέσσερα subblocks είναι XORed, προστιθέμενα, και πολλαπλασιασμένα το ένα με το άλλο και με έξι δεκαεξάμπιτα subkeys.Μεταξύ των κύκλων, ται δεύτερα και τα τρίτα υπο--block εναλλάσσονται. Τέλος, τα τέσσερα υπο--block συνδυάζονται με τέσσερα subkeys σε έναν μετασχηματισμό παραλαξης.. Σε κάθε κύκλο, η ακολουθία γεγονότων είναι η ακόλουθη:

1. Πολλαπλασιάστε X1 και το πρώτο subkey
2. Προσθέστε X2 και το δεύτερο subkey.
3. Προσθέστε X3 και το τρίτο subkey.
4. Πολλαπλασιάστε X4 και το τέταρτο subkey.

5. XOR τα αποτελέσματα των βημάτων (1) και (3).

6. XOR τα αποτελέσματα των βημάτων (2) και (4).

7. Πολλαπλασιάστε τα αποτελέσματα του βήματος (5) με το πέμπτο subkey.

8. Προσθέστε τα αποτελέσματα των βημάτων (6) και (7).

9. Πολλαπλασιάστε τα αποτελέσματα του βήματος (8) με το έκτο subkey.

10. Προσθέστε τα αποτελέσματα των βημάτων (7) και (9).

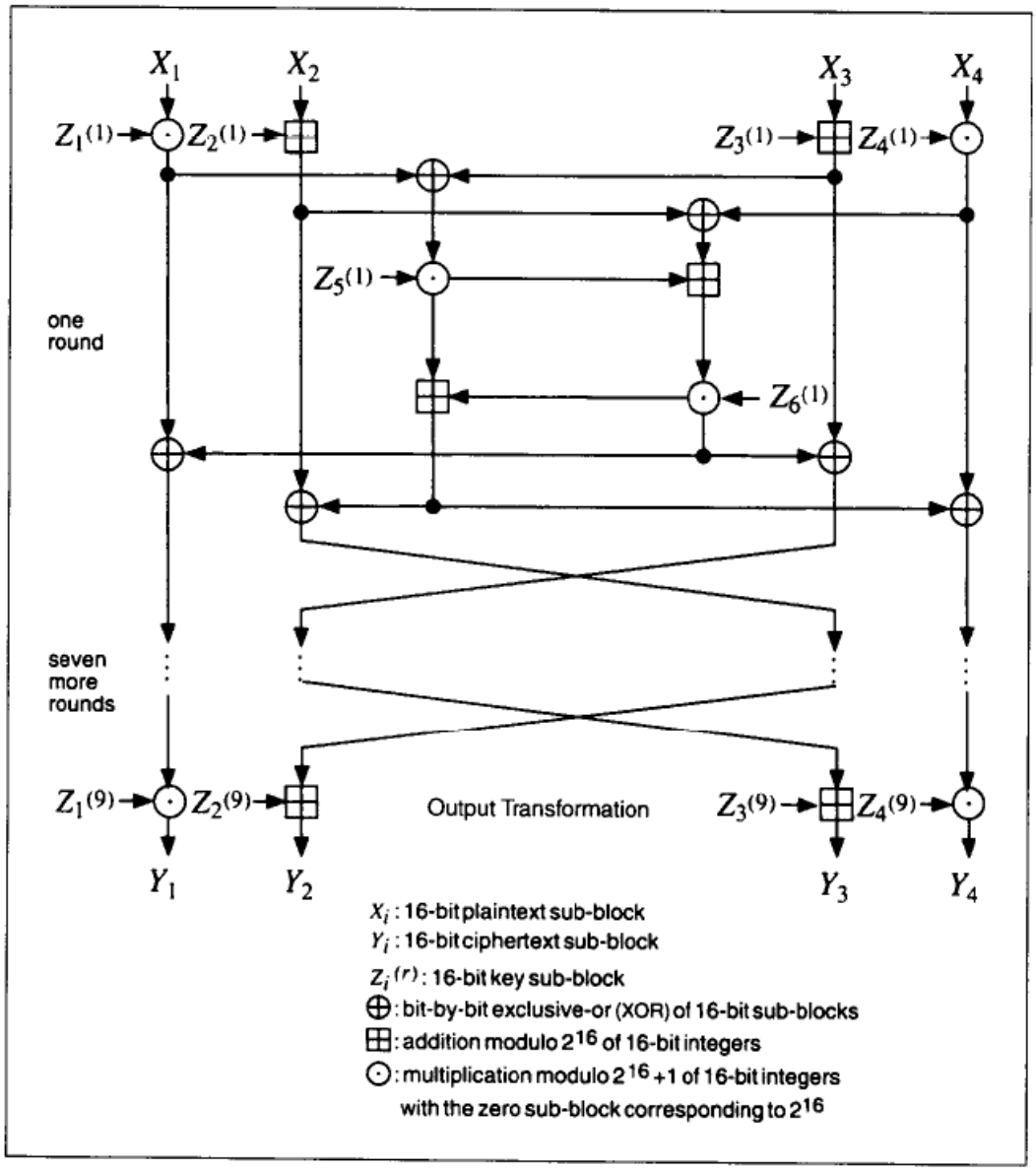
11. XOR τα αποτελέσματα των βημάτων (1) και (9).

12. XOR τα αποτελέσματα των βημάτων (3) και (9).

13. XOR τα αποτελέσματα των βημάτων (2) και (10).

14. XOR τα αποτελέσματα των βημάτων (4) και (10).

Πίνακας 3.4



Η παραγωγή του κύκλου είναι τέσσερα υπο--block που είναι τα αποτελέσματα των βημάτων (11),(12), (13), και (14). εναλλάξτε τα δύο εσωτερικά block (εκτός από τον τελευταίο κύκλο) ,και αυτή είναι η εισαγωγή στον επόμενο κύκλο. Μετά από τον όγδοο κύκλο, υπάρχει ένας τελικός μετασχηματισμός παραγωγής δεδομένων

1. Πολλαπλασιάστε X1 και το πρώτο subkey.
2. Προσθέστε X2 και το δεύτερο subkey.
3. Προσθέστε το X, και το τρίτο subkey.
4. Πολλαπλασιάστε το X, και το τέταρτο subkey.

Τέλος, τα τέσσερα υπο—block προσαρτούνται για να παραγάγουν το κρυπτογράφημα.

Η δημιουργία των subkeys είναι επίσης εύκολη. Ο αλγόριθμος χρησιμοποιεί 52 τους (έξι για κάθε έναν από τους οκτώ κύκλους και τέσσερα ακόμα για το μετασχηματισμό παραγωγής). Κατ' αρχάς, το εκατονεικοσαοκτάμπιτο κλειδί διαιρείται σε οκτώ δεκαεξάμπιτα subkeys. Αυτά είναι τα πρώτα οκτώ subkeys του αλγόριθμου (Τα έξι για τον πρώτο κύκλο, και πρώτα δύο για το δεύτερο κύκλο). Κατόπιν, το κλειδί περιστρέφεται 25 bit στο αριστερα και διαιρείται πάλι σε οκτώ subkeys. Τα τέσσερα πρώταχρησιμοποιούνται στην $2^{η}$ περιστροφή τελευταία τα τέσσερα χρησιμοποιούνται στην $3^{η}$. Το κλειδί περιστρέφεται άλλα 2.5bit στα αριστερά για τα επόμενα οκτώ subkeys, και τα λοιπά μέχρι το τέλος του αλγορίθμου.Η αποκρυπτογράφηση είναι ακριβώς η ίδια, εκτός από το ότι τα subkeys αντιστρέφονται και ελαφρώς διαφορετικα. Η αποκρυπτογράφηση των subkeys είναι είτε τα πρόσθετα είτε πολλαπλασιαστικά αντίστροφα της κρυπτογράφησης των subkeys.(Για τους σκοπούς του IDEA, το όλο μηδενικά sub-block θεωρείται οτι αντιπροσωπεύει το 2^{16} = -1 για πολλαπλασιασμό του modulo 2^{16} + 1 κατά συνέπεια το πολλαπλασιαστικόαντίστροφο 0 είναι 0.) Υπολογίζοντας αυτές παιρνει λιγο χρονο, αλλά εμείς πρέπει μόνο να το κάνουμεμιά φορά για κάθε κλειδί αποκρυπτογράφησης.

- **Κρυπτολογική ανάλυση του IDEA**

Το βασικό μήκος του IDEA είναι 128 bit δύο φορές μεγαλύτερος από τον DES. Υποθέτοντας ότι μια επίθεση bruteforce είναι η αποδοτικότερη, θα απαιτούσε 2^{128} (10^{38}) κρυπτογραφήσεις για να ανακτήσειτο κλειδί. Σχεδιάστε ένα τσιπ που μπορεί να εξετάσει δισεκατομμύριο κλειδιά ανά δευτερόλεπτο και να ρίξει ένα δισεκατομμύριο απόαυτα στο πρόβλημα, αυτό θα πάρει 10^{13} έτη -που είναι μεγαλύτερα από την ηλικίατου συμπατος. Μια σειρά 10^{24} τέτοιων τσιπ μπορεί να βρεί το κλειδί σε μια ημέρα, αλλά δεν υπάρχουν αρκετά άτομα πυριτίου στον κόσμο για να χτίσουν μια τέτοια μηχανή. Ίσως το brute force δεν είναι ο καλύτερος τρόπος να επιτεθείς στο IDEA. Ο αλγόριθμος είναι ακόμα πάρα πολύ νέος για οποιαδήποτε οριστικά cryptanalytic αποτελέσματα. Οι σχεδιαστές έχουν κάνει το καλύτερό τους για να καταστήσουν τον αλγόριθμο άνοσο στη διαφορική κρυπτολογική ανάλυση, καθόρισαν την έννοια Markov cipher και έδειξαν ότι η αντίσταση στη διαφορική κρυπτολογική ανάλυση μπορεί να διαμορφωθεί και να ποσολογηθεί [93 1.925]

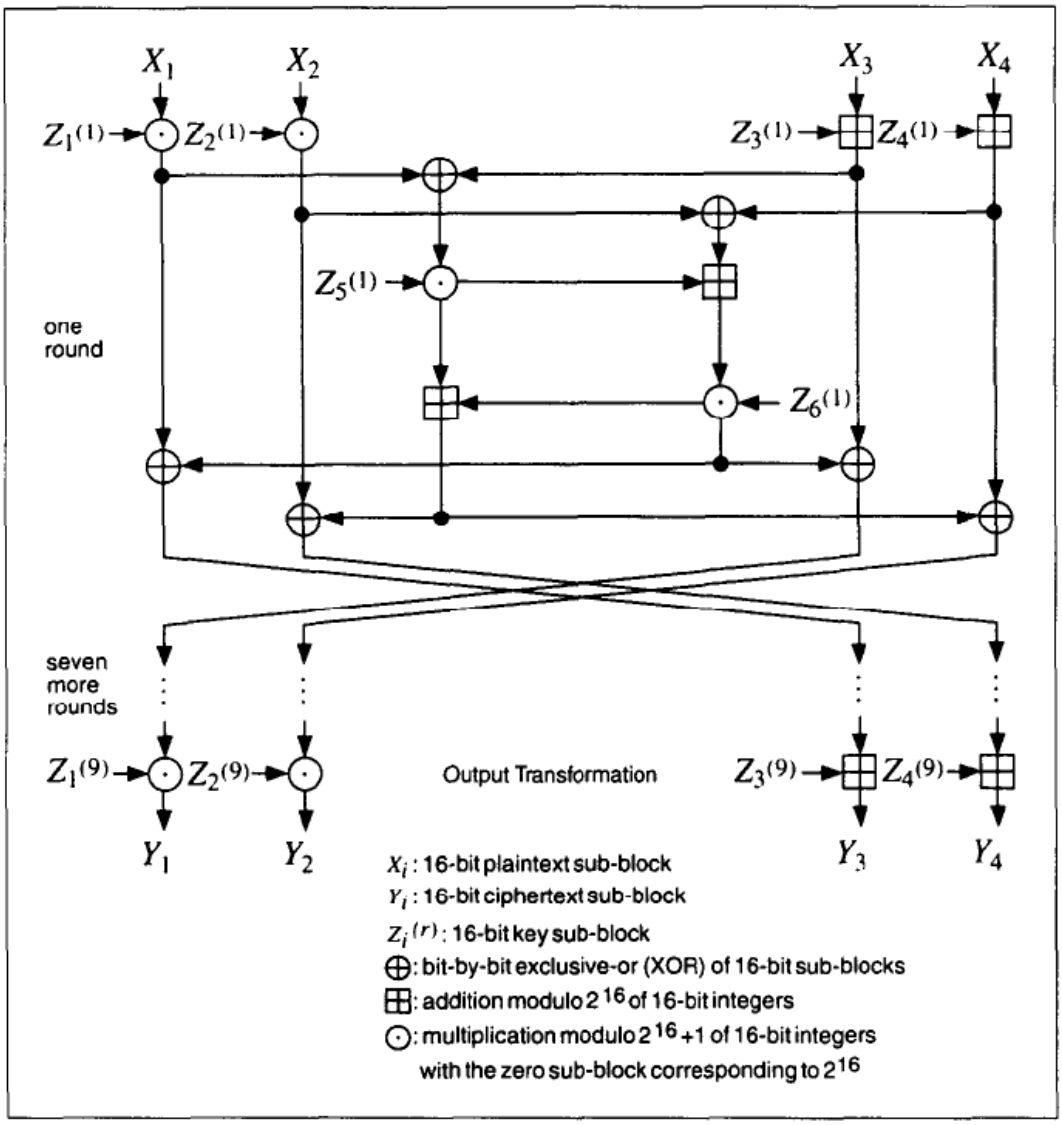
Καταπλήσσει πώς μερικές λεπτές αλλαγές μπορούν να κάνουν μια τέτοια μεγάλη διαφορά.) Στο [925], ο Lai υποστήριξε (έδωσε τα στοιχεία, όχι μια απόδειξη) ότι το Idea είναι άνοσο στη διαφορική κρυπτολογική ανάλυση μετά από 4 από τους 8 κύκλους του. Σύμφωνα με τον Biham, η με το συσχετιζομενο κλειδί cryptanalytic επίθεσή του δεν λειτουργεί ενάντια στο Idea.Ο Willi Meier εξέτασε τις τρεις αλγεβρικές διαδικασίες του Idea, και επισήμανε ότι ενώ είναι ασυμβίβαστοι, υπάρχουν περιπτώσεις όπου μπορούν να απλοποιηθούν με έναν τέτοιο τρόπο ώστε να διευκολυνθεί η κρυπτολογική ανάλυση σε κάποιο ποσοστό του χρόνου [1050]. Η επίθεσή του είναι αποδοτικότερη από brute-forceγια 2 κύκλους (λειτουργίας) του Idea (2^{42} διαδικασίες), αλλά λιγότερο αποδοτική για 3 κύκλους του Idea ή περισσοτέρους. Το κανονικό Idea, με 8 κύκλους (λειτουργίας), είναι ασφαλές.Ο Joan Daemen ανακάλυψε μια κατηγορία αδύνατων κλειδιών για το Idea [406.409]. Αυτά δεν είναι αδύνατα κλειδιά από την άποψη των αδύνατων κλειδιών DES δηλαδή η λειτουργία κρυπτογράφησης είναι selfinverse. Είναι αδύνατα υπό την έννοια ότι εάν χρησιμοποιούνται, ένας επιτιθέμενος μπορεί εύκολα να τα προσδιορίσει σε μια επιλεγμένη plaintext επίθεση.

