ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΡΗΤΗΣ
Παράρτημα Χανίων
Τμήμα Ηλεκτρονικής

Θέμα:

«Μελέτη Αλγορίθμων Κρυπτογράφησης και Υλοποίηση του DES και Triple-DES σε FPGA με τη χρήση της Γλώσσας Περιγραφής Υλικού VHDL»

Πτυχιακή Εργασία

Σπουδαστή: **Φιολιτάκη Αντωνίου**

Επιβλέπων: Δρ. Μηχ. Πετράκης Νικόλαος

Χανιά
Δεκέμβριος 2005

1 Εισαγωγή

Είναι γεγονός ότι, από το πολύ απόμακρο παρελθόν μέχρι σήμερα, η ανάγκη για αποθήκευση ή/και μεταφορά πληροφοριών με μυστικό τρόπο αποτελεί αναπόσπαστο τμήμα της ιστορίας του ανθρώπου. Η ανάγκη αυτή δημιουργήθηκε λόγω της συνύπαρξης διαφορετικών κοινωνικών- πολιτικών-στρατιωτικών ακόμα και θρησκευτικών συμφερόντων ή πεποιθήσεων.

Στις μέρες μας έχει γίνει επιτακτικότερη αυτή τη απαίτηση για ασφαλή αποθήκευση/μετάδοση των πληροφοριών λόγω της τεχνολογικής εξέλιξης των υπολογιστών και του Διαδικτύου, τα οποία προσφέρουν την δυνατότητα στο ευρύ κοινό να έχει πρόσβαση σε πληροφορίες που μέχρι πριν από μερικά χρόνια θα ήταν αδιανόητο. Κάθε πληροφορία που μεταδίδεται, είναι πλέον αναγκαίο να είναι σε τέτοια μορφή η οποία να μην είναι κατανοητή από κανέναν άλλο παρά μόνο από τους καθορισμένους παραλήπτες. Αυτόν τον ρόλο επιτελεί η επιστήμη της κρυπτολογίας, η οποία ασχολείται τόσο με την διαδικασία εύρεσης κρυπτογραφικών μεθόδων-αλγορίθμων, όσο και με τον έλεγχό τους για την ασφάλεια που μπορούν να προσφέρουν.

Είναι παραδεκτό σήμερα ότι η ασφάλεια των αλγορίθμων κρυπτογράφησης δεν πρέπει να βασίζεται σε μυστικές μεθόδους ή κρυφές διαδικασίες. Θεωρούμε δηλαδή ότι οι μέθοδοι και οι αλγόριθμοι κρυπτογράφησης είναι πλήρως γνωστοί και δεν αποτελούν πια μυστικό, ενώ η δύναμη της κάθε κρυπτογράφησης είναι το ίδιο το κλειδί το οποίο για να ανακαλυφθεί από τον επίδοξο κρυπταναλυτή απαιτεί αρκετά χρόνια και τεράστια υπολογιστική.

Η κρυπτογραφία στις μέρες μας έχει αποκτήσει τεράστια σημασία. Αυτό φαίνεται από το γεγονός ότι στις ΗΠΑ οι Ομοσπονδιακές Αρχές απαγορεύουν την παραγωγή και εξαγωγή από την χώρα κρυπτογραφικών μη κρατικών προγραμμάτων, εκτός από αυτά που είναι παρωχημένης εποχής-τεχνολογίας.

Η παρούσα πτυχιακή εργασία, με στόχο να παρουσιάσει αναλυτικά όλη την διαδικασία που ακολουθήθηκε για την υλοποίηση σε υλικό (Hardware) των αλγορίθμων κρυπτογράφησης DES και Triple-DES, εκτείνεται συνολικά σε έξι κεφάλαια και δύο παραρτήματα. Εκτός από παρόν κεφάλαιο που αποτελεί μια σύντομη εισαγωγή, στο δεύτερο κεφάλαιο παρουσιάζεται μία συνοπτική αλλά αναγκαία αναδρομή στην ιστορία της κρυπτογραφίας από την εμφάνισή της έως τις ημέρες μας.

Στο τρίτο κεφάλαιο, αναλύονται τόσο οι θεμελιώδεις έννοιες όσο και η ορολογία της κρυπτολογίας, εισάγοντας τον αναγνώστη σταδιακά στις τεχνικές της κρυπτογραφίας, στις αρχές σχεδιασμού και στα είδη κρυπτογράφησης. Δίνεται έμφαση στην κρυπτογραφία ιδιωτικού κλειδιού και στις τέσσερις βασικές παραλλαγές του τρόπου λειτουργίας της.

Το τέταρτο κεφάλαιο περιγράφει με λεπτομέρεια οτιδήποτε χρειάζεται κάποιος να γνωρίζει για την λειτουργία του προτύπου κρυπτογράφησης DES, στοιχεία που θα φανούν χρήσιμα, στην συνέχεια, προκειμένου να γίνει κατανοητή η υλοποίησή του σε ψηφιακό κύκλωμα.

Στο πέμπτο κεφάλαιο υλοποιούνται "Hardware" οι αλγόριθμοι DES και Triple-DES χρησιμοποιώντας την γλώσσα περιγραφής υλικού VHDL και το ολοκληρωμένο περιβάλλον λογισμικού ISE (Integrated Software Environment) της Xilinx, τα οποία έχουμε στην διάθεση μας από την συνδρομή-συμμετοχή του Ιδρύματός μας στον ευρωπαϊκό οργανισμό Europractice. Μετά την ολοκλήρωση της σχεδίασης έγινε ο απαραίτητος έλεγχος ορθότητας του κυκλώματος με τη διεξαγωγή προσομοιώσεων κρυπτογράφησης και αποκρυπτογράφησης, με την βοήθεια γνωστών παραδειγμάτων από την βιβλιογραφία.

Τέλος, στο έκτο κεφάλαιο εκθέτονται γενικά συμπεράσματα και παρατηρήσεις που προήλθαν κατά την ενασχόλησή μου με την παρούσα εργασία.

Ολόκληρος ο κώδικας VHDL που κατασκευάστηκε για την υλοποίηση αυτή υπάρχει στο συνοδευτικό CD (μαζί με τα αρχεία προσομοίωσης κάθε βαθμίδας) ενώ τμήμα του παρουσιάζεται στο παράρτημα Α. Στο ίδιο CD υπάρχει μέρος της βιβλιογραφίας που χρησιμοποιήθηκε καθώς επίσης και η παρούσα εργασία σε ηλεκτρονική μορφή.

Στο παράρτημα Β παρουσιάζεται συνοπτικά η πορεία της κρυπτογραφίας από την πρώτη καταγραφή της εμφάνισης της μέχρι πρόσφατα.

2 Η Ιστορία της Κρυπτογραφίας

Η ύπαρξη της κρυπτογραφίας είναι τόσο παλιά όσο και ο πολιτισμός. Η ανθρώπινη επιθυμία για μυστικότητα κατά την διάρκεια της επικοινωνίας οδήγησε αναπόφευκτα στην κρυπτογραφία. Ο όρος κρυπτογραφία συνδυάζει δύο ελληνικές λέξεις, την λέξη «κρυπτό» που σημαίνει μυστικό και την λέξη «γράφος», προκειμένου να αποδοθεί με ακρίβεια η λειτουργία της.

Η κρυπτογραφία, έχει μία πάρα πολύ μεγάλη και συναρπαστική ιστορία, της οποίας η αρχή εντοπίζεται σύμφωνα με το βιβλίο του Kahn "The Codebreakers" πριν από 4000 χρόνια περίπου και συνδέεται με τους Αιγυπτίους. Η ιστορία της μπορεί να χωριστεί σε τρεις περιόδους με σκοπό την καλύτερη ανάλυση και κατανόηση της.

Έτσι, η πρώτη περίοδος εκτείνεται από την περίοδο από την πρώτη εμφάνισή της μέχρι και τον 19ο αιώνα. Η επόμενη περίοδος χαρακτηρίζεται από την επέκταση της χρήσης του τηλέγραφου, δηλαδή στις αρχές του 20ου αιώνα και φτάνει μέχρι την δεκαετία του 1950 από όπου και αρχίζει η τελευταία περίοδος της ανάπτυξης της κρυπτογραφίας, η οποία συνεχίζεται στις μέρες μας.

Στα κεφάλαια που ακολουθούν θα αναφερθούν οι κυριότεροι σταθμοί στην πορεία της κρυπτογραφίας μέσα στους αιώνες. Συνοπτικά η εξέλιξη της κρυπτογραφίας από το 1900 π.Χ. έως και το τέλος του $20^{ου}$ αιώνα παρουσιάζεται στο Παράρτημα Β1.

2.1 Πρώτη Περίοδος Κρυπτογραφίας (1900 π.Χ. – 1900 μ.Χ.)

Κατά την διάρκεια αυτής της περιόδου αναπτύχθηκε μεγάλο πλήθος μεθόδων και αλγορίθμων κρυπτογράφησης, που βασιζόταν κυρίως σε απλές αντικαταστάσεις γραμμάτων. Όλες αυτές δεν απαιτούσαν εξειδικευμένες γνώσεις και πολύπλοκες συσκευές αλλά στηριζόταν στην ευφυΐα και την ευρηματικότητα των δημιουργών τους. Όλα αυτά τα συστήματα έχουν στις μέρες μας κρυπταναλυθεί και έχει αποδειχθεί, ότι, εάν μας είναι γνωστό ένα μεγάλο κομμάτι του κρυπτογραφημένου μηνύματος, τότε το αρχικό κείμενο μπορεί σχετικά εύκολα να επανακτηθεί.

Όπως προκύπτει, από μία μικρή σφηνοειδή επιγραφή, που ανακαλύφθηκε στις όχθες του ποταμού Τίγρη, οι πολιτισμοί που αναπτύχθηκαν στην Μεσοποταμία ασχολήθηκαν με την κρυπτογραφία ήδη από το 1500 π.Χ. Η επιγραφή αυτή περιγράφει μία μέθοδο κατασκευής σμάλτων για αγγειοπλαστική και θεωρείται ως το αρχαιότερο κρυπτογραφημένο κείμενο με βάση τον Kahn. Επίσης ως το αρχαιότερο βιβλίο κρυπτοκωδικών στον κόσμο θεωρείται μία σφηνοειδής επιγραφή στα Σούσα της Περσίας η οποία περιλαμβάνει τους αριθμούς 1 έως 8 και από το 32 έως

το 35, τοποθετημένους τον ένα κάτω από τον άλλο, ενώ απέναντι τους βρίσκονται τα αντίστοιχα για τον καθένα σφηνοειδή σύμβολα.

Η πρώτη στρατιωτική χρήση της κρυπτογραφίας αποδίδεται στους Σπαρτιάτες. Γύρω στον 5ο π.Χ. αιώνα εφεύραν τη «σκυτάλη», την πρώτη κρυπτογραφική συσκευή, στην οποία, χρησιμοποίησαν για την κρυπτογράφηση, τη μέθοδο της αντικατάστασης. Όπως αναφέρει ο Πλούταρχος, η «Σπαρτιατική Σκυτάλη» Σχήμα (2.1), ήταν μια ξύλινη ράβδος, ορισμένης διαμέτρου, γύρω από την οποία ήταν τυλιγμένη ελικοειδώς μια λωρίδα περγαμηνής. Το κείμενο ήταν γραμμένο σε στήλες, ένα γράμμα σε κάθε έλικα, όταν δε ξετύλιγαν τη λωρίδα, το κείμενο ήταν ακατάληπτο εξαιτίας της ανάμειξης των γραμμάτων. Το «κλειδί» ήταν η διάμετρος της σκυτάλης.

Σχήμα 2.1 : Η Σπαρτιατική Σκυτάλη, μια πρώιμη συσκευή για την κρυπτογράφηση.

Στην αρχαιότητα, χρησιμοποιήθηκαν κυρίως συστήματα τα οποία βασίζονταν στην στεγανογραφία και όχι τόσο στην κρυπτογραφία. Οι Έλληνες συγγραφείς δεν αναφέρουν αν και πότε χρησιμοποιήθηκαν συστήματα γραπτής αντικατάστασης γραμμάτων, αλλά τα βρίσκουμε στους Ρωμαίους, κυρίως την εποχή του Ιουλίου Καίσαρα. Ο Ιούλιος Καίσαρας έγραφε στον Κικέρωνα και σε άλλους φίλους του, αντικαθιστώντας τα γράμματα του κειμένου, με γράμματα, που βρίσκονται 3 θέσεις μετά, στο Λατινικό Αλφάβητο. Έτσι, σήμερα το σύστημα κρυπτογράφησης που στηρίζεται στην αντικατάσταση των γραμμάτων του αλφαβήτου με άλλα που βρίσκονται σε καθορισμένο αριθμό θέσης πριν ή μετά λέγεται κρυπτοσύστημα αντικατάστασης του Καίσαρα. Ο Καίσαρας, χρησιμοποίησε και άλλα πιο πολύπλοκα συστήματα κρυπτογράφησης, για τα οποία έγραψε ένα βιβλίο ο Valerius Probus, το οποίο δυστυχώς δεν διασώθηκε, αλλά αν και χαμένο, θεωρείται το πρώτο βιβλίο κρυπτολογίας. Το σύστημα αντικατάστασης του Καίσαρα, χρησιμοποιήθηκε ευρύτατα και στους επόμενους αιώνες.

Στην διάρκεια του Μεσαίωνα, η κρυπτολογία ήταν κάτι το απαγορευμένο και αποτελούσε μια μορφή αποκρυφισμού και μαύρης μαγείας, κάτι που συντέλεσε στην καθυστέρηση της ανάπτυξης της. Η εξέλιξη τόσο της κρυπτολογίας, όπως και των μαθηματικών, συνεχίζεται στον Αραβικό κόσμο. Στο γνωστό μυθιστόρημα «Χίλιες και μία νύχτες» κυριαρχούν οι λέξεις-αινίγματα,

οι γρίφοι, τα λογοπαίγνια και οι αναγραμματισμοί. Έτσι, εμφανίστηκαν βιβλία που περιείχαν κρυπταλφάβητα, όπως το αλφάβητο «Dawoudi» που πήρε το όνομα του από τον βασιλιά Δαυίδ. Οι Άραβες είναι οι πρώτοι που ανακάλυψαν αλλά και χρησιμοποίησαν μεθόδους κρυπτανάλυσης. Το κυριότερο εργαλείο στην κρυπτανάλυση, η χρησιμοποίηση των συχνοτήτων των γραμμάτων κειμένου, σε συνδυασμό με τις συχνότητες εμφάνισης στα κείμενα των γραμμάτων της γλώσσας, ανακαλύφθηκε από αυτούς γύρω στον 14ο αιώνα. Η κρυπτογραφία λόγω των στρατιωτικών εξελίξεων, σημείωσε σημαντική ανάπτυξη στους επόμενους αιώνες. Ο Ιταλός Giovanni Batista Porta, το 1563, δημοσίευσε το περίφημο για την κρυπτολογία βιβλίο «De furtivis literarum notis», με το οποίο έγιναν γνωστά τα πολύαλφαβητικά συστήματα κρυπτογράφησης και τα διγραφικά κρυπτογραφήματα, στα οποία, δύο γράμματα αντικαθίστανται από ένα. Σημαντικός εκπρόσωπος εκείνης της εποχής είναι και ο Γάλλος Vigenere, του οποίου ο πίνακας πολυαλφαβητικής αντικατάστασης, χρησιμοποιείται ακόμη και σήμερα.

Ο C.Wheatstone, γνωστός από τις μελέτες του στον ηλεκτρισμό, παρουσίασε την πρώτη μηχανική κρυπτοσυσκευή, η οποία απετέλεσε τη βάση για την ανάπτυξη των κρυπτομηχανών της δεύτερης ιστορικής περιόδου της κρυπτογραφίας.

Η μεγαλύτερη αποκρυπτογράφηση, ήταν αυτή των αιγυπτιακών ιερογλυφικών τα οποία επί αιώνες, παρέμεναν μυστήριο και οι αρχαιολόγοι μόνο εικασίες μπορούσαν να διατυπώσουν για τη σημασία τους. Ωστόσο, χάρη σε μία κρυπταναλυτική εργασία, τα ιερογλυφικά εν τέλει αναλύθηκαν και έκτοτε οι αρχαιολόγοι είναι σε θέση να διαβάζουν ιστορικές επιγραφές. Τα αρχαιότερα ιερογλυφικά , χρονολογούνται στο 3000 π.Χ. Τα σύμβολα των ιερογλυφικών ήταν υπερβολικά πολύπλοκα για την καταγραφή των συναλλαγών εκείνης της εποχής. Έτσι, παράλληλα με αυτά, αναπτύχθηκε για καθημερινή χρήση η ιερατική γραφή, που ήταν μία συλλογή συμβόλων, τα οποία ήταν εύκολα τόσο στο γράψιμο όσο και στην ανάγνωση. Τον 17ο αιώνα αναθερμάνθηκε το ενδιαφέρον για την αποκρυπτογράφηση των ιερογλυφικών, έτσι το 1652 ο Γερμανός ιερέας Α. Κίρχερ, εξέδωσε ένα λεξικό ερμηνείας τους, με τίτλο «Oedipous Aegyptiakus». Με βάση αυτό, προσπάθησε να ερμηνεύσει τις αιγυπτιακές γραφές, αλλά η προσπάθεια του αυτή ήταν κατά γενική ομολογία αποτυχημένη. Για παράδειγμα, το όνομα του Φαραώ Απρίη, το ερμήνευσε σαν «τα ευεργετήματα του θεϊκού Όσιρι εξασφαλίζονται μέσω των ιερών τελετών της αλυσίδας των πνευμάτων, ώστε να επιδαψιλευθούν τα δώρα του Νείλου». Παρόλα αυτά, η προσπάθεια του άνοιξε τον δρόμο προς την σωστή ερμηνεία των ιερογλυφικών, που προχώρησε χάρη στην ανακάλυψη της «Στήλης της Ροζέτας». Ήταν μια πέτρινη στήλη που βρήκαν τα στρατεύματα του Ναπολέοντα στην Αίγυπτο και είχε χαραγμένο πάνω της το ίδιο κείμενο, τρεις φορές. Μια στα ιερογλυφικά, μια στα ελληνικά και μια στα ιερατικά. Δύο μεγάλοι αποκρυπτογράφοι της εποχής, ο Γιάνγκ και ο Σαμπολιόν, μοιράστηκαν την δόξα της ερμηνείας τους. Οι προϊστορικοί πληθυσμοί χρησιμοποίησαν τρεις γραφές, μέχρι να επινοήσουν αλφάβητο, γύρω στο 850 π.Χ.

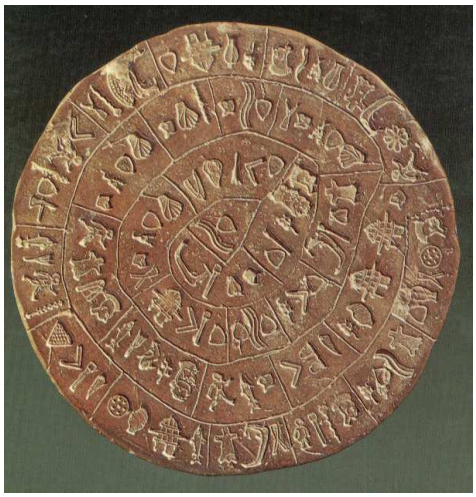
Χρονολογικά, οι γραφές αυτές κατατάσσονται ως εξής :

3000 1600 π.Χ. : Εικονογραφική (Ιερογλυφική) γραφή

1850 1450 π.Χ.: Γραμμική γραφή Α

1450 1200 π.Χ.: Γραμμική Γραφή Β

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 μυστηριώδεις αρχαία ευρωπαϊκή γραφή.



Σχήμα 2.2 : Ο Δίσκος της Φαιστού

Οι πρώτες επιγραφές με Γραμμική γραφή, ανακαλύφθηκαν από τον Sir Arthur Evans, το μεγάλο Άγγλο αρχαιολόγο, που άνεσκαψε συστηματικά την Κνωσό το 1900. Ο ίδιος, ονόμασε αυτή τη γραφή γραμμική, επειδή τα γράμματα της είναι γραμμές (ένα γραμμικό σχήμα) και όχι σφήνες, όπως στην σφηνοειδή γραφή ή εικόνες όντων, όπως στην αιγυπτιακή ιερατική. Η γραμμική γραφή Α είναι μάλλον η γραφή των Μινωιτών (από το μυθικό Μίνωα, βασιλιά της Κνωσού), των κατοίκων της αρχαίας Κρήτης και από αυτή ίσως να προήλθε το σημερινό ελληνικό αλφάβητο. Τα

γράμματα της γραμμικής γραφής χαράζονταν με αιχμηρό αντικείμενο πάνω σε πήλινες πλάκες, οι οποίες κατόπιν ξεραίνονταν σε φούρνους. Οι περισσότερες από τις
επιγραφές με Γραμμική γραφή Α (περίπου 1500) είναι λογιστικές και περιέχουν εικόνες ή συντομογραφίες των εμπορεύσιμων προϊόντων και αριθμούς για υπόδειξη της ποσότητας ή οφειλής.

Ο Έβανς κατέγραψε 135 σύμβολα της. Χρησιμοποιήθηκε κυρίως στην Κρήτη, αν και ορισμένα πρόσφατα ευρήματα καταδεικνύουν ότι μπορεί να αποτέλεσε μέσο γραφής και αλλού, αφού επιγραφές με Γραμμική Α έχουν βρεθεί στην Κνωσό και Φαιστό της Κρήτης, αλλά και στη Μήλο και Θήρα. Πλάκες με επιγραφές σε γραμμική Α, εκτίθενται στο Μουσείο Ηρακλείου. Παρά την πρόοδο που έχει σημειωθεί, η γραμμική γραφή Α δεν έχει αποκρυπτογραφηθεί ακόμη.

Ο Evans έδωσε και την ονομασία στην Γραμμική Γραφή Β, επειδή αναγνώρισε ότι πρόκειται για συγγενική γραφή με την γραμμική Α, πιο πρόσφατη ωστόσο και εξελιγμένη. Με βάση όσα γνωρίζουμε σήμερα, η γραφή αυτή υιοθετήθηκε αποκλειστικά για λογιστικούς σκοπούς. Πινακίδες χαραγμένες με την γραμμική γραφή Β, βρέθηκαν στην Κνωσό, στα Χανιά αλλά και στην Πύλο, Μηκύνες, Θήβα και Τίρυνθα. Σήμερα, αποτελούν ένα σύνολο 10.000 τεμαχίων. Τα σχήματα των πινακίδων της γραφής αυτής, ποικίλουν, επικρατούν όμως οι φυλλοειδείς και «σελιδόσχημες», οι οποίες διαφέρουν ως προς τις διαστάσεις, ανάλογα με τις προτιμήσεις του κάθε γραφέα. Έπλαθαν πηλό σε σχήμα κυλίνδρου, τον τοποθετούσαν σε λεία επιφάνεια και την πίεζαν, μέχρι να γίνει επίπεδη, επιμήκης και συμπαγής πινακίδα, σαφώς διαφοροποιημένη σε δύο επιφάνειες: μία επίπεδη λειασμένη, που επρόκειτο να αποτελέσει την κύρια γραφική επιφάνεια και μία κυρτή, που συνήθως έμενε άγραφη. Πολλές φορές, όταν τα κείμενα απαιτούσαν περισσότερες από μία πινακίδες, έχουμε τις αποκαλούμενες «ομάδες» ή «πολύπτυχα» πινακίδων, οι οποίες εμφανίζουν κοινά χαρακτηριστικά και ως προς την αποξήρανση και το μίγμα του πηλού και κυρίως, ως προς το γραφικό χαρακτήρα του ίδιου του γραφέα. Τα πολύπτυχα αυτά, φυλάσσονταν σε αρχειοφυλακεία και ταξινομούνταν κατά θέματα σε ξύλινα κιβώτια. Για να γνωρίζει ο ενδιαφερόμενος το περιεχόμενο των καλαθιών, κυρίως, χρησιμοποιούσαν ετικέτες: ένα σφαιρίδιο πηλού, εντυπωμένο στην πρόσθια πλευρά, στο οποίο καταγράφονταν συνοπτικές πληροφορίες. Συστηματικά, με την γραφή αυτή, με την οποία είχε πραγματικό πάθος, ασχολήθηκε ο Άγγλος αρχιτέκτονας και ερασιτέχνης αρχαιολόγος Μ. Βέντρις. Ήταν ο πρώτος που κατάλαβε, ότι επρόκειτο για κάποιο είδος ελληνικής γραφής αλλά η άποψη του αυτή δεν έγινε δεκτή αρχικά, από τους ειδικούς. Στην συνέχεια όμως, αρκετοί προσχώρησαν στην άποψη του. Ένας από αυτούς ήταν ο κρυπταναλυτής Τζον Τσάντγουικ, ο οποίος, στη διάρκεια του πολέμου είχε εργασθεί στην ανάλυση της γερμανικής κρυπτομηχανής Enigma.

Προσπάθησε να μεταφέρει την πείρα του στην κρυπτανάλυση της Γραμμικής Β, αλλά χωρίς επιτυχία μέχρι τότε. Όμως, ο συνδυασμός των δύο επιστημόνων, έφερε το πολυπόθητο

αποτέλεσμα. Το 1953 κατέγραψαν τα συμπεράσματα τους στο μνημειώδες έργο «Μαρτυρίες για την ελληνική διάλεκτο στα μυκηναϊκά αρχεία», που έγινε το πιο διάσημο άρθρο κρυπτανάλυσης. Η αποκρυπτογράφηση της Γραμμικής Β, απέδειξε ότι επρόκειτο για ελληνική γλώσσα, ότι οι Μινωίτες της Κρήτης μιλούσαν ελληνικά και ότι η δεσπόζουσα δύναμη εκείνη την εποχή ήταν οι Μυκήνες. Η αποκρυπτογράφηση της Γραμμικής Β θεωρήθηκε επίτευγμα, ανάλογο της κατάκτησης του Έβερεστ, που συνέβη την ίδια ακριβώς εποχή. Για αυτό και έγινε γνωστή σαν το «Έβερεστ της Ελληνικής αρχαιολογίας».

2.2 Δεύτερη Περίοδος Κρυπτογραφίας (1900 μ.Χ. – 1950 μ.Χ.)

Η δεύτερη περίοδος της κρυπτογραφίας όπως προαναφέραμε τοποθετείτε στις αρχές του 20ου αιώνα και φτάνει μέχρι το 1950. Καλύπτει, επομένως, τους δύο παγκόσμιους πολέμους, εξαιτίας των οποίων (λόγω της εξαιρετικά μεγάλης ανάγκης που υπήρξε για ασφάλεια κατά την μετάδοση ζωτικών πληροφοριών μεταξύ των στρατευμάτων των χωρών) αναπτύχθηκε η κρυπτογραφία τόσο όσο δεν είχε αναπτυχθεί τα προηγούμενα 3000 χρόνια. Τα κρυπτοσυστήματα αυτής της περιόδου αρχίζουν να γίνονται πολύπλοκα, και να αποτελούνται από μηχανικές και ηλεκτρομηχανικές κατασκευές, οι οποίες ονομάζονται «κρυπτομηχανές». Η κρυπτανάλυση τους, απαιτεί μεγάλο αριθμό προσωπικού, το οποίο εργαζόταν επί μεγάλο χρονικό διάστημα ενώ ταυτόχρονα γίνεται εξαιρετικά αισθητή η ανάγκη για μεγάλη υπολογιστική ισχύ. Παρά την πολυπλοκότητα που αποκτούν τα συστήματα κρυπτογράφησης κατά την διάρκεια αυτής της περιόδου η κρυπτανάλυση τους είναι συνήθως επιτυχημένη.

Οι Γερμανοί έκαναν εκτενή χρήση (σε διάφορες παραλλαγές) ενός συστήματος γνωστού ως Enigma Σχήμα (2.3). Ο Marian Rejewski, στην Πολωνία, επιτέθηκε και παραβίασε την πρώτη μορφή του γερμανικού στρατιωτικού συστήματος Enigma (μια ηλεκτρομηχανική κρυπτογραφική μηχανή) χρησιμοποιώντας θεωρητικά μαθηματικά το 1932. Ήταν η μεγαλύτερη σημαντική ανακάλυψη στην κρυπτολογική ανάλυση της χιλιετίας. Οι Πολωνοί συνέχισαν να παραβιάζουν τα μηνύματα που βασιζόταν στην κρυπτογράφηση με τον Enigma μέχρι το 1939. Τότε, ο γερμανικός στρατός έκανε κάποιες αλλαγές και οι Πολωνοί δεν μπόρεσαν να ακολουθήσουν γιατί η παραβίαση απαιτούσε περισσότερους πόρους από όσους μπορούσαν να διαθέσουν. Έτσι, εκείνο το καλοκαίρι μεταβίβασαν τη γνώση τους, μαζί με μερικές μηχανές που είχαν κατασκευάσει, στους Βρετανούς και τους Γάλλους. Ακόμη και ο Rejewski και οι μαθηματικοί και κρυπτογράφοι του, από τον Biuro Szyfrow, κατέληξαν με τους Βρετανούς και τους Γάλλους μετά από αυτή την εξέλιξη. Η εργασία αυτή, συνεχίστηκε από τον Alan Turing, τον Gordon Welchman, και από πολλούς άλλους στο Bletchley Park και οδήγησε σε συνεχείς παραβιάσεις των διαφόρων παραλλαγών του Enigma.

Σχήμα 2.3 : Η μηχανή Αίνιγμα χρησιμοποιήθηκε ευρέως από την Γερμανία

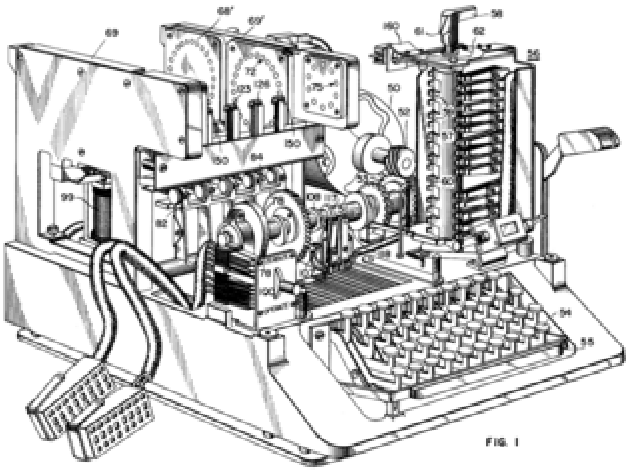
Οι κρυπτογράφοι του αμερικανικού ναυτικού (σε συνεργασία με Βρετανούς και Ολλανδούς κρυπτογράφους μετά από το 1940) έσπασαν αρκετά κρυπτό-συστήματα του Ιαπωνικού ναυτικού. Το σπάσιμο ενός από αυτά, του JN-25, οδήγησε στην αμερικανική νίκη στην μάχη του Midway.

Το Ιαπωνικό υπουργείο εξωτερικών χρησιμοποίησε ένα τοπικά αναπτυγμένο κρυπτογραφικό σύστημα, (που καλείται Purple), και χρησιμοποίησε επίσης διάφορες παρόμοιες μηχανές για τις συνδέσεις μερικών ιαπωνικών πρεσβειών. Μία από αυτές αποκαλέστηκε ως "Μηχανή-Μ" από τις ΗΠΑ, ενώ μια άλλη αναφέρθηκε ως «Red» (Κόκκινη). Μια ομάδα του αμερικανικού στρατού, η αποκαλούμενη SIS, κατάφερε να σπάσει το ασφαλέστερο ιαπωνικό διπλωματικό σύστημα κρυπτογράφησης (μια ηλεκτρομηχανική μηχανή αποκαλούμενη Purple από τους Αμερικανούς) πριν καν ακόμη αρχίσει ο δεύτερος παγκόσμιος πόλεμος. Οι Αμερικανοί αναφέρονται στο αποτέλεσμα της κρυπτανάλυσης, ειδικότερα της μηχανής Purple, αποκαλώντας το ως Magic (Μαγεία).

Οι συμμαχικές κρυπτό-μηχανές που χρησιμοποιήθηκαν στον δεύτερο παγκόσμιο πόλεμο περιλάμβαναν το βρετανικό TypeX και το αμερικανικό SIGABA (Σχήμα 2.4) και τα δύο ήταν ηλεκτρομηχανικά σχέδια παρόμοια στο πνεύμα με το Enigma, εν τούτοις με σημαντικές βελτιώσεις. Κανένα δεν έγινε γνωστό ότι παραβιάστηκε κατά τη διάρκεια του πολέμου. Τα στρατεύματα στο πεδίο μάχης χρησιμοποίησαν το M-209 και τη λιγότερη ασφαλή οικογένεια κρυπτό-μηχανών M-94. Οι Βρετανοί πράκτορες SOE χρησιμοποίησαν αρχικά ένα τύπο κρυπτογραφίας που βασιζόταν σε ποιήματα (τα απομνημονευμένα ποιήματα ήταν τα κλειδιά).

Οι Πολωνοί είχαν προετοιμαστεί για την εμπόλεμη περίοδο κατασκευάζοντας την κρυπτό-μηχανή LCD Lacida, η οποία κρατήθηκε μυστική ακόμη και από τον Rejewski. Όταν τον Ιούλιο

του 1941 ελέγχθηκε από τον Rejewski η ασφάλειά της, του πήρε ακριβώς μερικές ώρες για να την σπάσει και έτσι αναγκάστηκαν να την αλλάξουν βιαστικά. Τα μηνύματα που εστάλησαν με Lacida δεν ήταν, εντούτοις, συγκρίσιμα με αυτά του Αίνιγμα, αλλά η παρεμπόδιση θα μπορούσε να έχει σημάνει το τέλος της κρίσιμης κρυπταναλυτικής Πολωνικής προσπάθειας.



Σχήμα 2.4 : Κρυπτό-μηχανή SIGABA

2.3 Τρίτη Περίοδος Κρυπτογραφίας (1950 μ.Χ. - Σήμερα)

Αυτή η περίοδος χαρακτηρίζεται από την έξαρση της ανάπτυξης στους επιστημονικούς κλάδους των μαθηματικών, της μικροηλεκτρονικής και των υπολογιστικών συστημάτων.

Η εποχή της σύγχρονης κρυπτογραφίας αρχίζει ουσιαστικά με τον Claude Shannon, αναμφισβήτητα ο πατέρας των μαθηματικών συστημάτων κρυπτογραφίας. Το 1949 δημοσίευσε το έγγραφο «Θεωρία επικοινωνίας των συστημάτων μυστικότητας» (Communication Theory of Secrecy Systems) στο τεχνικό περιοδικό Bell System και λίγο αργότερα στο βιβλίο του, «Μαθηματική Θεωρία της Επικοινωνίας» (Mathematical Theory of Communication), μαζί με τον Warren Weaver. Αυτά, εκτός από τις άλλες εργασίες του επάνω στην θεωρία δεδομένων και επικοινωνίας καθιέρωσε μια στερεά θεωρητική βάση για την κρυπτογραφία και την κρυπτανάλυση. Εκείνη την εποχή η κρυπτογραφία εξαφανίζεται και φυλάσσεται από τις μυστικές υπηρεσίες κυβερνητικών επικοινωνιών όπως η NSA. Πολύ λίγες εξελίξεις δημοσιοποιήθηκαν ξανά μέχρι τα μέσα της δεκαετίας του '70, όταν όλα άλλαξαν.

Στα μέσα της δεκαετίας του '70 έγιναν δύο σημαντικές δημόσιες (δηλ. μη-μυστικές) πρόοδοι. Πρώτα ήταν η δημοσίευση του σχεδίου προτύπου κρυπτογράφησης DES (Data Encryption Standard)στον ομοσπονδιακό κατάλογο της Αμερικής στις 17 Μαρτίου 1975. Το προτεινόμενο DES υποβλήθηκε από την IBM, στην πρόσκληση του Εθνικού Γραφείου των Προτύπων (τώρα γνωστό ως NIST), σε μια προσπάθεια να αναπτυχθούν ασφαλείς ηλεκτρονικές εγκαταστάσεις επικοινωνίας για επιχειρήσεις όπως τράπεζες και άλλες μεγάλες οικονομικές οργανώσεις. Μετά

από τις συμβουλές και την τροποποίηση από την NSA, αυτό το πρότυπο υιοθετήθηκε και δημοσιεύθηκε ως ένα ομοσπονδιακή τυποποιημένο πρότυπο επεξεργασίας πληροφοριών το 1977 (αυτήν την περίοδο αναφέρεται σαν FIPS 46-3). Ο DES ήταν ο πρώτος δημόσια προσιτός αλγόριθμος κρυπτογράφησης που εγκρίνεται από μια εθνική αντιπροσωπεία όπως η NSA. Η απελευθέρωση της προδιαγραφής της από την NBS υποκίνησε μια έκρηξη δημόσιου και ακαδημαϊκού ενδιαφέροντος για τα συστήματα κρυπτογραφίας.

Ο DES αντικαταστάθηκε επίσημα από τον AES το 2001 όταν ανήγγειλε ο NIST το FIPS 197. Μετά από έναν ανοικτό διαγωνισμό, ο NIST επέλεξε τον αλγόριθμο Rijndael, που υποβλήθηκε από δύο Φλαμανδούς κρυπτογράφους, για να είναι το AES. Ο DES και οι ασφαλέστερες παραλλαγές του όπως ο 3DES ή TDES χρησιμοποιούνται ακόμα σήμερα, ενσωματωμένος σε πολλά εθνικά και οργανωτικά πρότυπα. Εντούτοις, το βασικό μέγεθος των 56-bit έχει αποδειχθεί ότι είναι ανεπαρκές να αντισταθεί στις επιθέσεις ωμής βίας (μια τέτοια επίθεση πέτυχε να σπάσει τον DES σε 56 ώρες ενώ το άρθρο που αναφέρεται ως το σπάσιμο του DES δημοσιεύτηκε από τον O'Reilly and Associates). Κατά συνέπεια, η χρήση απλής κρυπτογράφησης με τον DES είναι τώρα χωρίς την αμφιβολία επισφαλής για χρήση στα νέα σχέδια των κρυπτογραφικών συστημάτων και μηνύματα που προστατεύονται από τα παλαιότερα κρυπτογραφικά συστήματα που χρησιμοποιούν DES, και όλα τα μηνύματα που έχουν αποσταλεί από το 1976 με την χρήση DES, διατρέχουν επίσης σοβαρό κίνδυνο αποκρυπτογράφησης. Ανεξάρτητα από την έμφυτη ποιότητά του, το βασικό μέγεθος του DES (56-bit) ήταν πιθανά πάρα πολύ μικρό ακόμη και το 1976, πράγμα που είχε επισημάνει ο Whitfield Diffie. Υπήρξε επίσης η υποψία ότι κυβερνητικές οργανώσεις είχαν ακόμα και τότε ικανοποιητική υπολογιστική δύναμη ώστε να σπάσουν μηνύματα που είχαν κρυπτογραφηθεί με τον DES [TEX05].

3 Θεμελιώδεις Έννοιες Κρυπτολογίας

Η όλη διαδικασία της επεξεργασίας των μηνυμάτων καθώς και της κατασκευής διαφόρων αλγορίθμων και μηχανών κρυπτογράφησης περικλείονται μέσα στον όρο Κρυπτολογία. Έτσι, Κρυπτολογία είναι η επιστήμη η οποία περιλαμβάνει όλη την μελέτη της διαδικασία της κρυπτογράφησης και της κρυπτανάλυσης ενός μηνύματος προς αποστολή.

3.1 Κρυπτογράφηση- Αποκρυπτογράφηση

Ως Κρυπτογραφία, ορίζεται η διαδικασία μελέτης των μέσων που χρησιμοποιούνται για να μετατρέψουν μία πληροφορία από την κανονική της μορφή, η οποία είναι κατανοητή σε οποιονδήποτε, σε ένα ακατανόητο σύνολο που θα την καθιστά δυσανάγνωστη.

Εφαρμογή της Κρυπτογραφίας αποτελεί η Κρυπτογράφηση. Με τον όρο Κρυπτογράφηση καλούμε την μέθοδο μετασχηματισμού ενός απλού-μη κρυπτογραφημένου κειμένου (plaintext) σε ένα κρυπτογραφημένο κείμενο (ciphertext), ενώ ως Αποκρυπτογράφηση περιγράφεται η αντίστροφη διαδικασία. Ένα κρυπτογραφικό σύστημα είναι συνήθως μια ολόκληρη συλλογή αλγορίθμων. Σκοπός της κρυπτογράφησης είναι να εξασφαλίσει το απόρρητο των δεδομένων κρατώντας τα κρυφά από όλους όσους έχουν πρόσβαση σε αυτά.

Όλα τα προβλήματα που παρουσιάστηκαν κατά την διάρκεια της ιστορίας της κρυπτογραφίας οδήγησαν στο συμπέρασμα ότι η ασφάλεια της κρυπτογράφησης δεν θα πρέπει να εξαρτάται από την μυστικότητα της μεθόδου κρυπτογράφησης-αποκρυπτογράφησης. Ο μετασχηματισμός του απλού κειμένου σε κρυπτογραφημένο γίνεται πλέον με την βοήθεια μίας συνάρτησης η οποία όμως έχει ως παράμετρο ένα κλειδί. Ως κλειδί ορίζουμε ένα κομμάτι πληροφορίας το οποίο ελέγχει και καθορίζει τόσο την διαδικασία κρυπτογράφησης όσο και της αποκρυπτογράφησης.

Η προσπάθεια να κρατήσουμε έναν αλγόριθμο κρυφό, η οποία είναι ευρύτερα γνωστή ως ασφάλεια μέσω ασάφειας, δεν αποδίδει. Έτσι, έχει γίνει πλέον θεμελιώδης κανόνας της κρυπτογραφίας η αποδοχή ότι, η ασφάλεια και η μυστικότητα ενός αλγορίθμου έγκειται αποκλειστικά στα κλειδιά. Αυτή η αρχή ονομάζεται αρχή του Kerckoff (Kerckoff's principle) και πήρε το όνομα της από το Φλαμανδό στρατιωτικό κρυπτογράφο Auguste Kerckoff ο οποίος την διατύπωσε για πρώτη φορά το 1883 ως εξής:

«Η ασφάλεια ενός κρυπτοσυστήματος δεν εξαρτάται από τη μυστικότητα του αλγόριθμου κρυπτογράφησης. Η ασφάλεια του κρυπτοσυστήματος εξαρτάται μόνο από το να διατηρείται μυστικό το κλειδί»

3.2 Κρυπτογραφικές Αρχές

Παρά την ύπαρξη διαφόρων ειδών κρυπτογραφικών συστημάτων, πολλά από τα οποία θα μελετήσουμε στην συνέχεια, όλα ανεξαρτήτως ακολουθούν δύο θεμελιώδεις αρχές τις οποίες και θα αναλύσουμε.

Η πρώτη αρχή που διέπει τα κρυπτογραφικά μηνύματα είναι αυτή του «πλεονασμού». Τα κρυπτογραφημένα μηνύματα πρέπει να περιέχουν πληροφορίες οι οποίες δεν απαιτούνται για την κατανόηση του μηνύματος. Με αυτό τον τρόπο, γίνεται δύσκολο στους εισβολείς να στείλουν τυχαίες άχρηστες πληροφορίες με σκοπό να λαμβάνονται ως έγκυρα μηνύματα τα οποία έχουν αλλοιωθεί. Ωστόσο, η προσθήκη αυτή στα μηνύματα τα καθιστά πιο ευάλωτα σε τυχόν επιθέσεις. Ο πλεονασμός δεν πρέπει ποτέ να έχει την μορφή *n* μηδενικών στην αρχή ή στο τέλος ενός μηνύματος, επειδή η χρήση τέτοιων μηνυμάτων σε κάποιους κρυπτογραφικούς αλγορίθμους δίνει πιο προβλέψιμα αποτελέσματα, γεγονός που κάνει ευκολότερη την παραβίαση τους.

Μια δεύτερη κρυπτογραφική αρχή είναι η λεγόμενη «φρεσκάδα» την οποία θα πρέπει να έχουν τα λαμβανόμενα μηνύματα δηλαδή να έχουν αποσταλεί πάρα πολύ πρόσφατα. Με αυτό τον τρόπο αποτρέπεται η αναπαραγωγή παλιών μηνυμάτων από τους εισβολείς. Αυτή η αρχή γενικότερα εκφράζει την ανάγκη ύπαρξης κάποιας μεθόδου ματαίωσης των επιθέσεων αναπαραγωγής. Ένα τέτοιο μέτρο είναι να περιλαμβάνεται σε κάθε μήνυμα μια χρονοσφραγίδα που θα είναι έγκυρη για πάρα πολύ μικρό χρονικό διάστημα για παράδειγμα μόνο για 10 δευτερόλεπτα. Έτσι, ο παραλήπτης μπορεί να διατηρεί τα μηνύματα για το συγκεκριμένο χρονικό όριο ώστε να μπορεί να τα συγκρίνει με νεοεισερχόμενα και να φιλτράρει τα αντίγραφα. Τα μηνύματα εκείνα τα οποία θα είναι παλαιότερα από τα 10 δευτερόλεπτα θα απορρίπτονται.

3.3 Λειτουργίες της Κρυπτογράφησης

- **Εμπιστευτικότητα (Confidentiality)**

Είναι η προστασία από τη μη εξουσιοδοτημένη αποκάλυψη της πληροφορίας. Η εμπιστευτικότητα θα πρέπει να προσφέρεται με τέτοιο τρόπο ώστε να είναι αδύνατη η αποκάλυψη και πολλές φορές η ίδια η ύπαρξη της πληροφορίας σε μη εξουσιοδοτημένα άτομα. Η κρυπτογραφία χρησιμοποιείται για να μεταμορφώνει την πληροφορία που στέλνεται μέσω του Διαδικτύου (Internet) και αποθηκεύεται στους εξυπηρετητές (servers), έτσι ώστε να μην μπορούν να δουν το περιεχόμενο των δεδομένων αυτοί που υποκλέπτουν. Μερικοί ονομάζουν αυτή την ιδιότητα μυστικότητα (privacy) αλλά οι περισσότεροι χρησιμοποιούν αυτή τη λέξη για να αναφέρονται στην προστασία της ατομικής πληροφορίας.

- **Απόδειξη γνησιότητας ή Επικύρωση (Authentication)**

Είναι η εξασφάλιση του ότι γνωρίζουμε τον χρήστη ή γενικότερα την οντότητα που επικοινωνούμε (user/entity authentication). Οι Ψηφιακές Υπογραφές χρησιμοποιούνται για να εξακριβώνουν την ταυτότητα του αποστολέα ενός μηνύματος. Οι παραλήπτες ενός μηνύματος μπορούν να ελέγξουν την ταυτότητα του αποστολέα, ο οποίος υπέγραψε ψηφιακά το μήνυμα. Μπορούν να χρησιμοποιηθούν σε συνδυασμό με τα password ή και να τα αντικαταστήσουν.

Η Ψηφιακή υπογραφή είναι ένας τύπος μεθόδου για τις ψηφιακές πληροφορίες ανάλογη με την συνηθισμένη φυσική υπογραφή επάνω στο χαρτί, αλλά αναπαριστάμενη χρησιμοποιώντας τις τεχνικές από τον τομέα της κρυπτογράφησης δημόσιου κλειδιού. Μια μέθοδος ψηφιακών υπογραφών ορίζει γενικά δύο συμπληρωματικούς αλγορίθμους, έναν για την υπογραφή και ένα άλλο για την επαλήθευση, ενώ και η διαδικασία της παραγωγής της υπογραφής καλείται επίσης ψηφιακή υπογραφή.

Τα ψηφιακά χρήματα (ή ηλεκτρονικά χρήματα) αναφέρονται στα μετρητά και στις σχετικές συναλλαγές που γίνονται μεταξύ ατόμων η εταιριών χρησιμοποιώντας τα ηλεκτρονικά μέσα. Χαρακτηριστικά, αυτό περιλαμβάνει τη χρήση των δικτύων υπολογιστών (όπως το Διαδίκτυο) και των ψηφιακών συστημάτων αποθήκευσης δεδομένων. Πολλοί πολίτες στις μέρες μας έχουν χρησιμοποιήσει συστήματα κρυπτογραφίας επειδή η κρυπτογραφική τεχνολογία είναι σε ένα μεγάλο μέρος ενσωματωμένη στα σημερινά υπολογιστικά και τηλεπικοινωνιακά συστήματα.

- **Ακεραιότητα (Integrity)**

Είναι η προστασία από τη μη εξουσιοδοτημένη τροποποίηση των δεδομένων. Η ακεραιότητα θα πρέπει να παρέχει στον παραλήπτη και γενικότερα στον κάτοχο ενός μηνύματος την δυνατότητα να μπορεί να ανιχνεύσει πιθανές αλλαγές στο μήνυμα από μη εξουσιοδοτημένα άτομα. Υπάρχουν μέθοδοι που ελέγχουν αν ένα μήνυμα έχει μεταβληθεί την στιγμή της μεταφοράς. Συχνά αυτό γίνεται με τους κώδικες αποσύνθεσης μηνυμάτων ψηφιακά υπογεγραμμένων. Στον χώρο των τηλεπικοινωνιών και της θεωρίας της πληροφορίας η διαδικασία ελέγχου της ακεραιότητας είναι γνωστή ως ανίχνευση σφαλμάτων, όπου ένα μήνυμα μπορεί να υποστεί τροποποίηση λόγω του θορύβου του καναλιού επικοινωνίας.

- **Απαγόρευση απάρνησης (Non-repudiation)**

Είναι η υπηρεσία κατά την οποία ο παραλήπτης δεν μπορεί να απαρνηθεί ότι έλαβε (non-repudiation of destination) ή ότι έστειλε (non-repudiation of origin) ένα μήνυμα. Για τον λόγο αυτό δημιουργούνται οι κρυπτογραφικές αποδείξεις.

3.4 Βασικές αρχές σχεδιασμού κρυπτογραφημάτων ομάδας (block ciphers)

3.4.1 Τα μέτρα του Shannon

Ο Shannon ο θεμελιωτής της θεωρίας της πληροφορίας διατύπωσε το 1949 ένα σύνολο από μέτρα τα οποία χαρακτηρίζουν έναν ορθά σχεδιασμένο αλγόριθμο κρυπτογράφησης

- Βαθμός απαιτούμενης κρυπτογραφικής ασφάλειας. Το μέτρο αυτό αφορά το κέρδος του αντιπάλου σε πληροφορία, όταν παρατηρεί το κρυπτοκείμενο.
- Μήκος του κλειδιού. Η ευκολία χειρισμού του κλειδιού εξαρτάται από το μήκος του.
- Πρακτική εκτέλεση της κρυπτογράφησης και της αποκρυπτογράφησης. Η προσπάθεια που απαιτείται για την κρυπτογράφηση και την αποκρυπτογράφηση, σε χρόνο ή λειτουργίες.
- Διόγκωση του κρυπτοκειμένου. Είναι επιθυμητό το κρυπτοκείμενο να έχει το ίδιο μήκος (ή συγκρίσιμου μεγέθους) με το απλό κείμενο.
- Διάδοση των σφαλμάτων κρυπτογράφησης. Είναι επιθυμητό ένα σφάλμα κατά την κρυπτογράφηση να επηρεάζει σε όσο το δυνατό λιγότερο βαθμό την αποκρυπτογράφηση.

Η ύπαρξη των μέτρων σε ένα κρυπτοσύστημα είναι υποχρεωτική, αλλά συγχρόνως και αντιφατική, με αποτέλεσμα να μην υπάρχει στην πραγματικότητα κρυπτοσύστημα το οποίο να ικανοποιεί όλα τα μέτρα στο μέγιστο τους. Για παράδειγμα, πλήρης έλλειψη του πρώτου μέτρου σημαίνει ότι ο αντίπαλος μπορεί να ανακτήσει πλήρως το απλό κείμενο. Η πλήρης έλλειψη του τρίτου και τέταρτου μέτρου επιτρέπει κρυπτό-συστήματα που μπορούν να μεγιστοποιούν όλα τα άλλα μέτρα. Η πλήρης έλλειψη του πέμπτου μέτρου δέχεται ύπαρξη κρυπτοσυστήματος που μεγιστοποιεί όλα τα άλλα μέτρα, αλλά σε περίπτωση σφάλματος κατά την κρυπτογράφηση, η ανάκτηση του απλού κειμένου θα ήταν αδύνατη, ακόμη και για κάποιο τμήμα αυτού.

3.4.2 Σύγχυση (confusion) και Διάχυση (diffusion)

Δύο ιδιότητες που χρησιμοποιούνται στην σωστή σχεδίαση ενός κρυπταλγορίθμου είναι η σύγχυση (confusion)και η διάχυση (diffusion).

Έστω ένα απλό κείμενο το οποίο αντιστοιχεί σε ένα κρυπτοκείμενο μέσω ενός κρυπταλγόριθμου. Εάν αντικαταστήσουμε ένα σύμβολο του απλού κειμένου και κρυπτογραφήσουμε το νέο απλό κείμενο, τότε για ένα κρυπταλγόριθμο με υψηλή διάχυση, ο αντίπαλος δεν θα μπορεί να προβλέψει ποια σύμβολα του κρυπτοκειμένου θα μεταβληθούν η γενικότερα θα επηρεαστούν.

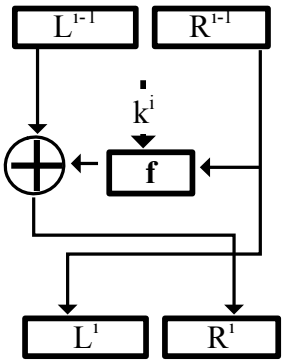
Σύγχυση είναι η ικανότητα του αλγορίθμου κρυπτογράφησης όπου ο αντίπαλος δεν είναι σε θέση να προβλέψει ποιες μεταβολές θα συμβούν στο κρυπτοκείμενο, δεδομένης μιας μεταβολής

στο απλό κείμενο. Δηλαδή, ένας αλγόριθμος έχει υψηλή σύγχυση όταν η σχέση μεταξύ του απλού κειμένου και του κρυπτοκειμένου είναι αρκετά πολύπλοκες, ώστε να χρειάζεται ο αντίπαλος να ξοδέψει σημαντικό χρόνο προκειμένου να τις προσδιορίσει.

Η διάχυση είναι η ικανότητα του αλγορίθμου κρυπτογράφησης όπου ένα τμήμα του απλού κειμένου να έχει την ευκαιρία να επηρεάζει όσο το δυνατόν περισσότερα τμήματά του κρυπτοκειμένου. Ένας αλγόριθμος έχει υψηλή διάχυση όταν ένα στοιχειώδες τμήμα του απλού κειμένου έχει την δυνατότητα να επηρεάσει όλα τα τμήματα του κρυπτοκειμένου ανεξαρτήτως της τοποθεσίας του τμήματος αυτού στο απλό κείμενο.

3.4.3 Δίκτυα Feistel

Η κρυπτογραφική πράξη τύπου Feistel είναι της μορφής του σχήματος που ακολουθεί, η οποία αποτελεί και ένα γύρο σε κρυπτοσύστημα γινομένου. Το βασικό χαρακτηριστικό ενός δικτύου Feistel είναι η πλήρης ελευθερία στην επιλογή της συνάρτησης γύρου f. Η δομή του δικτύου Feistel είναι τέτοια ώστε η αντίστροφη σχέση ορίζεται πάντοτε, ακόμα και αν η συνάρτηση δεν είναι ενριπτική. Επιπλέον, σε ορισμένες περιπτώσεις ένα δίκτυο Feistel μπορεί να είναι αποδείξιμα ασφαλές, όπως θα δείξουμε παρακάτω.



Σχήμα 3.1 : Ένας γύρος Feistel

Σε κάθε γύρο η είσοδος χωρίζεται στο αριστερό και στο δεξιό τμήμα. Τα δύο τμήματα της εισόδου του i-στου γύρου συμβολίζονται με L^{i-1} και R^{i-1}, ενώ οι έξοδοι συμβολίζονται με L^i και R^i. Στον πρώτο γύρο τα τμήματα L^0και R^0 αντιστοιχούν στο απλό κείμενο, ενώ στον τελικό γύρο τα τμήματα L^r και R^r αντιστοιχούν στο κρυπτοκείμενο.

Κατά το γύρο του i, η συνάρτηση γύρου f δέχεται ως είσοδο το δεξιό τμήμα της εισόδου και το κλειδί k^i το οποίο προέρχεται από το πρόγραμμα κλειδιού. Η έξοδος της συνάρτησης συνδυάζεται με το αριστερό τμήμα της εισόδου με αποκλειστική διάζευξη και το αποτέλεσμα της πράξης αντιστοιχίζεται στο δεξιό τμήμα της εξόδου, ενώ το δεξιό τμήμα της εισόδου αντιστοιχίζεται στο αριστερό τμήμα της εξόδου. Η ανταλλαγή του αριστερού τμήματος με το δεξί έχει ως αποτέλεσμα ο επόμενος γύρος να εφαρμόσει το αποτέλεσμα της συνάρτησης σε εκείνο το τμήμα της εισόδου το οποίο μεταφέρθηκε ατόφιο από την είσοδο στην έξοδο. Είναι φανερό ότι σε

κρυπτοσύστημα με έναν και μόνο γύρο, το δεξιό τμήμα του κρυπτοκειμένου θα είναι ίσο με το αριστερό τμήμα του απλού κειμένου. Αυτό είναι ένα χαρακτηριστικό της κρυπτογραφικής πράξης τύπου Feistel και θεωρητικά ένα δίκτυο όπου τα δύο τμήματα εισόδου έχουν το ίδιο μέγεθος, θα πρέπει να περιλαμβάνει τουλάχιστον τρεις γύρους προκειμένου το κρυπτοσύστημα να έχει τη δυνατότητα να αποκρύψει πλήρως το απλό κείμενο. Στην πράξη όμως απαιτούνται πολύ περισσότεροι γύροι για να είναι ένα κρυπτοσύστημα τύπου Feistel ασφαλές. Ο αριθμός των γύρων καθώς και η κρυπτογραφική δύναμη του κρυπτοσυστήματος εξαρτάται από τη συνάρτηση *f*.

Έστω n_L και n_R το μέγεθος του αριστερού και δεξιού τμήματος αντίστοιχα, με συνολικό μήκος εισόδου $n= n_L+ n_R$. Η συνάρτηση γύρου θα ορίζει την αντιστοιχία $f:\{0,1\}^n{}_R \rightarrow \{0,1\}^n{}_L$. Αν $n_L=n_R=n/2$ το δίκτυο Feistel ονομάζεται ισορροπημένο. Η πράξη κρυπτογράφησης ορίζεται από την επανάληψη της κρυπτογραφικής πράξης:

$$e^i{}_{ki}(L^i , R^i)= L^{i-1} \| (f(R^{i-1},k^i) \oplus L^{i-1}, \text{για } 0< i \leq r,$$

Για το απλό κείμενο θα είναι $p=L^0\|R^0$, ενώ για το κρυπτοκείμενο θα είναι $c=L^r\|R^r$. Το κλειδί επιλέγεται σε κάθε γύρο από το πρόγραμμα κλειδιού $\{k^1,k^2,\ldots,k^r\}$. Κατά την αποκρυπτογράφηση εφαρμόζεται η ίδια πράξη με τη διαφορά ότι το πρόγραμμα του κλειδιού ακολουθεί την αντίστροφη σειρά.

Το τμήμα της εισόδου το οποίο τροφοδοτείται στη συνάρτηση γύρου ονομάζεται προέλευση, ενώ το τμήμα της εισόδου στο οποίο εφαρμόζεται το αποτέλεσμα της συνάρτησης με αποκλειστική διάζευξη ονομάζεται στόχος. Αν το μέγεθος της πηγής είναι μεγαλύτερο από το μέγεθος του στόχου, τότε το δίκτυο ονομάζεται δίκτυο Feistel σημαίνουσας προέλευσης, ενώ στην περίπτωση που το μέγεθός του στόχου είναι μεγαλύτερο, το δίκτυο ονομάζεται δίκτυο Feistel σημαίνοντος στόχου. Αν το άθροισμα του μεγέθους της πηγής και του στόχου είναι ίσο με το μέγεθος της εισόδου, τότε το δίκτυο ονομάζεται τέλειο ενώ στην περίπτωση που το άθροισμα της πηγής και του στόχου είναι μικρότερο, το δίκτυο ονομάζεται ατελές. Σε ένα ατελές δίκτυο υπάρχει τμήμα της εισόδου το οποίο εμφανίζεται ατόφιο στην έξοδο και επιπλέον δεν συμπεριλαμβάνεται στην πράξη της συνάρτησης γύρου. Το τμήμα αυτό ονομάζεται μηδενικό.

Στη βιβλιογραφία το συντριπτικό ποσοστό στην έρευνα των δικτύων Feistel αποδίδεται σε ισορροπημένα δίκτυα Feistel, δηλαδή το αριστερό τμήμα της εισόδου είναι ο στόχος και είναι ίσο με το δεξιό τμήμα της εισόδου που είναι η προέλευσή. Ο βασικός λόγος εκτενούς μελέτης των ισορροπημένων δικτύων Feistel είναι επειδή τα πιο διαδεδομένα κρυπτοσυστήματα τα οποία βασίζονται σε δίκτυα Feistel είναι ισορροπημένα, όπως το κρυπτοσύστημα DES που θα υλοποιήσουμε στην συνέχεια. Ωστόσο, η ασύμμετρη κατανομή των τμημάτων της εισόδου σε δίκτυα Feistel σημαίνουσας προέλευσης και σημαίνοντος στόχου δημιουργεί υποψίες ότι ένα μη ισορροπημένο δίκτυο Feistel μπορεί να είναι κρυπτογραφικά αδύναμο. Στην περίπτωση του

δικτύου Feistel σημαίνοντος στόχου θα υπάρχουν σε κάθε γύρο γραμμικές σχέσεις μεταξύ ορισμένων bits εισόδου με ορισμένα bits εξόδου. Στην περίπτωση δικτύου Feistel σημαίνουσας προέλευσης απαιτούνται περισσότεροι γύροι για να εμφανιστεί κάθε bit στο τμήμα του στόχου.

3.4.3.1 Ασφάλεια δικτύων Feistel

Για να κατασκευαστεί ένα αποδείξιμα ασφαλές δίκτυο Feistel θα πρέπει να ακολουθηθεί μία συγκεκριμένη μεθοδολογία.. Η προσέγγιση σε ένα ασφαλές σύστημα ακολουθεί τα εξής βασικά στάδια:

1. Αναγνώριση του προβλήματος, όπου ξεχωρίζουμε ένα πρόβλημα κρυπτογραφίας. Ένα από τα κλασικά προβλήματα, για παράδειγμα, είναι: « Η f είναι μονόδρομη συνάρτηση».
2. Καθορισμός του προβλήματος. Αυτό είναι ίσως το πιο βασικό στάδιο στο οποίο περιγράφουμε μαθηματικά το πρόβλημα.
3. Ανάπτυξη του πρωτοκόλλου το οποίο καθορίζει τα βήματα τα οποία θα ακολουθήσουμε για να αποδείξουμε την ασφάλεια του προβλήματος.
4. Καθορισμός της υποθέσεως, η οποία αναφέρεται στις δυνατότητες του αντιπάλου.
5. Απόδειξη. Με βάση την υπόθεση εκτελούμε το πρωτόκολλο και να προσδιορίσουμε αν το πρόγραμμα το οποίο έχουμε καθορίσει οδηγεί σε ασφαλή κατασκευή.

3.5 Είδη Κρυπτογράφησης

Οι βασικές μέθοδοι κρυπτογράφησης, οι οποίες χρησιμοποιούνται, είναι δύο: η συμμετρική και η ασύμμετρη.

Η **Συμμετρική Κρυπτογραφία** ή Κρυπτογραφία Ιδιωτικού Κλειδιού (Private key cryptography), βασίζεται στην ύπαρξη ενός μοναδικού «κλειδιού», το οποίο χρησιμοποιείται τόσο για την κρυπτογράφηση όσο και για την αποκρυπτογράφηση των δεδομένων. Το κλειδί αυτό στην πραγματικότητα είναι ένα σύνολο χαρακτήρων, με βάση το οποίο τα προγράμματα κρυπτογράφησης, χρησιμοποιώντας ειδικούς αλγορίθμους, μετατρέπουν το κείμενο σε μη αναγνώσιμη μορφή. Στη συνέχεια το κείμενο αποστέλλεται στον παραλήπτη, ο οποίος για να το επαναφέρει στην αρχική μορφή του θα πρέπει να γνωρίζει το «κλειδί» αλλά και τους αλγορίθμους οι οποίοι χρησιμοποιήθηκαν για την κρυπτογράφησή του.

Τέτοιοι αλγόριθμοι κρυπτογράφησης υπάρχουν πολλοί, με πιο γνωστό τον DES (Data Encryption Standard), ο οποίος αναπτύχθηκε το 1977 από την IBM και χρησιμοποιήθηκε από την κυβέρνηση των ΗΠΑ για μεγάλο χρονικό διάστημα ως το επίσημο πρότυπο κρυπτογράφησης

απόρρητων πληροφοριών. Πολύ γνωστός είναι επίσης ο IDEA (International Data Encryption Algorithm).

Όμως, το σύστημα της συμμετρικής κρυπτογραφίας δεν μπορεί να χρησιμοποιηθεί ευρέως, διότι το κλειδί της κρυπτογράφησης θα πρέπει να το γνωρίζει ο παραλήπτης του μηνύματος, κάτι που προϋποθέτει ότι θα πρέπει να σταλεί σε αυτόν από τον αποστολέα μέσω ενός ασφαλούς διαύλου. Αν και έχουν αναπτυχθεί συστήματα τα οποία επιτρέπουν την ασφαλή ανταλλαγή κλειδιών μέσα από δημόσια δίκτυα, με κλασικό παράδειγμα το Kerberos που αναπτύχθηκε στο MIT, η κάλυψη των απαιτήσεων μεγάλου αριθμού χρηστών κάθε άλλο παρά εύκολη είναι, ενώ απαιτούνται και πρόσθετες διαδικασίες ασφαλείας, όπως η αποθήκευση των κλειδιών σε έναν κεντρικό, ασφαλή εξυπηρετητή (server).

Η αδυναμία αυτή της συμμετρικής κρυπτογράφησης οδήγησε στην καθιέρωση της **Ασύμμετρης Κρυπτογράφησης** ή αλλιώς Κρυπτογράφησης Δημόσιου Κλειδιού (Public key cryptography) ως του μοναδικού τρόπου που να μπορεί να εγγυηθεί την ασφαλή μεταφορά δεδομένων. Οι αλγόριθμοι και τα συστήματα της κατηγορίας αυτής χρησιμοποιούν για τις διαδικασίες κρυπτογράφησης και αποκρυπτογράφησης δύο ξεχωριστά κλειδιά, ένα δημόσιο και ένα ιδιωτικό, τα οποία σχετίζονται μεταξύ τους. Τα κλειδιά που ανήκουν στο ζεύγος αυτό έχουν τη σημαντική ιδιότητα ότι είναι πρακτικά αδύνατος ο υπολογισμός του ενός κλειδιού γνωρίζοντας το άλλο.