Παραδείγματος χάριν, ένα αδύνατο κλειδί είναι:

oooo,oooo,oxoo,oooo,oooo,ooox,xxxx,xooo

Ο αριθμός στις θέσεις «του X» μπορεί να είναι οποιοσδήποτε αριθμός. Εάν αυτό το κλειδί χρησιμοποιείται, τοτε το XOR ορισμένων ζευγαριών plaintext τα bit του XOR των επακόλουθων ζευγαριών κρυπτογραφημάτων. Σε κάθε περίπτωση, η τυχαία περίπτωση να βρεθεί ένα από αυτά τα αδύνατα κλειδιά είναι πολύ μικρή: Μια στις 2^{96}. Δεν υπάρχει κανένας κίνδυνος εάν επιλέγετε τα κλειδιά τυχαία. Και είναι εύκολο να τροποποιηθεί το Idea έτσι ώστε να μην έχει οποιαδήποτε αδύνατα κλειδιά: Κάνουμε XOR κάθε subkey με την αξία OxOdae [409].

Πίνακας 3.5



- **IDEA τρόποι λειτουργίας και παραλλαγών**

Το Idea μπορεί να λειτουργήσει με οποιοδήποτε cipher block. Εντούτοις, επειδή το βασικό μήκος του Idea είναι περισσότερο από το διπλάσιο του DES, η επίθεση είναι μη πρακτική. Θα απαιτούσε ένα χώρο αποθήκευσης 10^{39} bytes. Δεν υπάρχει επίσης κανένας λόγος για τον οποίο δεν μπορούσατε να εφαρμόσετε στο Idea ανεξάρτητα subkeys, ειδικά εάν έχετε key-management εργαλεία για να χειριστείτε το μεγαλύτερο δυνατό κλειδί. Το Idea χρειάζεται συνολικά 52 δεκαεξάμπιτα κλειδιά, για ένα συνολικό βασικό μήκος 832 μπιτ. Αυτή η παραλλαγή είναι σίγουρα ασφαλέστερη, αλλά κανένας δεν ξέρει για πόσο.

3.1.3 Blowfish

Το Blowfish είναι ένα εξηντατετράμπιτο block cipher με ένα κλειδί μεταβλητού-μήκους. Ο αλγόριθμος αποτελείται από δύο μέρη: Την επέκταση-μήκος του κλειδιού και κρυπτογράφηση στοιχείων. Η επέκταση μετατρέπει ένα κλειδί από 448 bit σε διάφορες σειρές subkeys που συμπληρώνουν συνολικά 4168 bytes.Η κρυπτογράφηση στοιχείων αποτελείται από μια απλή λειτουργία που επαναλαμβάνεται 16 φορές. Κάθε κύκλος αποτελείται από ένα κλειδί-εξαρτώμενο-μεταλλαγής, και από ένα κλειδί με στοιχείο-εξαρτώμενη αντικατάσταση. Όλες οι διαδικασίες είναι προσθήκες XORs με τριανταδυάμπιτες λέξεις. Οι μόνες πρόσθετες διαδικασίες είναι τέσσερις συνταγμένες αναζητήσεις στοιχείων σειράς ανά κύκλο.Το Blowfish χρησιμοποιεί έναν μεγάλο αριθμό subkeys. Αυτά τα κλειδιά πρέπει να είναι υπολογισμένα πριν από οποιαδήποτε δεδομένα κρυπτογραφηθούν ή αποκρυπτογραφηθούν.

Η P-σειρά αποτελείται από 18 τριανταδυάμπιτα subkeys:

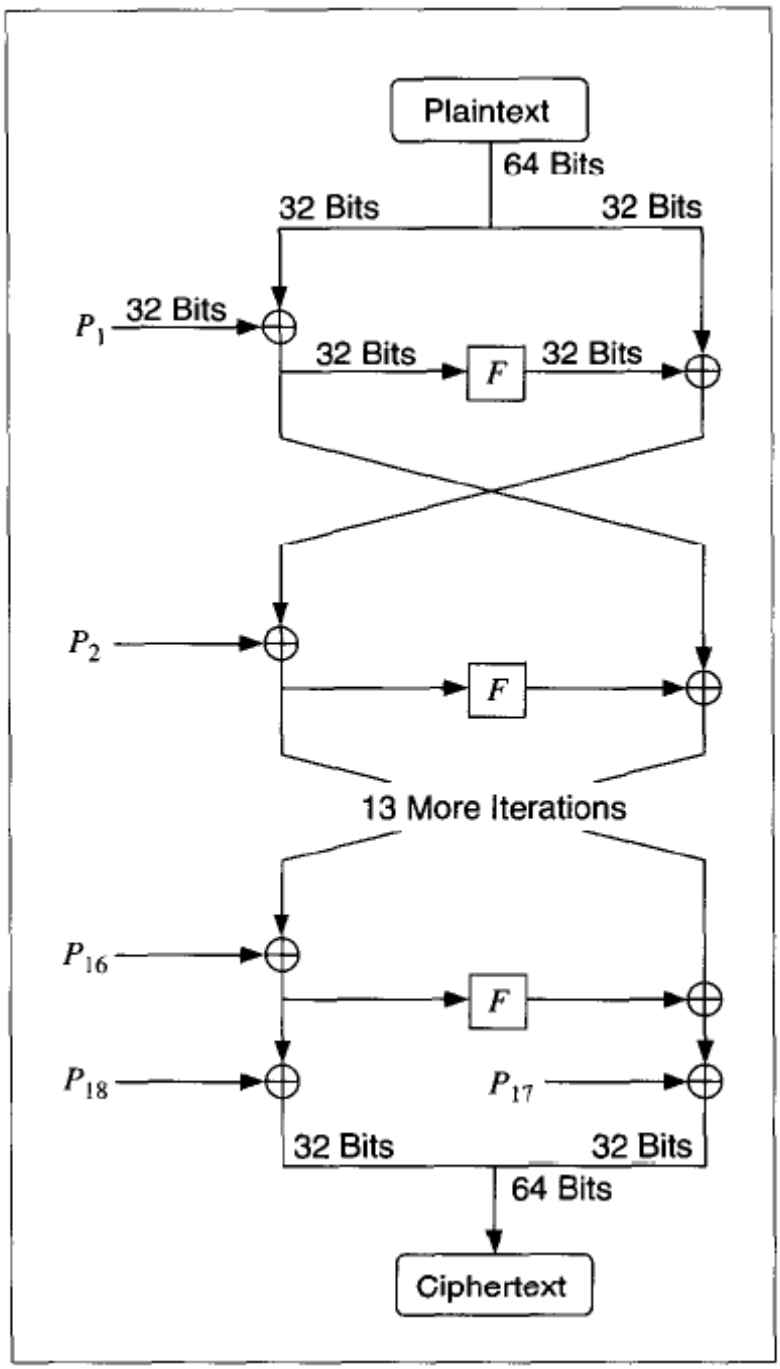
$$P_1, P_2, \ldots, P_{18}$$

Τέσσερα τριανταδυάμπιτα S-boxes έχουν 256 καταχωρήσεις το κάθε ένα:

$$S_{1,0}, S_{1,1}, \ldots, S_{1,255}$$
$$S_{2,0}, S_{2,1}, \ldots, S_{2,255}$$
$$S_{3,0}, S_{3,1}, \ldots, S_{3,255}$$
$$S_{4,0}, S_{4,1}, \ldots, S_{4,255}$$

Η ακριβής μέθοδος που χρησιμοποιείται για να υπολογίσει αυτά τα subkeys θα περιγραφεί αργότερα σε αυτό το τμήμα.

Πίνακας 3.6



Το Blowfish είναι ένα δίκτυο Feistel που αποτελείται από 16 κύκλους. Η εισαγωγή είναι ένα εξηντατετράμπιτο στοιχείο, Χ. Για να κρυπτογραφήσει:

Διαιρέστε το Χ σε δύο τριανταδυάμπιτα μισά: $X_L,\ X_R$

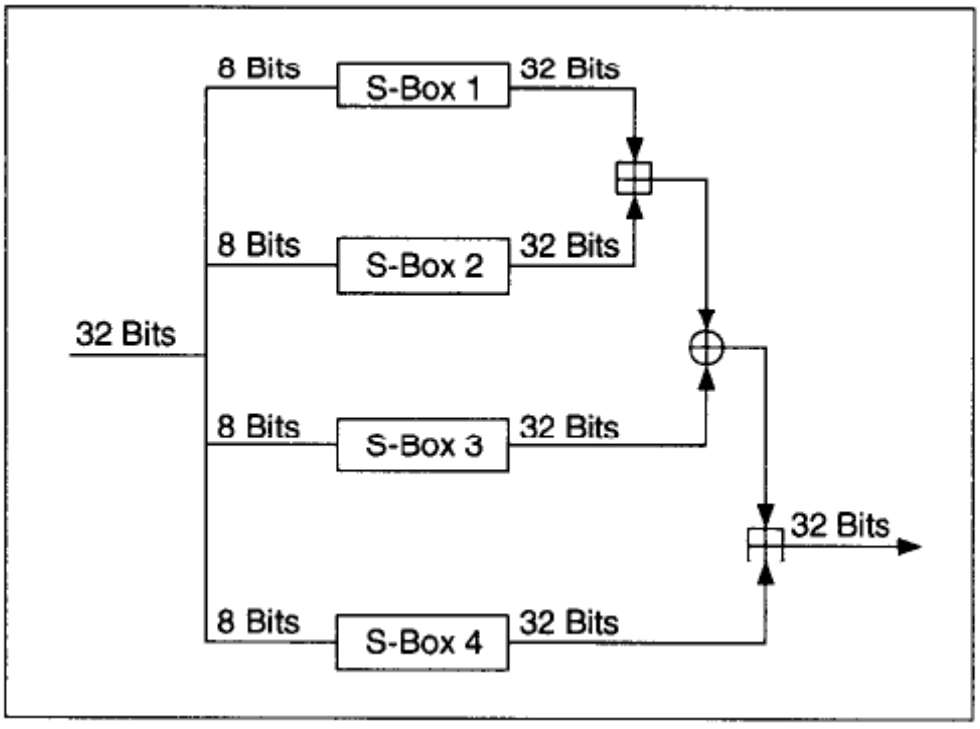
$$x_L = x_L \oplus P_i$$
$$x_R = \mathrm{F}(x_L) \oplus x_R$$
$$\text{Swap } x_L \text{ and } x_R$$

Ξανά αλλάξετε τα x_L, x_R

$$x_R = x_R \oplus P_{17}$$
$$x_L = x_L \oplus P_{18}$$

Επανασυνδυάστε τα x_L, x_R

Πίνακας 3.7



Η λειτουργία F είναι η ακόλουθη:

Διαίρεστε την x_L σε τέσσερα 8 bita τέταρτα:

$$a, b, c, \text{ and } d \quad \mathrm{F}(x_L) = ((S_{1,a} + S_{2,b} \bmod 2^{32}) \oplus S_{3,c}) + S_{4,d} \bmod 2^{32}$$

Η αποκρυπτογράφηση είναι ακριβώς η ίδια με την κρυπτογράφηση, εκτός από τα P1, P2. . . , P18 που χρησιμοποιούνται αντίστροφα. Οι εφαρμογές Blowfish που απαιτούν τις γρηγορότερες ταχύτητες πρέπει να ξετυλίξουν το βρόχο και να εξασφαλίσουν ότι όλα τα subkeys αποθηκεύονται στην μνήμη cache. Τα subkeys υπολογίζονται χρησιμοποιώντας τον αλγόριθμο Blowfish. Η ακριβής μέθοδος είναι αυτή που ακολουθεί.