Η πρώτη διατύπωση αυτής της μεθόδου κρυπτογράφησης έγινε το 1976 από τους Diffie και Hellman, ενώ μόλις έναν χρόνο αργότερα, το 1977, οι Rivest, Shamir και Adleman δημιούργησαν το πρώτο σύστημα ασύμμετρης κρυπτογραφίας, το κρυπτοσύστημα RSA. Από τότε ακολούθησαν και άλλες προτάσεις, με κάθε κρυπτοσύστημα να έχει τις δικές του ιδιαιτερότητες. Η φιλοσοφία, πάντως, των συστημάτων αυτών βασίζεται στη δυνατότητα δημοσίευσης των δημόσιων κλειδιών, καθώς όπως αναφέραμε είναι πρακτικά αδύνατη η εύρεση του αντίστοιχου ιδιωτικού κλειδιού. Το δημόσιο κλειδί, λοιπόν, κοινοποιείται από κάθε χρήστη όσο γίνεται ευρύτερα, συνήθως μέσω ειδικών καταλόγων ή απευθείας από τον ένα χρήστη στον άλλο. Ο αποστολέας χρησιμοποιεί το δημόσιο κλειδί του παραλήπτη για να κρυπτογραφήσει το μήνυμα, εξασφαλίζοντας έτσι ότι μόνο ο παραλήπτης που διαθέτει το σχετιζόμενο ιδιωτικό κλειδί μπορεί να το αποκρυπτογραφήσει. Επειδή όμως η μέθοδος αυτή, για να έχει τον ίδιο δείκτη ασφαλείας με τη συμμετρική, απαιτεί περίπου χίλιες φορές περισσότερο χρόνο και δέκα φορές μεγαλύτερα κλειδιά προκειμένου να εκτελέσει τη διαδικασία κρυπτογράφησης-αποκρυπτογράφησης, έχει επικρατήσει τελικά μία μέθοδος που συνδυάζει και τις δύο. Έτσι, για την κωδικοποίηση ενός μηνύματος χρησιμοποιείται ένα προσωρινό τυχαίο κλειδί, το οποίο στη συνέχεια, με τη βοήθεια του δημόσιου κλειδιού του παραλήπτη, κρυπτογραφείται και αποστέλλεται μαζί με το κωδικοποιημένο κείμενο. Ο παραλήπτης στη συνέχεια, κάνοντας χρήση του ιδιωτικού κλειδιού του, αποκρυπτογραφεί το κλειδί και στη

συνέχεια το μήνυμα. Με τη διαδικασία αυτή επιτυγχάνεται η κρυπτογράφηση αρχείων με ασφάλεια που είναι υψηλότερη όσο μεγαλώνει το μέγεθος των χρησιμοποιούμενων κλειδιών.

Η Κρυπτογράφηση Δημόσιου Κλειδιού είναι μια μορφή κρυπτογραφίας που γενικά επιτρέπει στους χρήστες να επικοινωνήσουν ασφαλώς, χωρίς την κατοχή προγενέστερης πρόσβασης σε ένα κοινό μυστικό κλειδί, με τη χρησιμοποίηση ενός ζευγαριού των κρυπτογραφικών κλειδιών, που ονομάζονται ως δημόσιο και ιδιωτικό κλειδί, τα οποία συσχετίζονται μαθηματικά.

Το 1976 στο πανεπιστήμιο Stanford, οι Diffie και Hellman, πρότειναν ένα ριζικά διαφορετικό είδος κρυπτοσυστήματος, στο οποίο τα κλειδιά κρυπτογράφησης και αποκρυπτογράφησης ήταν διαφορετικά.

Αυτή η μέθοδος εκθετικής ανταλλαγής κλειδιών, η οποία έγινε γνωστή ως ανταλλαγή Diffie-Hellman, ήταν η πρώτη δημοσιευμένη πρακτική μέθοδος για ένα κοινό μυστικό κλειδί μεταδιδόμενο από ένα μη προστατευμένο κανάλι επικοινωνιών χωρίς χρησιμοποίηση ενός προγενέστερου κοινού μυστικού.

Οι αλγόριθμοι δημόσιου κλειδιού είναι συνήθως βασισμένοι σε πολύπλοκα μαθηματικά προβλήματα. Ο RSA, παραδείγματος χάριν, στηρίζεται στην δυσκολία της παραγοντοποίησης. Για λόγους αποδοτικότητας, χρησιμοποιούνται στην πράξη τα υβριδικά συστήματα κρυπτογράφησης όπου ένα κλειδί ανταλλάσσεται χρησιμοποιώντας κρυπτογράφηση δημόσιου κλειδιού και το υπόλοιπο περιεχόμενο της επικοινωνίας κρυπτογραφείται χρησιμοποιώντας κρυπτογράφηση συμμετρικού κλειδιού (η οποία είναι πολύ γρηγορότερη).

Η Κρυπτογράφηση Δημόσιου Κλειδιού παρέχει επίσης μηχανισμούς για την δημιουργία ψηφιακών υπογραφών, οι οποίες είναι ένας τρόπος να εξασφαλιστεί υψηλό επίπεδο εμπιστευτικότητας ότι το λαμβανόμενο μήνυμα εστάλη από τον σωστό αποστολέα. Τέτοιες υπογραφές συχνά, θεωρούνται από την νομοθεσία ως το ψηφιακό ισοδύναμο των φυσικών υπογραφών. Υπό μια τεχνική έννοια, δεν είναι δεδομένο ότι δεν υπάρχει καμία φυσική επαφή ούτε σύνδεση μεταξύ του "υπογράφοντος" και "υπογεγραμμένου". Χρησιμοποιώντας τα κατάλληλα σχέδια και εφαρμογές υψηλής ποιότητας είναι δυνατό να επιτευχθεί ένας πολύ υψηλός βαθμός διαβεβαίωσης της υπογραφής. Παραδείγματα πρωτοκόλλων ψηφιακών υπογραφών αποτελούν τα DSA και ElGamal. Οι ψηφιακές υπογραφές αποτελούν κεντρικό σημείο στην λειτουργία της κρυπτογράφησης δημόσιου κλειδιού και πολλών συστημάτων δικτύων ασφάλειας όπως του Kerberos και πολλών άλλων.

3.6 Σύγκριση Συμμετρικής και Ασύμμετρης Κρυπτογραφίας

Η συμμετρική κρυπτογράφηση κλειδιού έχει ένα μειονέκτημα. Δύο άνθρωποι που επιθυμούν να ανταλλάξουν εμπιστευτικά μηνύματα πρέπει να μοιραστούν ένα μυστικό κλειδί. Το κλειδί πρέπει να ανταλλαχθεί με έναν ασφαλή τρόπο και όχι με τα μέσα που θα επικοινωνούσαν κανονικά. Αυτό είναι συνήθως το πιο δύσκολο σημείο και το σύστημα κρυπτογραφίας δημόσιων-κλειδιών (ή ασύμμετρο) παρέχει μια εναλλακτική λύση.

Γενικά, οι τεχνικές δημόσιου κλειδιού είναι πολύ ισχυρότερες υπολογιστικά από τους καθαρά συμμετρικούς αλγορίθμους, αλλά η σωστή χρήση αυτών των τεχνικών επιτρέπει μια ευρεία ποικιλία εφαρμογών τους.

Όσον αφορά την ασφάλεια, δεν υπάρχει τίποτα που να καθιστά ασφαλέστερους τους αλγόριθμους δημόσιου κλειδιού σε σύγκριση με τους συμμετρικούς αλγορίθμους κλειδιών. Υπάρχουν δημοφιλείς και μη δημοφιλείς αλγόριθμοι. Υπάρχουν αυτοί που έχουν παραβιαστεί και αυτοί που δεν έχουν ακόμα τουλάχιστον σπάσει. Δυστυχώς, η δημοτικότητα δεν είναι ένας αξιόπιστος δείκτης της ασφάλειας. Πολλές αποδείξεις υποστηρίζουν ότι το σπάσιμο ενός αλγορίθμου, όσον αφορά κάποιους καθορισμένους με σαφήνεια στόχους ασφάλειας, είναι ισοδύναμο με την επίλυση ενός από τα δημοφιλέστερα μαθηματικά προβλήματα που θεωρούνται για να είναι ισοδύναμα με την κατασκευή ενός διακεκριμένου λογάριθμου. Γενικά, κανένας από αυτούς τους αλγόριθμους δεν έχει αποδειχθεί να είναι απόλυτα ασφαλής. Όπως με όλους τους κρυπτογραφικούς αλγόριθμους, αυτοί οι αλγόριθμοι πρέπει να επιλεχτούν και να χρησιμοποιηθούν με εξαιρετική προσοχή.

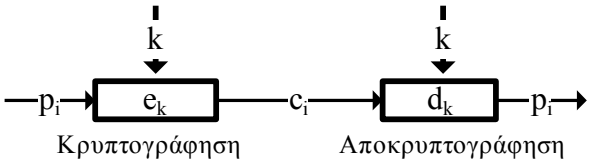
3.7 Καταστάσεις Λειτουργίας Συμμετρικών Κρυπταλγόριθμων

Οι τρόποι λειτουργίας είναι τρόποι διασύνδεσης κρυπταλγορίθμων τμήματος, με στόχο την περαιτέρω αύξηση της κρυπτογραφικής δύναμης και την αποτελεσματικότερη απόκρυψη πιθανών υπολειμμάτων πληροφορίας του απλού κειμένου που μπορεί να υπάρχει στο κρυπτοκείμενο. Υπάρχουν τέσσερις τυποποιημένοι τρόποι λειτουργίας σύμφωνα με το πρότυπο FIPS 81 και αρκετοί μη τυποποιημένοι τρόποι λειτουργίας. Οι τέσσερις τρόποι λειτουργίας είναι:

- Κατάσταση λειτουργίας ηλεκτρονικού βιβλίου κωδικών (Electronic Codebook, ECB)
- Κατάσταση λειτουργίας αλυσιδωτής σύνδεσης τμημάτων κρυπτογραφίας (cipher block chaining, CBC).
- Κατάσταση λειτουργίας κρυπτογραφίας ανάδρασης (cipher feedback mode, CFB).
- Ανάδραση εξόδου (output feedback, OFB)

3.7.1 Κατάσταση λειτουργίας ηλεκτρονικού βιβλίου κωδικών (ECB)

Ο τρόπος λειτουργίας ECB αποτελεί την ευθεία συνδεσμολογία όπου το απλό κείμενο τροφοδοτείται στον κρυπταλγόριθμο και το κρυπτοκείμενο προκύπτει από την έξοδο. Η ονομασία του τρόπου αυτού προέρχεται από την αναπαράσταση του κρυπτοσυστήματος ως ένα μεγάλο βιβλίο το οποίο περιέχει όλα τα ζεύγη απλού κειμένου και κρυπτοκειμένου για κάθε κλειδί. Έτσι για ένα κρυπταλγόριθμο τμήματος με μέγεθος απλού κειμένου και κρυπτοκειμένου n bits και μεγέθους κλειδιού k bits, μπορούμε να φανταστούμε ότι το βιβλίο περιέχει2^k κεφάλαια, ένα για το κάθε κλειδί και το περιεχόμενο κάθε κεφαλαίου θα αποτελούνταν από $2x2^k$ καταχωρίσεις, οι μισές ταξινομημένες ως προς το απλό κείμενο και οι υπόλοιπες ταξινομημένες ως προς το κρυπτοκείμενο.



Σχήμα 3.2 : Τρόπος Λειτουργίας ECB

Το απλό κείμενο χωρίζεται σε τμήματα $P=[p_1p_2\ldots p_l]$, όπου το κάθε τμήμα μεγέθους n bits τροφοδοτείται στον αλγόριθμο κρυπτογράφησης:

$$c_i = e_k\,(p_i),$$

Η αναγκαία κατάτμηση του απλού κειμένου είναι και το μειονέκτημα της ECB λειτουργίας. Για όμοια τμήματα του απλού κειμένου, τα αντίστοιχα κρυπτοκείμενα που προκύπτουν είναι επίσης όμοια. Αυτό καθιστά την ECB ακατάλληλη για τις εφαρμογές στις οποίες υπάρχουν επαναλαμβανόμενα μοτίβα δεδομένων. Για παράδειγμα, στα δίκτυα ηλεκτρονικών υπολογιστών, τα δεδομένα που ανταλλάσσονται μεταξύ των υπολογιστών βασίζονται σε πρωτόκολλα επικοινωνίας τα οποία χρησιμοποιούν τυποποιημένα μηνύματα. Έτσι ο αντίπαλος είναι σε θέση να αναγνωρίζει τα επαναλαμβανόμενα τμήματα του απλού κειμένου, καθώς και τις στιγμές κατά τις οποίες γίνεται η αλλαγή του κλειδιού κρυπτογράφησης. Αυτή η διαρροή της πληροφορίας για τις περισσότερες εφαρμογές δεν είναι επιτρεπτή.

Συμπεραίνουμε λοιπόν ότι, η αναγκαία κατάτμηση σε συνδυασμό με το γεγονός ότι το κάθε τμήμα κρυπτογραφείται ανεξάρτητα από τα υπόλοιπα τμήματα του απλού κειμένου δίνει το πλεονέκτημα στον αντίπαλο να αναγνωρίσει με αφαιρετικό τρόπο το περιεχόμενο του απλού κειμένου.

3.7.2 Κατάσταση λειτουργίας αλυσιδωτής σύνδεσης κρυπτογραφημάτων ομάδας (cipher block chaining CBC).

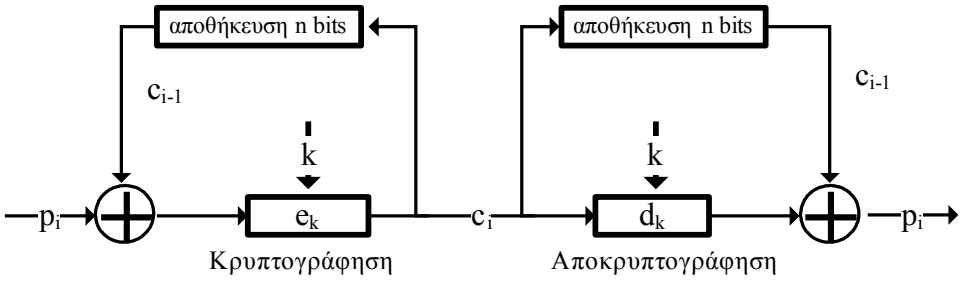
Η κρυπτογράφηση ενός τμήματος του απλού κειμένου p_i εξαρτάται και από το προηγούμενο τμήμα p_{i-1} με την ακόλουθη σχέση κρυπτογράφησης:

$$c_i = e_k (c_{i-1} \oplus p_i),$$

Ενώ η αποκρυπτογράφηση ορίζεται ως εξής:

$$p_i = d_k (c_i) \oplus c_{i-1})$$

Η λειτουργία CBC παριστάνεται στο παρακάτω σχήμα:



Σχήμα 3.3 : Τρόπος Λειτουργίας CBC

Η ορθότητα της σχέσης κρυπτογράφησης-αποκρυπτογράφησης είναι φανερή από:

$$p_i = d_k (c_i) \oplus c_{i-1}) = c_{i-1} \oplus p_i \oplus c_{i-1} = (c_{i-1} \oplus c_{i-1}) \oplus p_i = p_i$$

Μπορούμε να παρατηρήσουμε ότι η ποσότητα του κρυπτοκειμένου που εφαρμόζεται στην αποκλειστική διάζευξη είναι η ίδια τόσο στην πλευρά ένα κρυπτογράφησης, όσο και στην πλευρά ένα αποκρυπτογράφησης και είναι αυτή που προκύπτει από την προηγούμενη κρυπτογράφηση.

Για να καθοριστεί πλήρως η κρυπτογράφηση ένα λειτουργίας CBC, θα πρέπει να ορισθούν η αρχική τιμή c_0, καθώς και το τελικό τμήμα του απλού κειμένου, στην περίπτωση που το μέγεθός του απλού κειμένου δεν είναι πολλαπλάσιο του n . Στην περίπτωση ένα κρυπτογράφησης του πρώτου τμήματος p_i είναι:

$$c_1 = e_k (c_0 \oplus p_i),$$

που σημαίνει ότι απαιτείται η ποσότητα c_0. Αυτή η ποσότητα είναι το διάνυσμα αρχικοποίησης το οποίο θα πρέπει να είναι γνωστό τόσο κατά τη διαδικασία ένα κρυπτογράφησης, όσο και κατά τη διαδικασία ένα αποκρυπτογράφησης. Αν και η εμπιστευτικότητα του δεν είναι υποχρεωτική, αποτελεί κοινή πρακτική να στέλνεται στον αποδέκτη κρυπτογραφημένο με ECB.

Ένα δεύτερος λόγος που προτιμάται ένα κρυπτογραφημένη αποστολή του διανύσματος αρχικοποίησης, είναι η προστασία ένα ακεραιότητας του, η οποία είναι σημαντικότερη από την εμπιστευτικότητα του διανύσματος. Η προστασία ένα ακεραιότητάς

πραγματοποιείται με τη χρήση συνάρτησης ακεραιότητας, σε συνδυασμό με την κρυπτογράφηση ECB. Το διάνυσμα αρχικοποίησης διαιρείται σε δύο τμήματα, το πρώτο μήκους n-α bits και το δεύτερο μήκους α bits. Στο πρώτο τμήμα εφαρμόζεται κάποια μονόδρομη συνάρτηση hash, και τα πρώτα α bits του αποτελέσματος απαρτίζουν το δεύτερο τμήμα του διανύσματος. Με αυτόν τον τρόπο το δεύτερο τμήμα του διανύσματος αποτελεί τη σύνοψη του πρώτου τμήματος. Στη συνέχεια το διάνυσμα κρυπτογραφείται και αποστέλλεται στον αποδέκτη. Ο αποδέκτης με τη σειρά του το αποκρυπτογραφεί και ελέγχει αν το δεύτερο τμήμα του διανύσματος είναι η σύνοψη του πρώτου. Στην περίπτωση που ο αντίπαλος προσβάλλει την ακεραιότητα του διανύσματος, αυτό θα γίνει αντιληπτό από τον αποδέκτη.

Όσον αφορά το τελευταίο τμήμα του απλού κειμένου, υπάρχει το ενδεχόμενο το τμήμα αυτό να είναι μικρότερο του n. Αυτό συμβαίνει όταν το μέγεθος του κρυπτοκειμένου δεν είναι πολλαπλάσιο του n. Στην περίπτωση αυτή προστίθενται μηδενικά στο τέλος, έως ότου το τμήμα έχει μέγεθος ίσο με n bits. Οι εφαρμογές στις οποίες τα δεδομένα ακολουθούν αυστηρή τυποποίηση και δεν είναι επιτρεπτό να συμπεριλαμβάνονται τα επιπλέον μηδενικά, τα τελευταία bits του τμήματος χρησιμοποιούνται για την καταγραφή του πλήθους των επιπρόσθετων μηδενικών. Για z μηδενικά, απαιτούνται $\log_2(z)$ δυαδικές θέσεις.

3.7.3 Κατάσταση λειτουργίας κρυπτογραφίας ανάδρασης (cipher feedback mode CFB)

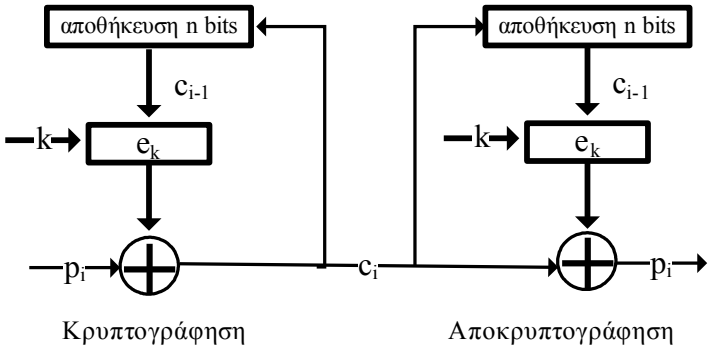
Το κρυπτοκείμενο προκύπτει από την επανακρυπτογράφηση του προηγουμένου κρυπτοκειμένου, συνδυασμένο με αποκλειστική διάζευξη με το απλό κείμενο.
Έτσι η κρυπτογράφηση ορίζεται ως εξής:

$$c_i = e_k (c_{i-1}) \oplus p_i$$

Ενώ κατά την αποκρυπτογράφηση ισχύει:

$$p_i = e_k (c_{i-1}) \oplus c_i$$

Ένα κρυπτοσύστημα σε λειτουργία CFB παρουσιάζεται στο ακόλουθο σχήμα:



Σχήμα 3.4 : Τρόπος Λειτουργίας CFB

Η λειτουργία CFB μπορεί να θεωρηθεί ως κρυπταλγόριθμος ροής, όπου τα σύμβολα του απλού κειμένου είναι δυαδικές λέξεις μεγέθους n bits, που σημαίνει ότι το αλφάβητο του απλού κειμένου και του κρυπτοκειμένου αποτελείται από 2^n σύμβολα. Επίσης σε ένα κρυπταλγόριθμο ροής, οι γεννήτριες της κλειδοροής του αποστολέα και του αποδέκτη θα πρέπει να παράγουν την ίδια ακολουθία. Αυτό φαίνεται και από την πράξη αποκρυπτογράφησης, όπου ο κρυπταλγόριθμος τμήματος εφαρμόζεται σε λειτουργία κρυπτογράφησης και όχι αποκρυπτογράφησης.

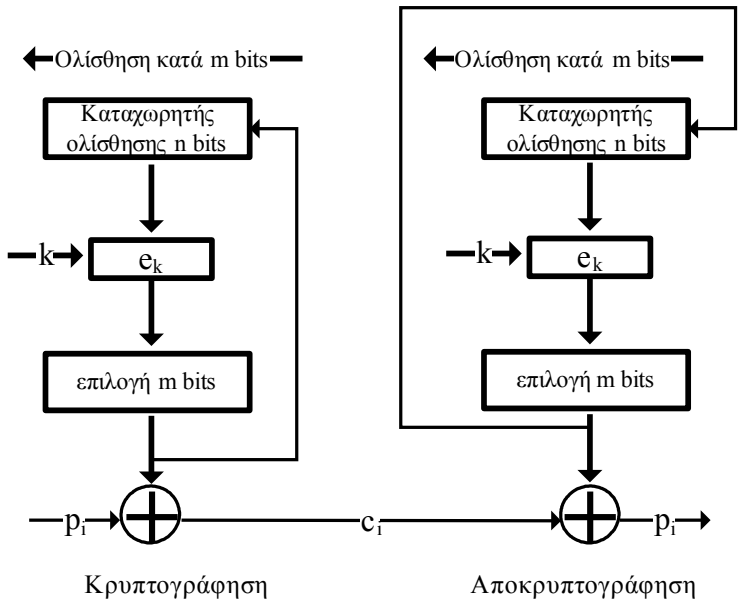
Παρόμοια με την λειτουργία CBC, για την πλήρη εκτέλεση της CFB απαιτείται διάνυσμα αρχικοποίησης. Το διάνυσμα αυτό αποθηκεύεται στον καταχωρητή αποθήκευσης που τροφοδοτεί τον κρυπταλγόριθμο τμήματος. Η μετάδοσή του διανύσματος αρχικοποίησης δεν απαιτεί εμπιστευτικότητα. Αντίθετα, μπορεί να προηγηθεί του μηνύματος και να σταλεί κρυπτογραφημένο με το ίδιο το κρυπτοσύστημα της CFB. Οι δύο καταχωρητές αποθήκευσης μπορούν να έχουν μηδενικές τιμές κατά τη μετάδοση του διανύσματος αρχικοποίησης.

3.7.4 Ανάδραση εξόδου (output feedback, OFB)

Η κρυπτογράφηση πραγματοποιείται με την αποκλειστική διάζευξη του απλού κειμένου με την ακολουθία της κλειδοροής. Αντίστοιχα, η αποκρυπτογράφηση πετυχαίνετε με την αποκλειστική διάζευξη το κρυπτοκειμένου με την ακολουθία της κλειδοροής. Η διαδικασία δημιουργίας της κλειδοροής είναι ανεξάρτητη από το απλό κείμενο και το κρυπτοκείμενο.

Θα μπορούσαμε να υποστηρίξουμε ότι η ασφάλεια της OFB είναι μικρότερη από την ασφάλεια των υπόλοιπων τριών τρόπων λειτουργίας και αυτό γιατί ο μόνος τρόπος διασύνδεσής του απλού κειμένου και του κρυπτοκειμένου είναι η πράξη της αποκλειστικής διάζευξης. Εφόσον η γεννήτρια κλειδοροής είναι μηχανή πεπερασμένων καταστάσεων, η κλειδοροή που παράγει θα είναι περιοδική. Επομένως υπάρχει κάποια χρονική στιγμή όπου η κλειδοροή εμφανίζει τα ίδια κλειδιά. Αν ο αντίπαλος εντοπίσει την αρχή της περιόδου, τότε μπορεί να απομακρύνει την πληροφορία της κλειδοροής από το κρυπτοκείμενο με τον ακόλουθο τρόπο.

Η λειτουργία OFB παρουσιάζεται στο παρακάτω σχήμα:



Σχήμα 3.5 : Τρόπος Λειτουργίας OFB

Έστω τα τμήματα του κρυπτοκειμένου c_i και c_j τα οποία προκύπτουν από την εφαρμογή ιδίων τμημάτων της κλειδοροής. Αν συνδυάσουμε τα κρυπτοκείμενα με αποκλειστική διάζευξη προκύπτει ότι:

$$c_i \oplus c_j = (p_i \oplus k_i) \oplus (p_j \oplus k_i) = p_i \oplus p_j$$

Δηλαδή, η κλειδοροή έχει εξαλειφθεί. Αν το απλό κείμενο είναι κάποια φυσική γλώσσα με υψηλή περίσσεια, τότε ο αντίπαλος μπορεί να εντοπίσει την περίοδο με στατιστικές μεθόδους. Στη συνέχεια, μπορεί να συνδυάζει με αποκλειστική διάζευξη απλά κείμενα της γλώσσας στα παραπάνω άθροισμα για να ξεχωρίσει τα p_i και p_j.

Επομένως στην περίπτωση της λειτουργίας OFB δεν υπάρχει η ίδια ελευθερία επιλογής κρυπταλγόριθμου τμήματος που υπάρχει στους υπόλοιπους τρόπους λειτουργίας. Ο κρυπταλγόριθμος τμήματος θα πρέπει από τη μια να έχει μεγάλη περίοδο, ενώ από την άλλη θα πρέπει να αλλάζει το κλειδί προτού ξεπεραστεί αυτή η περίοδος. Η αλλαγή του κλειδιού του κρυπταλγόριθμου τμήματος θα έχει ως αποτέλεσμα η γεννήτρια κλειδοροής να μεταπίπτει σε άλλη ακολουθία κλειδοροής.

3.8 Κρυπτανάλυση

Ως Κρυπτανάλυση ορίζεται η μελέτη των μαθηματικών τεχνικών, οι οποίες επιχειρούν την αναίρεση των τεχνικών της Κρυπτογραφίας και γενικότερα την αναίρεση της ασφαλούς μεταδόσεως πληροφοριών. Η Κρυπτανάλυση βασίζεται, πέραν των μαθηματικών και στο εμπειρικό γεγονός ότι, στην πράξη, ο κρυπταναλυτής έχει στη διάθεσή του πάρα πολύ μεγάλο αριθμό κρυπτογραφημένων κειμένων, τα οποία κρυπτογραφήθηκαν με τον ίδιο τρόπο. Ενώ επίσης

θεωρούμε σαν δεδομένο το γεγονός ότι ο Κρυπταναλυτής γνωρίζει πλήρως τη διαδικασία κρυπτογράφησης-αποκρυπτογράφησης που χρησιμοποιήθηκε. Οι διαδικασίες κρυπτανάλυσης ονομάζονται επιθέσεις (attacks).

Σκοπός του κρυπταναλυτή είναι να αποκτήσει :

1. Ένα μέρος από κάποιο αρχικό καθαρό κείμενο ή ολόκληρο.
2. Ένα μέρος από κάποιο κρυπτογραφημένο κείμενο ή ολόκληρο.
3. Συνδυασμό των προηγουμένων.
4. Συνδυασμό των προηγουμένων από διαφορετικά μηνύματα.
5. Γνώση παραγωγής ή απόκτησης των κλειδιών κρυπτογράφησης.

3.9 Μέθοδοι Επιθέσεων

Ανάλογα με τις πληροφορίες και τις μεθόδους που χρησιμοποιούνται, έχουμε κατηγοριοποίηση των διαφόρων μεθόδων επίθεσης. Οι δυνατότητες επίθεσης σε ένα κρυπτοσύστημα χωρίζονται στις ακόλουθες κατηγορίες.

Επίθεση στο κρυπτοκείμενο (ciphertext-only). Ο αντίπαλος έχει πρόσβαση μόνο σε ορισμένα κομμάτια του κρυπτοκειμένου και ο αντικειμενικός σκοπός είναι να αποκρυπτογραφήσει το κρυπτοκείμενο αυτό, η να ανακαλύψει το αντίστοιχο κλειδί. Ένα κρυπτοσύστημα το οποίο είναι ευάλωτο σε μια τέτοια επίθεση θεωρείται ανασφαλές.

Επίθεση με γνωστό απλό κείμενο (known-plaintext). Ο αντίπαλος γνωρίζει αντιστοιχίες κρυπτοκειμένου με απλό κείμενο και ο αντικειμενικός του σκοπός είναι η ανακάλυψη του αντίστοιχου κλειδιού. Στον κόσμο των δικτύων των υπολογιστών τα πρωτόκολλα επικοινωνίας εμφανίζουν συστηματικά τυποποιημένα μηνύματα.

Επίθεση με επιλεγμένο απλό κείμενο (chosen-plaintext). Ο αντίπαλος έχει τη δυνατότητα πρόσβασης στο κρυπτοσύστημα όπου δεν γνωρίζει το κλειδί και μπορεί να ζητά την κρυπτογράφηση μηνυμάτων. Με αυτόν τον τρόπο μπορεί να ανακαλύψει την αντιστοιχία του απλού κειμένου με το άγνωστο κρυπτοκείμενο.

Επίθεση προσαρμόσιμου επιλεγμένου απλού κειμένου (adaptive chosen-plaintext). Ο αντίπαλος είναι σε θέση να πραγματοποιήσει επίθεση με επιλεγμένο απλό κείμενο, αλλά επιπλέον μπορεί να εφαρμόσει τη μεθοδολογία σύμφωνα με την οποία η επόμενη επιλογή του απλού κειμένου εξαρτάται από τις προηγούμενες, προκειμένου να ανακαλύψει γρηγορότερα το κλειδί, από μια εξαντλητική αναζήτηση (exhaustive search).

Επίθεση με επιλεγμένο κρυπτοκείμενο (chosen-ciphertext). Υποθέτοντας ότι ο αντίπαλος έχει πρόσβαση στον αλγόριθμο αποκρυπτογράφησης, ο αντικειμενικός σκοπός του είναι να ανακαλύψει το κλειδί αποκρυπτογράφησης προκειμένου να μπορεί στο μέλλον να αποκρυπτογραφεί τα νέα κρυπτοκείμενα, όταν δεν θα έχει πρόσβαση στον αλγόριθμο

αποκρυπτογράφησης. Στα περισσότερα συμμετρικά κρυπτοσυστήματα η επίθεση αυτή έχει την ίδια ισχύ με την επίθεση του επιλεγμένου απλού κειμένου. Η επίθεση με επιλεγμένο κρυπτοκείμενο θεωρείται ως η πιο αυστηρή επίθεση.

Επίθεση προσαρμόσιμου επιλεγμένου κρυπτοκειμένου (adaptive chosen-ciphertext). Η επίθεση αυτή είναι αντίστοιχη του προσαρμόσιμου επιλεγμένου απλού κειμένου, με τη διαφορά ότι ο αντίπαλος έχει πρόσβαση στον αλγόριθμο αποκρυπτογράφησης [ΚΓΣ03].

4 Ο Αλγόριθμος DES

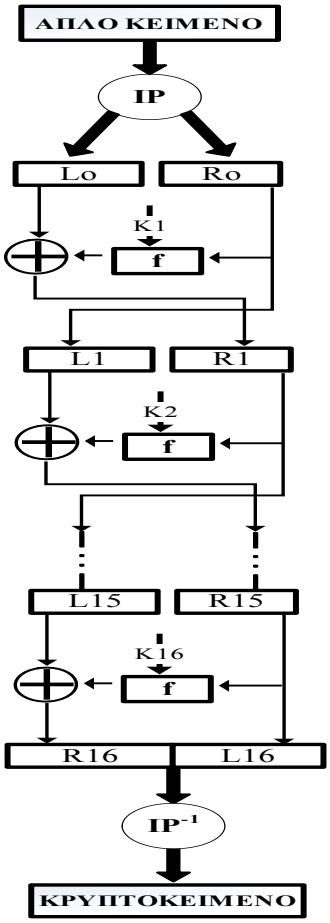
Τον Αύγουστο, του 1974, η IBM σε απάντηση στη δεύτερη προκήρυξη του Εθνικού Γραφείου Προτύπων (National Bureau of Standards, NBS), τώρα γνωστό ως Εθνικό Ίδρυμα Προτύπων & Τεχνολογίας (National Institute of Standards & Technology, NIST), υπέβαλε υποψηφιότητα για την ανάδειξη ενός κρυπτογραφικού αλγόριθμου, του οποίου στόχος ήταν να προστατεύσει τα στοιχεία κατά τη διάρκεια της μετάδοσης και της αποθήκευσης. Η IBM υπέβαλε ένα αλγόριθμο με το όνομα LUCIFER ο οποίος είχε μέγεθος κειμένου καθώς και μέγεθος κλειδιού 128 bits. Το NBS άρχισε την διαδικασία αξιολόγησης με τη βοήθεια της Υπηρεσίας Εθνικής Ασφάλειας (National Security Agency, NSA) από την οποία ζήτησε να διενεργήσει μια εξαντλητική τεχνική ανάλυση του αλγορίθμου. Επίσης κατά την διάρκεια αυτής της έρευνας, η IBM διαπραγματεύτηκε με το NBS τους όρους με βάση τους οποίους θα μπορούσε να διατεθεί αυτό το κατοχυρωμένο προϊόν της σε άλλους κατασκευαστές. Η συμφωνία τελικά επήλθε στις 13 Μαΐου 1975.

Τον Μάρτιο του 1975 δημοσιεύθηκε περιγραφή του αλγορίθμου προκειμένου να υπάρξουν σχόλια και προτάσεις από το ευρύ κοινό ενώ τον Αύγουστο της ίδιας χρονιάς δημοσιοποιήθηκε και ένα σχεδιάγραμμα του προτεινόμενου προτύπου. Τα σχόλια που ακολούθησαν ήταν ποικίλα. Πολλά από αυτά επικεντρωνόταν στους τρόπους χρησιμοποίησης και ενσωμάτωσης του συγκεκριμένου προτύπου σε διάφορες αρχιτεκτονικές υπολογιστών ενώ άλλα αναφέρονταν στην κρυπτογραφική δύναμη του αλγορίθμου. Μετά την διαδικασία αυτή ανταλλαγής απόψεων, η οποία διήρκησε περίπου 18 μήνες, στις 23 Νοεμβρίου του 1976 ο αλγόριθμος αυτός κρυπτογράφησης υιοθετήθηκε ως Εθνικό Πρότυπο ενώ δημοσιεύθηκε επίσημα σαν έκδοση FIPS-46 με το όνομα DES στις 15 Ιανουαρίου του 1977. Έξι μήνες αργότερα, στις 15 Ιουλίου του 1977, κατέστη υποχρεωτικός και όλες οι Εθνικές Υπηρεσίες, εκτός και εάν πρόβαλαν έναν αρκετά σοβαρό και βάσιμο λόγο, θα έπρεπε να συμμορφωθούν με τις προδιαγραφές του.

Ο αλγόριθμος DES σχεδιάστηκε με βάση κάποια συγκεκριμένα κριτήρια. Κατ' αρχήν ο προτεινόμενος αλγόριθμος θα έπρεπε να έχει εξαιρετικά υψηλό επίπεδο ασφαλείας το οποίο όμως δεν θα εξαρτάται από την μυστικότητα του κρυπταλγορίθμου ενώ οι προδιαγραφές του επιβάλλεται να είναι πλήρεις και διαφανείς. Να είναι κατάλληλο για ποικιλία εφαρμογών με χαμηλό κόστος υλοποίησης, να επιτρέπεται η εξαγωγή του (μία έκδοση του DES για εξαγωγή από τις ΗΠΑ είναι η Commercial Data Masking Facility ή CDMF της IBM, η οποία χρησιμοποιεί ένα κλειδί των 40 bits) και τέλος να καθίσταται δυνατή η αξιολόγηση του.

Ωστόσο όλα αυτά τα κριτήρια ήταν καθαρά σε θεωρητικό επίπεδο, στην πράξη πολλά από αυτά τροποποιήθηκαν ή ακόμα και αγνοήθηκαν. Πρώτα απ' όλα ο DES είχε πολύ μικρότερο κλειδί

(ο κλειδοχώρος ορίζεται από 2^{56} κλειδιά) από αυτό του προκατόχου του LUCIFER (2^{128} κλειδιά) γεγονός που από μόνο του τον καθιστούσε λιγότερο ασφαλή. Σε αυτό το γεγονός όμως ευθύνεται αποκλειστικά η NSA των ΗΠΑ η οποία άσκησε πιέσεις για μικρό μήκος κλειδιού. Το κριτήριο διαφάνειας των προδιαγραφών ουσιαστικά αγνοήθηκε αφού τα κριτήρια σχεδιασμού των κουτιών αντικατάστασης που περιέχονται στη συνάρτηση γύρου του DES αποκαλύφθηκαν στα μέσα περίπου της δεκαετίας του `90 και τα κριτήρια σχεδιασμού τους περιείχαν ενδείξεις ότι η τεχνική της διαφορικής κρυπτανάλυσης που αποκαλύφθηκε στις αρχές της ίδιας δεκαετίας ήταν γνωστή πριν από 15 χρόνια. Αυτό αποδεικνύει εν' μέρει και η επιλογή των γύρων του δικτύου Feistel του DES η οποία έγινε έτσι ώστε ο αλγόριθμος να είναι ανθεκτικός σε περίπτωση διαφορικής κρυπτανάλυσης.



Σχήμα 4.1 : Ο αλγόριθμος DES

Ο DES είναι ο μοναδικός αλγόριθμος κρυπτογράφησης ο οποίος έχει υποστεί την μεγαλύτερη έρευνα ούτως ώστε να εξακριβωθεί η κρυπτογραφική του δύναμη. Οι απόπειρες κρυπτανάλυσης του DES είχαν σαν αποτέλεσμα την ανακάλυψη και καθιέρωση ποικίλων αρχών σχεδίασης των κρυπταλγορίθμων τμήματος.

4.1 Περιγραφή του DES

Ο αλγόριθμος DES έχει σχεδιαστεί με σκοπό να κρυπτογραφεί και να αποκρυπτογραφεί σύνολα δεδομένων που αποτελούνται από 64-bits με την βοήθεια ενός 64-bits κλειδιού. Η αποκρυπτογράφηση θα πρέπει να επιτυγχάνεται με την χρησιμοποίηση του ίδιου κλειδιού που χρησιμοποιήθηκε για την κρυπτογράφηση, αλλά με αλλαγμένο το πρόγραμμα εφαρμογής των κλειδιών έτσι ώστε η διαδικασία αποκρυπτογράφησης να είναι η αντιστροφή της διαδικασίας κρυπτογράφησης. Εάν δηλαδή κατά την κρυπτογράφηση το πρόγραμμα κλειδιού είναι $\{k_1,k_2,\ldots,k_{16}\}$, κατά την αποκρυπτογράφηση θα είναι $k_{16},k_{15},\ldots,k_1\}$.

Το αρχικό προς κρυπτογράφηση κείμενο υποβάλλεται σε μια μετάθεση σύμφωνα με τον πίνακα, κατόπιν σε έναν σύνθετο υπολογισμό ο οποίος βασίζεται και εξαρτάται από το κλειδί και τελικά σε μια μετάθεση $\mathbf{IP^{-1}}$ η οποία είναι το αντίστροφο της αρχικής μετάθεσης **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

Πίνακας 4.1: IP

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

Πίνακας 4.2: $\mathbf{IP^{-1}}$

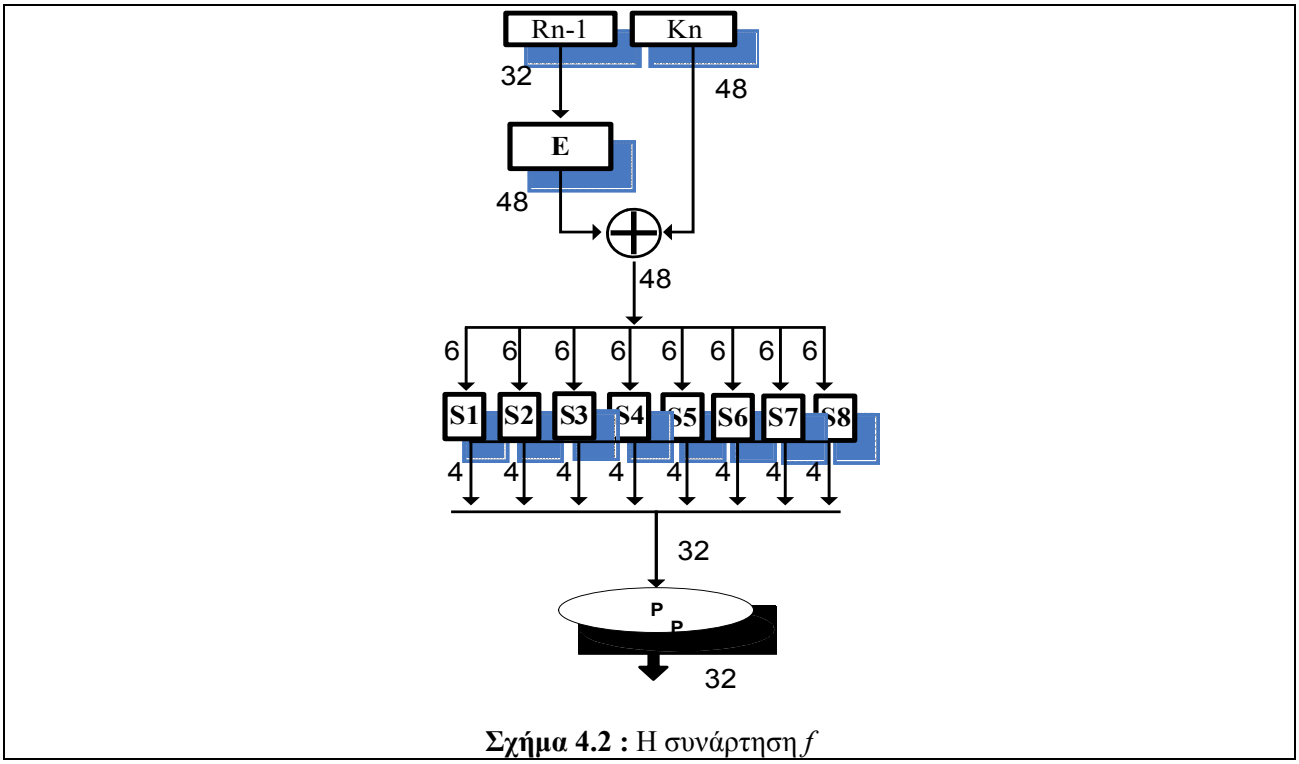
Ο λόγος που επιλέχθηκε η εξάρτηση αυτή των δύο μεταθέσεων δεν έχει να κάνει με την προσθήκη κάποιας επιπλέον ασφάλειας, αφού σε αυτές τις μεταθέσεις δεν παίρνει μέρος το κλειδί, αλλά για να μπορεί να χρησιμοποιηθεί η ίδια διαδικασία και στην κρυπτογράφηση και στην αποκρυπτογράφηση. Επίσης, η ύπαρξη και μόνο των δύο αυτών μεταθέσεων προσδίδει την δυνατότητα αποθήκευσης τόσο του απλού κειμένου όσο και του κρυπτοκειμένου πριν και μετά την εφαρμογή του δικτύου Feistel.

Ένας άλλος λόγος είναι ότι δημιουργούνται ομαδοποιήσεις. Τα bits που βρίσκονται σε θέση πολλαπλάσια του 8 δημιουργούν δυαδικές λέξεις, έτσι όταν το απλό κείμενο αποτελείται από χαρακτήρες ASCII διαχωρίζεται η πληροφορία του χαρακτήρα από το όγδοο bit αρτιότητας.

Ο υπολογισμός ο οποίος βασίζεται και εξαρτάται από το κλειδί μπορεί απλά να περιγραφεί σε όρους μίας συνάρτησης *f*, η οποία καλείται συνάρτηση κρυπτογράφησης και μία συνάρτηση για το πρόγραμμα του κλειδιού η οποία καλείται KS. Μια περιγραφή της συνάρτησης *f* δίνεται στην συνέχεια μαζί με τις λεπτομέρειες ως προς τον τρόπο με τον οποίο ο αλγόριθμος χρησιμοποιείται για κρυπτογράφηση.

4.1.1 Η συνάρτηση *f*

Μετά την αρχική μετάθεση IP τα 64-bit κειμένου χωρίζονται σε δύο κείμενα των 32-bits τα οποία οδηγούνται σε δύο καταχωρητές L και R, από τις λέξεις Left (Αριστερά) και Right (Δεξιά), οι οποίοι αποτελούν βασικό κομμάτι του γύρου και της συνάρτησης του DES.



Σχήμα 4.2 : Η συνάρτηση *f*

Από τον καταχωρητή R τα 32 bits προχωρούν στην συνάρτηση επέκτασης E, η οποία περιλαμβάνει μία μετάθεση κατά την οποία ορισμένα bits της εισόδου εμφανίζονται περισσότερες φορές από μία έτσι ώστε η έξοδος τελικά να αποτελείται από 48 bits. Τα bit εισόδου που εμφανίζονται διπλά είναι τα 4,5,8,9,12,13…,28,29. Αναλυτικότερα όλη η διαδικασία της επέκτασης φαίνεται στον πίνακα που ακολουθεί.

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

Πίνακας 4.3: Η Συνάρτηση Επέκτασης **E**

Η έξοδος των 48-bits από την συνάρτηση επέκτασης μέσω μίας πύλης XOR μέσω της οποίας συνδυάζεται με το αντίστοιχο κλειδί κρυπτογράφησης. Στην συνέχεια τα 48 αυτά bit από την έξοδο της XOR χωρίζονται σε οκτώ εξάδες και κάθε μία από αυτές οδηγείται στην είσοδο των

οκτώ κουτιών αντικατάστασης (S-Boxes), έτσι τα κουτιά αυτά έχουν είσοδο 6-bit αλλά παράγουν μόνο 4-bit στην έξοδο τους (η ακριβής λειτουργία των κουτιών αντικατάστασης περιγράφεται στο Κεφάλαιο 5.2.2).

Οι έξοδοι των 4-bit από όλα τα κουτιά αντικατάστασης (δηλαδή συνολικά 32 bit) συγκεντρώνονται και μπαίνουν σαν είσοδο στην μετάθεση P η οποία είναι μια πολύ ασυνήθιστη μετάθεση η οποία διαφέρει πολύ από τις μεταθέσεις IP και E που περιγράψαμε παραπάνω. Αυτή η μετάθεση ούτε αφαιρεί αλλά ούτε και αντιγράφει bit σε περισσότερες από μία θέσεις άρα και η έξοδος της θα είναι πάλι 32-bit. Ο τρόπος με τον οποίο γίνεται η μετάθεση αυτή φαίνεται στον πίνακα που ακολουθεί.

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

Πίνακας 4.4: Η Μετάθεση **P**

Μετά την ολοκλήρωση της συνάρτησης f τα 32 αυτά bit, από την έξοδο της μετάθεσης P τα οποία είναι και η τελική έξοδος της συνάρτησης, γίνονται XOR με αυτά του καταχωρητή L.

Όλη η παραπάνω διαδικασία του γύρου που περιγράψαμε θα επαναληφθεί 16 φορές και τότε τα περιεχόμενα των καταχωρητών L και R θα συγκεντρωθούν σε ένα σύνολο 64-bit (με την σειρά που φαίνεται στο σχήμα 4.3) και θα υποβληθούν στην τελική μετάθεση IP^{-1} και στην συνέχεια το αποτέλεσμα που θα προκύψει θα είναι η τελική μας έξοδος.

R	**L**

{----32-bit----}{----32-bit----}

{----------64-bit----------}

Σχήμα 4.3 Η σύνδεση των περιεχομένων των καταχωρητών μετά τον $16^{ο}$ γύρο επεξεργασίας.

4.1.2 Ανάλυση S-box

Τα οκτώ κουτιά αντικατάστασης είναι το πιο σημαντικό κομμάτι του DES. Μέχρι το 1994 δεν είχε κοινοποιηθεί τίποτα για τα κριτήρια σχεδιασμού τους. Αυτό και μόνο το γεγονός δείχνει

ότι η κρυπτογραφική δύναμη του αλγορίθμου ήταν άρρηκτα συνδεδεμένη με αυτά. Τα κουτιά αυτά επίσης προσδίδουν μη-γραμμικότητα στον τρόπο κατασκευής του κρυπταλγόριθμου.

Τα κριτήρια σχεδιασμού όπως έγιναν γνωστά σε μία δημοσίευση του Coppersmith είναι τα ακόλουθα:

- Αν τα δύο πρώτα bits και τα δύο τελευταία bits είναι σταθερά και τα ενδιάμεσα αλλάζουν οι έξοδοι που προκύπτουν θα πρέπει να είναι μοναδικές.
- Κάθε κουτί αντικατάστασης έχει είσοδο των 6-bits ενώ η έξοδος του είναι των 4-bits.
- Κανένα από τα bits της εξόδου δεν θα πρέπει να έχει γραμμική σχέση με οποιοδήποτε από τα bits εξόδου.
- Αν η απόσταση Hamming δύο εισόδων είναι ίση με 1, τότε η απόσταση Hamming των αντιστοίχων εξόδων θα πρέπει να είναι το λιγότερο ίση με 2. Ως απόσταση Hamming ορίζεται ο αριθμός των θέσεων ανάμεσα σε δύο ακολουθίες bit (string) ίσου μήκους των οποίων τα επιμέρους στοιχεία διαφέρουν ή αλλιώς ο αριθμός των μετατοπίσεων που απαιτείται ούτως ώστε να γίνει το ένα ίδιο με το άλλο.

  Για παράδειγμα:

  Η απόσταση Hamming ανάμεσα στο **10<u>1</u>1<u>1</u>01** και στο **10<u>0</u>1<u>0</u>01** είναι 2.
- Εάν δύο είσοδοι διαφέρουν στα δύο μεσαία bits τότε οι αντίστοιχες έξοδοι θα πρέπει να διαφέρουν το λιγότερο σε δύο bits.
- Αν δύο είσοδοι έχουν τα δύο πρώτα bits διαφορετικά ενώ τα δύο τελευταία bits είναι ίδια, τότε οι αντίστοιχες έξοδοι θα πρέπει να είναι διαφορετικές.
- Για οποιαδήποτε μη μηδενική διαφορά των 6 bits της εισόδου, θα πρέπει το πολύ 8 από τα 32 ζευγάρια να προκαλούν την ίδια διαφορά εξόδου.
- Όμοια με το παραπάνω κριτήριο, αλλά θα πρέπει να εφαρμόζεται συγχρόνως σε οποιαδήποτε 3 από τα 8 κουτιά αντικατάστασης.

Έτσι με βάση αυτά τα κριτήρια σχεδιάστηκαν τα κουτιά S_1 έως S_8 και οι τιμές που δόθηκαν σε κάθε κουτί φαίνονται στους παρακάτω πίνακες.

Κουτί Αντικατάστασης S_1

	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

Κουτί Αντικατάστασης S_2

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

Κουτί Αντικατάστασης S_3

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

Κουτί Αντικατάστασης S_4

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

Κουτί Αντικατάστασης S_5

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

Κουτί Αντικατάστασης S_6

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

Κουτί Αντικατάστασης S_7

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

Κουτί Αντικατάστασης S_8

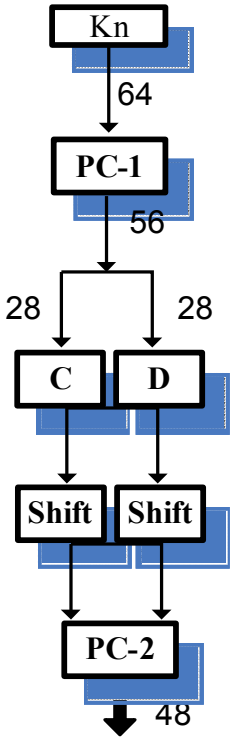
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

4.1.3 Πρόγραμμα κλειδιού του DES

Όπως προαναφέραμε η είσοδος κάθε γύρου εκτός από τα 32 bits δεδομένων απαιτεί και 48bits του κλειδιού. Το κλειδί κάθε γύρου είναι διαφορετικό και προκύπτει από ένα συγκεκριμένο πρόγραμμα κλειδιού το οποίο φαίνεται στο παρακάτω σχήμα 4.4.

Το πρόγραμμα του κλειδιού αποτελείται από δύο συναρτήσεις μετάθεσης επιλογής. Ως μετάθεση επιλογής ορίζεται η μετάθεση κατά την οποία ορισμένα bits της εισόδου αγνοούνται και δεν εμφανίζονται στην έξοδο. Οι δύο συναρτήσεις επιλογής είναι οι ακόλουθες:

PC-1: $\{ 0,1 \}^{64} \rightarrow \{ 0,1\}^{56}$ και PC-2: $\{ 0,1 \}^{56} \rightarrow \{ 0,1\}^{48}$



Σχήμα 4.4: Το Πρόγραμμα Κλειδιού του DES

Η PC-1 ξεχωρίζει όλα τα bits του αρχικού κλειδιού που θα συμμετάσχουν στο πρόγραμμα του κλειδιού για την δημιουργία των επιμέρους κλειδιών. Έτσι το κάθε όγδοο bit του αρχικού κλειδιού αγνοείται με αποτέλεσμα να διατίθενται για την κρυπτογράφηση 64-8 = 56bits. Οι μεταθέσεις PC-1 και PC-2 παρουσιάζονται στους παρακάτω πίνακες.

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

Πίνακας 4.5: PC-1

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

Πίνακας 4.6: PC-2

Στην συνέχεια τα 56bits χωρίζονται σε δύο λέξεις των 28bits και οδηγούνται σε καταχωρητές ολίσθησης. Στην συνέχεια ανάλογα με τον γύρο πραγματοποιείται μια ολίσθηση n προκαθορισμένων θέσεων όπως φαίνονται στο παρακάτω σχήμα.

Γύρος	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Αρ.Ολισθήσεων	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Πίνακας 4.7: Ολισθήσεις Κλειδιού

Κατά την διαδικασία της κρυπτογράφησης η ολίσθηση εκτελείται κατά την αριστερή φορά ενώ κατά την αποκρυπτογράφηση προς την δεξιά φορά. Ο αριθμός των ολισθήσεων έχει ως αποτέλεσμα το κλειδί να επανέλθει στην αρχική του θέση. Έτσι κατά την αποκρυπτογράφηση εκτελούνται οι αντίστροφες ολισθήσεις με την διαφορά ότι η αρχική ολίσθηση είναι μηδενική. Δηλαδή : 0,1,2,2…..,1.

Μετά την ολοκλήρωση κάθε ολίσθησης επιλέγονται τα 48 από τα 56 bits που βρίσκονται στον καταχωρητή ολίσθησης σύμφωνα με την μετάθεση επιλογής PC-2 [SCH96].

4.2 Ασφάλεια DES

Ο DES, είναι ο κρυπταλγόριθμος που έχει μελετηθεί περισσότερο όσον αφορά το πλήθος των δημοσιευμένων μελετών και το χρονικό διάστημα. Πολλά πορίσματα που διατυπώθηκαν χάριν του DES μετατράπηκαν αργότερα σε αρχές σχεδιασμού συμμετρικών αλγορίθμων τμήματος τύπου DES.

Το πρώτο βήμα ανάλυσης του κρυπταλγόριθμου, είναι η εξέταση του μεγέθους κλειδιού που ορίζει και το μέγεθος του κλειδοχώρου. Το DES επιτρέπει 2^{56} διαφορετικά κλειδιά. Αρχικά, ήταν σχεδιασμένο για 2^{64} κλειδιά, αλλά το μέγεθος αυτό μειώθηκε για να συμπεριληφθεί η πληροφορία των bits αρτιότητας. Ο κλειδοχώρος του DES κατακρινόταν πάντοτε για το μικρό του μέγεθος. Κατά την δεκαετία του 80 ήταν οικονομικά επιτρεπτό σε μια κυβέρνηση να σπάσει το κρυπτοσύστημα με εξαντλητική αναζήτηση. Σήμερα, η κατανεμημένη υπολογιστική επιτρέπει τη συνεργασία μελών κοινότητας του διαδικτύου, όπου το κάθε μέλος συνεισφέρει με την υπολογιστική ισχύ που διαθέτει για να επιτύχει η εξαντλητική αναζήτηση σε λογικό χρονικό διάστημα. Η κοινότητα γνωστή με το όνομα distributed.net πραγματοποίησε τον Ιανουάριο του 1999 εξαντλητική αναζήτηση στην πρόκληση "DES Challenge III" της εταιρείας RSA Laboratories και το κλειδί βρέθηκε μετά από 22 ώρες και 15 λεπτά. Στην αναζήτηση συμμετείχαν 100.000 ηλεκτρονικοί υπολογιστές, με δυνατότητα να αναζήτησης 245 δισεκατομμύρια κλειδιών ανά δευτερόλεπτο. Για να συνειδητοποιήσουμε πόσο ευάλωτος είναι ο DES όσον αφορά το

μέγεθος του κλειδιού του, αρκεί να αναγάγουμε την υπολογιστική ισχύ του 1999 στα σημερινά δεδομένα (2005).

Μια άλλη παράμετρος η οποία είναι εξίσου κατακριτέα είναι το μικρό μέγεθος των κουτιών αντικατάστασης. Η διαδικασία καθορισμού των κουτιών αντικατάστασης ήταν κρυφή για αρκετά χρόνια και η παρέμβαση της Υπηρεσίας Εθνικής Ασφάλειας στο σχεδιασμό, δημιούργησε υποψίες ύπαρξης "μυστικής πόρτας", η οποία επιτρέπει την αποκρυπτογράφηση χωρίς τη γνώση του κλειδιού. Ωστόσο, η ανάλυση των κουτιών έδειξε ότι η επιλογή αυτών έγινε με ιδιαίτερη προσοχή, αφού στην περίπτωση χρησιμοποίησης τυχαίων κουτιών, η ασφάλεια του DES μειώνονταν σημαντικά. Βέβαια, κάτι τέτοιο δεν αποδεικνύει την ύπαρξη ή όχι κάποιας μυστικής πόρτας [AST03].

4.2.1 Αδύναμα και Ημιαδύναμα Κλειδιά

Το πρόγραμμα κλειδιού του DES είναι αδύναμο, όχι μόνο επειδή από οποιοδήποτε κλειδί γύρου μπορεί να βρεθεί με σχετική ευκολία το αρχικό κλειδί, αλλά και επειδή υπάρχουν κλειδιά τα οποία παράγουν το ίδιο πρόγραμμα κλειδιού κατά την κρυπτογράφηση και την αποκρυπτογράφηση. Υπό κανονικές συνθήκες, το πρόγραμμα κλειδιού της αποκρυπτογράφησης έχει αντίστροφη σειρά του προγράμματος κλειδιού της κρυπτογράφησης.

01	01	01	01	01	01	01	01
FE	FE	FE	FE	FE	FE	FE	FE
1F	1F	1F	1F	1F	1F	1F	1F
E0	E0	E0	E0	E0	E0	E0	E0

Πίνακας 4.8: Τα αδύναμα κλειδιά του DES σε δεκαεξαδική αναπαράσταση

01	FE	01	FE	01	FE	01	FE	1F	E0	1F	E0	0E	F1	0E	F1
FE	01	FE	01	FE	01	FE	01	E0	1F	E0	1F	F1	0E	F1	0E
01	E0	01	E0	01	F1	01	F1	1F	FE	1F	FE	0E	FE	0E	FE
E0	01	E0	01	F1	01	F1	01	FE	1F	FE	1F	FE	0E	FE	0E
01	1F	01	1F	01	0E	01	0E	E0	FE	E0	FE	F1	FE	F1	FE
1F	01	1F	01	0E	01	0E	01	FE	E0	FE	E0	FE	F1	FE	F1

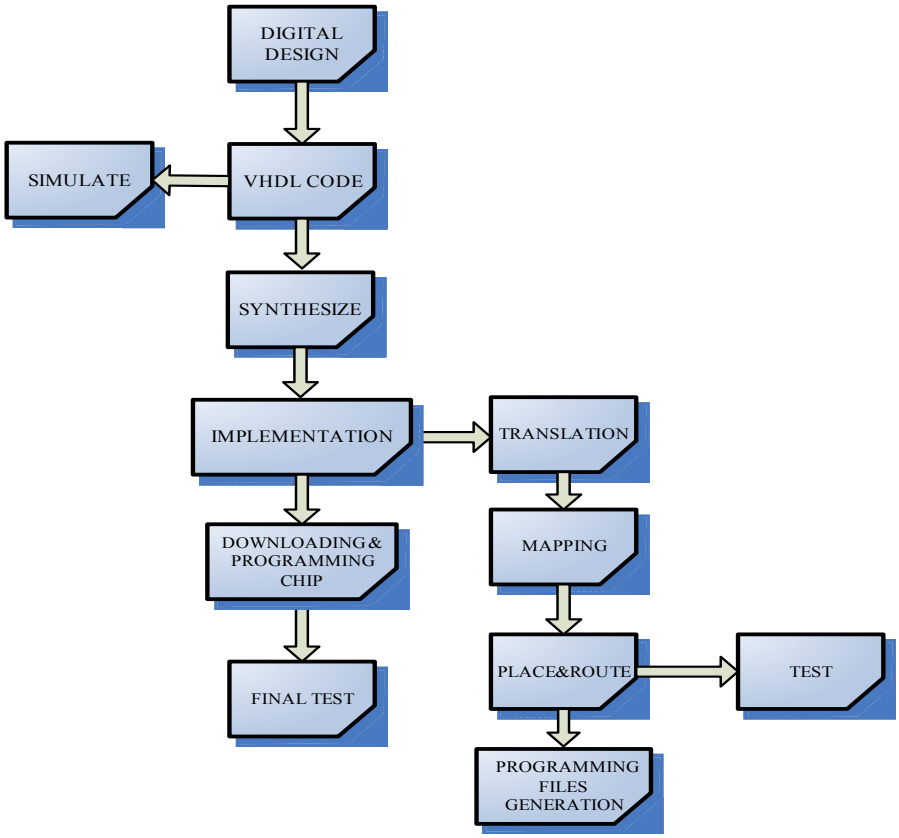
Πίνακας 4.9: Τα ημιαδύναμα κλειδιά του DES

Στην περίπτωση όμως που το κλειδί είναι κάποιο από τα κλειδιά του πίνακα 4.8, τότε το πρόγραμμα κλειδιών που προκύπτει είναι το ίδιο τόσο στην κρυπτογράφηση, όσο και στην αποκρυπτογράφηση. Αυτό σημαίνει ότι στην περίπτωση που ένα σύστημα χρησιμοποιεί διπλή κρυπτογράφηση, το κρυπτοκείμενο που προκύπτει θα είναι ίδιο με το απλό κείμενο. Δηλαδή, η δεύτερη κρυπτογράφηση θα αναιρεί την πρώτη. Τα κλειδιά αυτά ονομάζονται αδύναμα κλειδιά και θα πρέπει να αποφεύγονται σε εφαρμογές που απαιτείται υψηλή ασφάλεια, όπως για παράδειγμα στη δημιουργία κλειδιών από ένα αρχικό κλειδί.

Εκτός από τα αδύναμα κλειδιά, υπάρχουν και τα ημιαδύναμα κλειδιά. Τα κλειδιά αυτά εμφανίζεται σε ζευγάρια, όπου το πρόγραμμα του κλειδιού του ενός είναι ισοδύναμο με το πρόγραμμα κλειδιού του άλλου, με αντίστροφη σειρά. Έτσι, η κρυπτογράφηση του ενός κλειδιού ακολουθούμενη από την κρυπτογράφηση του δεύτερου κλειδιού που ορίζουν το ζευγάρι, έχει ως αποτέλεσμα το κρυπτοκείμενο να είναι ίσο με το αρχικό απλό κείμενο. Η ύπαρξη τόσο των αδύναμων κλειδιών, όσο και των ζευγαριών των ημιαδύναμων κλειδιών, οφείλεται στην απλοϊκή κατασκευή του αλγορίθμου του προγράμματος κλειδιών.

5 Μελέτη και Υλοποίηση του Αλγορίθμου DES και Triple-DES

Για την υλοποίηση του αλγορίθμου DES με χρήση της γλώσσας περιγραφής υλικού VHDL (Very High Speed Integrated Circuits Hardware Description Language) ακολουθήθηκε η διαδικασία η οποία συνοπτικά παρουσιάζεται στο παρακάτω σχήμα.



Σχήμα 5.1: Διάγραμμά ροής υλοποίησης

Αρχικά, σχεδιάστηκε το ψηφιακό διάγραμμα υλοποίησης του αλγορίθμου DES (Digital Design) με χρήση της αρχιτεκτονικής συνεχόμενης ροής (pipeline), βάση του οποίου θα έπρεπε να κατασκευαστεί ούτως ώστε να πληροί όλες τις απαραίτητες απαιτήσεις (υψηλή ταχύτητα δεδομένων (data rate)) και προϋποθέσεις (ενσωμάτωση σε PCI αρχιτεκτονική και κατ' επέκταση «κατάληψη» περιορισμένου χώρου του Chip- λιγότερη χρήση τόσο καταχωρητών όσο και ψηφιακών πυλών γενικότερα.), για την σωστή λειτουργία του.

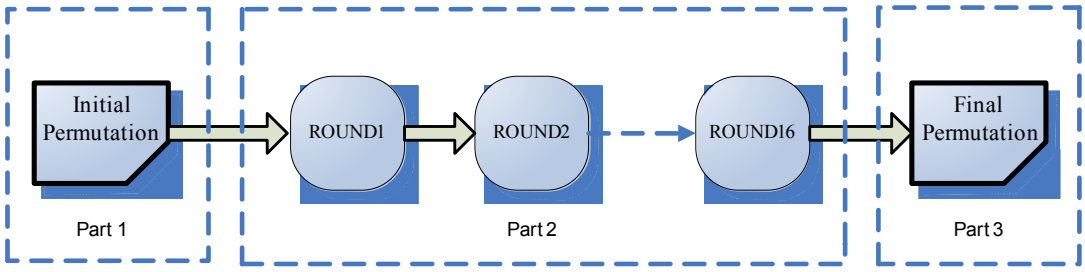
Έπειτα, δημιουργήθηκαν όλα τα απαραίτητα μέρη του σχεδίου σε κώδικα με χρήση της γλώσσας VHDL, τα οποία προσομοιώθηκαν σε επίπεδο λογισμικού (simulation) και ελέγχθηκαν για την σωστή λειτουργία τους. Στην συνέχεια έγινε η σύνθεση (synthesis) του κώδικα και η υλοποίηση (implementation) του αλγορίθμου. (βλέπε Σχήμα 5.1).

Η υλοποίηση αποτελείται από διάφορα επιμέρους στάδια. Αρχικά, πραγματοποιήθηκε η μετάφραση (translation) του κώδικα σε ψηφιακό επίπεδο πυλών. Έπειτα έγινε η απαραίτητη δημιουργία αρχείων για τη χαρτογράφηση του Chip (Mapping). Στη συνέχεια πραγματοποιήθηκε η τοποθέτηση των ψηφιακών στοιχείων (Components) στο Chip και η διασύνδεση αυτών (Place and route.). Σε αυτό το σημείο έγινε και ο έλεγχος (Testing) του Chip σε επίπεδο RTL (Register Transfer Level) λαμβάνοντας υπόψη και τις χρονοκαθυστερήσεις των ψηφιακών πυλών. Τέλος δημιουργήθηκαν τα κατάλληλα αρχεία (source files) για τον προγραμματισμό του Chip (βλέπε Σχήμα 5.1).