1. Μονογράψτε πρώτα την P-σειρά και έπειτα τέσσερα S-boxes, στη σειρα, σε ένα συγκεκριμένο string. Αυτή η σειρά αποτελείται από τα δεκαεξαδικά ψηφία του π.

2. XOR P1 με τα πρώτα 32 μπιτ του κλειδιού, XOR το P2 με τα δεύτερα 32 μπιτ του κλειδιού ,και τα λοιπά για όλα τα κομμάτια του κλειδιού (μέχρι το P18). Ο κύκλος θα επαναλαμβάνεται στα bits των κλειδιών μέχρι την ολόκληρη η P-σειρά να γινει XORed με των κλειδιων τα bits.

3. Κρυπτογραφήστε την all-zero string με τον αλγόριθμο Blowfish, χρησιμοποιώντας τα subkeys που περιγράφονται στα βήματα (1) και (2).

Αντικαταστήστε το P1 και P2 με το αποτέλεσμα του βήματος (3)

4. Κρυπτογραφήστε το αποτέλεσμα του βήματος (3) χρησιμοποιώντας τον αλγόριθμο Blowfish με τροποποιημένα subkeys.

5. Αντικαταστήστε P3 και P4 με το αποτέλεσμα του βήματος (5).

6. Συνεχίστε τη διαδικασία, αντικαθιστοντας όλα τα στοιχεία της P-σειράς, και έπειτα τα
7. τέσσερα S-boxes στην σειρά , με τα αποτελέσματα του συνεχώς μεταβαλλόμενου αλγορίθμου Blowfish.

Στο σύνολο, 521 επαναλήψεις απαιτούνται για να παραγουν όλα που απαιτούμενα subkeys. Οι εφαρμογές μπορούν να υποθηκεύσουν τα subkeys-εκεί δεν είναι ανάγκη να εκτελεστεί αυτή η διαδικασία πολλές φορές.

Ασφάλεια του Blowfish

Ο Serge Vaudenay εξέτασε τον Blowfish με τα γνωστά s-boxes και τις ρουτίνες r. Μια διαφορική επίθεση μπορεί να ανακτήσει την P-σειρά με 2^{8r+1} με επιλεγμένα plaintexts [1568]. Για ορισμένα αδύνατα κλειδιά που παράγουν κακά S-boxes (οι πιθανότητες να πάρουν αυτα τυχαία είναι 1 στις 2^{14}),η ίδια επίθεση απαιτεί μόνο 2^{4r+1} επιλεγμένα plaintexts να ανακτήσει την P- σειρά. Με τα άγνωστα S-boxes αυτή η επίθεση μπορεί να ανιχνεύσει εάν ένα αδύνατο κλειδί χρησιμοποιείται, αλλά δεν μπορεί να καθορίσει τι είναι (ούτε τα S-boxes ούτε η P-σειρά). Αυτή η επίθεση λειτουργεί μόνο ενάντια στις μειωμένες-στρογγυλές μεταβλητές είναι απολύτως αναποτελεσματικό ενάντια στους 16 κύκλους εργασίας του Blowfish. Φυσικά, η ανακάλυψη των αδύνατων κλειδιών είναι σημαντική, ακόμα κι αν φαίνονται αδύνατο να εκμεταλλευτούν. Ένα αδύνατο κλειδί είναι ένα στο οποίο δύο από τις καταχωρήσεις για το S-box είναι ίδιες. Δεν υπάρχει κανένας τρόπος να ελέγξει για τα αδύνατα κλειδιά πριν ελέγξει το κυρίως κλειδί. Για να είστε ασφαλής, μην εφαρμόστε στον Blowfish έναν μειωμένο αριθμό κύκλων.

3.1.4 RC5

Το RC5 είναι ένα block cipher με ποικίλες παραμέτρους: μέγεθος block, μέγεθος κλειδιού, και αριθμός κύκλων. Εφευρέθηκε από Ron Rivest και αναλύθηκε από τα εργαστήρια RSA [1324.1325].

Υπάρχουν τρεις διαδικασίες XOR, η προσθήκη, και οι περιστροφές. Οι περιστροφές είναι συνεχόμενες διαδικασίες στους περισσότερους επεξεργαστές και οι μεταβλητές περιστροφές είναι μια μη γραμμική λειτουργία. Αυτές οι περιστροφές, που εξαρτώνται και από το κλειδί και από τα στοιχεία, είναι η ενδιαφέρουσα λειτουργία.
RC5 έχει ένα block μεταβλητού-μήκους, αλλά αυτό το παράδειγμα θα εστιαστεί σε ένα εξηντατετράμπιτο block δεδομένων. Η κρυπτογράφηση χρησιμοποιεί ένα κλειδι 2r + 2 εξαρτώμενης τριανταδυάμπιτης λέξης $S_0, S_1, S_2, \ldots, S_{2r+1}$ οπου r είναι ο αριθμός των κύκλων. Για να κρυπτογραφήσει, διαιρεί αρχικά το block plaintext σε δύο τριανταδυάμπιτες λέξεις: A και B. (Το RC5 κάνει μια μικρή αλλαγή , τα bytes σε λέξεις: Η πρώτη ψηφιολέξη πηγαίνει στα loworder bits της καταχώρισης A, κ.λπ.)
Έτσι :

$$A = A + S_0$$
$$B = B + S_1$$

For $i = 1$ to r:

$$A = ((A \oplus B) <<< B) + S_{2i}$$
$$B = ((B \oplus A) <<< A) + S_{2i+1}$$

Το αποτέλεσμα είναι στους καταχωριστές A και B.

Η αποκρυπτογράφηση είναι εξίσου εύκολη. Διαιρέστε το block plaintext σε δύο λέξεις, A και B, και έπειτα:

For $i = r$ down to 1:

$$B = ((B - S_{2i+1}) >>> A) \oplus A$$
$$A = ((A - S_{2i}) >>> B) \oplus B$$

$$B = B - S_1$$
$$A = A - S_0$$

Το σύμβολο «>>>» είναι μια σωστή κυκλική μετατόπιση. Φυσικά, όλες οι προσθήκες και οι αφαιρέσεις είναι mod 2^{32}.

Η δημιουργία της σειράς κλειδιών είναι πιό περίπλοκη, αλλά και απλή. Κατ' αρχάς, αντιγράψτε τις bytes του κλειδιού σε μια σειρά, L, των τριανταδυάμπιτων λέξεων c , γεμίζοντας την τελική λέξη με τα μηδενικά εάν είναι απαραίτητο. Κατόπιν, μονογράψτε μια σειρά, S, χρησιμοποιώντας μια γραμμική γεννήτρια mod 2^{32}.

$$S_0 = \mathrm{P}$$

for $i = 1$ to $2(r+1) - 1$:

$$S_i = (S_{i-1} + \mathrm{Q}) \bmod 2^{32}$$

Τα P = Oxb7e15163 και Q = Ox9e3779b9 είναι ι σταθερές που είναι βασισμένες στη δυαδική αντιπροσώπευση του e και phi.

Τέλος αναμειγνύουμε το L στο S.

$$i = j = 0$$
$$A = B = 0$$

Κάνει 3n φορές (οπού το n είναι το μέγιστο του 2(r+1) και του c)

$$A = S_i = (S_i + A + B) <<< 3$$
$$B = L_j = (L_j + A + B) <<< (A + B)$$
$$i = (i + 1) \bmod 2(r + 1)$$
$$j = (j + 1) \bmod c$$

Το RC5 είναι βασικά μια οικογένεια αλγορίθμων. Καθορίσαμε ακριβώς RC5 με ένα τριανταδυάμπιτο μέγεθος λέξης και έναν εξηντατετράμπιτο block δεν υπάρχει κανένας λόγος για τον οποίο ο ίδιος αλγόριθμος δεν μπορεί να έχει ένα εξηντατετράμπιτο μέγεθος λέξης και ενα εκατονεικοσαοκτάμπιτο μέγεθος block. Για w=24 τα P και Q είναι Oxb7e15 1628aed2a6b και Ox9e3779b97f4a7c15 αντίστοιχα. Το Rivest υποδεικνύει τις ιδιαίτερες εφαρμογές του RC5 ως RC5-w/r/b, όπου το W είναι το μέγεθος λέξης, r είναι ο αριθμός κύκλων, και το b είναι το μήκος του κλειδιού σε bytes.O RC5 είναι νέος, αλλά τα εργαστήρια στο RSA έχουν ξοδεψει τον αρκετο χρόνο που αναλύοντας τον με ένα εξηντατετράμπιτο block. Μετά από 5 κύκλους, οι στατιστικές φαίνονται πολύ καλές. Μετά από 8 κύκλους, το bit plaintext έχει επιπτώσεις τουλάχιστον σε μια περιστροφή. Υπάρχει μια διαφορική επίθεση που απαιτεί 2^{24} επιλεγμένα plaintexts για 5 κύκλους, 2^{45} για 10 κύκλους, 2^{53} για 12 κύκλους, και 2^{68} για 15 κύκλους. Φυσικά, υπάρχουν μόνο 2^{64} πιθανά επιλεγμένα plaintexts, έτσι αυτή η επίθεση δεν θα λειτουργήσει για 15 ή περισσότερους κύκλους. Οι γραμμικές εκτιμήσεις κρυπτολογικής ανάλυσης δείχνουν ότι είναι ασφαλής μετά από 6 κύκλους. Το Rivest συστήνει τουλάχιστον 12 κύκλους, και ενδεχομένως 16.
Αυτό το νουμερο πιθανος να αλλαξει.

3.1.5 RSA

Το RSA παίρνει την ασφάλειά του από τη δυσκολία δημιουργίας μεγάλων αριθμών. Τα δημόσια και ιδιωτικά κλειδιά (The public and private keys) είναι λειτουργίες ζευγαριών μεγάλων (100 έως 200 ψηφία ή ακόμα και μεγαλύτερα) βασικών αριθμών. Η ανάκτηση του plaintext από ένα δημόσιο κλειδί και το κρυπτογράφημα

υποτίθεται πως θα ισοδυναμεί με την δημιουργία του προϊόντος (ζητούμενο κλειδί) από δυο άλλα βασικά.
Για να παραγάγετε τα δύο κλειδιά, επιλέξτε δύο τυχαίους μεγάλους πρωταρχικούς αριθμούς, τους p και q, και για μέγιστη ασφάλεια επιλέξτε να έχουν το ίδιο μήκος. Υπολογίστε το αποτέλεσμα :

n=pq

Κατόπιν επιλέξτε τυχαία το κλειδί κρυπτογράφησης, e , έτσι ώστε το e και (p - 1) (q - 1) να είναι σχετικά μεταξύ τους (βασικά κλειδιά). Τέλος, χρησιμοποιήστε τον εκτεταμένο Euclidean αλγόριθμο για να υπολογίσετε το κλειδί αποκρυπτογράφησης, d , έτσι ώστε

ed= 1 mod(p- l)(q- 1)

με άλλα λόγια

d = e-l mod ((p - l)(q - 1))

Σημειώστε επίσης ότι οι αριθμοί d και n είναι επίσης συσχετιζόμενα πρωταρχικά (κλειδιά). Οι αριθμοί e και n είναι το δημόσιο κλειδί ο αριθμός d είναι το ιδιωτικό κλειδί. Τα δύο πρωταρχικά κλειδιά, το p και το q, δεν χρειάζονται πλέον. Δεν πρέπει να απορριφθούν, αλλά ούτε και να αποκαλυφθούν. Για να κρυπτογραφήσετε ένα μήνυμα m, πρώτα διαιρέστε αυτό στους αριθμητικά block μικρότερα από το n (με τα δυαδικά στοιχεία, επιλέξτε το μεγαλύτερο δυνατό από τα 2 και μικρότερο από το n).
Δηλαδή εάν και το p και το q είναι βασικά 100-ψηφία , τότε το n θα έχει κάτω από 200 ψηφία και κάθε block μηνυμάτων, m, θα πρέπει να έχει πάνω από 200 ψηφία .
(Εάν πρέπει να κρυπτογραφήσετε έναν σταθερό αριθμό block, μπορείτε να τους γεμίσετε με μερικά μηδενικά από τα αριστερα για να εξασφαλίσετε ότι θα είναι πάντα λιγότερα από n.). Το κρυπτογραφημένο μήνυμα, c, αποτελείται από τα ομοίως μεγέθους blocks μηνυμάτων, Ci, περίπου του ίδιου μήκους. Ο τύπος κρυπτογράφησης είναι απλά :

$$c_i = m_i^e \bmod n$$

Για να αποκρυπτογραφήσετε ένα μήνυμα, πάρτε κάθε κρυπτογραφημένο block Ci και υπολογίστε

$$m_i = c_i^d \bmod n$$

Οπότε

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i; \text{ all } (\bmod n)$$

Ο τύπος επαναχτεί το μήνυμα .