5.1 Υλοποίηση του Αλγορίθμου DES και Triple-DES

Ο τρόπος σχεδίασης που ακολουθήθηκε για την υλοποίηση του DES και του Triple-DES θα αναλυθούν παρακάτω.

Η υλοποίηση του DES έγινε κάτω από ορισμένες προϋποθέσεις και προδιαγραφές (όπως αναφέρονται στη εισαγωγή του παρόντος κεφαλαίου- βλέπε Κεφ.5.1). Με βάση όλα τα παραπάνω το σχέδιο χωρίστηκε σε τρία μέρη τα οποία και παρουσιάζονται στο σχήμα που ακολουθεί:

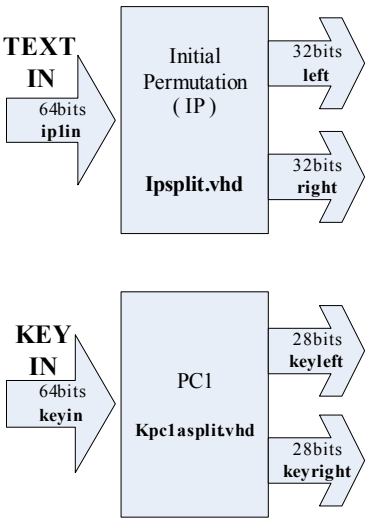


Σχήμα 5.2: Λειτουργία του DES

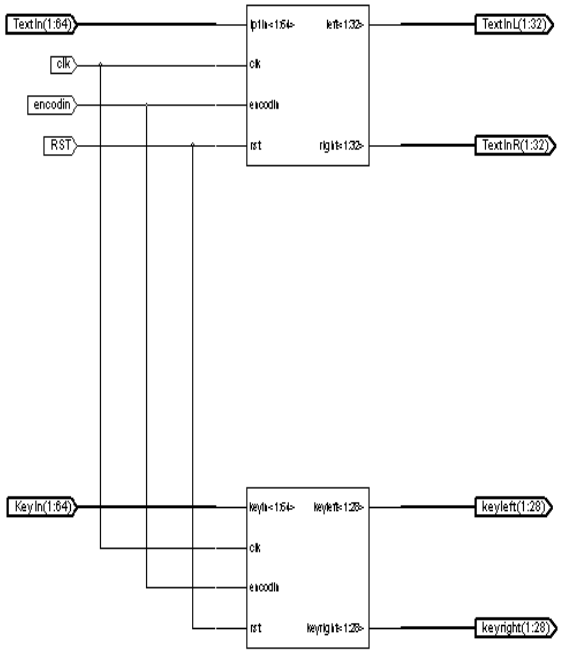
Το πρώτο κομμάτι της σχεδίασης (Part 1) χωρίζεται στην αρχική αντιμετάθεση των 64bits των δεδομένων και των 64bits του χρησιμοποιούμενου κλειδιού, προτού αυτά προχωρήσουν στους δεκαέξι συνεχόμενους γύρους επεξεργασίας τους (Part 2) οι οποίοι είναι όμοιοι μεταξύ τους. Μετά από αυτή την διαδικασία το τελικό αποτέλεσμα υπόκειται σε μία τελική αντιμετάθεση (Part 3) από την οποία και θα προκύψουν τα τελικά κρυπτογραφημένα δεδομένα (Βλέπε Σχήμα 5.2).

Αναλυτικότερα τα 3 μέρη της υλοποίησης του DES περιγράφονται στις επόμενες παραγράφους.

Όπως προαναφέρθηκε στο πρώτο μέρος της σχεδίασης γίνεται η απαραίτητη αντιμετάθεση τόσο του κλειδιού όσο και των εισερχόμενων δεδομένων. Συνοπτικά το πρώτο μέρος της σχεδίασης (Part 1- Βλέπε επίσης Σχήμα 5.2) παρουσιάζεται στο Σχήμα 5.3 που ακολουθεί ενώ στο Σχήμα 5.4 φαίνεται η λειτουργία των δύο καταχωρητών σε επίπεδο RTL.

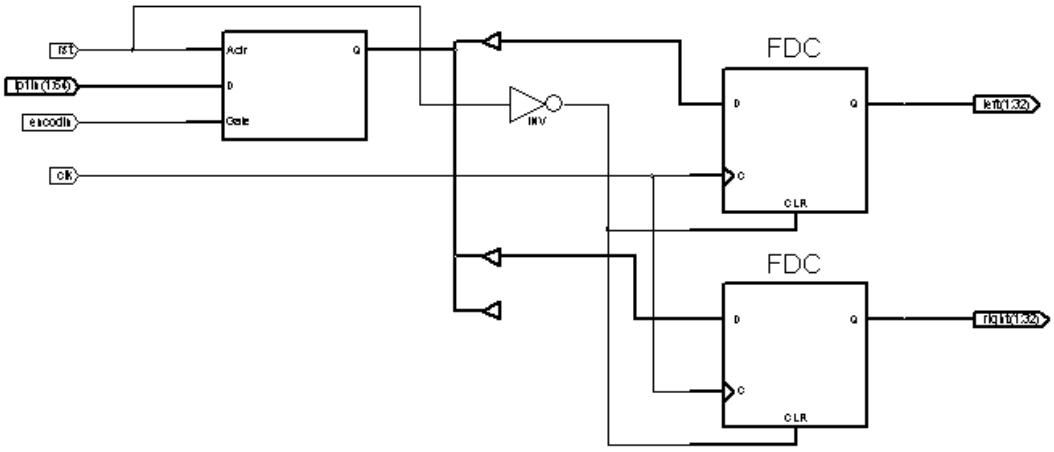


Σχήμα 5.3 :
Αρχική Αντιμετάθεση Δεδομένων-Κλειδιού



Σχήμα 5.4: RTL διάγραμμα

Μετά την ενεργοποίηση του απαραίτητου σήματος (*encoding*), το οποίο επιτρέπει την είσοδο των δεδομένων και του κλειδιού προς κρυπτογράφηση, γίνεται η απαραίτητη bit προς bit αντιμετάθεση τους. Τα δεδομένα αποθηκεύονται προσωρινά σε μία εσωτερική 64bits μεταβλητή, σύμφωνα με τον πίνακα 4.1 (βλέπε Κεφ 4.2). Ενώ στην συνέχεια τα 64 αυτά bit χωρίζονται σε δύο ισάριθμες 32bits μεταβλητές, οι οποίες και είναι η τελική έξοδος του καταχωρητή IP.



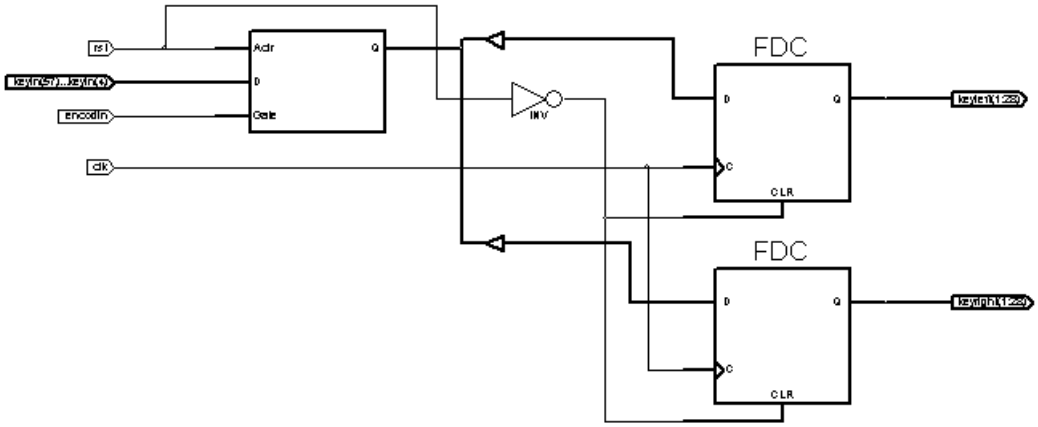
Σχήμα 5.5: Καταχωρητής **IP**

Παρόμοια διαδικασία ακολουθείται και κατά την επεξεργασία του κλειδιού προκειμένου τα 64 bits του κλειδιού να υποστούν την απαραίτητη αντιμετάθεση ενώ στην συνέχεια το μέγεθος του να μειωθεί στα 56bit.

Έτσι, πιο αναλυτικά το κλειδί αποθηκεύεται σε μία εσωτερική μεταβλητή και στην συνέχεια γίνεται αντιμετάθεση των bits του (Πίνακας 4.5 Κεφ.4.2.3) από όπου και προκύπτει ένα νέο κλειδί

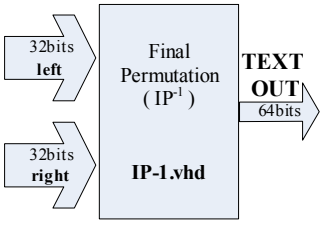
μεγέθους 56bits το οποίο στην συνέχεια χωρίζεται σε δύο ίσα μέρη τα οποία και είναι η τελική έξοδος του καταχωρητή PC1.

Στα RTL σχηματικά διαγράμματα των καταχωρητών IP και PC1 των σχημάτων 5.5 και 5.6 φαίνεται όλη η διαδικασία που περιγράψαμε σε επίπεδο ψηφιακής σχεδίασης.

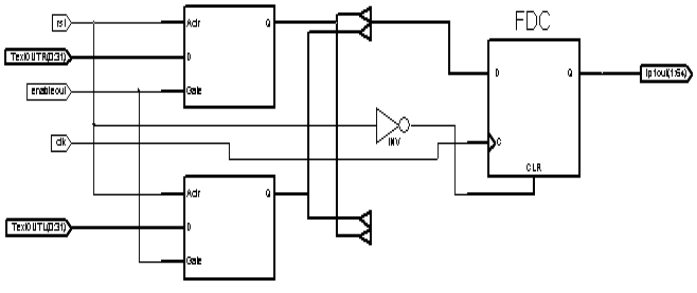


Σχήμα 5.6: Καταχωρητής **PC1**

Μετά την αντιμετάθεση που υπέστησαν τόσο το κλειδί όσο και τα εισερχόμενα προς κρυπτογράφηση δεδομένα γίνεται η επεξεργασία των εξόδων των καταχωρητών στους δεκαέξι συνεχόμενους γύρους που ακολουθούν (Part 2). Στο ακόλουθο υποκεφάλαιο θα γίνει πλήρης ανάλυση της λειτουργίας ενός γύρου DES. Στην συνέχεια μετά τους 16 αυτούς γύρους στα τελικά δεδομένα που προκύπτουν, στο τρίτο και τελευταίο μέρος της σχεδίασης (Part3), γίνεται μία τελευταία αντιμετάθεση από όπου και έχουμε την τελική μορφή του κρυπτογραφήματος μας. Αυτό το κομμάτι της σχεδίασης αποτελείται από την βαθμίδα IP^{-1} , η οποία όμως για την σωστή λειτουργία και τον απαραίτητο συγχρονισμό της, θα πρέπει να αναφερθεί ότι ελέγχεται από το κατάλληλο σήμα (*enableout*) που καθορίζει σε ποια χρονική στιγμή να κάνει εισαγωγή δεδομένων προς επεξεργασία. Η αντιμετάθεση αυτή γίνεται σύμφωνα με τον πίνακα 4.2 (Βλέπε Κεφ.4.2).



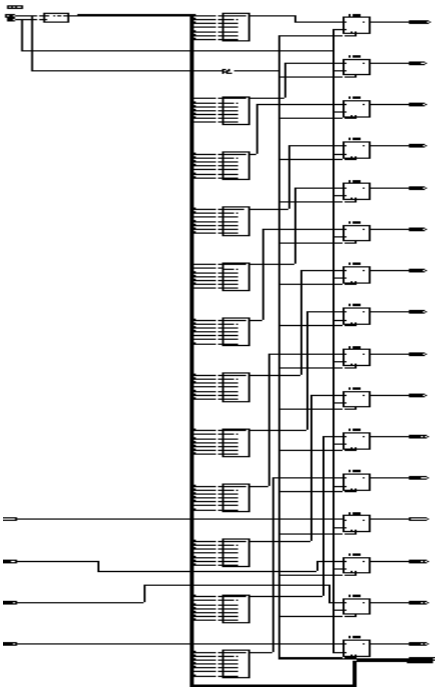
Σχήμα 5.7: Τελική Αντιμετάθεση



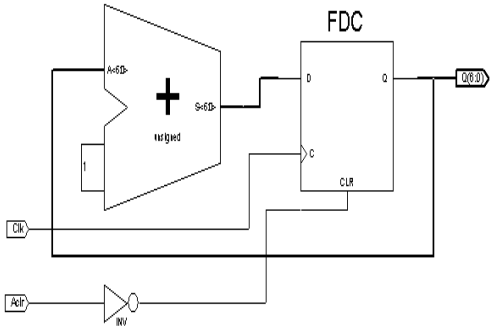
Σχήμα 5.8: RTL Διάγραμμα **IP-1**

Συνοπτικά το τρίτο μέρος της σχεδίασης (Part 3- Βλέπε επίσης Σχήμα 5.2) παρουσιάζεται στο σχήμα 5.7, ενώ στο σχήμα 5.8 φαίνεται η λειτουργία της βαθμίδας σε επίπεδο RTL.

Ο έλεγχος όλων των απαραίτητων σημάτων ελέγχου γίνεται από την βαθμίδα ελέγχου (Σχήμα 5.9), η οποία ελέγχει την όλη διαδικασία ενεργοποιώντας τα κατάλληλα σήματα στην σωστή χρονική στιγμή, ούτως ώστε να μην υπάρξει απώλεια των δεδομένων και του κλειδιού και να επιτύχουμε τον καλύτερο δυνατό συγχρονισμό.



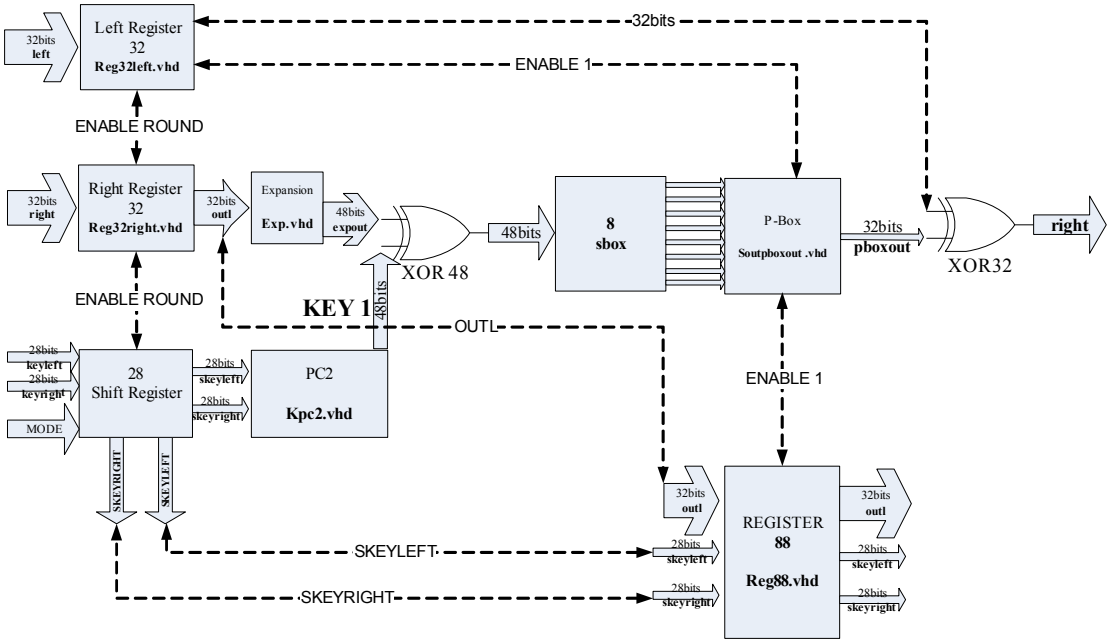
Σχήμα 5.9: Βαθμίδα Ελέγχου (Control Unit)



Σχήμα 5.10: Μετρητής (Counter)

5.1.1 Εσωτερική λειτουργία κύκλου DES (One Round Function)

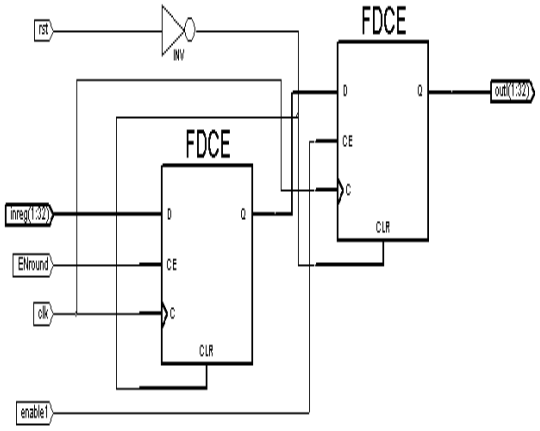
Για τον απαραίτητο συγχρονισμό εισόδου των δεδομένων και του κλειδιού στους γύρους επεξεργασίας τους, χρησιμοποιείται ένα σήμα ενεργοποίησης (*enableround*). Με την ενεργοποίηση του παραπάνω σήματος γίνεται ταυτόχρονη είσοδος των δεδομένων και του κλειδιού.



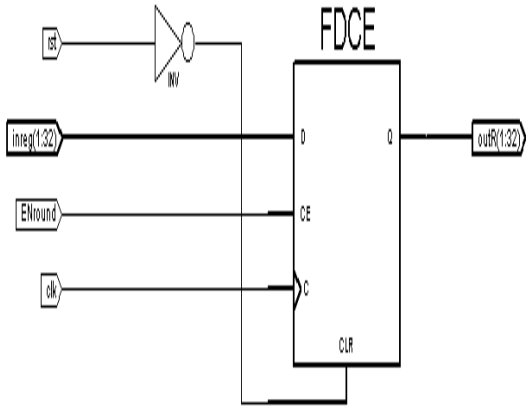
Σχήμα 5.11: Block Διάγραμμα Περιγραφής ενός γύρου DES

Τα δεδομένα αρχικά αποθηκεύονται σε δύο καταχωρητές (left και right register) της τάξης των 32bits. Ο καταχωρητής left register αποθηκεύει προσωρινά τα δεδομένα σε μία εσωτερική μεταβλητή και περιμένει το κατάλληλο σήμα ενεργοποίησης της εξόδου (*enable1*). Αυτό συμβαίνει προκειμένου κατά την έξοδο των δεδομένων να υπάρχει σωστός συγχρονισμός και η επεξεργασία τους να προχωρήσει σύμφωνα με το σχέδιο υλοποίησης (Σχήμα 5.11). Τα δεδομένα που έχουν εισαχθεί στον καταχωρητή right register αποθηκεύονται προσωρινά για ένα παλμό ρολογιού. Κατά την επόμενη θετική ακμή του ρολογιού εξέρχονται και οδηγούνται στην επόμενη βαθμίδα, η οποία είναι μία βαθμίδα επέκτασης δεδομένων (expansion). Ταυτόχρονα όμως οδηγούνται και σε ένα καταχωρητή, ο οποίος και τα αποθηκεύει (Register 88). Την ίδια χρονική στιγμή ο ίδιος καταχωρητής λαμβάνει και τα δύο 28bits μέρη του κλειδιού. Ο λόγος ύπαρξης του συγκεκριμένου καταχωρητή είναι να αποθηκεύει τόσο τα δεδομένα όσο και τα μέρη του κλειδιού μέχρι να γίνει όλη η απαραίτητη διαδικασία και στην συνέχεια αφού ενεργοποιηθεί το κατάλληλο σήμα (*enable1*) να έχουμε ταυτόχρονη έξοδο στο τέλος κάθε γύρου όλων των δεδομένων και των μερών των κλειδιών.

Ακολουθούν τα RTL σχηματικά διαγράμματα των καταχωρητών left register και right register στα οποία φαίνεται η ακριβής τους λειτουργία.

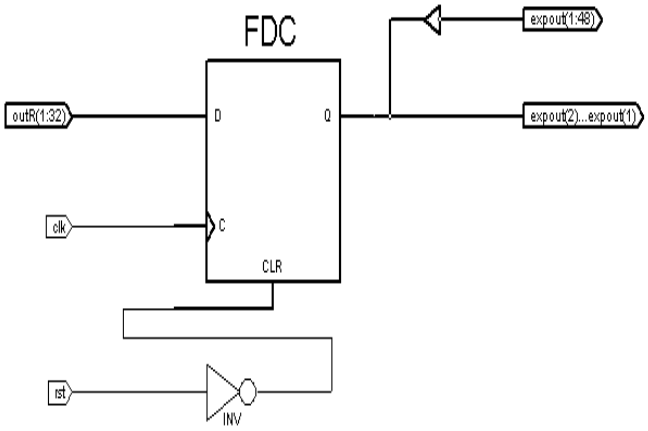


Σχήμα 5.12: Καταχωρητής Left



Σχήμα 5.13: Καταχωρητής Right

Σ’ αυτήν την χρονική στιγμή πραγματοποιείται η απαραίτητη επέκταση των δεδομένων από τα 32bits στα 48bits. Πρακτικά, εσωτερικά της βαθμίδας expansion γίνεται bit προς bit αντιγραφή των δεδομένων εισόδου σε μία μεγαλύτερη μεταβλητή σύμφωνα με τον Πίνακα 4.3 (Κεφάλαιο 4.2.1). Η μεταβλητή αυτή θα αποτελέσει και την τελική της έξοδο.



Σχήμα 5.14 Επέκταση Δεδομένων

Στην συνέχεια τα 48 bits δεδομένων γίνονται αποκλειστική διάζευξη (XOR) με τα 48 bits του κλειδιού που αντιστοιχεί στον συγκεκριμένο γύρο επεξεργασίας. Η διαδικασία για το πώς προκύπτει το κλειδί για κάθε γύρο, θα αναλυθεί σε ακόλουθο υποκεφάλαιο. Στην συνέχεια το αποτέλεσμα αυτό εισέρχεται στην μνήμη ROM που έχουμε κατασκευάσει. Εσωτερικά της μνήμης αυτής υλοποιούνται τα S-Boxes, τα οποία όπως έχουμε αναφέρει αποτελούν το πιο σημαντικό μέρος του αλγόριθμου κρυπτογράφησης, αφού από αυτά εξαρτάται σε μεγάλο βαθμό η κρυπτογραφική δύναμη του αλγορίθμου.

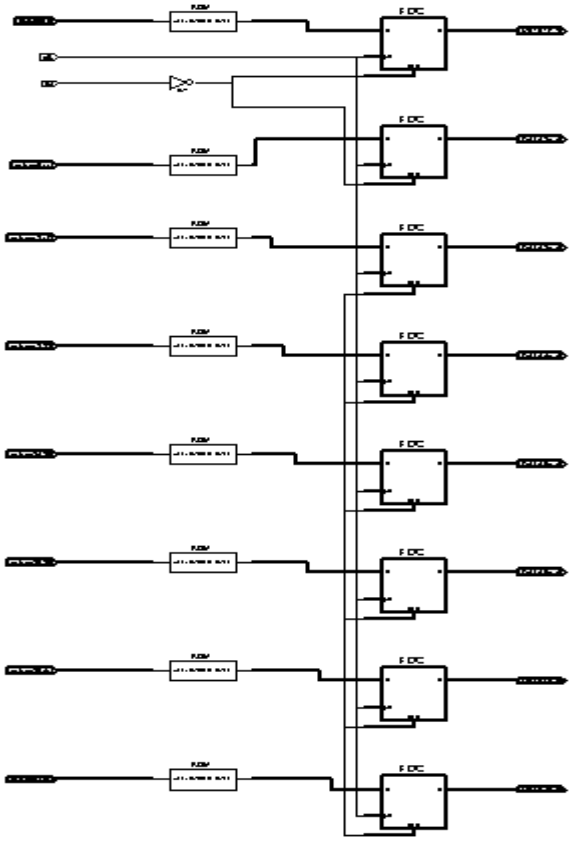
Η μνήμη που κατασκευάστηκε για αυτή την λειτουργία αποτελείται από οκτώ πίνακες αναζήτησης (look up tables). Με την εισαγωγή των δεδομένων στην μνήμη γίνεται διαχωρισμός των 48bits σε οκτώ ομάδες των 6bits με ταυτόχρονη αποθήκευση τους σε ισάριθμες εσωτερικές μεταβλητές. Κάθε μία από αυτές τις μεταβλητές θα αποτελέσει την συγκριτική τιμή για κάθε ένα από τους οκτώ πίνακες αναζήτησης – δηλαδή τα δεδομένα εισόδου των S-Boxes λειτουργούν σαν γεννήτρια διευθύνσεων. Έτσι, ανάλογα με τα δεδομένα που θα λαμβάνει κάθε φορά θα ανατρέχει στους σταθερούς πίνακες τιμών που έχουμε εισάγει (Βλέπε Κεφάλαιο 4.2.2) και θα βρίσκει την αντίστοιχη τιμή η οποία είναι της τάξης των 4bits την οποία και θα εξάγει ως τελικό αποτέλεσμα.

Επομένως στην έξοδο της μνήμης ROM θα λάβουμε οκτώ μεταβλητές των 4bits, δηλαδή συνολικά θα έχουμε 32bits δεδομένων. Οι μεταβλητές αυτές στην συνέχεια οδηγούνται στην βαθμίδα P-Box (Σχήμα 5.11) όπου μετά την είσοδο τους αποθηκεύονται σε μία 32bits μεταβλητή, προκειμένου να υποστούν μία τελική αντιμετάθεση η οποία γίνεται σύμφωνα με τον πίνακα 4.2 (Βλέπε Κεφ4.2). Το αποτέλεσμα της αντιμετάθεσης αυτής δεν θα εξαχθεί από την βαθμίδα αμέσως αλλά την χρονική στιγμή κατά την οποία θα γίνει ενεργοποίηση του απαραίτητου για τον συγχρονισμό σήματος (*enable1*). Το συγκεκριμένο αυτό σήμα ενεργοποιεί ταυτόχρονα και τις εξόδους των καταχωρητών left register και register 88, όπως έχουμε ήδη αναφέρει στην αρχή του κεφαλαίου.

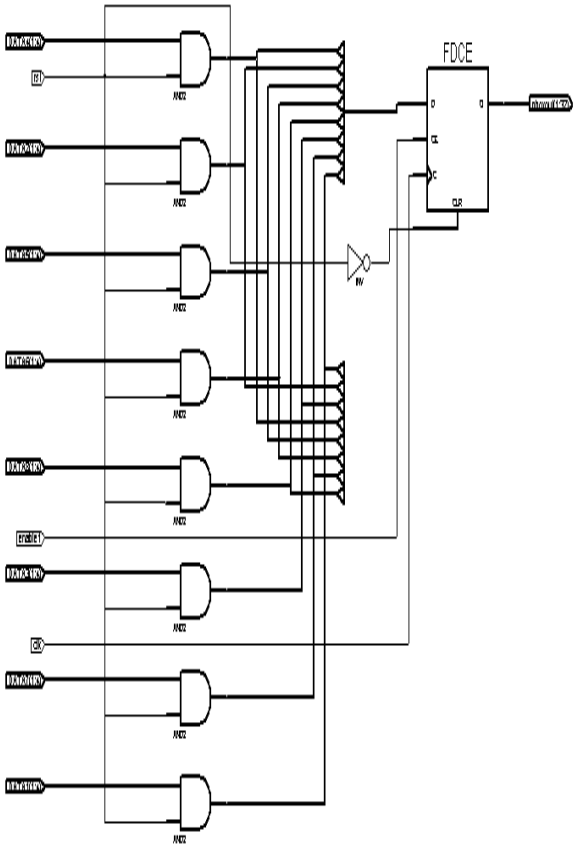
Με την ταυτόχρονη ενεργοποίηση των δύο αυτών βαθμιδών επιτυγχάνουμε τον συγχρονισμό των δύο εξόδων. Αυτό, γίνεται προκειμένου τα δεδομένα που θα λάβουμε να εισαχθούν στην επερχόμενη 32bit XOR χωρίς καμία χρονική διαφορά ούτως ώστε να εξασφαλίσουμε σωστό τελικό αποτέλεσμα (συγχρονισμό). Η αποκλειστική αυτή διάζευξη είναι και η τελική επεξεργασία των δεδομένων στο γύρο αυτό. Επομένως, το αποτέλεσμα που θα προκύψει (*outxor32*) θα είναι η τελική έξοδος του συγκεκριμένου γύρου κρυπτογράφησης.

Το σήμα ελέγχου (*enable1*) όπως είπαμε, ενεργοποιεί και τον καταχωρητή register88. Έτσι, έχουμε ταυτόχρονη έξοδο στο τέλος κάθε γύρου όλων των δεδομένων και των μερών των κλειδιού.

Στα RTL σχηματικά διαγράμματα των S-Boxes και του P-Box που ακολουθούν φαίνεται όλη η λειτουργία που αναλύθηκε παραπάνω.

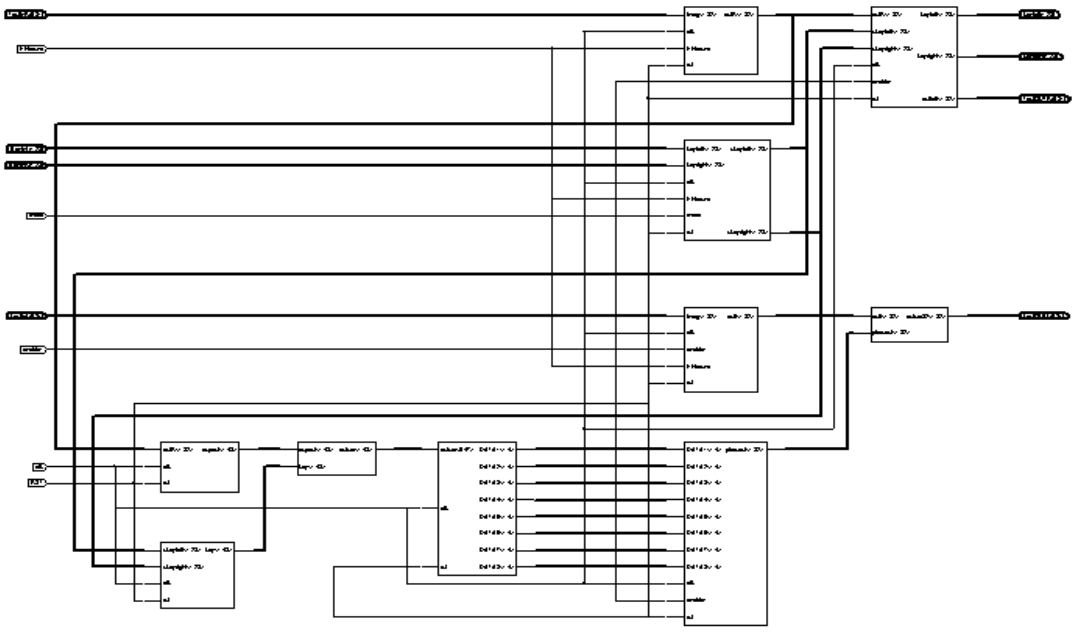


Σχήμα 5.15 8ROMS (S-Boxes)



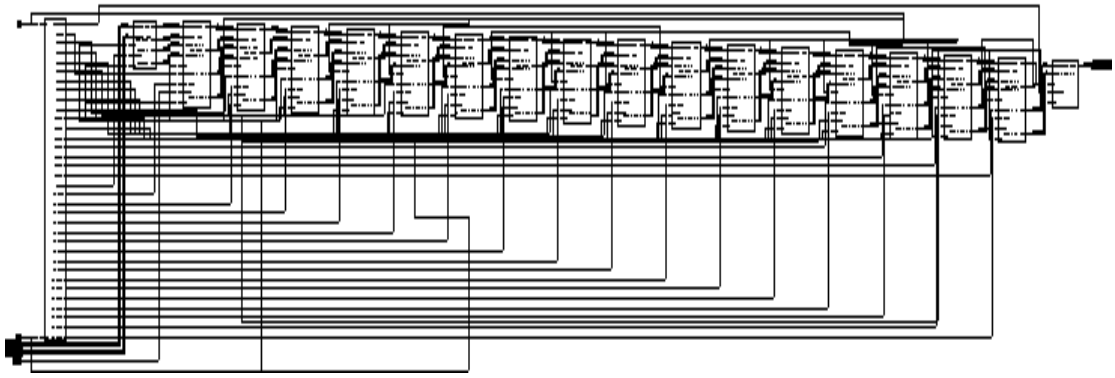
Σχήμα 5.16 P-BOX

Όλη η διαδικασία που αναλύθηκε σε αυτό το κεφάλαιο, από την είσοδο των δεδομένων και του κλειδιού μέχρι και την τελική έξοδο του γύρου επεξεργασίας φαίνεται αναλυτικά στο παρακάτω RTL διάγραμμα.



Σχήμα 5.17 RTL Διάγραμμα Ενός Γύρου DES

Τα δεδομένα πριν την τελική τους αντιμετάθεση όπως έχουμε ήδη αναφέρει υπόκεινται σε δεκαέξι γύρους επεξεργασίας. Η σύνδεση όλων των γύρων καθώς και όλη η διαδικασία κρυπτογράφησης που σχεδιάστηκε (Σχήμα 5.2) παρουσιάζεται υλοποιημένη στο παρακάτω RTL σχηματικό διάγραμμα.



Σχήμα 5.18 Συνολική Λειτουργία του DES

5.1.2 Επεξεργασία Κλειδιού DES ενός κύκλου (One round Key function)

Η επεξεργασία του απαραίτητου για την κρυπτογράφηση κλειδιού και η παραγωγή των αντίστοιχων για κάθε γύρο υποκλειδιών, σε πληθώρα σχεδιάσεων-υλοποιήσεων του αλγορίθμου DES, πραγματοποιείται κατά κόρον συνολικά. Πιο συγκεκριμένα στην αρχή παράγονται και τα 16 υποκλειδιά τα οποία ουσιαστικά αποθηκεύονται σε ένα καταχωρητή. Έτσι, στην συνέχεια ανάλογα με τον γύρο επεξεργασίας που βρίσκεται ο αλγόριθμος γίνεται η εφαρμογή του αντίστοιχου κλειδιού. Η ίδια διαδικασία ακολουθείται και κατά την διάρκεια της αποκρυπτογράφησης.

Στην παρούσα υλοποίηση λόγω της αρχιτεκτονικής συνεχόμενης ροής που ακολουθήθηκε κατά την σχεδίαση, όλη η παραπάνω διαδικασία θα αποτελούσε ουσιαστικά μία αρκετά μεγάλη

χρόνο-καθυστέρηση στην ταχύτητα επεξεργασίας των δεδομένων. Αυτό συμβαίνει γιατί τα δεδομένα, μέχρι να παραχθούν τα αντίστοιχα υποκλειδιά, θα έπρεπε να τηρούν στάση αναμονής, γεγονός που θα επιβάρυνε την συνολική ταχύτητα του αλγορίθμου. Για τον λόγο αυτό η παραγωγή των υποκλειδιών θα έπρεπε να γίνει με άλλο τρόπο ο οποίος θα εκμεταλλεύεται την σχεδίαση και δεν θα λειτουργεί επιβαρυντικά ως προς την ταχύτητα του.

Έτσι, στην συγκεκριμένη σχεδίαση η παραγωγή του κάθε υποκλειδιού γίνεται στον συγκεκριμένο γύρο επεξεργασίας των δεδομένων όπου και θα χρησιμοποιηθεί προκειμένου να έχουμε την ελάχιστη δυνατή χρονική καθυστέρηση.

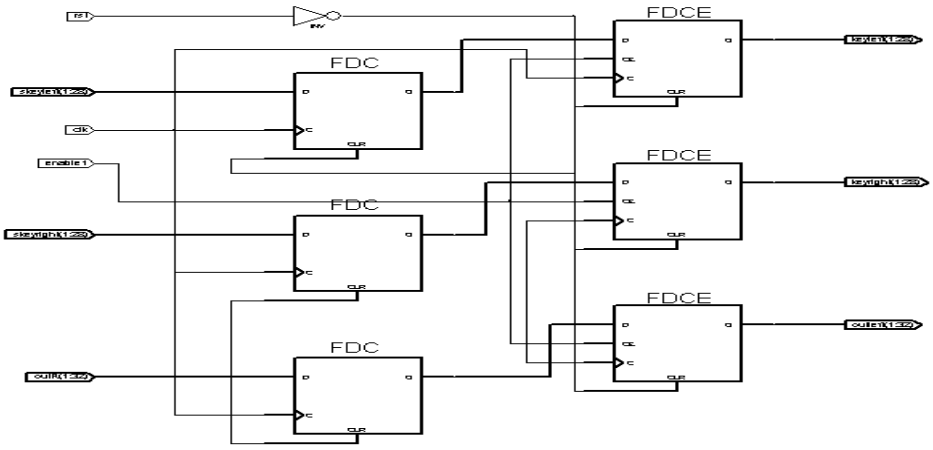
Όπως αναφέραμε παραπάνω, κατά το πρώτο μέρος της σχεδίασης (Part1) γίνεται η αρχική αντιμετάθεση των δεδομένων και του κλειδιού. Από τις εξόδους του καταχωρητή PC1 έχουμε το κλειδί μας μειωμένο από τα 64bits στα 56bits και χωρισμένο σε δύο μέρη των 28bits. Στην συνέχεια αυτά τα δύο μέρη οδηγούνται στο δεύτερο μέρος της σχεδίασης (Part2) όπου και εισέρχονται σε ένα καταχωρητή ολίσθησης, ο οποίος ανάλογα με τον γύρο επεξεργασίας που βρισκόμαστε κάνει ένα συγκεκριμένο αριθμό αριστερών ή δεξιών ολισθήσεων όταν έχουμε κρυπτογράφηση ή αποκρυπτογράφηση αντιστοίχως. Ο αριθμός των ολισθήσεων του κάθε bit ορίζεται σύμφωνα με τον Πίνακα 4.7 (Βλέπε Κεφ.4.2.3) ενώ για το αν βρισκόμαστε σε διαδικασία κρυπτογράφησης ή αποκρυπτογράφησης χρησιμοποιείται ένα σήμα ελέγχου (*mode*).

Κατά την επόμενη θετική ακμή του ρολογιού τα ολισθημένα πλέον μέρη του κλειδιού εξέρχονται και οδηγούνται στην επόμενη βαθμίδα, η οποία είναι μία βαθμίδα αντιμετάθεσης δεδομένων (PC2-βλέπε Σχήμα 5.11). Μετά την είσοδο τους αποθηκεύονται σε μία 56bits μεταβλητή προκειμένου να υποστούν μία τελική αντιμετάθεση η οποία γίνεται σύμφωνα με τον πίνακα 4.6 (Βλέπε Κεφ.4.2.3). Η αντιμετάθεση αυτή έχει ως σκοπό να μειώσει το μήκος του κλειδιού από τα 56bits στα 48bits. Το αποτέλεσμα της αντιμετάθεσης αυτής είναι το τελικό κλειδί που θα χρησιμοποιηθεί σε αυτό τον γύρο και το οποίο θα γίνει αποκλειστική διάζευξη (XOR) όπως αναφέραμε και πιο πάνω με τα αντίστοιχα bits δεδομένων (Σχήμα5.11).

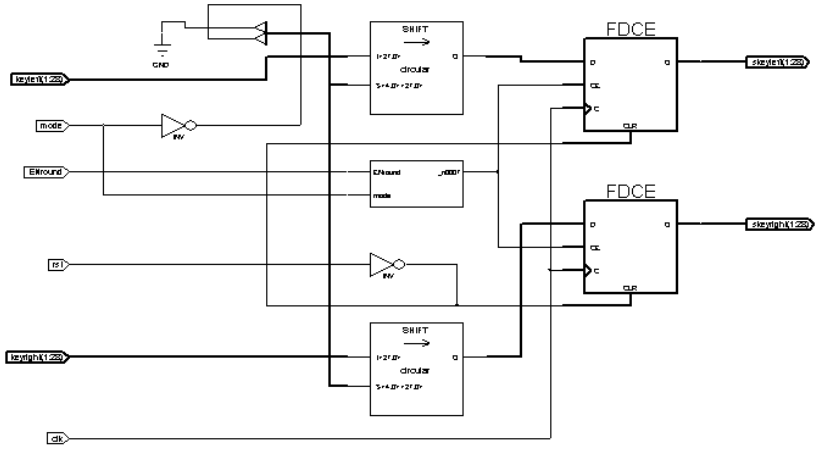
Ταυτόχρονα όμως, τα δύο μέρη του κλειδιού πριν μπουν στην βαθμίδα PC2 οδηγούνται και σε ένα καταχωρητή ο οποίος και τα αποθηκεύει (Register 88). Η λειτουργία του συγκεκριμένου καταχωρητή αναλύθηκε στο κεφάλαιο 5.2.1 ενώ το αντίστοιχο RTL σχηματικό του διάγραμμα παρουσιάζεται στο τέλος του κεφαλαίου (Σχήμα 5.19).

Η όλη διαδικασία παραγωγής του συγκεκριμένου κλειδιού επαναλαμβάνεται σε κάθε ένα γύρο ξεχωριστά και με αυτό τον τρόπο γίνεται η παραγωγή και των 16 κλειδιών.

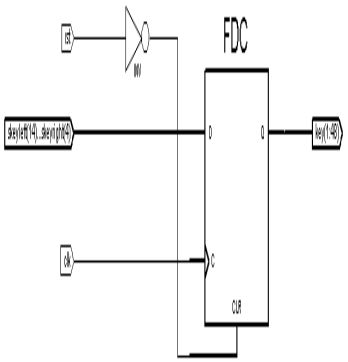
Ακολουθούν τα RTL σχηματικά διαγράμματα του καταχωρητή 88 (register 88), του καταχωρητή ολίσθησης (shift register) όσο και της βαθμίδας PC2 στα οποία φαίνεται η ακριβής τους λειτουργία.



Σχήμα 5.19 Καταχωρητής 88



Σχήμα 5.20 Καταχωρητής ολίσθησης



Σχήμα 5.21 PC2

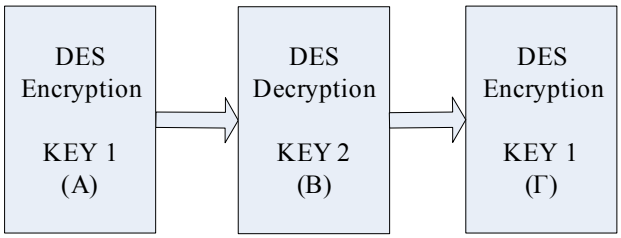
5.1.3 Triple DES

Για την μετάβαση από την υλοποίηση του DES σε αυτήν του Triple-Des ήταν αναγκαίο να γίνουν κάποιες αλλαγές τόσο στην μονάδα ελέγχου των σημάτων ενεργοποίησης όσο και στα αρχικά δεδομένα και κλειδιά τα οποία θα πρέπει να λάβουμε.

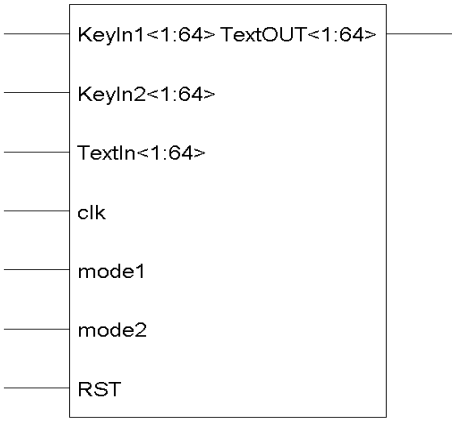
Δηλαδή, ουσιαστικά τροποποιήθηκε ο μετρητής, ο οποίος ελέγχει τα σήματα ενεργοποίησης και επεκτάθηκε έτσι ώστε η είσοδος των δεδομένων και του κλειδιού σε κάθε βαθμίδα να γίνεται σε μία καθορισμένη χρονική στιγμή και να μην έχουμε απώλεια του συγχρονισμού. Επίσης, η εφαρμογή του Triple-Des απαιτεί την είσοδο δύο διαφορετικών κλειδιών κρυπτογράφησης όπως επίσης και την δημιουργία διαφορετικού σήματος ελέγχου της διαδικασίας κρυπτογράφησης-αποκρυπτογράφησης (*mode1*) για την κάθε βαθμίδα. Όπως φαίνεται και στο ακόλουθο σχήμα, ουσιαστικά ο Triple-Des είναι ο Des υλοποιημένος τρεις συνεχόμενες φορές, με την μόνη διαφορά ότι η δεύτερη υλοποίηση του (Σχήμα 5.22, (B)) κάνει αποκρυπτογράφηση των δεδομένων εισόδου με διαφορετικό κλειδί (Key 2), με απώτερο σκοπό να αυξήσει ακόμα περισσότερο την

πολυπλοκότητα-ασφάλεια του αλγορίθμου. Ο πρώτος DES (Σχήμα 5.22, (Α)) καθώς και ο τρίτος (Σχήμα 5.22, (Γ)) λειτουργούν κανονικά κάνοντας κρυπτογράφησης με το ίδιο αρχικό κλειδί (Key 1). Στην αποκρυπτογράφηση γίνεται αλλαγή των ρόλων της κάθε DES βαθμίδας. Δηλαδή η βαθμίδα που προηγουμένως έκανε κρυπτογράφηση τώρα κάνει αποκρυπτογράφηση και αντιστρόφως.

Έτσι, όπως παρατηρούμε στο συνοπτικό σχήμα RTL υλοποίησης (Σχήμα 5.23) του Triple-Des εκτός από τα δεδομένα, το ένα κλειδί και το σήμα ελέγχου για την λειτουργία της κρυπτογράφησης-αποκρυπτογράφησης, τα οποία είχαμε στην απλή υλοποίηση του αλγορίθμου, τώρα έχουμε ένα δεύτερο κλειδί κρυπτογράφησης (*Keyin2*) και ένα νέο σήμα ελέγχου (*mode2*).

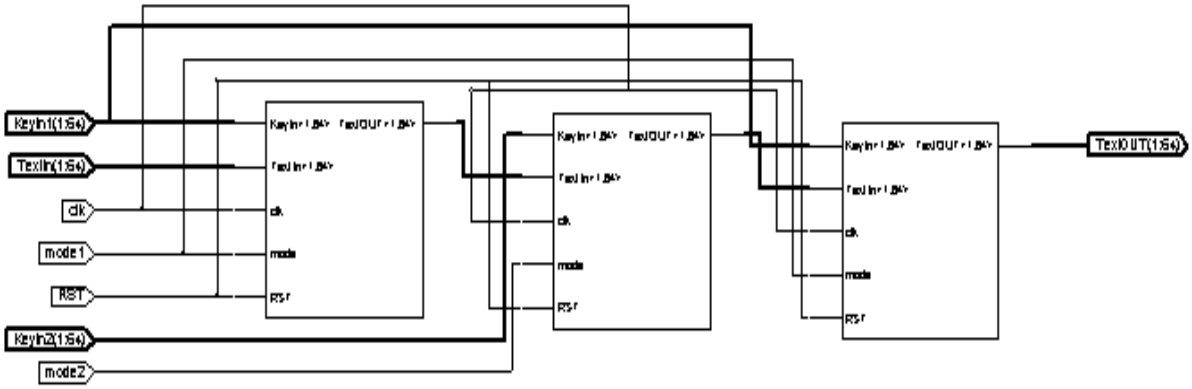


Σχήμα 5.22: Triple DES



Σχήμα 5.23: Top Triple DES RTL

Στο παρακάτω σχήμα (5.24) φαίνεται η συνολική λειτουργία του Triple DES σε επίπεδο RTL.

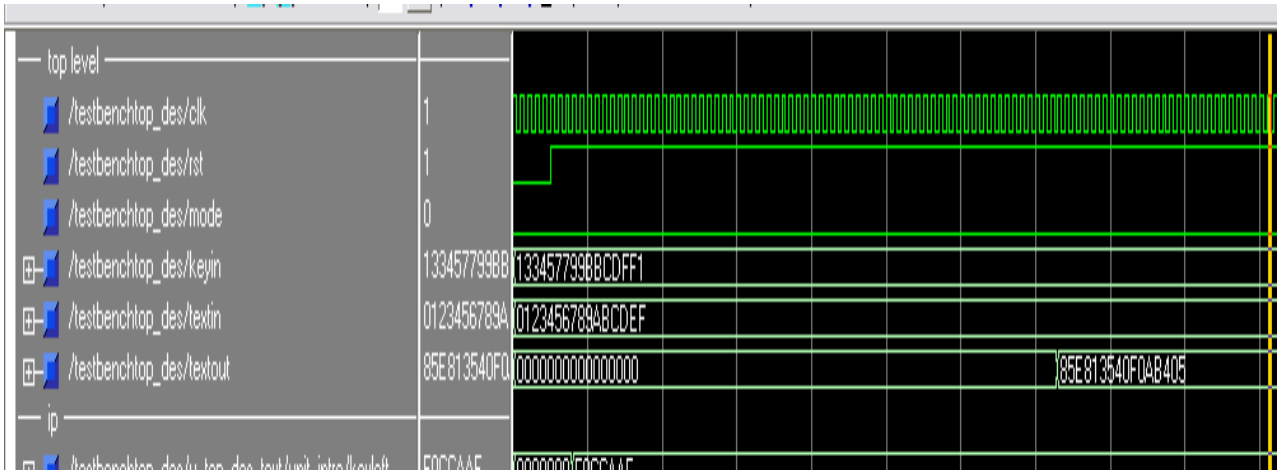


Σχήμα 5.24: RTL Triple DES

5.2 Υλοποίηση και Προσομοίωση

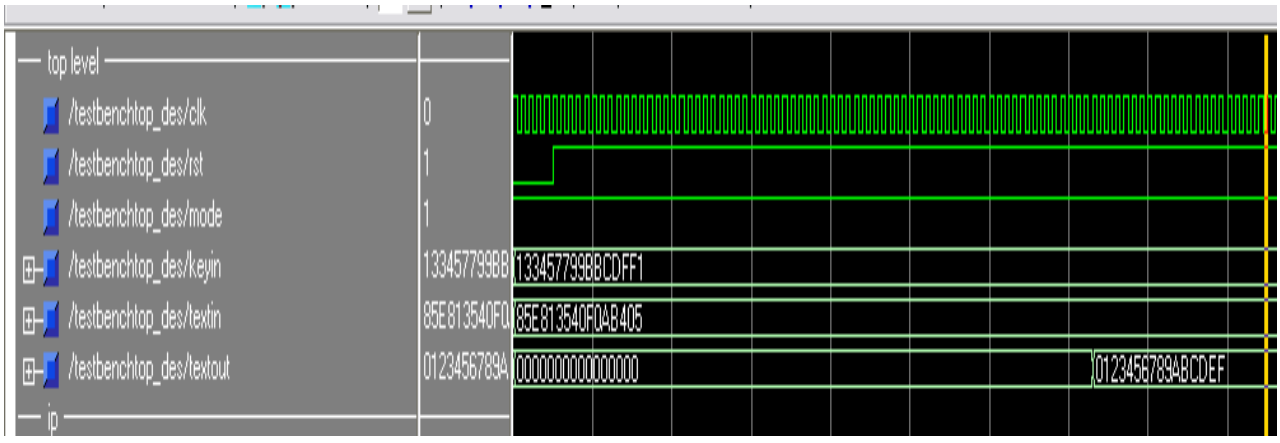
Για την εξακρίβωση της σωστής λειτουργίας του DES, στη συνέχεια παρουσιάζεται μία πλήρης διαδικασία προσομοίωσης του (Σχήμα 5.27, 5.28), που έγινε σε πραγματικό χρόνο.

Τα εισερχόμενα δεδομένα που δίνονται προς κρυπτογράφηση (σε δεκαεξαδική μορφή) είναι: Textin=«0123456789ABCDEF», ενώ το αντίστοιχο κλειδί κρυπτογράφησης είναι Keyin=«133457799BBCDFF1». Τα τελικά κρυπτογραφημένα δεδομένα που λαμβάνουμε είναι «85E813540F0AB405».



Σχήμα 5.25: Διαδικασία Κρυπτογράφησης DES

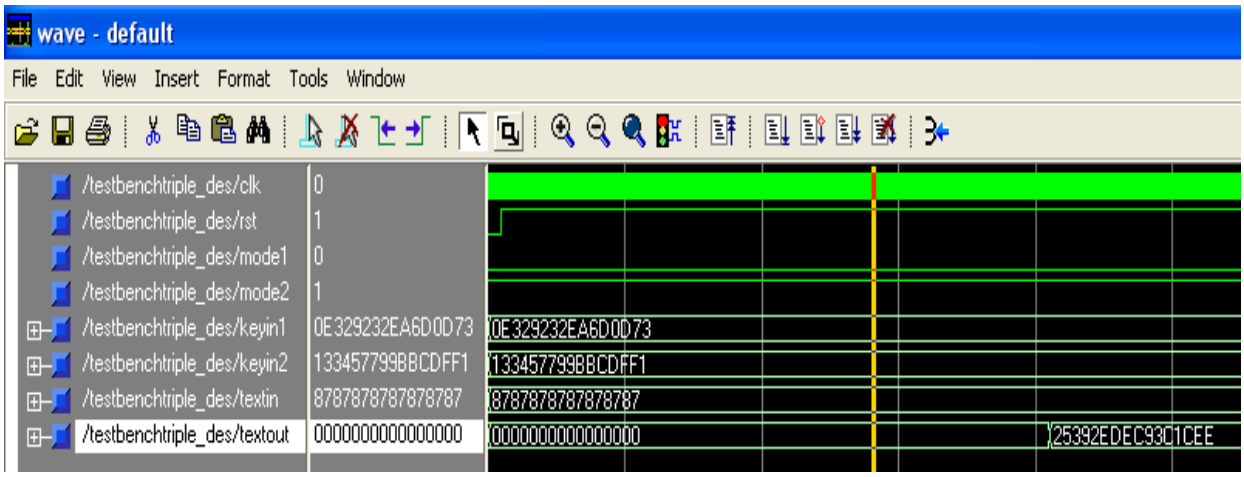
Για τον έλεγχο της διαδικασίας αποκρυπτογράφησης, δόθηκαν τα δεδομένα που λάβαμε κατά την παραπάνω διαδικασία κρυπτογράφησης. Το κλειδί, όπως φαίνεται στο Σχήμα 5.28 είναι ακριβώς το ίδιο με προηγουμένως ενώ ορίσαμε την λειτουργία του προγράμματος σε κατάσταση αποκωδικοποίησης αλλάζοντας το σήμα ελέγχου mode. Αποτέλεσμα της διαδικασίας αποκρυπτογράφησης είναι τα αρχικά δεδομένα που είχαμε εισάγει («0123456789ABCDEF»).



Σχήμα 5.26: Decryption DES

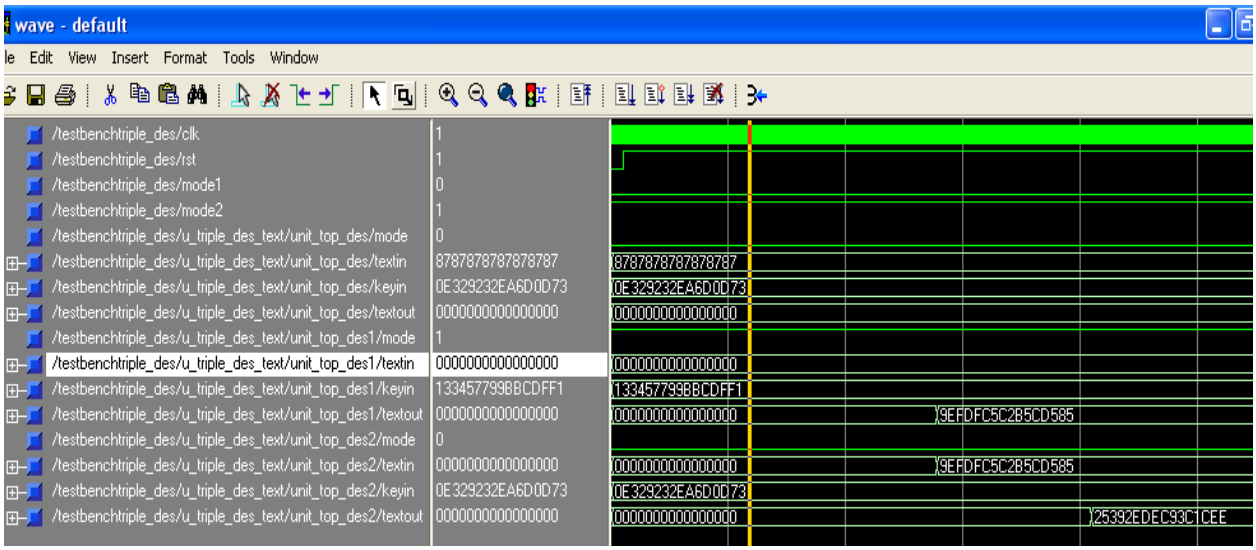
Η ίδια διαδικασία ακολουθήθηκε και για τον έλεγχο της υλοποίησης του Triple-DES. Ακολουθούν τα σχήματα στα οποία φαίνεται αναλυτικά η διαδικασία προσομοίωσης (Σχήμα 5.27, 5.28, 5.29) του Triple DES σε πραγματικό χρόνο.

Τα εισερχόμενα δεδομένα που δίνονται προς κρυπτογράφηση (σε δεκαεξαδική μορφή) είναι: Textin=«8787878787878787», ενώ τα αντίστοιχα κλειδιά κρυπτογράφησης Keyin1=«0E329232EA60D73» και Keyin2=«133457799BBCDFF1». Τα τελικά κρυπτογραφημένα δεδομένα που λαμβάνουμε είναι «25392EDEC93C1CEE».



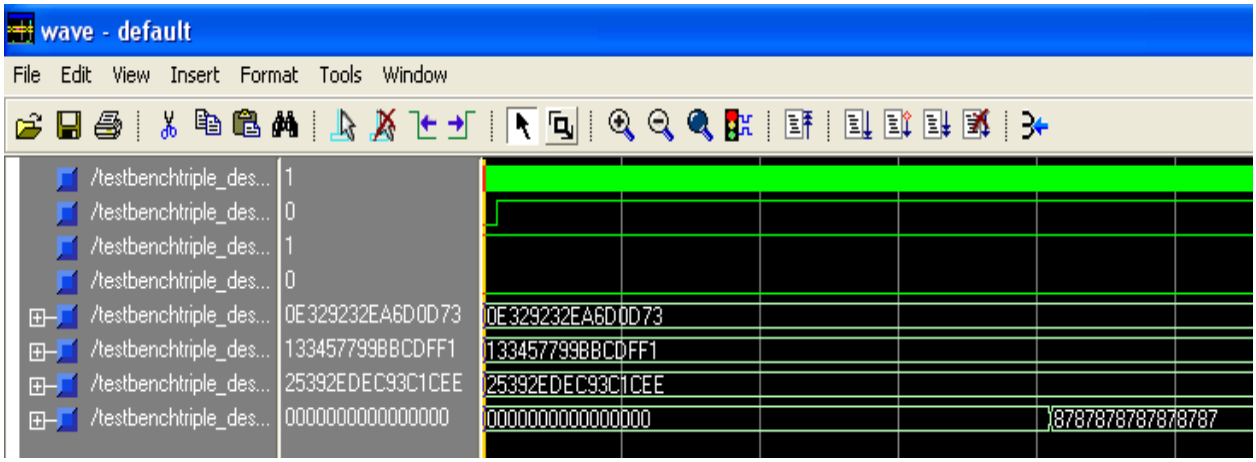
Σχήμα 5.27: Διαδικασία Κρυπτογράφησης Triple DES

Η αναλυτική διαδικασία και των τριών υποβαθμίδων DES του Triple DES (Σχήμα 5.22) κατά την διαδικασία κρυπτογράφησης που κάναμε παραπάνω παρουσιάζεται στο ακόλουθο σχήμα.



Σχήμα 5.28: Αναλυτική Διαδικασία Κρυπτογράφησης Triple DES

Για τον έλεγχο της διαδικασίας αποκρυπτογράφησης, δόθηκαν τα δεδομένα που λάβαμε κατά την παραπάνω διαδικασία κρυπτογράφησης. Τα κλειδιά, όπως φαίνεται στο Σχήμα 5.32 είναι ακριβώς τα ίδια με προηγουμένως ενώ ορίσαμε την λειτουργία του προγράμματος σε κατάσταση αποκωδικοποίησης αλλάζοντας τα απαραίτητα σήματα ελέγχου mode1 και mode2. Αποτέλεσμα της διαδικασίας αποκρυπτογράφησης ήταν τα αρχικά δεδομένα που είχαμε εισάγει («8787878787878787»).



Σχήμα 5.29: Διαδικασία Αποκρυπτογράφησης Triple DES

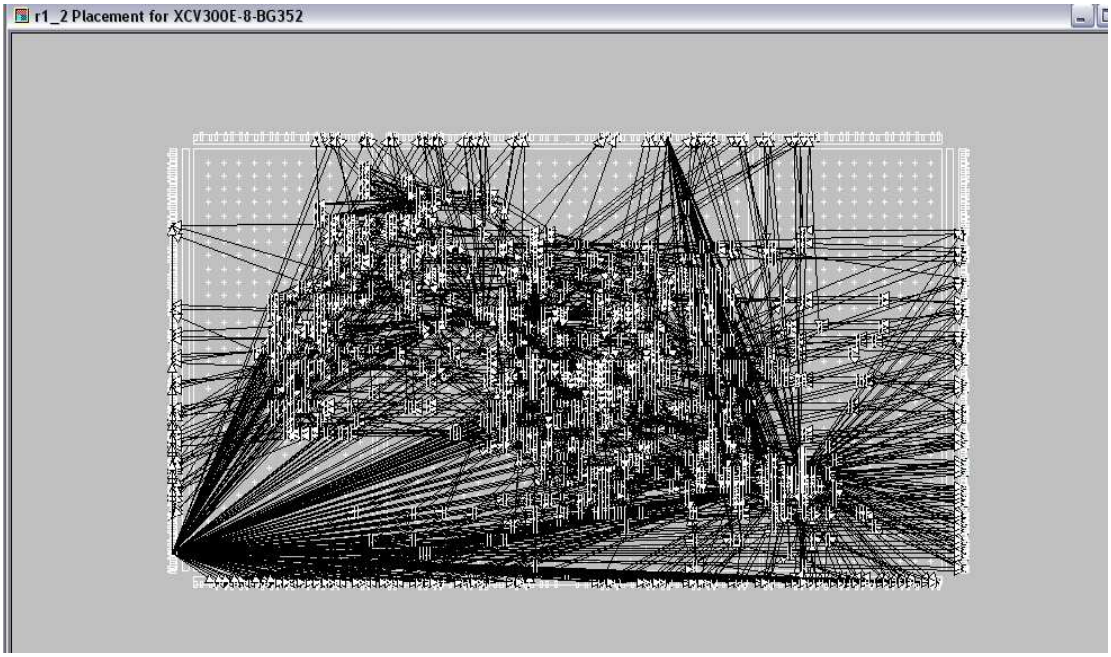
Στον πίνακα (Πίνακας 5.1) παρουσιάζονται περιληπτικά τα στατιστικά στοιχεία της υλοποίησης του DES για το Chip XCV300ebg352-8 C (device family Virtex E, device type 300E, Package type BG, number of pins 352, Speed grade -8, Temperature range C).

Number of Slices:	670 out of 3072 21%
Number of Slice Flip Flops:	1000 out of 6144 16%
Number of 4 input LUTs:	642 out of 6144 10%
Number of bonded IOBs:	186 out of 264 70%
Number of GCLKs:	1 out of 4 25%
Minimum period	6.129ns (Maximum Frequency: 163.159MHz)
Minimum input arrival time before clock	8.236ns
Maximum output required time after clock	6.917ns

Πίνακας 5.1 Ποσοστό χρήσης των πόρων της FPGA που χρησιμοποιήθηκε

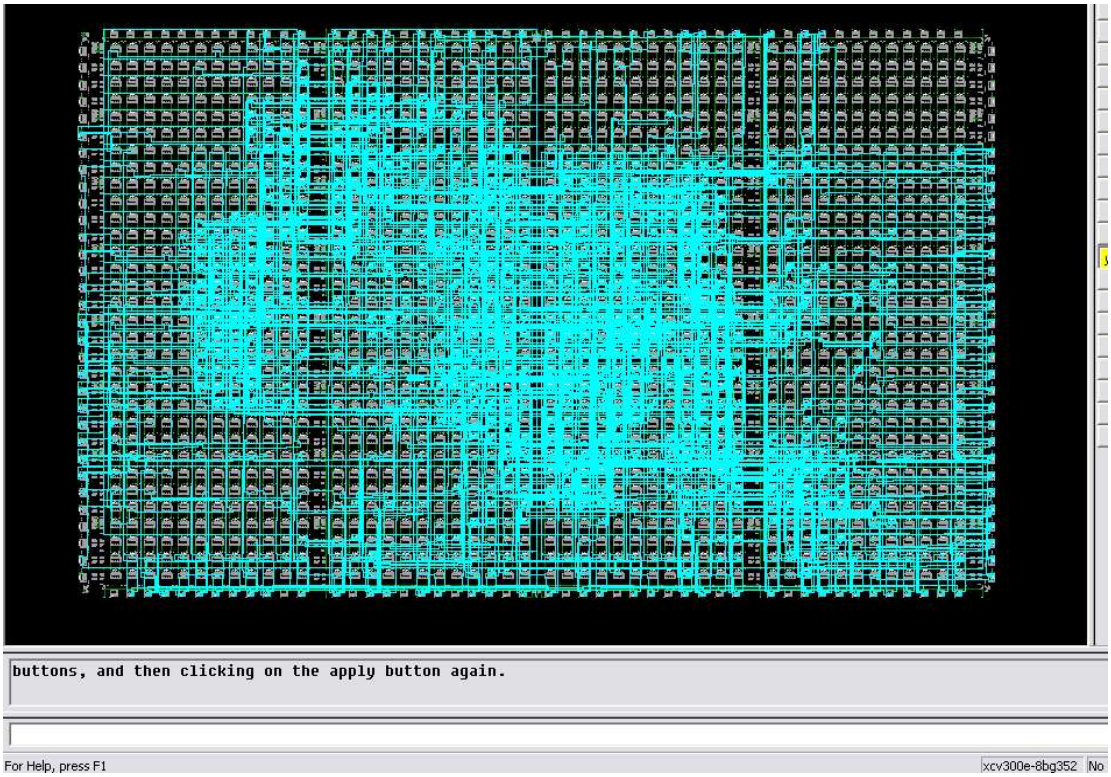
Όπως φαίνεται, το ποσοστό κάλυψης του αριθμού των Flip Flops του συγκριμένου Chip βρίσκεται στο 16%, χρησιμοποιήθηκαν 186 pins από τα συνολικά 264 δηλαδή το 70%, ένα από τα τέσσερα συνολικά pins ρολογιού του FPGA (GCLKs), 670 από τα 3072 (21%) CLBs λογικές προγραμματιζόμενες κυψέλες (Slices), 642 από τους 6144 πίνακες αναζήτησης (LUBs). Η μέγιστη λειτουργία που μπορεί να λειτουργήσει το συγκεκριμένο Chip είναι 163.159MHz, ο ελάχιστος χρόνος άφιξης δεδομένων πριν το ρολόι είναι 8.236ns, ενώ τέλος 6.917ns είναι ο μέγιστος χρόνος που απαιτείται μετά την ακμή ρολογιού για να έχουμε έξοδο.