Το μήνυμα θα μπορούσε εύκολα να έχει κρυπτογραφηθεί με το d και έχει αποκρυπτογραφηθεί με το e. η επιλογή είναι ελεύθερη. Ένα σύντομο παράδειγμα θα το αρκετά πιο κατανοητό. Εάν p = 47 και q= 71, τότε

Δημόσιο κλειδί (Puplic key)

n το αποτέλεσμα των δυο πρωτευόντων κλειδιών, p και q (όπου το p και το q πρέπει να παραμείνουν μυστικά).

e το σχετικά πρωταρχικό στα (p - 1)(q - 1)

Ιδιωτικό κλειδι (Private key)

d $e^{-1} \bmod ((p-1)(q-1))$

Κρυπτογράφηση (Encrypting)

$$c = m^e \bmod n$$

Αποκρυπτογράφηση (Decrypting)

$$m = c^d \bmod n$$

n=pq=3337

Το κλειδί κρυπτογράφησης, ε, δεν πρέπει να έχει κανέναν παράγοντα κοινό με

$$(p-1)(q-1) = 46 * 70 = 3220$$

Επιλέξτε το e (τυχαία) να είναι το 79. Σε αυτή την περίπτωση

$$d = 79^{-1} \bmod 3220 = 1019$$

Αυτός ο αριθμός υπολογίστηκε χρησιμοποιώντας τον εκτεταμένο Euclidean αλγόριθμο. Δημοσιεύστε το e και το n, και κρατήστε το d μυστικό. Απορρίψτε το p και το q.

Για να κρυπτογραφήσει το μήνυμα πρώτα το σπάμε σε μικρά blocks

m=6882326879666683

Τα τριψήφια blocks δουλεύουν ωραία σε αυτήν την περίπτωση. Το μήνυμα είναι χωρισμένο σε έξι blocks, m, στα οποία:

$$m_1 = 688$$
$$m_2 = 232$$
$$m_3 = 687$$
$$m_4 = 966$$
$$m_5 = 668$$
$$m_6 = 003$$

Ο πρώτος φραγμός κρυπτογραφείται όπως παρακάτω :

$$688^{79} \bmod 3337 = 1570 = c_1$$

Η εκτέλεση της ίδιας λειτουργίας στα επόμενα block παράγει ένα κρυπτογραφημένο μήνυμα:

$$c = 1570\ 2756\ 2091\ 2276\ 2423\ 158$$

Η αποκρυπτογράφηση του μηνύματος απαιτεί το ίδιο exponentiation χρησιμοποιώντας το κλειδί αποκρυπτογράφησης 1019, έτσι

$$1570^{1019} \bmod 3337 = 688 = m_1$$

Το υπόλοιπο του μηνύματος μπορεί να ανακτηθεί κατά αυτόν τον τρόπο.

- **Η ασφάλεια του RSA**

Η ασφάλεια του RSA εξαρτάται πλήρως από το πρόβλημα της δημιουργίας μεγάλων αριθμών. Τεχνικά, αυτό είναι ένα ψέμα. Υποτίθεται ότι η ασφάλεια του RSA εξαρτάται από το πρόβλημα δημιουργίας μεγάλων αριθμών. Δεν έχει αποδειχθεί ποτέ από μαθηματική άποψη ότι χρειάζεστε τον παράγοντα n για να υπολογίσετε το m από το c και το e. Είναι κατανοητό ότι ένας εξ ολοκλήρου διαφορετικός τρόπος για την κρυπτανάλυση του RSA μπορεί να ανακαλυφθεί . Εντούτοις, εάν αυτός ο νέος τρόπος επιτρέπει στην κρυπτανάλυση να μεγαλώνει το d, θα μπορούσε επίσης να χρησιμοποιηθεί ως νέος τρόπος παραγωγής μεγάλων αριθμών**.**

Είναι επίσης δυνατό να επιτεθούμε στο RSA μαντεύοντας τα (p - 1) (q - 1). Αυτή η επίθεση δεν είναι όχι ευκολότερη από την παραγωγή του n .

Για τον ultraskeptical, μερικές παραλλαγές του RSA έχουν αποδειχθεί τόσο δύσκολες όσο η παραγωγή (factoring). Η ανάκτηση ακόμη και ορισμένων bits των πληροφοριών από ένα Rsa-κρυπτογραφημένο κρυπτογράφημα είναι τόσο δύσκολη σαν να αποκρυπτογραφείς ολόκληρο μήνυμα. Η παραγωγή του n είναι ο προφανέστερος τρόπος της επίθεσης. Οποιοσδήποτε αντίπαλος θα μπορούσε να έχει το δημόσιο κλειδί, το e, και το συντελεστή, n. Για να βρει το κλειδί αποκρυπτογράφησης, d, πρέπει παράγει το n. Έτσι , ένας συντελεστής δεκαδικός-129 ψηφίων - είναι η αιχμή της τεχνολογίας παραγωγής (factoring). Έτσι το n πρέπει να είναι μεγαλύτερο από αυτήν (key length).

Είναι βεβαίως πιθανό για ένα cryptanalyst (κρυπταναλυτή) να δοκιμαστεί κάθε πιθανό d έως ότου να σκοντάψει στο σωστό. Αυτή η επίθεση brute-force είναι λιγότερο αποδοτική από το να προσπαθήσουμε να παράγουμε (factor) το n. Από καιρό σε καιρό, οι άνθρωποι υποστηρίζουν ότι έχουν βρεί τους εύκολους τρόπους να σπάσουν το RSA, αλλά καμία τέτοια αξίωση δεν έχει επαληθευτεί . Υπάρχει μια άλλη ανησυχία. Οι περισσότεροι κοινοί αλγόριθμοι για υπολογίzουν το p και το q με βάση πιθανοτήτων. Τι θα συμβεί όμως εάν το p ή το q είναι σύνθετα; Οι πιθανότητες τότε είναι παρά πολύ μικρές.

Αλλά ακόμα και αν συμβεί, οι πιθανότητες είναι ότι η κρυπτογράφηση και η αποκρυπτογράφηση δεν θα λειτουργήσουν καλά και θα παρατηρηθεί αμέσως. Υπάρχουν μερικοί αριθμοί, αποκαλούμενοι αριθμοί Carmichael, για τους οποίους ορισμένοι πιθανολογικοί αλγόριθμοι primality θα αποτύχουν να ανιχνεύσουν. Αυτοί είναι υπερβολικά σπάνιοι, αλλά είναι επισφαλείς.

4. ΠΑΡΑΔΕΙΓΜΑΤΑ ΑΛΓΟΡΙΘΜΩΝ

- **DES**

DES

```
#define ENO    0       /* MODE == encrypt */
#define DE1    1       /* MODE == decrypt */

typedef struct {
        unsigned long ek[32];
        unsigned long dk[32];
} des_ctx;

extern void deskey(unsigned char *, short);
/*                  hexkey[8]     MODE
 * Sets the internal key register according to the hexadecimal
 * key contained in the 8 bytes of hexkey, according to the DES,
 * for encryption or decryption according to MODE.
 */

extern void usekey(unsigned long *);
/*                 cookedkey[32]
```

```
 * Loads the internal key register with the data in cookedkey.
 */

extern void cpkey(unsigned long *);
/*               cookedkey[32]
 * Copies the contents of the internal key register into the storage
 * located at &cookedkey[0].
 */

extern void des(unsigned char *, unsigned char *);
/*                from[8]         to[8]
 * Encrypts/Decrypts (according to the key currently loaded in the
 * internal key register) one block of eight bytes at address 'from'
 * into the block at address 'to'.  They can be the same.
 */

static void scrunch(unsigned char *, unsigned long *);
static void unscrun(unsigned long *, unsigned char *);
static void desfunc(unsigned long *, unsigned long *);
static void cookey(unsigned long *);

static unsigned long KnL[32] = { 0L };
static unsigned long KnR[32] = { 0L };
static unsigned long Kn3[32] = { 0L };
static unsigned char Df_Key[24] = {
       0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
       0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10,
       0x89,0xab,0xcd,0xef,0x01,0x23,0x45,0x67 };

static unsigned short bytebit[8]    = {
       0200, 0100, 040, 020, 010, 04, 02, 01 };

static unsigned long bigbyte[24] = {
       0x800000L,     0x400000L,     0x200000L,     0x100000L,
       0x80000L,      0x40000L,      0x20000L,      0x10000L,
       0x8000L,       0x4000L,       0x2000L,       0x1000L,
       0x800L,              0x400L,              0x200L,              0x100L,
       0x80L,               0x40L,               0x20L,               0x10L,
       0x8L,          0x4L,          0x2L,          0x1L    };

/* Use the key schedule specified in the Standard (ANSI X3.92-1981). */

static unsigned char pc1[56] = {
       56, 48, 40, 32, 24, 16,  8,   0, 57, 49, 41, 33, 25, 17,
        9,  1, 58, 50, 42, 34, 26,  18, 10,  2, 59, 51, 43, 35,
       62, 54, 46, 38, 30, 22, 14,   6, 61, 53, 45, 37, 29, 21,
       13,  5, 60, 52, 44, 36, 28,  20, 12,  4, 27, 19, 11,  3 };

static unsigned char totrot[16] = {
       1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 };

static unsigned char pc2[48] = {
       13, 16, 10, 23,  0,  4,       2, 27, 14,  5, 20,  9,
       22, 18, 11,  3, 25,  7,      15,  6, 26, 19, 12,  1,
       40, 51, 30, 36, 46, 54,      29, 39, 50, 44, 32, 47,
       43, 48, 38, 55, 33, 52,      45, 41, 49, 35, 28, 31 };
```

```
void deskey(key, edf)       /* Thanks to James Gillogly & Phil Karn! */
unsigned char *key;
short edf;
{
        register int i, j, l, m, n;
        unsigned char pc1m[56], pcr[56];
        unsigned long kn[32];

        for ( j = 0; j < 56; j++ ) {
                l = pc1[j];
                m = l & 07;
                pc1m[j] = (key[l >> 3] & bytebit[m]) ? 1 : 0;
                }
        for( i = 0; i < 16; i++ ) {
                if( edf == DE1 ) m = (15 - i) << 1;
                else m = i << 1;
                n = m + 1;
                kn[m] = kn[n] = 0L;
                for( j = 0; j < 28; j++ ) {
                        l = j + totrot[i];
                        if( l < 28 ) pcr[j] = pc1m[l];
                        else pcr[j] = pc1m[l - 28];
                        }
                for( j = 28; j < 56; j++ ) {
                    l = j + totrot[i];
                    if( l < 56 ) pcr[j] = pc1m[l];
                    else pcr[j] = pc1m[l - 28];
                    }
                for( j = 0; j < 24; j++ ) {
                        if( pcr[pc2[j]] ) kn[m] |= bigbyte[j];
                        if( pcr[pc2[j+24]] ) kn[n] |= bigbyte[j];
                        }
                }
        cookey(kn);
        return;
}

static void cookey(raw1)
register unsigned long *raw1;
{
        register unsigned long *cook, *raw0;
        unsigned long dough[32];
        register int i;

        cook = dough;
        for( i = 0; i < 16; i++, raw1++ ) {
                raw0 = raw1++;
                *cook    = (*raw0 & 0x00fc0000L) << 6;
                *cook   |= (*raw0 & 0x00000fc0L) << 10;
                *cook   |= (*raw1 & 0x00fc0000L) >> 10;
                *cook++         |= (*raw1 & 0x00000fc0L) >> 6;
                *cook    = (*raw0 & 0x0003f000L) << 12;
                *cook   |= (*raw0 & 0x0000003fL) << 16;
                *cook   |= (*raw1 & 0x0003f000L) >> 4;
                *cook++         |= (*raw1 & 0x0000003fL);
```

```
                }
        usekey(dough);
        return;
}

void cpkey(into)
register unsigned long *into;
{
        register unsigned long *from, *endp;

        from = KnL, endp = &KnL[32];
        while( from < endp ) *into++ = *from++;
        return;
}

void usekey(from)
register unsigned long *from;
{
        register unsigned long *to, *endp;

        to = KnL, endp = &KnL[32];
        while( to < endp ) *to++ = *from++;
        return;
}

void des(inblock, outblock)
unsigned char *inblock, *outblock;
{
        unsigned long work[2];

        scrunch(inblock, work);
        desfunc(work, KnL);
        unscrun(work, outblock);
        return;
}

static void scrunch(outof, into)
register unsigned char *outof;
register unsigned long *into;
{
        *into   = (*outof++ & 0xffL) << 24;
        *into  |= (*outof++ & 0xffL) << 16;
        *into  |= (*outof++ & 0xffL) << 8;
        *into++ |= (*outof++ & 0xffL);
        *into   = (*outof++ & 0xffL) << 24;
        *into  |= (*outof++ & 0xffL) << 16;
        *into  |= (*outof++ & 0xffL) << 8;
        *into  |= (*outof   & 0xffL);
        return;
}

static void unscrun(outof, into)
register unsigned long *outof;
register unsigned char *into;
{
```

```
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >>  8) & 0xffL;
        *into++ =  *outof++           & 0xffL;
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >>  8) & 0xffL;
        *into   =  *outof     & 0xffL;
        return;
}

static unsigned long SP1[64] = {
        0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
        0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
        0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
        0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
        0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
        0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
        0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
        0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
        0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
        0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
        0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
        0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
        0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
        0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
        0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L };

static unsigned long SP2[64] = {
        0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
        0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
        0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
        0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
        0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
        0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
        0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
        0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
        0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
        0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
        0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
        0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
        0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
        0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
        0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
        0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L };

static unsigned long SP3[64] = {
        0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
        0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
        0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
        0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
        0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
        0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
```

```
        0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
        0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
        0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
        0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
        0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
        0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
        0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
        0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
        0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
        0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L };

static unsigned long SP4[64] = {
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
        0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
        0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
        0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
        0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
        0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
        0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
        0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
        0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L };

static unsigned long SP5[64] = {
        0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
        0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
        0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
        0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
        0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
        0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
        0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
        0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
        0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
        0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
        0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
        0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
        0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
        0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
        0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
        0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L };

static unsigned long SP6[64] = {
        0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
        0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
        0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
        0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
        0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
```

```
        0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
        0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
        0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
        0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
        0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
        0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
        0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
        0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
        0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L };

static unsigned long SP7[64] = {
        0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
        0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
        0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
        0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
        0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
        0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
        0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
        0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
        0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
        0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
        0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
        0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
        0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
        0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
        0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
        0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L };

static unsigned long SP8[64] = {
        0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
        0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
        0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
        0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
        0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
        0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
        0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
        0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
        0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
        0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
        0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
        0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
        0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
        0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
        0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
        0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L };

static void desfunc(block, keys)
register unsigned long *block, *keys;
{
        register unsigned long fval, work, right, leftt;
        register int round;

        leftt = block[0];
```

```
right = block[1];
work = ((leftt >> 4) ^ right) & 0x0f0f0f0fL;
right ^= work;
leftt ^= (work << 4);
work = ((leftt >> 16) ^ right) & 0x0000ffffL;
right ^= work;
leftt ^= (work << 16);
work = ((right >> 2) ^ leftt) & 0x33333333L;
leftt ^= work;
right ^= (work << 2);
work = ((right >> 8) ^ leftt) & 0x00ff00ffL;
leftt ^= work;
right ^= (work << 8);
right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffl
work = (leftt ^ right) & 0xaaaaaaaaL;
leftt ^= work;
right ^= work;
leftt = ((leftt << 1) | ((leftt >> 31) & 1L)) & 0xffffffffl

for( round = 0; round < 8; round++ ) {
       work  = (right << 28) | (right >> 4);
       work ^= *keys++;
       fval  = SP7[ work             & 0x3fL];
       fval |= SP5[(work >>  8) & 0x3fL];
       fval |= SP3[(work >> 16) & 0x3fL];
       fval |= SP1[(work >> 24) & 0x3fL];
       work  = right ^ *keys++;
       fval |= SP8[ work             & 0x3fL];
       fval |= SP6[(work >>  8) & 0x3fL];
       fval |= SP4[(work >> 16) & 0x3fL];
       fval |= SP2[(work >> 24) & 0x3fL];
       leftt ^= fval;
       work  = (leftt << 28) | (leftt >> 4);
       work ^= *keys++;
       fval  = SP7[ work             & 0x3fL];
       fval |= SP5[(work >>  8) & 0x3fL];
       fval |= SP3[(work >> 16) & 0x3fL];
       fval |= SP1[(work >> 24) & 0x3fL];
       work  = leftt ^ *keys++;
       fval |= SP8[ work             & 0x3fL];
       fval |= SP6[(work >>  8) & 0x3fL];
       fval |= SP4[(work >> 16) & 0x3fL];
       fval |= SP2[(work >> 24) & 0x3fL];
       right ^= fval;
       }

right = (right << 31) | (right >> 1);
work = (leftt ^ right) & 0xaaaaaaaaL;
leftt ^= work;
right ^= work;
leftt = (leftt << 31) | (leftt >> 1);
work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
right ^= work;
leftt ^= (work << 8);
```

```
        work = ((leftt >> 2) ^ right) & 0x33333333L;
        right ^= work;
        leftt ^= (work << 2);
        work = ((right >> 16) ^ leftt) & 0x0000ffffL;
        leftt ^= work;
        right ^= (work << 16);
        work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
        leftt ^= work;
        right ^= (work << 4);
        *block++ = right;
        *block = leftt;
        return;
}

/* Validation sets:
 *
 * Single-length key, single-length plaintext -
 * Key     : 0123 4567 89ab cdef
 * Plain   : 0123 4567 89ab cde7
 * Cipher : c957 4425 6a5e d31d
 *
 ********************************************************************/

void des_key(des_ctx *dc, unsigned char *key){
        deskey(key,EN0);
        cpkey(dc->ek);
        deskey(key,DE1);
        cpkey(dc->dk);
}

/* Encrypt several blocks in ECB mode.  Caller is responsible for
   short blocks. */
void des_enc(des_ctx *dc, unsigned char *data, int blocks){
        unsigned long work[2];
        int i;
        unsigned char *cp;

        cp = data;
        for(i=0;i<blocks;i++){
                scrunch(cp,work);
                desfunc(work,dc->ek);
                unscrun(work,cp);
                cp+=8;
        }
}

void des_dec(des_ctx *dc, unsigned char *data, int blocks){
        unsigned long work[2];
        int i;
        unsigned char *cp;

        cp = data;
        for(i=0;i<blocks;i++){
                scrunch(cp,work);
                desfunc(work,dc->dk);
```

```
                unscrun(work,cp);
                cp+=8;
        }
}

void main(void){
        des_ctx dc;
        int i;
        unsigned long data[10];
        char *cp,key[8] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef};
        char x[8] = {0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xe7};

        cp = x;

        des_key(&dc,key);
        des_enc(&dc,cp,1);
        printf("Enc(0..7,0..7) = ");
        for(i=0;i<8;i++) printf("%02x ", ((unsigned int) cp[i])&0x00ff);
        printf("\n");

        des_dec(&dc,cp,1);

        printf("Dec(above,0..7) = ");
        for(i=0;i<8;i++) printf("%02x ",((unsigned int)cp[i])&0x00ff);
        printf("\n");

        cp = (char *) data;
        for(i=0;i<10;i++)data[i]=i;

        des_enc(&dc,cp,5); /* Enc 5 blocks. */
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                i/2,data[i],data[i+1]);

        des_dec(&dc,cp,1);
        des_dec(&dc,cp+8,4);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                i/2,data[i],data[i+1]);

}
```