Παρακάτω ακολουθούν τα σχήματα 5.30- 5.31 από το αναπτυξιακό λογισμικό της Xilinx του υλοποιημένου DES Chip.



Σχήμα 5.30 Απεικόνιση χώρου Chip XCV1600ebg560-8 C

Στο παραπάνω σχήμα φαίνεται η κατανομή όλων των ψηφιακών στοιχειών και συνδέσεων επάνω στο FPGA Chip ενώ στο ακόλουθο σχήμα μπορούμε να δούμε την συνολική τοποθέτηση των στοιχείων σε όλα τα επίπεδα του FPGA chip καθώς και τις διαδρομές των συνδέσεων αυτών.



Σχήμα 5.31 Απεικόνιση δρομολογήσεων Chip XCV300ebg352-8 C

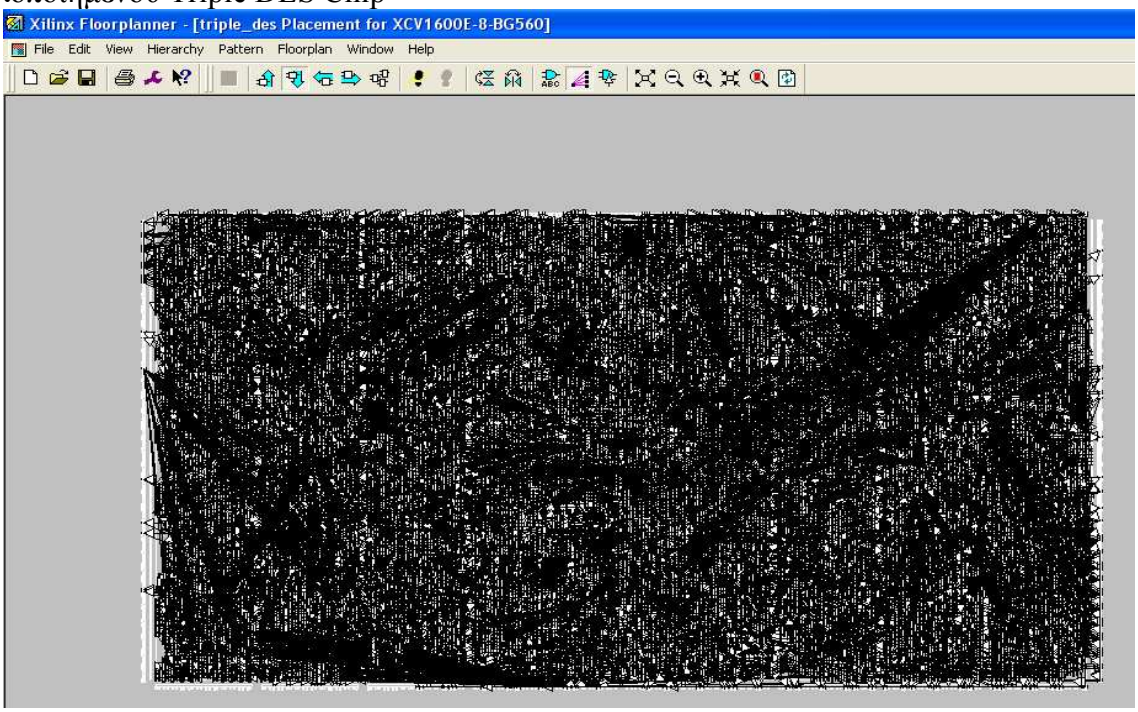
Όμοια, για την υλοποίηση του Triple DES (Πίνακας 5.2), παρουσιάζονται περιληπτικά τα στατιστικά στοιχεία για το Chip XCV1600ebg560-8 C (device family Virtex E, device type 1600E, Package type BG, number of pins 560, Speed grade -8, Temperature range C).

Number of Slices:	12635 out of 14039 90%
Number of Slice Flip Flops:	20505 out of 31104 65%
Number of 4 input LUTs:	15518 out of 31104 49%
Number of bonded IOBs:	243 out of 408 59%
Number of GCLKs:	1 out of 4 25%
Minimum period	6.564ns (Maximum Frequency: 152.346MHz)
Minimum input arrival time before clock	10.116ns
Maximum output required time after clock	5.783ns

Πίνακας 5.2 Ποσοστό χρήσης των πόρων της FPGA που χρησιμοποιήθηκε

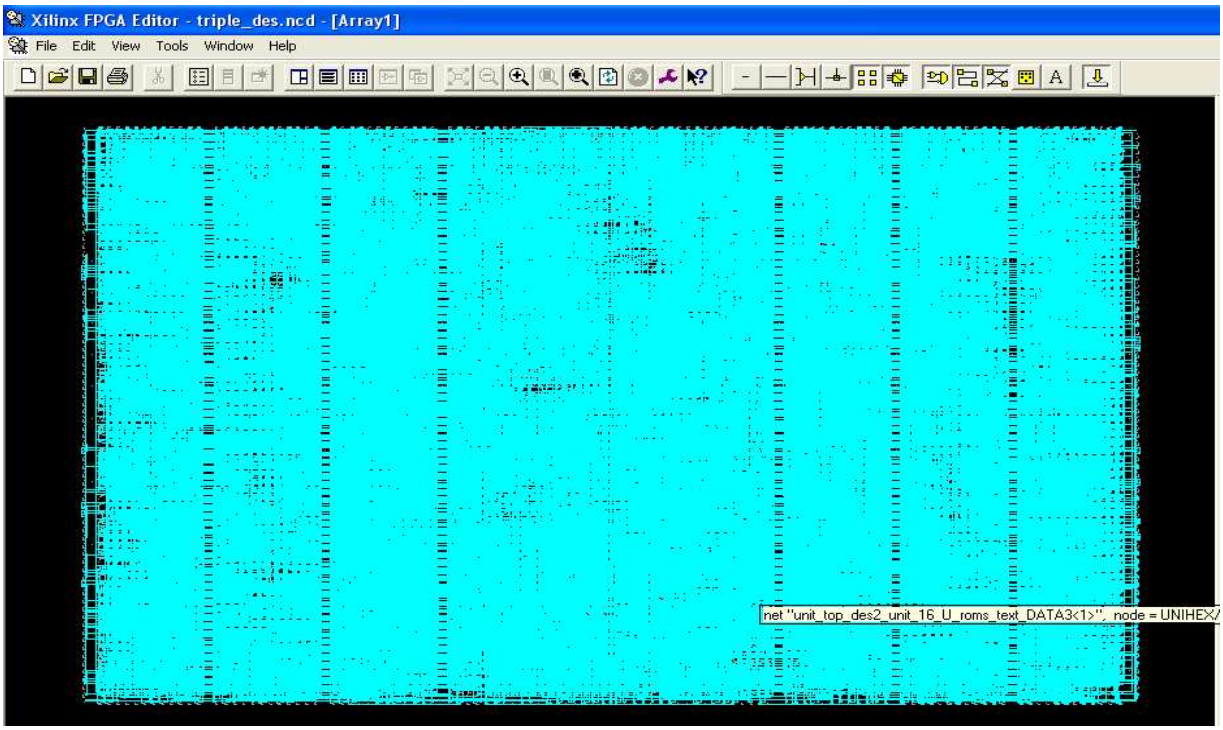
Η κάλυψη των Flip Flops του συγκριμένου Chip βρίσκεται στο 65%, χρησιμοποιήθηκαν 243 pins από τα συνολικά 408 δηλαδή το 59%, ένα από τα τέσσερα συνολικά pins ρολογιού του FPGA (GCLKs), 12635 από τα 14039 (90%) CLBs λογικές προγραμματιζόμενες κυψέλες (Slices), 15518 από τους 31104 πίνακες αναζήτησης (LUTs) δηλαδή 49%. Η μέγιστη λειτουργία που μπορεί να λειτουργήσει το συγκεκριμένο Chip είναι 152.346MHz, ο ελάχιστος χρόνος άφιξης δεδομένων πριν την ακμή του ρολογιού είναι 10.116ns, ενώ τέλος 5.783ns είναι ο μέγιστος χρόνος που απαιτείται μετά την ακμή ρολογιού για να έχουμε έξοδο.

Παρακάτω ακολουθούν τα σχήματα 5.29- 5.30 από το αναπτυξιακό λογισμικό της Xilinx του υλοποιημένου Triple DES Chip



Σχήμα 5.32 Απεικόνιση χώρου Chip XCV1600ebg560-8 C

Στο παραπάνω σχήμα φαίνεται η κατανομή όλων των ψηφιακών στοιχειών και συνδέσεων επάνω στο FPGA Chip ενώ στο ακόλουθο σχήμα μπορούμε να δούμε την συνολική τοποθέτηση των στοιχείων σε όλα τα επίπεδα του FPGA chip καθώς και τις διαδρομές που ακολουθούν όλες οι συνδέσεις.



Σχήμα 5.33 Απεικόνιση δρομολογήσεων Chip XCV1600ebg560-8 C

Με βάση την ελάχιστη περίοδο λειτουργίας μπορούμε να κάνουμε μια εκτίμηση των επιδόσεων του κυκλώματος που σχεδιάσαμε για τον αλγόριθμο DES. Έτσι, αν θεωρήσουμε μέγιστη συχνότητα λειτουργίας τα 150MHz αυτό σημαίνει ότι θα έχουμε ελάχιστη περίοδο ρολογιού 6,6ns. Γνωρίζοντας ότι για να κρυπτογραφήσουμε ή αποκρυπτογραφήσουμε μια ομάδα μεγέθους 64-bits χρειαζόμαστε ένα παλμό στην αρχή, δεκαέξι γύρους των τεσσάρων παλμών ο καθένας και ένα παλμό στο τέλος, δηλαδή συνολικά 66 παλμούς. Άρα, για κάθε ομάδα 64-bits (8bytes) απαιτούνται 435,6ns. Αυτό σημαίνει ότι χρειάζονται λιγότερα από 60ms για την κρυπτογράφηση ή αποκρυπτογράφηση πληροφορίας μεγέθους 1MByte, δηλαδή έχουμε ρυθμό περίπου 18Mbytes/sec.

Οι επιδόσεις του κυκλώματος για των Triple-DES πέφτουν στο 1/3 διότι χρειάζεται τους τριπλάσιους παλμούς αφού υλοποιεί τον DES τρεις φορές. Εύκολα όμως μπορεί κάποιος να βελτιώσει τον ρυθμό κρυπτογράφησης-αποκρυπτογράφησης χρησιμοποιώντας τεχνικές σχεδίασης συνεχόμενης ροής (pipeline).

6 Συμπεράσματα

Στη παρούσα εργασία παρουσιάστηκαν αναλυτικά ένα προς ένα τα βήματα για την υλοποίηση του αλγορίθμου κρυπτογράφησης DES και Triple-DES σε FPGA Chip καθώς επίσης και τα πιο σημαντικά στοιχεία από τη θεωρία της κρυπτογραφίας και κρυπτανάλυσης.

Συγκεκριμένα, αρχικά συντέθηκε μια σύντομη περιγραφή της ιστορίας της κρυπτογραφίας από το 1900 π.Χ. μέχρι και σήμερα. Στην συνέχεια γίνεται αναφορά στις βασικές έννοιες της κρυπτογραφίας-κρυπτανάλυσης, ενώ στο τέλος μελετήθηκαν σε βάθος οι αλγόριθμοι κρυπτογράφησης DES και Triple-DES και υλοποιήθηκαν σε επίπεδο υλικού (Hardware). Προσομοιώθηκε επιτυχώς η λειτουργία του κυκλώματος και εκτιμήσαμε ότι η ταχύτητα λειτουργίας του DES, που είναι 18Mbytes/sec, είναι περίπου δέκα φορές μεγαλύτερη από αυτήν που μπορεί να επιτευχθεί επιστρατεύοντας μεθόδους software και σύγχρονους υπολογιστές (π.χ. με επεξεργαστή AMD Athlon 64 3400+ και με 512MB RAM)[XPI05].

Συμπερασματικά, η παρούσα εργασία με τη βοήθεια και των προσομοιώσεων απέδειξε ότι μπορεί να λειτουργήσει αξιόπιστα, πληρώντας όλες τις αρχικές προδιαγραφές που είχαμε θέσει. Συνεπώς μπορεί να εφαρμοστεί και στην πράξη με τις απαραίτητες μετατροπές (πχ κατάλληλο Interface) σε πληθώρα εφαρμογών, όπως ασφαλή και γρήγορη μεταφορά αρχείων κειμένου με σχετικά μικρό κλειδί (64 bits Key). Επιπλέον ένα από τα προτερήματα της παρούσας υλοποίησης που θα πρέπει να ληφθεί υπόψη στην εφαρμογή της, είναι η υλοποίηση σε Hardware (γρηγορότερη και ασφαλέστερη επεξεργασία δεδομένων) καθώς επίσης και της συμμετρικότητας του αλγορίθμου (γρήγορη κωδικοποίηση- αποκωδικοποίηση με το ίδιο Hardware). Σε αυτό το σημείο, θα πρέπει να αναφερθεί και η αδυναμία πλέον του αλγορίθμου DES /Triple-DES των 64bits στην κρυπτανάλυση (λόγω της αυξανόμενης υπολογιστικής ισχύς και της ακριβής γνώσης της μεθόδου λειτουργίας του). Παρά ταύτα ο Triple DES εξακολουθεί να χρησιμοποιείται σε πολλές εφαρμογές κυρίως όμως στα128bits.

Οι ανάγκες της σημερινής εποχής έχουν οδηγήσει σε νέα είδη κρυπτογράφησης όπως η ελλειπτική και η κβαντική κρυπτογραφία. Επίσης, τα τελευταία χρόνια λόγω της ευρείας χρήσης του Διαδικτύου και του Παγκόσμιου Ιστού, η κρυπτογράφηση βασίζεται πλέον σε τεχνικές που συνδυάζουν συμμετρικούς και ασύμμετρους αλγορίθμους κρυπτογράφησης για την φύλαξη ή την μετάδοση της πληροφορίας.

Κάθε μία από τις παραπάνω κρυπτογραφήσεις έχει τα δικά της προτερήματα και μειονεκτήματα. Κατά την γνώμη μου, μια πιθανή λύση για την κατασκευή μίας κατηγορίας ισχυρών τεχνικών κρυπτογράφησης έγκειται στην μετάδοση μόνο της κρυπτογραφημένης

πληροφορίας, αποτρέποντας την μετάδοση του κλειδιού. Τέτοια συστήματα μπορούν να δημιουργηθούν χρησιμοποιώντας συγχρονισμένες γεννήτριες κλειδιών.

Τέλος, θα μπορούσαμε να πούμε ότι, η κρυπτογραφία γενικότερα θα αποτελέσει ένα από τα μεγαλύτερα ζητήματα συζητήσεων και εφαρμογών σε παγκόσμιο επίπεδο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

- **Βιβλία-Άρθρα**

[DKA67] **D. KAHN, "The Codebreakers"**, Macmillan Publishing Company, New York, 1967**.**

[MEN96] **A. Menezes**, **P. van Oorschot**, and **S.Vanstone, "Handbook of Applied Cryptography"**, CRC Press, 1996.

[SCH96] **Bruce Schneier,"*Applied Cryptography Protocols, Algorithms, and Source Code in C"*,**Second Edition, John Wiley & Sons, 1996.

[ΚΓΣ03] **Β.Α. Κάτος - Γ.Χ. Στεφανίδης**, *Τεχνικές Κρυπτογραφίας & Κρυπτανάλυσης*, ΖΥΓΟΣ, 396, 2003.

[XPI05] Χριστόφιλος Κ., «*Μελέτη Αλγορίθμων Κρυπτογράφησης/Αποκρυπτογράφησης και Υλοποίηση του Αλγορίθμου DES μέσω λογισμικού», Πτυχιακή Εργασία Τμήματος Ηλεκτρονικής του Τ.Ε.Ι Κρήτης, Χανιά 2005*.

[AST03] **Andrew S. Tanenbaum, "Computer Networks",** Ελληνικός Τίτλος:**"Δίκτυα Υπολογιστών",** Μετάφραση: Γ.Ξηλωμένος, Εκδόσεις Κλειδαριθμος 2003.

[TEX05] **Τεχνική Εκλογή, "Αφιέρωμα στην Κρυπτογραφία",** (Φεβρουάριος, Μάρτιος, Απρίλιος, Μάιος) 2005.

[PAS90] **Peter J. Ashenden, "VHDL Cookbook",** First Edition, July, 1990.

[ARU98] **Andrew Rushton, "VHDL for Logic Synthesis"** John Wiley & Sons, Second edition 1998.

[IFBAN] **Implementation of an FPGA Based Accelerator for Virtual Private Networks,** O.Y.H. Cheung, P.H.W. Leong Department of Computer Science and Engineering .The Chinese University of Hong Kong Shatin, NT Hong Kong.

[ADH00] **Amit Dhir, "Data Encryption using DES/Triple-DES Functionality in Spartan-II FPGAs",** WP115 (v1.0) March 9, 2000.

[SLI03] **MSc CompSci Sebastian Link, "Paper Script Information Systems Security 157.738"**, Massey University Department of Information Systems, New Zealand, Version March 24, 2003.

[CDK85] **Cipher A. Deavours and Louis Kruh, "Machine Cryptography and Modern Cryptanalysis"**, Artech House, 1985.

[WDH76] **Whitfield Diffie and Martin Hellman, "New Directions in Cryptography",** IEEE Transactions on Information Theory, Nov 1976.

[STI02] **D. Stinson**.**"Cryptography: Theory and Practice",** 2nd Edition, Chapman and Hall/CRC, 2002

[HFIBM] **Horst Feistel, "Cryptographic Coding for Data-Bank Privacy",** IBM Research Report RC2827.

[SIG95] **Simson Garfinkel, "PGP: Pretty Good Privacy",**O'Reilly & Associates, Inc., 1995.

[RSA78] **Rivest, Shamir and Adleman, "A method for obtaining digital signatures and public key cryptosystems"**, Communications of the ACM, Feb. 1978

[BCO2E] **Ben Cohen**. **"VHDL- Coding Styles And Methodologies",** 2th Edition

[ALDEW] **Allen Dewey**. **"Analysis And Design Of Digital Systems With VHDL".**

- **Διαδικτυακοί Τόποι**

http://www.shmoo.com/crypto/Cracking_DES/cracking-des.htm

Cracking DES: Secrets of Encryption Research, Wiretap Politics, and Chip Design, by the Electronic Frontier Foundation Distributed by O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol, CA 95472 May 1998: First Edition. ISBN: 1-56592-520-3

http://www.itl.nist.gov/fipspubs/

Federal Information Processing Standards Publications (FIPS PUBS) FIPS Home Page

http://www-cse.ucsd.edu/users/mihir/cse207/

Introduction to Modern Cryptography

Ronald L. Rivest, "The RC5 Encryption Algorithm", κείμενο το οποίο έγινε διαθέσιμο μέσω του διαδικτύου το 1994.

Παραρτήματα

Α. Αποσπάσματα Κώδικα VHDL

Στο παρόν παράρτημα παρατίθεται μέρος του κώδικα που δημιουργήθηκε.

Triple_DES.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_misc.all;
entity triple_des is
  port(      TextIn:in Std_logic_vector(1 to 64);
             KeyIn1:in Std_logic_vector(1 to 64);
             KeyIn2:in Std_logic_vector(1 to 64);
             clk :in STD_LOGIC;
             RST: in STD_LOGIC;
             mode1 :in STD_LOGIC;
             mode2 :in STD_LOGIC;
              TextOUT:out Std_logic_vector(1 to 64)
             );
end;

architecture BEHAVIOR of triple_des is

component top_des is

  port(       TextIn:in Std_logic_vector(1 to 64);
              KeyIn:in Std_logic_vector(1 to 64);
             clk :in STD_LOGIC;
             RST: in STD_LOGIC;
             mode :in STD_LOGIC;
              TextOUT:out Std_logic_vector(1 to 64)
             );
end component;

component top_des1 is

  port(      TextIn:in Std_logic_vector(1 to 64);
             KeyIn:in Std_logic_vector(1 to 64);
             clk :in STD_LOGIC;
             RST: in STD_LOGIC;
             mode :in STD_LOGIC;
              TextOUT:out Std_logic_vector(1 to 64)
             );
end component;

component top_des2 is

  port(       TextIn:in Std_logic_vector(1 to 64);
              KeyIn:in Std_logic_vector(1 to 64);
             clk :in STD_LOGIC;
             RST: in STD_LOGIC;
             mode :in STD_LOGIC;
              TextOUT:out Std_logic_vector(1 to 64)
             );
end component;
signal TextOUT1,TextOUT2 :Std_logic_vector(1 to 64);
```

```
begin
-----------------------------------------
unit_top_des : top_des
  port map(
             TextIn=>TextIn,
             KeyIn=>KeyIn1,
             clk=>clk,
             RST=>RST,
             mode=>mode1,
             TextOUT=>TextOUT1
             );

unit_top_des1 : top_des1
  port map(
             TextIn=>TextOUT1,
             KeyIn=>KeyIn2,
             clk=>clk,
             RST=>RST,
             mode=>mode2,
             TextOUT=>TextOUT2
             );

unit_top_des2 : top_des2
  port map(
             TextIn=>TextOUT2,
              KeyIn=>KeyIn1,
             clk=>clk,
             RST=>RST,
             mode=>mode1,
              TextOUT=>TextOUT
             );
end BEHAVIOR;
```

Topdes.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_misc.all;

entity top_des is
  port(       TextIn:in Std_logic_vector(1 to 64);
              KeyIn:in Std_logic_vector(1 to 64);
             clk :in STD_LOGIC;
             RST: in STD_LOGIC;
             mode :in STD_LOGIC;
              TextOUT:out Std_logic_vector(1 to 64)
             );
end;

architecture BEHAVIOR of top_des is

component Control is
  port(
             clk :in STD_LOGIC;
             rst: in STD_LOGIC;

ENround1,ENround2,ENround3,ENround4,ENround5,
ENround6,ENround7,ENround8,ENround9,ENround10,E
Nround11,ENround12,ENround13,ENround14,
ENround15,ENround16 : out STD_LOGIC;

enable1,enable2,enable3,enable4,enable5,enable6,enable7,
enable8,enable9,enable10,enable11,enable12,enable13,ena
ble14,enable15,enable16,enableout: out STD_LOGIC;
     encodin: out STD_LOGIC
             );
end component;

component intro is
  port(
              TextIn:in Std_logic_vector(1 to 64);
              KeyIn:in Std_logic_vector(1 to 64);
              clk :in STD_LOGIC;
              RST: in STD_LOGIC;
              encodin:in std_logic;
            keyleft: out STD_LOGIC_VECTOR (1 to 28);
            keyright: out STD_LOGIC_VECTOR (1 to 28);
            TextInL: out STD_LOGIC_VECTOR (1 to 32);
            TextInR: out STD_LOGIC_VECTOR (1 to 32)
              );
end component;

component round1 is
   port(
             TextInL:in Std_logic_vector(0 to 31);
             TextInR:in Std_logic_vector(0 to 31);
             KeyInL:in Std_logic_vector(0 to 27);
             KeyInR:in Std_logic_vector(0 to 27);
```

```
  clk :in STD_LOGIC;
              RST: in STD_LOGIC;
              ENround:in STD_LOGIC;
              enable1:in STD_LOGIC;
              mode: in STD_LOGIC;
      keyleft:out Std_logic_vector(0 to 27);
              keyright:out Std_logic_vector(0 to 27);
              TextOUTL:out Std_logic_vector(0 to 31);
              TextOUTR:out Std_logic_vector(0 to 31)
              );
end component;

component round2 is
  port(       TextInL:in Std_logic_vector(0 to 31);
              TextInR:in Std_logic_vector(0 to 31);
               KeyInL:in Std_logic_vector(0 to 27);
              KeyInR:in Std_logic_vector(0 to 27);
              clk :in STD_LOGIC;
              RST: in STD_LOGIC;
              enable1:in STD_LOGIC;
              ENround:in STD_LOGIC;
              mode: in STD_LOGIC;
             keyleft:out Std_logic_vector(0 to 27);
              keyright:out Std_logic_vector(0 to 27);
              TextOUTL:out Std_logic_vector(0 to 31);
              TextOUTR:out Std_logic_vector(0 to 31)
              );
end component;

component round3 is
  port(       TextInL:in Std_logic_vector(0 to 31);
             TextInR:in Std_logic_vector(0 to 31);
              KeyInL:in Std_logic_vector(0 to 27);
              KeyInR:in Std_logic_vector(0 to 27);
              clk :in STD_LOGIC;
              RST: in STD_LOGIC;
              enable1:in STD_LOGIC;
              ENround:in STD_LOGIC;
              mode: in STD_LOGIC;
             keyleft:out Std_logic_vector(0 to 27);
              keyright:out Std_logic_vector(0 to 27);
              TextOUTL:out Std_logic_vector(0 to 31);
              TextOUTR:out Std_logic_vector(0 to 31)
              );
end component;
component round4 is
  port(      TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
             KeyInL:in Std_logic_vector(0 to 27);
              KeyInR:in Std_logic_vector(0 to 27);
              clk :in STD_LOGIC;
              RST: in STD_LOGIC;
              enable1:in STD_LOGIC;
              ENround:in STD_LOGIC;
              mode: in STD_LOGIC;
              keyleft:out Std_logic_vector(0 to 27);
       keyright:out Std_logic_vector(0 to 27);
              TextOUTL:out Std_logic_vector(0 to 31);
              TextOUTR:out Std_logic_vector(0 to 31)
              );
end component;
```

```
component round5 is
  port(TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
             keyleft:out Std_logic_vector(0 to 27);
             keyright:out Std_logic_vector(0 to 27);
           TextOUTL:out Std_logic_vector(0 to 31);
           TextOUTR:out Std_logic_vector(0 to 31)
             );
end component;

component round6 is
  port(       TextInL:in Std_logic_vector(0 to 31);
              TextInR:in Std_logic_vector(0 to 31);
              KeyInL:in Std_logic_vector(0 to 27);
              KeyInR:in Std_logic_vector(0 to 27);
              clk :in STD_LOGIC;
              RST: in STD_LOGIC;
              enable1:in STD_LOGIC;
              ENround:in STD_LOGIC;
              mode: in STD_LOGIC;
              keyleft:out Std_logic_vector(0 to 27);
              keyright:out Std_logic_vector(0 to 27);
              TextOUTL:out Std_logic_vector(0 to 31);
              TextOUTR:out Std_logic_vector(0 to 31)
              );
end component;

component round7 is
  port(TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;

  RST: in STD_LOGIC;
              ENround:in STD_LOGIC;
              enable1:in STD_LOGIC;
              mode: in STD_LOGIC;
              keyleft:out Std_logic_vector(0 to 27);
              keyright:out Std_logic_vector(0 to 27);
              TextOUTL:out Std_logic_vector(0 to 31);
              TextOUTR:out Std_logic_vector(0 to 31)
      );
end component;
```

```
component round8 is
  port(TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
             RST: in STD_LOGIC;
             enable1:in STD_LOGIC;
             ENround:in STD_LOGIC;
             mode: in STD_LOGIC;
            keyleft:out Std_logic_vector(0 to 27);
             keyright:out Std_logic_vector(0 to 27);
             TextOUTL:out Std_logic_vector(0 to 31);
             TextOUTR:out Std_logic_vector(0 to 31)
             );
end component;

component round9 is
  port(TextInL:in Std_logic_vector(0 to 31);
       TextInR:in Std_logic_vector(0 to 31);
       KeyInL:in Std_logic_vector(0 to 27);
       KeyInR:in Std_logic_vector(0 to 27);
       clk :in STD_LOGIC;
       RST: in STD_LOGIC;
       enable1:in STD_LOGIC;
       ENround:in STD_LOGIC;
       mode: in STD_LOGIC;
       keyleft:out Std_logic_vector(0 to 27);
       keyright:out Std_logic_vector(0 to 27);
       TextOUTL:out Std_logic_vector(0 to 31);
       TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;

component round10 is
  port(TextInL:in Std_logic_vector(0 to 31);
        TextInR:in Std_logic_vector(0 to 31);
        KeyInL:in Std_logic_vector(0 to 27);
        KeyInR:in Std_logic_vector(0 to 27);
        clk :in STD_LOGIC;
        RST: in STD_LOGIC;
        enable1:in STD_LOGIC;
        ENround:in STD_LOGIC;
        mode: in STD_LOGIC;
        keyleft:out Std_logic_vector(0 to 27);
        keyright:out Std_logic_vector(0 to 27);
        TextOUTL:out Std_logic_vector(0 to 31);
        TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;
component round11 is
  port(   TextInL:in Std_logic_vector(0 to 31);
          TextInR:in Std_logic_vector(0 to 31);
          KeyInL:in Std_logic_vector(0 to 27);
          KeyInR:in Std_logic_vector(0 to 27);
          clk :in STD_LOGIC;
          RST: in STD_LOGIC;
          enable1:in STD_LOGIC;
          ENround:in STD_LOGIC;
          mode: in STD_LOGIC;
```

```
            keyleft:out Std_logic_vector(0 to 27);
            keyright:out Std_logic_vector(0 to 27);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;

component round12 is
  port(     TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
            keyleft:out Std_logic_vector(0 to 27);
            keyright:out Std_logic_vector(0 to 27);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;

component round13 is
  port(     TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
            keyleft:out Std_logic_vector(0 to 27);
            keyright:out Std_logic_vector(0 to 27);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;

component round14 is
  port(     TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
            keyleft:out Std_logic_vector(0 to 27);
            keyright:out Std_logic_vector(0 to 27);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;
```

```
component round15 is
  port(     TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
            keyleft:out Std_logic_vector(0 to 27);
            keyright:out Std_logic_vector(0 to 27);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;

component round16 is
  port(     TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
            KeyInL:in Std_logic_vector(0 to 27);
            KeyInR:in Std_logic_vector(0 to 27);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
            keyleft:out Std_logic_vector(0 to 27);
            keyright:out Std_logic_vector(0 to 27);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end component;

component IP1 is
   port (
            TextOUTL:in Std_logic_vector(0 to 31);
            TextOUTR:in Std_logic_vector(0 to 31);
            clk: in STD_LOGIC;
            rst:in std_logic;
            enableout:in std_logic;
            ip1out: out STD_LOGIC_VECTOR (1 to 64)
   );
end component;

signal TextInL,TextInR :Std_logic_vector(0 to 31);
signal keyleft,keyright:Std_logic_vector(0 to 27);
signal TextOUTL,TextOUTR :Std_logic_vector(0 to
31);
signal keyleft1,keyright1:Std_logic_vector(0 to 27);
signal TextOUTL1,TextOUTR1 :Std_logic_vector(0 to
31);
signal keyleft2,keyright2:Std_logic_vector(0 to 27);
signal TextOUTL2,TextOUTR2 :Std_logic_vector(0 to
31);
signal keyleft3,keyright3:Std_logic_vector(0 to 27);
signal TextOUTL3,TextOUTR3 :Std_logic_vector(0 to
31);
```

```
signal keyleft4,keyright4:Std_logic_vector(0 to 27);
signal TextOUTL4,TextOUTR4 :Std_logic_vector(0 to
31);
signal keyleft5,keyright5:Std_logic_vector(0 to 27);
signal TextOUTL5,TextOUTR5 :Std_logic_vector(0 to
31);
signal keyleft6,keyright6:Std_logic_vector(0 to 27);
signal TextOUTL6,TextOUTR6 :Std_logic_vector(0 to
31);
signal keyleft7,keyright7:Std_logic_vector(0 to 27);
signal TextOUTL7,TextOUTR7 :Std_logic_vector(0 to
31);
signal keyleft8,keyright8:Std_logic_vector(0 to 27);
signal TextOUTL8,TextOUTR8 :Std_logic_vector(0 to
31);
signal keyleft9,keyright9:Std_logic_vector(0 to 27);
signal TextOUTL9,TextOUTR9 :Std_logic_vector(0 to
31);
signal keyleft10,keyright10:Std_logic_vector(0 to 27);
signal TextOUTL10,TextOUTR10 :Std_logic_vector(0
to 31);
signal keyleft11,keyright11:Std_logic_vector(0 to 27);
signal TextOUTL11,TextOUTR11 :Std_logic_vector(0
to 31);
signal keyleft12,keyright12:Std_logic_vector(0 to 27);
signal TextOUTL12,TextOUTR12 :Std_logic_vector(0
to 31);
signal keyleft13,keyright13:Std_logic_vector(0 to 27);
signal TextOUTL13,TextOUTR13 :Std_logic_vector(0
to 31);
signal keyleft14,keyright14:Std_logic_vector(0 to 27);
signal TextOUTL14,TextOUTR14 :Std_logic_vector(0
to 31);
signal keyleft15,keyright15:Std_logic_vector(0 to 27);
signal TextOUTL15,TextOUTR15 :Std_logic_vector(0
to 31);
signal keyleft16,keyright16:Std_logic_vector(0 to 27);
signal TextOUTL16,TextOUTR16 :Std_logic_vector(0
to 31);
signal
enable1,enable2,enable3,enable4,enable5,enable6,enable
7
,enable8,enable9,enable10,enable11,enable12,enable13,e
nable14,
enable15,enable16,enableout : STD_LOGIC;
signal
ENround1,ENround2,ENround3,ENround4,ENround5,E
Nround6,
ENround7,ENround8,ENround9,ENround10,ENround11
,ENround12,
ENround13,ENround14,ENround15,ENround16 :
STD_LOGIC;
signal encodin : STD_LOGIC;

begin
-------------------------------------------
unit_control : Control
  port map(
```

```
            clk=>clk,
            rst=>rst,
            encodin=>encodin,
            enable1=>enable1,
            enable2=>enable2,
            enable3=>enable3,
            enable4=>enable4,
            enable5=>enable5,
            enable6=>enable6,
            enable7=>enable7,
            enable8=>enable8,
            enable9=>enable9,
            enable10=>enable10,
            enable11=>enable11,
            enable12=>enable12,
            enable13=>enable13,
            enable14=>enable14,
            enable15=>enable15,
            enable16=>enable16,
            enableout=>enableout,
            ENround1=>ENround1,
            ENround2=>ENround2,
            ENround3=>ENround3,
            ENround4=>ENround4,
            ENround5=>ENround5,
            ENround6=>ENround6,
            ENround7=>ENround7,
            ENround8=>ENround8,
            ENround9=>ENround9,
            ENround10=>ENround10,
            ENround11=>ENround11,
            ENround12=>ENround12,
            ENround13=>ENround13,
            ENround14=>ENround14,
            ENround15=>ENround15,
            ENround16=>ENround16

    );

--DATA/TEXT declarations----------

unit_intro: intro
          port map
                    (TextIn=>TextIn,
                    KeyIn=>KeyIn,
                    clk =>clk,
                    RST=>RST,
                    encodin=>encodin,
                     keyleft=>keyleft,
                    keyright=>keyright,
                    TextInL=>TextInL,
                     TextInR=>TextInR
           );

unit_1: round1
  port map
  (         KeyInL=>keyleft,
            KeyInR=>keyright,
```

```
            TextInL=>TextInL,
            TextInR=>TextInR,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable1,
            ENround=>ENround1,
            keyleft=>keyleft1,
            keyright=>keyright1,
            TextOUTL=>TextOUTL1,
            TextOUTR=>TextOUTR1
            );

unit_2: round2
 port map
 (          TextInL=>TextOUTR1,
            TextInR=>TextOUTL1,
            KeyInL=>keyleft1 ,
            KeyInR=> keyright1,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable2,
            ENround=>ENround2,
            keyleft=>keyleft2,
            keyright=>keyright2,
            TextOUTL=>TextOUTL2,
            TextOUTR=>TextOUTR2
          );

unit_3: round3
 port map
 (          TextInL=>TextOUTR2,
            TextInR=>TextOUTL2,
            KeyInL=>keyleft2 ,
            KeyInR=> keyright2,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable3,
            ENround=>ENround3,
            keyleft=>keyleft3,
            keyright=>keyright3,
            TextOUTL=>TextOUTL3,
            TextOUTR=>TextOUTR3
          );
unit_4: round4
 port map
 (          TextInL=>TextOUTR3,
            TextInR=>TextOUTL3,
            KeyInL=>keyleft3 ,
            KeyInR=> keyright3,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable4,
            ENround=>ENround4,
            keyleft=>keyleft4,
            keyright=>keyright4,
            TextOUTL=>TextOUTL4,
            TextOUTR=>TextOUTR4
          );

unit_5: round5
 port map
 (          TextInL=>TextOUTR4,
            TextInR=>TextOUTL4,
            KeyInL=>keyleft4 ,
            KeyInR=> keyright4,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable5,
            ENround=>ENround5,
            keyleft=>keyleft5,
            keyright=>keyright5 ,
            TextOUTL=>TextOUTL5,
            TextOUTR=>TextOUTR5
          );

unit_6: round6
 port map
 (          TextInL=>TextOUTR5,
            TextInR=>TextOUTL5,
            KeyInL=>keyleft5 ,
            KeyInR=> keyright5,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable6,
            ENround=>ENround6,
            keyleft=>keyleft6,
            keyright=>keyright6 ,
            TextOUTL=>TextOUTL6,
            TextOUTR=>TextOUTR6
          );

unit_7: round7
 port map
 (          TextInL=>TextOUTR6,
            TextInR=>TextOUTL6,
            KeyInL=>keyleft6 ,
            KeyInR=> keyright6,
            clk =>clk,
            RST=>rst,
            mode=>mode,
            enable1=>enable7,
            ENround=>ENround7,
            keyleft=>keyleft7,
            keyright=>keyright7 ,
            TextOUTL=>TextOUTL7,
            TextOUTR=>TextOUTR7
 );
unit_8: round8
 port map
 (          TextInL=>TextOUTR7,
            TextInR=>TextOUTL7,
```

```
                KeyInL=>keyleft7 ,
                KeyInR=> keyright7,
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable8,
                ENround=>ENround8,
                 keyleft=>keyleft8,
                keyright=>keyright8     ,
                TextOUTL=>TextOUTL8,
                TextOUTR=>TextOUTR8
              );

unit_9: round9
  port map
  (             TextInL=>TextOUTR8,
                TextInR=>TextOUTL8,
                 KeyInL=>keyleft8 ,
                KeyInR=> keyright8,
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable9,
                ENround=>ENround9,
                 keyleft=>keyleft9,
                keyright=>keyright9     ,
                TextOUTL=>TextOUTL9,
                TextOUTR=>TextOUTR9
              );

unit_10: round10
  port map
  (            TextInL=>TextOUTR9,
               TextInR=>TextOUTL9,
                KeyInL=>keyleft9,
                KeyInR=> keyright9,
                clk =>clk,
                RST=>rst,
                 mode=>mode,
                enable1=>enable10,
                ENround=>ENround10,
                 keyleft=>keyleft10,
                keyright=>keyright10    ,
                TextOUTL=>TextOUTL10,
                TextOUTR=>TextOUTR10
              );
unit_11: round11
  port map
  (              TextInL=>TextOUTR10,
                 TextInR=>TextOUTL10,
                 KeyInL=>keyleft10 ,
                KeyInR=> keyright10,
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable11,
                ENround=>ENround11,
                 keyleft=>keyleft11,
                keyright=>keyright11,
```

```
                TextOUTL=>TextOUTL11,
                TextOUTR=>TextOUTR11
              );

unit_12: round12
  port map
  (             TextInL=>TextOUTR11,
                TextInR=>TextOUTL11,
                KeyInL=>keyleft11 ,
                KeyInR=> keyright11,
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable12,
                ENround=>ENround12,
                 keyleft=>keyleft12,
                keyright=>keyright12    ,
                TextOUTL=>TextOUTL12,
                TextOUTR=>TextOUTR12
              );

unit_13: round13
  port map
  (             TextInL=>TextOUTR12,
                TextInR=>TextOUTL12,
                KeyInL=>keyleft12 ,
                 KeyInR=> keyright12,
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable13,
                ENround=>ENround13,
                keyleft=>keyleft13,
                keyright=>keyright13    ,
                TextOUTL=>TextOUTL13,
                TextOUTR=>TextOUTR13
              );
unit_14: round14
  port map
  (             TextInL=>TextOUTR13,
                TextInR=>TextOUTL13,
                KeyInL=>keyleft13 ,
                 KeyInR=> keyright13,
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable14,
                ENround=>ENround14,
                 keyleft=>keyleft14,
    keyright=>keyright14    ,
                TextOUTL=>TextOUTL14,
                TextOUTR=>TextOUTR14
              );
 unit_15: round15
  port map
  (             TextInL=>TextOUTR14,
                TextInR=>TextOUTL14,
                KeyInL=>keyleft14 ,
                 KeyInR=> keyright14,
```

```
                clk =>clk,
                RST=>rst,
                mode=>mode,
                enable1=>enable15,
                ENround=>ENround15,
                keyleft=>keyleft15,
                keyright=>keyright15,
                TextOUTL=>TextOUTL15,
                TextOUTR=>TextOUTR15
              );

unit_16: round16
  port map(
                TextInL=>TextOUTR15,
                TextInR=>TextOUTL15,
                KeyInL=>keyleft15,
                KeyInR=>keyright15,
                clk =>clk,
                RST=>rst,
                enable1=>enable16,
                ENround=>ENround16,
                mode=>mode,
                keyleft=>keyleft16,
                keyright=>keyright16,
                TextOUTL=>TextOUTL16,
                TextOUTR=>TextOUTR16
              );
UNIT_IP1: IP1

  port map(
              TextOUTL=>TextOUTL16,
              TextOUTR=>TextOUTR16,
              clk =>clk,
              RST=>rst,
              enableout=>enableout,
               ip1out=>TextOUT
             );
end BEHAVIOR;
```