- **IDEA**

IDEA

```
typedef unsigned char boolean;      /* values are TRUE or FALSE */
typedef unsigned char byte; /* values are 0-255 */
typedef byte *byteptr;      /* pointer to byte */
```

```
typedef char *string;/* pointer to ASCII character string */
typedef unsigned short word16;      /* values are 0-65535 */
typedef unsigned long word32;       /* values are 0-4294967295 */

#ifndef TRUE
#define FALSE 0
#define TRUE (!FALSE)
#endif /* if TRUE not already defined */

#ifndef min   /* if min macro not already defined */
#define min(a,b) ( (a)<(b) ? (a) : (b) )
#define max(a,b) ( (a)>(b) ? (a) : (b) )
#endif /* if min macro not already defined */

#define IDEAKEYSIZE 16
#define IDEABLOCKSIZE 8

#define IDEAROUNDS 8
#define IDEAKEYLEN (6*IDEAROUNDS+4)

typedef struct{
       word16 ek[IDEAKEYLEN],dk[IDEAKEYLEN];
}idea_ctx;

/* End includes for IDEA.C */
#ifdef IDEA32        /* Use >16-bit temporaries */
#define low16(x) ((x) & 0xFFFF)
typedef unsigned int uint16;/* at LEAST 16 bits, maybe more */
#else
#define low16(x) (x) /* this is only ever applied to uint16's */
typedef word16 uint16;
#endif

#ifdef SMALL_CACHE
static uint16
mul(register uint16 a, register uint16 b)
{
       register word32 p;

       p = (word32)a * b;
       if (p) {
              b = low16(p);
              a = p>>16;
              return (b - a) + (b < a);
       } else if (a) {
              return 1-b;
       } else {
              return 1-a;
       }
} /* mul */
#endif /* SMALL_CACHE */

static uint16
mulInv(uint16 x)
{
```

```
        uint16 t0, t1;
        uint16 q, y;

        if (x <= 1)
                return x;       /* 0 and 1 are self-inverse */
        t1 = 0x10001L / x;      /* Since x >= 2, this fits into 16 bits */
        y = 0x10001L % x;
        if (y == 1)
                return low16(1-t1);
        t0 = 1;
        do {
                q = x / y;
                x = x % y;
                t0 += q * t1;
                if (x == 1)
                        return t0;
                q = y / x;
                y = y % x;
                t1 += q * t0;
        } while (y != 1);
        return low16(1-t1);
} /* mukInv */

static void
ideaExpandKey(byte const *userkey, word16 *EK)
{
        int i,j;

        for (j=0; j<8; j++) {
                EK[j] = (userkey[0]<<8) + userkey[1];
                userkey += 2;
        }
        for (i=0; j < IDEAKEYLEN; j++) {
                i++;
                EK[i+7] = EK[i & 7] << 9 | EK[i+1 & 7] >> 7;
                EK += i & 8;
                i &= 7;
        }
} /* ideaExpandKey */

static void
ideaInvertKey(word16 const *EK, word16 DK[IDEAKEYLEN])
{
        int i;
        uint16 t1, t2, t3;
        word16 temp[IDEAKEYLEN];
        word16 *p = temp + IDEAKEYLEN;

        t1 = mulInv(*EK++);
        t2 = -*EK++;
        t3 = -*EK++;
        *--p = mulInv(*EK++);
        *--p = t3;
        *--p = t2;
```

```
        *--p = t1;

        for (i = 0; i < IDEAROUNDS-1; i++) {
                t1 = *EK++;
                *--p = *EK++;
                *--p = t1;

                t1 = mulInv(*EK++);
                t2 = -*EK++;
                t3 = -*EK++;
                *  p = mulInv(*EK++);
                *--p = t2;
                *--p = t3;
                *--p = t1;
        }
        t1 = *EK++;
        *--p = *EK++;
        *--p = t1;

        t1 = mulInv(*EK++);
        t2 = -*EK++;
        t3 = -*EK++;
        *--p = mulInv(*EK++);
        *--p = t3;
        *--p = t2;
        *--p = t1;
/* Copy and destroy temp copy */
        memcpy(DK, temp, sizeof(temp));
        for(i=0;i<IDEAKEYLEN;i++)temp[i]=0;
} /* ideaInvertKey */

#ifdef SMALL_CACHE
#define MUL(x,y) (x = mul(low16(x),y))
#else /* !SMALL_CACHE */
#ifdef AVOID JUMPS
#define MUL(x,y) (x = low16(x-1), t16 = low16((y)-1), \
                t32 = (word32)x*t16 + x + t16 + 1, x = low16(t32), \
                t16 = t32>>16, x = (x-t16) + (x<t16) )
#else /* !AVOID_JUMPS (default) */
#define MUL(x,y) \
        ((t16 = (y)) ? \
                (x=low16(x)) ? \
                        t32 = (word32)x*t16, \
                        x = low16(t32), \
                        t16 = t32>>16, \
                        x = (x-t16)+(x<t16) \
                : \
                        (x = 1-t16) \
        : \
                (x = 1-x))
#endif
#endif

static void
```

```
ideaCipher(byte *inbuf, byte *outbuf, word16 *key)
{
        register uint16 x1, x2, x3, x4, s2, s3;
        word16 *in, *out;
#ifndef SMALL_CACHE
        register uint16 t16; /* Temporaries needed by MUL macro */
        register word32 t32;
#endif
        int r = IDEAROUNDS;

        in = (word16 *)inbuf;
        x1 = *in++;  x2 = *in++;
        x3 = *in++;  x4 = *in;
#ifndef HIGHFIRST
        x1 = (x1 >>8) | (x1<<8);
        x2 = (x2 >>8) | (x2<<8);
        x3 = (x3 >>8) | (x3<<8);
        x4 = (x4 >>8) | (x4<<8);
#endif
        do {
                MUL(x1,*key++);
                x2 += *key++;
                x3 += *key++;
                MUL(x4, *key++);

                s3 = x3;
                x3 ^= x1;
                MUL(x3, *key++);
                s2 = x2;
                x2 ^= x4;
                x2 += x3;
                MUL(x2, *key++);
                x3 += x2;

                x1 ^= x2;  x4 ^= x3;

                x2 ^= s3;  x3 ^= s2;
        } while (--r);
        MUL(x1, *key++);
        x3 += *key++;
        x2 += *key++;
        MUL(x4, *key);

        out = (word16 *)outbuf;
#ifdef HIGHFIRST
        *out++ = x1;
        *out++ = x3;
        *out++ = x2;
        *out = x4;
#else /* !HIGHFIRST */
        *out++ = (x1 >>8) | (x1<<8);
        *out++ = (x3 >>8) | (x3<<8);
        *out++ = (x2 >>8) | (x2<<8);
        *out = (x4 >>8) | (x4<<8);
```

```
#endif
} /* ideaCipher */

void idea_key(idea_ctx *c, unsigned char *key){
        ideaExpandKey(key,c->ek);
        ideaInvertKey(c->ek,c->dk);
}

void idea_enc(idea_ctx *c, unsigned char *data, int blocks){
        int i;
        unsigned char *d = data;

        for(i=0;i<blocks;i++){
                ideaCipher(d,d,c->ek);
                d+=8;
        }
}

void idea_dec(idea_ctx *c, unsigned char *data, int blocks){
        int i;
        unsigned char *d = data;

        for(i=0;i<blocks;i++){
                ideaCipher(d,d,c->dk);
                d+=8;
        }
}

#include <stdio.h>

#ifndef BLOCKS
#ifndef KBYTES
#define KBYTES 1024
#endif
#define BLOCKS (64*KBYTES)
#endif

int
main(void)
{       /* Test driver for IDEA cipher */
        int i, j, k;
        idea_ctx c;
        byte userkey[16];
        word16 EK[IDEAKEYLEN], DK[IDEAKEYLEN];
        byte XX[8], YY[8], ZZ[8];
        word32 long_block[10]; /* 5 blocks */
        long l;
        char *lbp;

        /* Make a sample user key for testing... */
        for(i=0; i<16; i++)
                userkey[i] = i+1;

        idea_key(&c,userkey);

        /* Make a sample plaintext pattern for testing... */
```

```
        for (k=0; k<8; k++)
                XX[k] = k;

        idea_enc(&c,XX,1); /* encrypt */

        lbp = (unsigned char *) long_block;
        for(i=0;i<10;i++) long_block[i] = i;
        idea_enc(&c,lbp,5);
        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                  i/2,long_block[i],long_block[i+1]);

        idea_dec(&c,lbp,3);
        idea_dec(&c,lbp+24,2);

        for(i=0;i<10;i+=2) printf("Block %01d = %08lx %08lx.\n",
                                  i/2,long_block[i],long_block[i+1]);

        return 0;       /* normal exit */
} /* main */
```

- **BLOWFISH**

BLOWFISH

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#ifdef little_endian   /* Eg: Intel */
   #include <alloc.h>
#endif

#include <ctype.h>

#ifdef little_endian   /* Eg: Intel */
   #include <dir.h>
   #include <bios.h>
#endif

#ifdef big_endian
   #include <Types.h>
#endif

typedef struct {
        unsigned long S[4][256],P[18];
} blf_ctx;

#define MAXKEYBYTES 56          /* 448 bits */
// #define little_endian 1              /* Eg: Intel */
#define big_endian 1            /* Eg: Motorola */

void Blowfish_encipher(blf_ctx *,unsigned long *xl, unsigned long *xr);
void Blowfish_decipher(blf_ctx *,unsigned long *xl, unsigned long *xr);

#define N               16
#define noErr            0
#define DATAERROR         -1
#define KEYBYTES         8

FILE*         SubkeyFile;

unsigned long F(blf_ctx *bc, unsigned long x)
{
   unsigned short a;
   unsigned short b;
   unsigned short c;
   unsigned short d;
   unsigned long  y;
```

```
    d = x & 0x00FF;
    x >>= 8;
    c = x & 0x00FF;
    x >>= 8;
    b = x & 0x00FF;
    x >>= 8;
    a = x & 0x00FF;
    //y = ((S[0][a] + S[1][b]) ^ S[2][c]) + S[3][d];
    y = bc->S[0][a] + bc->S[1][b];
    y = y ^ bc->S[2][c];
    y = y + bc->S[3][d];

    return y;
}

void Blowfish_encipher(blf_ctx *c,unsigned long *xl, unsigned long *xr)
{
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short          i;

    Xl = *xl;
    Xr = *xr;

    for (i = 0; i < N; ++i) {
        Xl = Xl ^ c >P[i];
        Xr = F(c,Xl) ^ Xr;

        temp = Xl;
        Xl = Xr;
        Xr = temp;
    }

    temp = Xl;
    Xl = Xr;
    Xr = temp;

    Xr = Xr ^ c->P[N];
    Xl = Xl ^ c->P[N + 1];

    *xl = Xl;
    *xr = Xr;
}

void Blowfish_decipher(blf_ctx *c, unsigned long *xl, unsigned long *xr)
{
    unsigned long  Xl;
    unsigned long  Xr;
    unsigned long  temp;
    short          i;

    Xl = *xl;
    Xr = *xr;
```

```
   for (i = N + 1; i > 1; --i) {
      Xl = Xl ^ c->P[i];
      Xr = F(c,Xl) ^ Xr;

      /* Exchange Xl and Xr */
      temp = Xl;
      Xl = Xr;
      Xr = temp;
   }

   /* Exchange Xl and Xr */
   temp = Xl;
   Xl = Xr;
   Xr = temp;

   Xr = Xr ^ c->P[1];
   Xl = Xl ^ c->P[0];

   *xl = Xl;
   *xr = Xr;
}

short InitializeBlowfish(blf_ctx *c, char key[], short keybytes)
{
   short          i;
   short          j;
   short          k;
   short          error;
   short          numread;
   unsigned long  data;
   unsigned long  datal;
   unsigned long  datar;

unsigned long ks0[] = {
0xd1310ba6, 0x98dfb5ac, 0x2ffd72db, 0xd01adfb7, 0xb8e1afed, 0x6a267e96,
0xba7c9045, 0xf12c7f99, 0x24a19947, 0xb3916cf7, 0x0801f2e2, 0x858efc16,
0x636920d8, 0x71574e69, 0xa458fea3, 0xf4933d7e, 0x0d95748f, 0x728eb658,
0x718bcd58, 0x82154aee, 0x7b54a41d, 0xc25a59b5, 0x9c30d539, 0x2af26013,
0xc5d1b023, 0x286085f0, 0xca417918, 0xb8db38ef, 0x8e79dcb0, 0x603a180e,
0x6c9e0e8b, 0xb01e8a3e, 0xd71577c1, 0xbd314b27, 0x78af2fda, 0x55605c60,
0xe65525f3, 0xaa55ab94, 0x57489862, 0x63e81440, 0x55ca396a, 0x2aab10b6,
0xb4cc5c34, 0x1141e8ce, 0xa15486af, 0x7c72e993, 0xb3ee1411, 0x636fbc2a,
0x2ba9c55d, 0x741831f6, 0xce5c3e16, 0x9b87931e, 0xafd6ba33, 0x6c24cf5c,
0x7a325381, 0x28958677, 0x3b8f4898, 0x6b4bb9af, 0xc4bfe81b, 0x66282193,
0x61d809cc, 0xfb21a991, 0x487cac60, 0x5dec8032, 0xef845d5d, 0xe98575b1,
0xdc262302, 0xeb651b88, 0x23893e81, 0xd396acc5, 0x0f6d6ff3, 0x83f44239,
0x2e0b4482, 0xa4842004, 0x69c8f04a, 0x9e1f9b5e, 0x21c66842, 0xf6e96c9a,
0x670c9c61, 0xabd388f0, 0x6a51a0d2, 0xd8542f68, 0x960fa728, 0xab5133a3,
0x6eef0b6c, 0x137a3be4, 0xba3bf050, 0x7efb2a98, 0xa1f1651d, 0x39af0176,
0x66ca593e, 0x82430e88, 0x8cee8619, 0x456f9fb4, 0x7d84a5c3, 0x3b8b5ebe,
0xe06f75d8, 0x85c12073, 0x401a449f, 0x56c16aa6, 0x4ed3aa62, 0x363f7706,
0x1bfedf72, 0x429b023d, 0x37d0d724, 0xd00a1248, 0xdb0fead3, 0x49f1c09b,
0x075372c9, 0x80991b7b, 0x25d479d8, 0xf6e8def7, 0xe3fe501a, 0xb6794c3b,
```

```
0x976ce0bd, 0x04c006ba, 0xc1a94fb6, 0x409f60c4, 0x5e5c9ec2, 0x196a2463,
0x68fb6faf, 0x3e6c53b5, 0x1339b2eb, 0x3b52ec6f, 0x6dfc511f, 0x9b30952c,
0xcc814544, 0xaf5ebd09, 0xbee3d004, 0xde334afd, 0x660f2807, 0x192e4bb3,
0xc0cba857, 0x45c8740f, 0xd20b5f39, 0xb9d3fbdb, 0x5579c0bd, 0x1a60320a,
0xd6a100c6, 0x402c7279, 0x679f25fe, 0xfb1fa3cc, 0x8ea5e9f8, 0xdb3222f8,
0x3c7516df, 0xfd616b15, 0x2f501ec8, 0xad0552ab, 0x323db5fa, 0xfd238760,
0x53317b48, 0x3e00df82, 0x9e5c57bb, 0xca6f8ca0, 0x1a87562e, 0xdf1769db,
0xd542a8f6, 0x287effc3, 0xac6732c6, 0x8c4f5573, 0x695b27b0, 0xbbca58c8,
0xe1ffa35d, 0xb8f011a0, 0x10fa3d98, 0xfd2183b8, 0x4afcb56c, 0x2dd1d35b,
0x9a53e479, 0xb6f84565, 0xd28e49bc, 0x4bfb9790, 0xe1ddf2da, 0xa4cb7e33,
0x62fb1341, 0xcee4c6e8, 0xef20cada, 0x36774c01, 0xd07e9efe, 0x2bf11fb4,
0x95dbda4d, 0xae909198, 0xeaad8e71, 0x6b93d5a0, 0xd08ed1d0, 0xafc725e0,
0x8e3c5b2f, 0x8e7594b7, 0x8ff6e2fb, 0xf2122b64, 0x8888b812, 0x900df01c,
0x4fad5ea0, 0x688fc31c, 0xd1cff191, 0xb3a8c1ad, 0x2f2f2218, 0xbe0e1777,
0xea752dfe, 0x8b021fa1, 0xe5a0cc0f, 0xb56f74e8, 0x18acf3d6, 0xce89e299,
0xb4a84fe0, 0xfd13e0b7, 0x7cc43b81, 0xd2ada8d9, 0x165fa266, 0x80957705,
0x93cc7314, 0x211a1477, 0xe6ad2065, 0x77b5fa86, 0xc75442f5, 0xfb9d35cf,
0xebcdaf0c, 0x7b3e89a0, 0xd6411bd3, 0xae1e7e49, 0x00250e2d, 0x2071b35e,
0x226800bb, 0x57b8e0af, 0x2464369b, 0xf009b91e, 0x5563911d, 0x59dfa6aa,
0x78c14389, 0xd95a537f, 0x207d5ba2, 0x02e5b9c5, 0x83260376, 0x6295cfa9,
0x11c81968, 0x4e734a41, 0xb3472dca, 0x7b14a94a, 0x1b510052, 0x9a532915,
0xd60f573f, 0xbc9bc6e4, 0x2b60a476, 0x81e67400, 0x08ba6fb5, 0x571be91f,
0xf296ec6b, 0x2a0dd915, 0xb6636521, 0xe7b9f9b6, 0xff34052e, 0xc5855664,
0x53b02d5d, 0xa99f8fa1, 0x08ba4799, 0x6e85076a};
unsigned long ks1[] = {
0x4b7a70e9, 0xb5b32944, 0xdb75092e, 0xc4192623, 0xad6ea6b0, 0x49a7df7d,
0x9cee60b8, 0x8fedb266, 0xecaa8c71, 0x699a17ff, 0x5664526c, 0xc2b19ee1,
0x193602a5, 0x75094c29, 0xa0591340, 0xe4183a3e, 0x3f54989a, 0x5b429d65,
0x6b8fe4d6, 0x99f73fd6, 0xa1d29c07, 0xefe830f5, 0x4d2d38e6, 0xf0255dc1,
0x4cdd2086, 0x8470eb26, 0x6382e9c6, 0x021ecc5e, 0x09686b3f, 0x3ebaefc9,
0x3c971814, 0x6b6a70a1, 0x687f3584, 0x52a0e286, 0xb79c5305, 0xaa500737,
0x3e07841c, 0x7fdeae5c, 0x8e7d44ec, 0x5716f2b8, 0xb03ada37, 0xf0500c0d,
0xf01c1f04, 0x0200b3ff, 0xae0cf51a, 0x3cb574b2, 0x25837a58, 0xdc0921bd,
0xd19113f9, 0x7ca92ff6, 0x94324773, 0x22f54701, 0x3ae5e581, 0x37c2dadc,
0xc8b57634, 0x9af3dda7, 0xa9446146, 0x0fd0030e, 0xecc8c73e, 0xa4751e41,
0xe238cd99, 0x3bea0e2f, 0x3280bba1, 0x183eb331, 0x4e548b38, 0x4f6db908,
0x6f420d03, 0xf60a04bf, 0x2cb81290, 0x24977c79, 0x5679b072, 0xbcaf89af,
0xde9a771f, 0xd9930810, 0xb38bae12, 0xdccf3f2e, 0x5512721f, 0x2e6b7124,
0x501adde6, 0x9f84cd87, 0x7a584718, 0x7408da17, 0xbc9f9abc, 0xe94b7d8c,
0xec7aec3a, 0xdb851dfa, 0x63094366, 0xc464c3d2, 0xef1c1847, 0x3215d908,
0xdd433b37, 0x24c2ba16, 0x12a14d43, 0x2a65c451, 0x50940002, 0x133ae4dd,
0x71dff89e, 0x10314e55, 0x81ac77d6, 0x5f11199b, 0x043556f1, 0xd7a3c76b,
0x3c11183b, 0x5924a509, 0xf28fe6ed, 0x97f1fbfa, 0x9ebabf2c, 0x1e153c6e,
0x86e34570, 0xeae96fb1, 0x860e5e0a, 0x5a3e2ab3, 0x771fe71c, 0x4e3d06fa,
0x2965dcb9, 0x99e71d0f, 0x803e89d6, 0x5266c825, 0x2e4cc978, 0x9c10b36a,
0xc6150eba, 0x94e2ea78, 0xa5fc3c53, 0x1e0a2df4, 0xf2f74ea7, 0x361d2b3d,
0x1939260f, 0x19c27960, 0x5223a708, 0xf71312b6, 0xebadfe6e, 0xeac31f66,
0xe3bc4595, 0xa67bc883, 0xb17f37d1, 0x018cff28, 0xc332ddef, 0xbe6c5aa5,
0x65582185, 0x68ab9802, 0xeecea50f, 0xdb2f953b, 0x2aef7dad, 0x5b6e2f84,
0x1521b628, 0x29076170, 0xecdd4775, 0x619f1510, 0x13cca830, 0xeb61bd96,
0x0334fe1e, 0xaa0363cf, 0xb5735c90, 0x4c70a239, 0xd59e9e0b, 0xcbaade14,
0xeecc86bc, 0x60622ca7, 0x9cab5cab, 0xb2f3846e, 0x648b1eaf, 0x19bdf0ca,
0xa02369b9, 0x655abb50, 0x40685a32, 0x3c2ab4b3, 0x319ee9d5, 0xc021b8f7,
0x9b540b19, 0x875fa099, 0x95f7997e, 0x623d7da8, 0xf837889a, 0x97e32d77,
```

```
0x11ed935f, 0x16681281, 0x0e358829, 0xc7e61fd6, 0x96dedfa1, 0x7858ba99,
0x57f584a5, 0x1b227263, 0x9b83c3ff, 0x1ac24696, 0xcdb30aeb, 0x532e3054,
0x8fd948e4, 0x6dbc3128, 0x58ebf2ef, 0x34c6ffea, 0xfe28ed61, 0xee7c3c73,
0x5d4a14d9, 0xe864b7e3, 0x42105d14, 0x203e13e0, 0x45eee2b6, 0xa3aaabea,
0xdb6c4f15, 0xfacb4fd0, 0xc742f442, 0xef6abbb5, 0x654f3b1d, 0x41cd2105,
0xd81e799e, 0x86854dc7, 0xe44b476a, 0x3d816250, 0xcf62a1f2, 0x5b8d2646,
0xfc8883a0, 0xc1c7b6a3, 0x7f1524c3, 0x69cb7492, 0x47848a0b, 0x5692b285,
0x095bbf00, 0xad19489d, 0x1462b174, 0x23820e00, 0x58428d2a, 0x0c55f5ea,
0x1dadf43e, 0x233f7061, 0x3372f092, 0x8d937e41, 0xd65fecf1, 0x6c223bdb,
0x7cde3759, 0xcbee7460, 0x4085f2a7, 0xce77326e, 0xa6078084, 0x19f8509e,
0xe8efd855, 0x61d99735, 0xa969a7aa, 0xc50c06c2, 0x5a04abfc, 0x800bcadc,
0x9e447a2e, 0xc3453484, 0xfdd56705, 0x0e1e9ec9, 0xdb73dbd3, 0x105588cd,
0x675fda79, 0xe3674340, 0xc5c43465, 0x713e38d8, 0x3d28f89e, 0xf16dff20,
0x153e21e7, 0x8fb03d4a, 0xc6c39f2b, 0xdb83adf7};
unsigned long ks2[] = {
0xe93d5a68, 0x948140f7, 0xf64c261c, 0x94692934, 0x411520f7, 0x7602d4f7,
0xbcf46b2e, 0xd4a20068, 0xd4082471, 0x3320f46a, 0x43b7d4b7, 0x500061af,
0x1e39f62e, 0x97244546, 0x14214f74, 0xbf8b8840, 0x4d95fc1d, 0x96b591af,
0x70f4ddd3, 0x66a02f45, 0xbfbc09ec, 0x03bd9785, 0x7fac6dd0, 0x31cb8504,
0x96eb27b3, 0x55fd3941, 0xda2547e6, 0xabca0a9a, 0x28507825, 0x530429f4,
0x0a2c86da, 0xe9b66dfb, 0x68dc1462, 0xd7486900, 0x680ec0a4, 0x27a18dee,
0x4f3ffea2, 0xe887ad8c, 0xb58ce006, 0x7af4d6b6, 0xaace1e7c, 0xd3375fec,
0xce78a399, 0x406b2a42, 0x20fe9e35, 0xd9f385b9, 0xee39d7ab, 0x3b124e8b,
0x1dc9faf7, 0x4b6d1856, 0x26a36631, 0xeae397b2, 0x3a6efa74, 0xdd5b4332,
0x6841e7f7, 0xca7820fb, 0xfb0af54e, 0xd8feb397, 0x454056ac, 0xba489527,
0x55533a3a, 0x20838d87, 0xfe6ba9b7, 0xd096954b, 0x55a867bc, 0xa1159a58,
0xcca92963, 0x99e1db33, 0xa62a4a56, 0x3f3125f9, 0x5ef47e1c, 0x9029317c,
0xfdf8e802, 0x04272f70, 0x80bb155c, 0x05282ce3, 0x95c11548, 0xe4c66d22,
0x48c1133f, 0xc70f86dc, 0x07f9c9ee, 0x41041f0f, 0x404779a4, 0x5d886e17,
0x325f51eb, 0xd59bc0d1, 0xf2bcc18f, 0x41113564, 0x257b7834, 0x602a9c60,
0xdff8e8a3, 0x1f636c1b, 0x0e12b4c2, 0x02e1329e, 0xaf664fd1, 0xcad18115,
0x6b2395e0, 0x333e92e1, 0x3b240b62, 0xeebeb922, 0x85b2a20e, 0xe6ba0d99,
0xde720c8c, 0x2da2f728, 0xd0127845, 0x95b794fd, 0x647d0862, 0xe7ccf5f0,
0x5449a36f, 0x877d48fa, 0xc39dfd27, 0xf33e8d1e, 0x0a476341, 0x992eff74,
0x3a6f6eab, 0xf4f8fd37, 0xa812dc60, 0xa1ebddf8, 0x991be14c, 0xdb6e6b0d,
0xc67b5510, 0x6d672c37, 0x2765d43b, 0xdcd0e804, 0xf1290dc7, 0xcc00ffa3,
0xb5390f92, 0x690fed0b, 0x667b9ffb, 0xcedb7d9c, 0xa091cf0b, 0xd9155ea3,
0xbb132f88, 0x515bad24, 0x7b9479bf, 0x763bd6eb, 0x37392eb3, 0xcc115979,
0x8026e297, 0xf42e312d, 0x6842ada7, 0xc66a2b3b, 0x12754ccc, 0x782ef11c,
0x6a124237, 0xb79251e7, 0x06a1bbe6, 0x4bfb6350, 0x1a6b1018, 0x11caedfa,
0x3d25bdd8, 0xe2e1c3c9, 0x44421659, 0x0a121386, 0xd90cec6e, 0xd5abea2a,
0x64af674e, 0xda86a85f, 0xbebfe988, 0x64e4c3fe, 0x9dbc8057, 0xf0f7c086,
0x60787bf8, 0x6003604d, 0xd1fd8346, 0xf6381fb0, 0x7745ae04, 0xd736fccc,
0x83426b33, 0xf01eab71, 0xb0804187, 0x3c005e5f, 0x77a057be, 0xbde8ae24,
0x55464299, 0xbf582e61, 0x4e58f48f, 0xf2ddfda2, 0xf474ef38, 0x8789bdc2,
0x5366f9c3, 0xc8b38e74, 0xb475f255, 0x46fcd9b9, 0x7aeb2661, 0x8b1ddf84,
0x846a0e79, 0x915f95e2, 0x466e598e, 0x20b45770, 0x8cd55591, 0xc902de4c,
0xb90bace1, 0xbb8205d0, 0x11a86248, 0x7574a99e, 0xb77f19b6, 0xe0a9dc09,
0x662d09a1, 0xc4324633, 0xe85a1f02, 0x09f0be8c, 0x4a99a025, 0x1d6efe10,
0x1ab93d1d, 0x0ba5a4df, 0xa186f20f, 0x2868f169, 0xdcb7da83, 0x573906fe,
0xa1e2ce9b, 0x4fcd7f52, 0x50115e01, 0xa70683fa, 0xa002b5c4, 0x0de6d027,
0x9af88c27, 0x773f8641, 0xc3604c06, 0x61a806b5, 0xf0177a28, 0xc0f586e0,
0x006058aa, 0x30dc7d62, 0x11e69ed7, 0x2338ea63, 0x53c2dd94, 0xc2c21634,
0xbbcbee56, 0x90bcb6de, 0xebfc7da1, 0xce591d76, 0x6f05e409, 0x4b7c0188,
```

```
0x39720a3d, 0x7c927c24, 0x86e3725f, 0x724d9db9, 0x1ac15bb4, 0xd39eb8fc,
0xed545578, 0x08fca5b5, 0xd83d7cd3, 0x4dad0fc4, 0x1e50ef5e, 0xb161e6f8,
0xa28514d9, 0x6c51133c, 0x6fd5c7e7, 0x56e14ec4, 0x362abfce, 0xddc6c837,
0xd79a3234, 0x92638212, 0x670efa8e, 0x406000e0};
unsigned long ks3[] = {
0x3a39ce37, 0xd3faf5cf, 0xabc27737, 0x5ac52d1b, 0x5cb0679e, 0x4fa33742,
0xd3822740, 0x99bc9bbe, 0xd5118e9d, 0xbf0f7315, 0xd62d1c7e, 0xc700c47b,
0xb78c1b6b, 0x21a19045, 0xb26eb1be, 0x6a366eb4, 0x5748ab2f, 0xbc946e79,
0xc6a376d2, 0x6549c2c8, 0x530ff8ee, 0x468dde7d, 0xd5730a1d, 0x4cd04dc6,
0x2939bbdb, 0xa9ba4650, 0xac9526e8, 0xbe5ee304, 0xa1fad5f0, 0x6a2d519a,
0x63ef8ce2, 0x9a86ee22, 0xc089c2b8, 0x43242ef6, 0xa51e03aa, 0x9cf2d0a4,
0x83c061ba, 0x9be96a4d, 0x8fe51550, 0xba645bd6, 0x2826a2f9, 0xa73a3ae1,
0x4ba99586, 0xef5562e9, 0xc72fefd3, 0xf752f7da, 0x3f046f69, 0x77fa0a59,
0x80e4a915, 0x87b08601, 0x9b09e6ad, 0x3b3ee593, 0xe990fd5a, 0x9e34d797,
0x2cf0b7d9, 0x022b8b51, 0x96d5ac3a, 0x017da67d, 0xd1cf3ed6, 0x7c7d2d28,
0x1f9f25cf, 0xadf2b89b, 0x5ad6b472, 0x5a88f54c, 0xe029ac71, 0xe019a5e6,
0x47b0acfd, 0xed93fa9b, 0xe8d3c48d, 0x283b57cc, 0xf8d56629, 0x79132e28,
0x785f0191, 0xed756055, 0xf7960e44, 0xe3d35e8c, 0x15056dd4, 0x88f46dba,
0x03a16125, 0x0564f0bd, 0xc3eb9e15, 0x3c9057a2, 0x97271aec, 0xa93a072a,
0x1b3f6d9b, 0x1e6321f5, 0xf59c66fb, 0x26dcf319, 0x7533d928, 0xb155fdf5,
0x03563482, 0x8aba3cbb, 0x28517711, 0xc20ad9f8, 0xabcc5167, 0xccad925f,
0x4de81751, 0x3830dc8e, 0x379d5862, 0x9320f991, 0xea7a90c2, 0xfb3e7bce,
0x5121ce64, 0x774fbe32, 0xa8b6e37e, 0xc3293d46, 0x48de5369, 0x6413e680,
0xa2ae0810, 0xdd6db224, 0x69852dfd, 0x09072166, 0xb39a460a, 0x6445c0dd,
0x586cdecf, 0x1c20c8ae, 0x5bbef7dd, 0x1b588d40, 0xccd2017f, 0x6bb4e3bb,
0xdda26a7e, 0x3a59ff45, 0x3e350a44, 0xbcb4cdd5, 0x72eacea8, 0xfa6484bb,
0x8d6612ae, 0xbf3c6f47, 0xd29be463, 0x542f5d9e, 0xaec2771b, 0xf64e6370,
0x740e0d8d, 0xe75b1357, 0xf8721671, 0xaf537d5d, 0x4040cb08, 0x4eb4e2cc,
0x34d2466a, 0x0115af84, 0xe1b00428, 0x95983a1d, 0x06b89fb4, 0xce6ea048,
0x6f3f3b82, 0x3520ab82, 0x011a1d4b, 0x277227f8, 0x611560b1, 0xe7933fdc,
0xbb3a792b, 0x344525bd, 0xa08839e1, 0x51ce794b, 0x2f32c9b7, 0xa01fbac9,
0xe01cc87e, 0xbcc7d1f6, 0xcf0111c3, 0xa1e8aac7, 0x1a908749, 0xd44fbd9a,
0xd0dadecb, 0xd50ada38, 0x0339c32a, 0xc6913667, 0x8df9317c, 0xe0b12b4f,
0xf79e59b7, 0x43f5bb3a, 0xf2d519ff, 0x27d9459c, 0xbf97222c, 0x15e6fc2a,
0x0f91fc71, 0x9b941525, 0xfae59361, 0xceb69ceb, 0xc2a86459, 0x12baa8d1,
0xb6c1075e, 0xe3056a0c, 0x10d25065, 0xcb03a442, 0xe0ec6e0e, 0x1698db3b,
0x4c98a0be, 0x3278e964, 0x9f1f9532, 0xe0d392df, 0xd3a0342b, 0x8971f21e,
0x1b0a7441, 0x4ba3348c, 0xc5be7120, 0xc37632d8, 0xdf359f8d, 0x9b992f2e,
0xe60b6f47, 0x0fe3f11d, 0xe54cda54, 0x1edad891, 0xce6279cf, 0xcd3e7e6f,
0x1618b166, 0xfd2c1d05, 0x848fd2c5, 0xf6fb2299, 0xf523f357, 0xa6327623,
0x93a83531, 0x56cccd02, 0xacf08162, 0x5a75ebb5, 0x6e163697, 0x88d273cc,
0xde966292, 0x81b949d0, 0x4c50901b, 0x71c65614, 0xe6c6c7bd, 0x327a140a,
0x45e1d006, 0xc3f27b9a, 0xc9aa53fd, 0x62a80f00, 0xbb25bfe2, 0x35bdd2f6,
0x71126905, 0xb2040222, 0xb6cbcf7c, 0xcd769c2b, 0x53113ec0, 0x1640e3d3,
0x38abbd60, 0x2547adf0, 0xba38209c, 0xf746ce76, 0x77afa1c5, 0x20756060,
0x85cbfe4e, 0x8ae88dd8, 0x7aaaf9b0, 0x4cf9aa7e, 0x1948c25c, 0x02fb8a8c,
0x01c36ae4, 0xd6ebe1f9, 0x90d4f869, 0xa65cdea0, 0x3f09252d, 0xc208e69f,
0xb74e6132, 0xce77e25b, 0x578fdfe3, 0x3ac372e6};

/* Initialize s-boxes without file read. */
        for(i=0;i<256;i++){
                c->S[0][i] = ks0[i];
                c->S[1][i] = ks1[i];
                c->S[2][i] = ks2[i];
```

```
                c->S[3][i] = ks3[i];
        }

        j = 0;
        for (i = 0; i < N + 2; ++i) {
                data = 0x00000000;
                for (k = 0; k < 4; ++k) {
                        data = (data << 8) | key[j];
                        j = j + 1;
                        if (j >= keybytes) {
                                j = 0;
                        }
                }
        c->P[i] = c->P[i] ^ data;
        }

     datal = 0x00000000;
        datar = 0x00000000;

        for (i = 0; i < N + 2; i += 2) {
                Blowfish_encipher(c,&datal, &datar);

                c->P[i] = datal;
                c->P[i + 1] = datar;
        }

        for (i = 0; i < 4; ++i) {
                for (j = 0; j < 256; j += 2) {

                        Blowfish_encipher(c,&datal, &datar);

                        c->S[i][j] = datal;
                        c->S[i][j + 1] = datar;
                }
        }
}

void blf_key(blf_ctx *c, char *k, int len){
        InitializeBlowfish(c,k,len);
}

void blf_enc(blf_ctx *c, unsigned long *data, int blocks){
        unsigned long *d;
        int i;

        d = data;
        for(i=0;i<blocks;i++){
                Blowfish_encipher(c,d,d+1);
                d += 2;
        }
}

void blf_dec(blf_ctx *c, unsigned long *data, int blocks){
        unsigned long *d;
        int i;
```

```
        d = data;
        for(i=0;i<blocks;i++){
                Blowfish_decipher(c,d,d+1);
                d += 2;
        }
}

void main(void){
        blf_ctx c;
        char key[]="AAAAA";
        unsigned long data[10];
        int i;

        for(i=0;i<10;i++) data[i] = i;

        blf_key(&c,key,5);
        blf_enc(&c,data,5);
        blf_dec(&c,data,1);
        blf_dec(&c,data+2,4);
        for(i=0;i<10;i+=2) printf("Block %01d decrypts to: %08lx %08lx.\n",
                                  i/2,data[i],data[i+1]);
}
```

- **RC5**

RC5

```
#include <stdio.h>

/* An RC5 context needs to know how many rounds it has, and its subkeys. */
typedef struct {
        u4 *xk;
        int nr;
} rc5_ctx;

/* Where possible, these should be replaced with actual rotate instructions.
   For Turbo C++, this is done with _lrotl and _lrotr. */

#define ROTL32(X,C) (((X)<<(C))|((X)>>(32-(C))))
#define ROTR32(X,C) (((X)>>(C))|((X)<<(32-(C))))
```

```
/* Function prototypes for dealing with RC5 basic operations. */
void rc5_init(rc5_ctx *, int);
void rc5_destroy(rc5_ctx *);
void rc5_key(rc5_ctx *, u1 *, int);
void rc5_encrypt(rc5_ctx *, u4 *, int);
void rc5_decrypt(rc5_ctx *, u4 *, int);

/* Function implementations for RC5. */

/* Scrub out all sensitive values. */
void rc5 destroy(rc5 ctx *c){
        int i;
     for(i=0;i<(c->nr)*2+2;i++) c->xk[i]=0;
     free(c->xk);
}

/* Allocate memory for rc5 context's xk and such. */
void rc5_init(rc5_ctx *c, int rounds){
     c->nr = rounds;
     c->xk = (u4 *) malloc(4*(rounds*2+2));
}

void rc5_encrypt(rc5_ctx *c, u4 *data, int blocks){
        u4 *d,*sk;
        int h,i,rc;

     d = data;
        sk = (c->xk)+2;
        for(h=0;h<blocks;h++){
                d[0] += c->xk[0];
                d[1] += c->xk[1];
                for(i=0;i<c->nr*2;i+=2){
                        d[0] ^= d[1];
                        rc = d[1] & 31;
                        d[0] = ROTL32(d[0],rc);
                        d[0] += sk[i];
               d[1] ^= d[0];
                        rc = d[0] & 31;
                        d[1] = ROTL32(d[1],rc);
                        d[1] += sk[i+1];
/*printf("Round %03d : %08lx %08lx  sk= %08lx %08lx\n",i/2,
                                d[0],d[1],sk[i],sk[i+1]);*/
                }
          d+=2;
        }
}

void rc5_decrypt(rc5_ctx *c, u4 *data, int blocks){
     u4 *d,*sk;
        int h,i,rc;

     d = data;
        sk = (c->xk)+2;
     for(h=0;h<blocks;h++){
                for(i=c->nr*2-2;i>=0;i-=2){
```

```
/*printf("Round %03d: %08lx %08lx  sk: %08lx %08lx\n",
        i/2,d[0],d[1],sk[i],sk[i+1]); */
                        d[1] -= sk[i+1];
                        rc = d[0] & 31;
                        d[1] = ROTR32(d[1],rc);
                        d[1] ^= d[0];

                        d[0] -= sk[i];
                        rc = d[1] & 31;
                        d[0] = ROTR32(d[0],rc);
                d[0] ^= d[1];
                 }
                 d[0] -= c->xk[0];
                 d[1] -= c->xk[1];
        d+=2;
     }
}

void rc5_key(rc5_ctx *c, u1 *key, int keylen){
     u4 *pk,A,B; /* padded key */
     int xk_len, pk_len, i, num_steps,rc;
     u1 *cp;

     xk_len = c->nr*2 + 2;
     pk_len = keylen/4;
     if((keylen%4)!=0) pk_len += 1;

     pk = (u4 *) malloc(pk_len * 4);
     if(pk==NULL) {
          printf("An error occurred!\n");
          exit(-1);
     }

     /* Initialize pk -- this should work on Intel machines, anyway.... */
     for(i=0;i<pk_len;i++) pk[i]=0;
     cp = (u1 *)pk;
     for(i=0;i<keylen;i++) cp[i]=key[i];

     /* Initialize xk. */
     c->xk[0] = 0xb7e15163; /* P32 */
     for(i=1;i<xk_len;i++) c->xk[i] = c->xk[i-1] + 0x9e3779b9; /* Q32 */

     /* TESTING */
     A = B = 0;
     for(i=0;i<xk_len;i++) {
          A = A + c->xk[i];
          B = B ^ c->xk[i];
     }

     /* Expand key into xk. */
     if(pk_len>xk_len) num_steps = 3*pk_len;else num_steps = 3*xk_len;

     A = B = 0;
     for(i=0;i<num_steps;i++){
          A = c->xk[i%xk_len] = ROTL32(c->xk[i%xk_len] + A + B,3);
          rc = (A+B) & 31;
```

```
            B = pk[i%pk_len] = ROTL32(pk[i%pk_len] + A + B,rc);
        }

        /* Clobber sensitive data before deallocating memory. */
        for(i=0;i<pk_len;i++) pk[i] =0;

        free(pk);
}

void main(void){
        rc5_ctx c;
        u4 data[8];
        char key[] = "ABCDE";
        int i;

        printf("------------------------------------------------\n");

           for(i=0;i<8;i++) data[i] = i;
        rc5_init(&c,10); /* 10 rounds */
        rc5_key(&c,key,5);

           rc5_encrypt(&c,data,4);
           printf("Encryptions:\n");
           for(i=0;i<8;i+=2) printf("Block %01d = %08lx %08lx\n",
                                    i/2,data[i],data[i+1]);
           rc5_decrypt(&c,data,2);
        rc5_decrypt(&c,data+4,2);
           printf("Decryptions:\n");
           for(i=0;i<8;i+=2) printf("Block %01d = %08lx %08lx\n",
                                    i/2,data[i],data[i+1]);

}
```

5. ΕΠΙΛΟΓΟΣ

Η κρυπτογραφία είναι ίσως από τις ποιο εξελίξιμες αρχαίες επιστήμες παγκοσμιος, και αυτό δεν είναι τυχαίο αν αναλογιστεί κανείς την χρήση της και την συμβολή της στην καθημερινότητα μας όπως και στην ιστορία γενικότερα. Σίγουρα ο κόσμος δεν θα ήταν ο ίδιος χωρίς την κρυπτογραφία, η ιδία η ιστορία θα ήταν πολύ διαφορετικά γραμμένη . Αυτό φαίνεται κυρίως στους πιο προσφάτους πολέμους, βλέπε πρώτο και δεύτερο παγκόσμιο που είναι και οι πιο γνωστοί. Ίσως η έκβαση τους να είχε πάρει άλλη τροπή αν δεν υπήρχε η κρυπτογραφία η αν δεν είχε χρησιμοποιηθεί κατάλληλα.

Καθημερινά στην πλειοψηφεία των καθημερινών μας συναλλαγών κυρίως ηλεκτρονικών η κρυπτογραφία παίζει σημαντικότατο ρόλο και είναι αναπόσπαστο κομμάτι της ζωής μας πλέον ,π.χ στις τράπεζες (παραδείγματα χρήσης της κρυπτογραφίας αναφέρω στο κεφαλαίο 1), στις τηλεπικοινωνίες που είναι καθημερινά και κοντινά μας παραδείγματα .

Σήμερα η κρυπτογραφία μαζί με την άνθηση της πληροφορίας γνωρίζει μια παράλληλη εξελικτική άνθηση. Είναι ένα χρήσιμο εργαλείο το οποίο όταν χρησιμοποιείται ορθά, χρήση : (ηθική, ασφαλεία, ειρήνη) μας εξασφαλίζει μια καλύτερη zωή.

Στην εργασία αυτή δεν επιχειρώ να εκβαθύνω στην επιστήμη της κρυπτογραφίας καθ’ αυτής (Δεν είμαι γνώστης άλλωστε) αλλά να δώσω μια γενική εικόνα αυτής με κάποια παραδείγματα έτσι ώστε ο αναγνώστης να μπορέσει να καταλάβει κάποιες βασικές αρχές της.

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

1. U.S. DEPARTMENT OF COMMERCE/National Institute of Standards and Technology. **FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION.**

2. Καρατζάς Ε., Κασσιός Α., Χριστοδούλου Δ. (2001). **Τεχνικές κρυπτογραφίας και η εφαρμογή τους στο χώρο του Διαδικτύου.** Μεταπτυχιακή εργασία στα πλαίσια του μαθήματος «Αλγοριθμικά θέματα Δικτύων και Τηλεματικής». Τμήμα Μηχανικών Η/Υ και Πληροφορικής Πανεπιστημίου Πατρών.

3. Singh S. (1999). **Κώδικες και Μυστικά.**

4. Tatersall J. (2001). E1. **Number Theory in 9 Chapters**. C.UP.

5. Schneier B. (1996). **Applied Chryptography**. Second edition.

ΔΙΑΔΙΚΤΥΑΚΕΣ ΔΙΕΥΘΥΝΣΕΙΣ

www.wikipedia.com