intro.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_misc.all;
entity intro is
  port(
            TextIn:in Std_logic_vector(1 to 64);
            KeyIn:in Std_logic_vector(1 to 64);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            encodin:in std_logic;
          keyleft: out STD_LOGIC_VECTOR (1 to 28);
          keyright: out STD_LOGIC_VECTOR (1 to 28);
          TextInL: out STD_LOGIC_VECTOR (1 to 32);
          TextInR: out STD_LOGIC_VECTOR (1 to 32)
             );
end;
architecture BEHAVIOR of intro is

component ipsplit is
    port (
       ip1in: in STD_LOGIC_VECTOR (1 to 64);
       clk: in STD_LOGIC;
       rst:in std_logic;
       encodin:in std_logic;
       left: out STD_LOGIC_VECTOR (1 to 32);
       right: out STD_LOGIC_VECTOR (1 to 32)
    );
end component;

component kpc1asplit is
    port ( keyin: in STD_LOGIC_VECTOR (1 to 64);
            clk: in STD_LOGIC;
            rst:in std_logic;
            encodin:in std_logic;
            keyleft: out STD_LOGIC_VECTOR (1 to 28);
           keyright: out STD_LOGIC_VECTOR (1 to 28)
    );
end component;
begin
--DATA/TEXT declarations----------
U_ipsplit : ipsplit
  port map(  ip1in=>TextIn,
                clk =>clk,
               RST=>rst,
               encodin=>encodin,
                left=>TextInL,
                right=>TextInR
              );
U_kpc1asplit : kpc1asplit
  port map(  keyin=>keyin,
                clk =>clk,
               RST=>rst,
              encodin=>encodin,
               keyleft=>keyleft,
               keyright=>keyright
             ); end BEHAVIOR;
```

Ipsplit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ipsplit is
   port (  ip1in: in STD_LOGIC_VECTOR (1 to 64);
           clk: in STD_LOGIC;
           rst:in std_logic;
           encodin:in std_logic;
            left: out STD_LOGIC_VECTOR (1 to 32);
            right: out STD_LOGIC_VECTOR (1 to 32)
   );
end ipsplit;

architecture ipsplit_arch of ipsplit is

signal ip1out: STD_LOGIC_VECTOR (1 to 64);

begin

process(clk,rst)

begin

if rst='0' then

ip1out <=(others=>'0');
left<=(others=>'0');
right<=(others=>'0');
    else
                    if encodin='1' then

                     ip1out(1)  <= ip1in(58);
                     ip1out(2)  <= ip1in(50) ;
                     ip1out(3)  <= ip1in(42);
                     ip1out(4)  <= ip1in(34);
                     ip1out(5)  <= ip1in(26);
                     ip1out(6)  <= ip1in(18);
                     ip1out(7)  <= ip1in(10);
                     ip1out(8)  <= ip1in(2);
                     ip1out(9)  <= ip1in(60);
                     ip1out(10) <= ip1in(52) ;
                     ip1out(11) <= ip1in(44);
                     ip1out(12) <= ip1in(36);
                     ip1out(13) <= ip1in(28);
                     ip1out(14) <= ip1in(20);
                     ip1out(15) <= ip1in(12);
                     ip1out(16) <= ip1in(4);
                     ip1out(17) <= ip1in(62);
                     ip1out(18) <= ip1in(54) ;
                     ip1out(19) <= ip1in(46);
                     ip1out(20) <= ip1in(38);
                     ip1out(21) <= ip1in(30);
                     ip1out(22) <= ip1in(22);
                     ip1out(23) <= ip1in(14);
                     ip1out(24) <= ip1in(6);
                     ip1out(25) <= ip1in(64);
```

```
                     ip1out(26) <= ip1in(56) ;
                     ip1out(27) <= ip1in(48);
                     ip1out(28) <= ip1in(40);
                     ip1out(29) <= ip1in(32);
                     ip1out(30) <= ip1in(24);
                     ip1out(31) <= ip1in(16);
                     ip1out(32) <= ip1in(8);
                     ip1out(33) <= ip1in(57);
                     ip1out(34) <= ip1in(49) ;
                     ip1out(35) <= ip1in(41);
                     ip1out(36) <= ip1in(33);
                     ip1out(37) <= ip1in(25);
                     ip1out(38) <= ip1in(17);
                     ip1out(39) <= ip1in(9);
                     ip1out(40) <= ip1in(1);
                     ip1out(41) <= ip1in(59);
                     ip1out(42) <= ip1in(51) ;
                     ip1out(43) <= ip1in(43);
                     ip1out(44) <= ip1in(35);
                     ip1out(45) <= ip1in(27);
                     ip1out(46) <= ip1in(19);
                     ip1out(47) <= ip1in(11);
                     ip1out(48) <= ip1in(3);
                     ip1out(49) <= ip1in(61);
                     ip1out(50) <= ip1in(53) ;
                     ip1out(51) <= ip1in(45);
                     ip1out(52) <= ip1in(37);
                     ip1out(53) <= ip1in(29);
                     ip1out(54) <= ip1in(21);
                     ip1out(55) <= ip1in(13);
                     ip1out(56) <= ip1in(5);
                     ip1out(57) <= ip1in(63);
                     ip1out(58) <= ip1in(55) ;
                     ip1out(59) <= ip1in(47);
                     ip1out(60) <= ip1in(39) ;
                     ip1out(61) <= ip1in(31);
                     ip1out(62) <= ip1in(23);
                     ip1out(63) <= ip1in(15);
                     ip1out(64) <= ip1in(7);
             else
                     null;
            end if;

           if clk'event and clk='1' then
                     left (1 to 32)<= ip1out( 1 to 32);
                     right(1 to 32)<= ip1out(33 to 64);

           end if;
end if;

end process;

end ipsplit_arch;
```

Kpc1asplit.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity kpc1asplit is
    port (
        keyin: in STD_LOGIC_VECTOR (1 to 64);
        clk: in STD_LOGIC;
        rst:in std_logic;
        encodin:in std_logic;
        keyleft: out STD_LOGIC_VECTOR (1 to 28);
        keyright: out STD_LOGIC_VECTOR (1 to 28)
    );
end kpc1asplit;

architecture kpc1asplit_arch of kpc1asplit is

signal keytemp:STD_LOGIC_VECTOR (1 to 56);

begin

process (clk,rst)

begin


if rst='0' then
            keytemp<=(others=>'0');
            keyleft<=(others=>'0');
            keyright<=(others=>'0');
   else
             if encodin='1' then

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

          if (clk'event and clk='1') then

              keyleft (1 to 28)<= keytemp( 1 to 28);
      keyright(1 to 28)<= keytemp(29 to 56);

             end if;

end if;

end process;

end kpc1asplit_arch;
```

round1.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.std_logic_misc.all;

entity round1 is
   port(
            TextInL:in Std_logic_vector(0 to 31);
            TextInR:in Std_logic_vector(0 to 31);
             KeyInL:in Std_logic_vector(1 to 28);
            KeyInR:in Std_logic_vector(1 to 28);
            clk :in STD_LOGIC;
            RST: in STD_LOGIC;
            enable1:in STD_LOGIC;
            ENround:in STD_LOGIC;
            mode: in STD_LOGIC;
           keyleft:out Std_logic_vector(1 to 28);
            keyright:out Std_logic_vector(1 to 28);
            TextOUTL:out Std_logic_vector(0 to 31);
            TextOUTR:out Std_logic_vector(0 to 31)
            );
end;

architecture BEHAVIOR of round1 is

component reg32right is
   port (
      inreg: in STD_LOGIC_VECTOR (1 to 32);
      clk: in STD_LOGIC;
      rst:in std_logic;
      ENround:in std_logic;
      outR: out STD_LOGIC_VECTOR (1 to 32)
   );
end component;

component reg32left is
   port (
            inreg: in STD_LOGIC_VECTOR (1 to 32);
            clk: in STD_LOGIC;
            enable1 : in STD_LOGIC;
            ENround : in STD_LOGIC;
            rst:in std_logic;
            outl: out STD_LOGIC_VECTOR (1 to 32)
   );
end component;
component EXP is
   port (
      outR: in STD_LOGIC_VECTOR (1 to 32);
      clk: in STD_LOGIC;
      rst:in std_logic;
      expout: out STD_LOGIC_VECTOR (1 to 48)
   );
end component;
```

```
component xor48 is

   port (
     expout: in STD_LOGIC_VECTOR (47 downto 0);
     key: in STD_LOGIC_VECTOR (47 downto 0);
     outxor: out STD_LOGIC_VECTOR (47 downto 0)
   );
end component;

component rom_64x4 is port (

outxor: in std_logic_vector(0 to 47);
DATA1,DATA2,DATA3,DATA4,DATA5,DATA6,
DATA7,DATA8: out std_logic_vector(3 downto 0);
rst:in std_logic;
clk:in std_logic
            );

end component;

component Pbox is
   port (
DATA1,DATA2,DATA3,DATA4,DATA5,DATA6,
DATA7,DATA8: in STD_LOGIC_VECTOR (1 to 4);
clk: in STD_LOGIC;
RST: in STD_LOGIC;
enable1 : in STD_LOGIC;
pboxout: out STD_LOGIC_VECTOR (1 to 32)
   );
end component;

component xor32 is
   port (
      pboxout: in STD_LOGIC_VECTOR (1 to 32);
      outl: in STD_LOGIC_VECTOR (1 to 32);
      outxor32: out STD_LOGIC_VECTOR (1 to 32)
   );
end component;

component reg88 is
   port (
      skeyleft: in STD_LOGIC_VECTOR (1 to 28);
      skeyright: in STD_LOGIC_VECTOR (1 to 28);
      outR: in STD_LOGIC_VECTOR (1 to 32);
      clk: in STD_LOGIC;
      enable1: in STD_LOGIC;
      RST: in STD_LOGIC;
      keyleft: out STD_LOGIC_VECTOR (1 to 28);
      keyright: out STD_LOGIC_VECTOR (1 to 28);
      outleft: out STD_LOGIC_VECTOR (1 to 32)
   );
end component;
component SReg28x1 is
   port (
      keyleft: in STD_LOGIC_Vector ( 1 to 28);
      keyright: in STD_LOGIC_Vector ( 1 to 28);
      clk: in STD_LOGIC;
```

```
            rst: in STD_LOGIC;
            mode: in STD_LOGIC;
            ENround: in STD_LOGIC;
            skeyleft: out STD_LOGIC_Vector ( 1 to 28);
            skeyright: out STD_LOGIC_Vector (1 to 28)
    );
end component;

component kpc2 is
    port (
            skeyleft: in STD_LOGIC_VECTOR (1 to 28);
                skeyright: in STD_LOGIC_VECTOR (1 to
28);
          key: out STD_LOGIC_VECTOR (1 to 48);
                rst: in STD_LOGIC;
                clk: in STD_LOGIC
    );
end component;

signal outkeyleft,outkeyright,skeyleft,skeyright:
STD_LOGIC_VECTOR (1 to 28);
signal outl,outR,pboxout,out32:STD_LOGIC_VECTOR
(1 to 32);
signal expout,outxor,keyout:STD_LOGIC_VECTOR (1
to 48);
STD_LOGIC_VECTOR (1 to 6);
Signal
DATA1,DATA2,DATA3,DATA4,DATA5,DATA6,
DATA7,DATA8: STD_LOGIC_VECTOR (1 to 4);

begin
-----------------------------------------
--DATA/TEXT declarations------------------

--Register left unit-----------------------
U_reg32left_text: reg32left
    port map(
        inreg=>TextInL,
                clk=>clk,
                enable1=>enable1,
                ENround=>ENround,
                rst=>rst,
                 outl=>outl    );
--Register right unit----------------------
U_reg32right_text: reg32right
    port map(
                inreg=>TextInR,
                clk=>clk,
                rst=>rst,
                ENround=>ENround,
                outR=>outR
    );
--expansion unit-----------------------------
U_exp_text: EXP
    port map(
        outR=>outR,    ---- outR
```

```
--expansion unit-----------------------------
U_exp_text: EXP
    port map(
        outR=>outR,     ---- outR ->input expansion unit
        clk=>clk,
         rst=>rst,
         expout=>expout
    );
--xor 48bits unit-- link key with data/text--
U_xor48_text: xor48

    port map(
        expout=>expout,
        key=>keyout,
        outxor=> outxor
    );

--ROM 64x4 --8 S_BOXES-----------------------
U_roms_text : rom_64x4

   port map (
        outxor=>outxor,
        DATA1=>DATA1,
        DATA2=>DATA2,
        DATA3=>DATA3,
        DATA4=>DATA4,
        DATA5=>DATA5,
        DATA6=>DATA6,
        DATA7=>DATA7,
        DATA8=>DATA8,
        rst=>rst,
        clk=>clk
      );
----------P_BOX-----------------------------
U_pbox_text: Pbox
    port map (
        clk=>clk,
        enable1=>enable1,
        rst=>rst,
        DATA1=>DATA1,
        DATA2=>DATA2,
        DATA3=>DATA3,
        DATA4=>DATA4,
        DATA5=>DATA5,
        DATA6=>DATA6,
        DATA7=>DATA7,
        DATA8=>DATA8,
        pboxout=>pboxout
    );
--final xor 32 unit---------------------------
U_xor32_text: xor32
port map(
        pboxout=>pboxout,
        outl=>outl,
        outxor32=>out32
    );
```

```
-------------------------------------
--DATA/KEY declarations---------------

--shift register key (x1) ----------------------

U_Sreg28x1_text: SReg28x1
   port map(
            keyleft=>KeyInL,
            keyright=>KeyInR,
            clk=>clk,
            mode=>mode,
            rst=>rst,
            ENround=>ENround,
            skeyleft=>skeyleft,
            skeyright=>skeyright
   );
--key  round1 unit------------------------------

U_kpc2_text: kpc2
   port map(
            skeyleft=>skeyleft,
            skeyright=>skeyright,
            key=>keyout,
            clk=>clk,
            rst=>rst
   );
-------------------------------------------------
U_reg88_text: reg88

port map(
             skeyleft=>skeyleft,
            skeyright=>skeyright,
             outR=>outR,
             clk=>clk,
            rst=>rst,
            enable1=>enable1,
            keyleft=>outkeyleft,
            keyright=>outkeyright,
            outleft=>TextOUTR
   );

-------------------------------------------------
--EXIT ROUND1-------------------------------

TextOUTL<=outleft1;

TextOUTL<=out32;
keyleft<=outkeyleft;
keyright<=outkeyright;


end BEHAVIOR;
```

Shift Register

```
library IEEE;
use IEEE.std_logic_1164.all;

entity SReg28x1 is
   port (
      keyleft: in STD_LOGIC_Vector ( 1 to 28);
      keyright: in STD_LOGIC_Vector ( 1 to 28);
      clk: in STD_LOGIC;
      rst: in STD_LOGIC;
      mode: in STD_LOGIC;
      ENround : in STD_LOGIC;
      skeyleft: out STD_LOGIC_Vector ( 1 to 28);
      skeyright: out STD_LOGIC_Vector (1 to 28)
   );
end SReg28x1;

architecture SReg28x1_arch of SReg28x1 is

begin

process (clk,mode,rst)

begin

if rst='0' then

skeyleft<=(others=>'0');
skeyright<=(others=>'0');

 else
  if mode='0' then

            if  (clk'event and clk='1') then
                     if ENround='1' then
skeyleft <=to_stdlogicvector (to_bitvector(keyleft) rol
1);
skeyright<=to_stdlogicvector (to_bitvector(keyright)
rol 1);
                     else
                              null;
                     end if;
            end if;
  else
             if  (clk'event and clk='1') then
                              if ENround='1' then
skeyleft <=to_stdlogicvector (to_bitvector(keyleft) ror
0);
skeyright<=to_stdlogicvector (to_bitvector(keyright)
ror 0);
                  else
                              null;
                             end if;
              end if;
end if;
end if;
end process;
end SReg28x1_arch;
```

EXPANSION

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity EXP is
   port (
      outR: in STD_LOGIC_VECTOR (1 to 32);
      clk: in STD_LOGIC;
      rst:in std_logic;
      expout: out STD_LOGIC_VECTOR (1 to 48)
   );
end EXP;

architecture EXP_arch of EXP is

begin

process (clk,rst)

begin

if rst='0' then
          expout <=(others=>'0');
        else
        if clk'event and clk='1' then

                  expout(1)  <= outR(32);
                  expout(2)  <= outR(1) ;
                  expout(3)  <= outR(2) ;
                  expout(4)  <= outR(3) ;
                  expout(5)  <= outR(4) ;
                  expout(6)  <= outR(5) ;
                  expout(7)  <= outR(4) ;
                  expout(8)  <= outR(5) ;
                  expout(9)  <= outR(6) ;
                  expout(10) <= outR(7) ;
                  expout(11) <= outR(8) ;
                  expout(12) <= outR(9) ;
                  expout(13) <= outR(8) ;
                  expout(14) <= outR(9) ;
                  expout(15) <= outR(10);
                  expout(16) <= outR(11);
                  expout(17) <= outR(12);
                  expout(18) <= outR(13);
                  expout(19) <= outR(12);
                  expout(20) <= outR(13);
                  expout(21) <= outR(14);
                  expout(22) <= outR(15);
                  expout(23) <= outR(16);
                  expout(24) <= outR(17);
                  expout(25) <= outR(16);
                  expout(26) <= outR(17);
                  expout(27) <= outR(18);
                  expout(28) <= outR(19);
```

```
                  expout(29) <= outR(20);
                  expout(30) <= outR(21);
                  expout(31) <= outR(20);
                  expout(32) <= outR(21);
                  expout(33) <= outR(22);
                  expout(34) <= outR(23);
                  expout(35) <= outR(24);
                  expout(36) <= outR(25);
                  expout(37) <= outR(24);
                  expout(38) <= outR(25);
                  expout(39) <= outR(26);
                  expout(40) <= outR(27);
                  expout(41) <= outR(28);
                  expout(42) <= outR(29);
                  expout(43) <= outR(28);
                  expout(44) <= outR(29);
                  expout(45) <= outR(30);
                  expout(46) <= outR(31);
                  expout(47) <= outR(32);
                  expout(48) <= outR(1) ;

                  end if;
end if;
end process;
end EXP_arch;
```

P-BOX

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Pbox is
   port (

DATA1,DATA2,DATA3,DATA4,DATA5,DATA6,
DATA7,DATA8: in STD_LOGIC_VECTOR (1 to 4);
clk: in STD_LOGIC;
rst: in STD_LOGIC;
enable1 : in STD_LOGIC;
pboxout: out STD_LOGIC_VECTOR (1 to 32)
   );
end Pbox;

architecture Pbox_arch of Pbox is

signal sboxout: STD_LOGIC_VECTOR (1 to 32);

begin

process (clk,rst)

begin

if rst='0' then
sboxout<=(others=>'0');
pboxout<=(others=>'0');

else
                    sboxout( 1 to  4)<=DATA1(1 to 4);
                    sboxout( 5 to  8)<=DATA2(1 to 4);
                    sboxout( 9 to 12)<=DATA3(1 to 4);
                    sboxout(13 to 16)<=DATA4(1 to 4);
                    sboxout(17 to 20)<=DATA5(1 to 4);
                    sboxout(21 to 24)<=DATA6(1 to 4);
                    sboxout(25 to 28)<=DATA7(1 to 4);
                    sboxout(29 to 32)<=DATA8(1 to 4);

          if (clk'event and clk='1') then
                    if enable1 ='1' then
                     pboxout(1) <= sboxout(16);
                     pboxout(2) <= sboxout(7);
                     pboxout(3) <= sboxout(20);
                     pboxout(4) <= sboxout(21);
                     pboxout(5) <= sboxout(29);
                     pboxout(6) <= sboxout(12);
                     pboxout(7) <= sboxout(28);
                     pboxout(8) <= sboxout(17);
                     pboxout(9) <= sboxout(1);
                     pboxout(10) <= sboxout(15);
                     pboxout(11) <= sboxout(23);
                     pboxout(12) <= sboxout(26);
                      pboxout(13) <= sboxout(5);
                     pboxout(14) <= sboxout(18);
```

```
                     pboxout(15) <= sboxout(31);
                     pboxout(16) <= sboxout(10);
                     pboxout(17) <= sboxout(2);
                     pboxout(18) <= sboxout(8);
                     pboxout(19) <= sboxout(24);
                     pboxout(20) <= sboxout(14);
                     pboxout(21) <= sboxout(32);
                     pboxout(22) <= sboxout(27);
                     pboxout(23) <= sboxout(3);
                     pboxout(24) <= sboxout(9);
                     pboxout(25) <= sboxout(19);
                     pboxout(26) <= sboxout(13);
                     pboxout(27) <= sboxout(30);
                     pboxout(28) <= sboxout(6);
                     pboxout(29) <= sboxout(22);
                     pboxout(30) <= sboxout(11);
                     pboxout(31) <= sboxout(4);
                     pboxout(32) <= sboxout(25);
                     else
                     null;

                    end if;
end if;

end if;
end process;

end Pbox_arch;
```

S-BOXES

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity rom_64x4 is port (

   outxor: in STD_LOGIC_VECTOR (0 to 47);
   DATA1,DATA2,DATA3,DATA4,DATA5,DATA6,
DATA7,DATA8: out std_logic_vector(1 to 4);
   clk:in std_logic;
   rst:in std_logic

            );

end rom_64x4;

architecture behaviour of rom_64x4 is

signal
ADDR1,ADDR2,ADDR3,ADDR4,ADDR5,ADDR6,
ADDR7,ADDR8: STD_LOGIC_VECTOR (0 to 5);

begin

process(clk,outxor)

begin

addr1 <= STD_LOGIC_VECTOR(outxor( 0 to  5));
addr2 <= STD_LOGIC_VECTOR(outxor( 6 to 11));
addr3 <= STD_LOGIC_VECTOR(outxor(12 to 17));
addr4 <= STD_LOGIC_VECTOR(outxor(18 to 23));
addr5 <= STD_LOGIC_VECTOR(outxor(24 to 29));
addr6 <= STD_LOGIC_VECTOR(outxor(30 to 35));
addr7 <= STD_LOGIC_VECTOR(outxor(36 to 41));
addr8 <= STD_LOGIC_VECTOR(outxor(42 to 47));

  if rst='0' then


            data1 <=(others=>'0');
            data2 <=(others=>'0');
            data3 <=(others=>'0');
            data4 <=(others=>'0');
            data5 <=(others=>'0');
            data6 <=(others=>'0');
            data7 <=(others=>'0');
            data8 <=(others=>'0');

 else
   if clk'event and clk='1' then
    case ADDR1 is

            when "000000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"e"));
            when "000010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"4"));
            when "000100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"d"));
            when "000110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"1"));
            when "001000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"2"));
            when "001010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"f"));
            when "001100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"b"));
            when "001110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"8"));
            when "010000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"3"));
            when "010010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"a"));
            when "010100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"6"));
            when "010110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"c"));
            when "011000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"5"));
            when "011010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"9"));
            when "011100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"0"));
            when "011110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"7"));
            when "000001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"0"));
            when "000011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"f"));
            when "000101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"7"));
            when "000111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"4"));
            when "001001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"e"));
            when "001011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"2"));
            when "001101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"d"));
            when "001111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"1"));
            when "010001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"a"));
            when "010011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"6"));
            when "010101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"c"));
            when "010111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"b"));
            when "011001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"9"));
            when "011011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"5"));
```

```
when "011101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"3"));
          when "011111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"8"));
          when "100000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"4"));
          when "100010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"1"));
          when "100100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"e"));
          when "100110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"8"));
          when "101000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"d"));
          when "101010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"6"));
          when "101100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"2"));
          when "101110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"b"));
          when "110000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"f"));
          when "110010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"c"));
          when "110100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"9"));
          when "110110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"7"));
          when "111000"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"3"));
          when "111010"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"a"));
          when "111100"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"5"));
          when "111110"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"0"));
          when "100001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"f"));
          when "100011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"c"));
          when "100101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"8"));
          when "100111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"2"));
          when "101001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"4"));
          when "101011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"9"));
          when "101101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"1"));
          when "101111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"7"));
          when "110001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"5"));
          when "110011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"b"));
          when "110101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"3"));
```

```
          when "110111"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"e"));
          when "111001"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"a"));
          when "111011"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"0"));
          when "111101"=>
DATA1<=To_StdLogicVector(Bit_Vector'(x"6"));
          when others  =>
DATA1<=To_StdLogicVector(Bit_Vector'(x"d"));
 end case;

case ADDR2 is

                    .
                    .
                    .


case ADDR8 is
                    .
                    .
                    .

end process;
end;
```

Ο Πλήρης Κώδικας βρίσκεται στο συνοδευτικο CD

Kpc2.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity kpc2 is
    port (
        clk: in STD_LOGIC;
        rst: in STD_LOGIC;
        skeyleft: in STD_LOGIC_VECTOR (1 to 28);
        skeyright: in STD_LOGIC_VECTOR (1 to 28);
        key: out STD_LOGIC_VECTOR (1 to 48)
    );
end kpc2;

architecture kpc2_arch of kpc2 is

signal keytemp:STD_LOGIC_VECTOR (1 to 56);

begin

process(rst,clk)
begin

if rst='0' then
  keytemp(1 to 56)<= (others=>'0');

   elsif (clk'event and clk='1') then

            keytemp( 1 to 28)<= skeyleft( 1 to 28);
            keytemp(29 to 56)<= skeyright(1 to 28);
  end if;
end process;

                        key(1)  <= keytemp(14);
                        key(2)  <= keytemp(17);
                        key(3)  <= keytemp(11);
                        key(4)  <= keytemp(24);
                        key(5)  <= keytemp(1) ;
                        key(6)  <= keytemp(5) ;
                        key(7)  <= keytemp(3) ;
                        key(8)  <= keytemp(28);
                        key(9)  <= keytemp(15);
                        key(10) <= keytemp(6) ;
                        key(11) <= keytemp(21);
                        key(12) <= keytemp(10);
                        key(13) <= keytemp(23);
                        key(14) <= keytemp(19);
                        key(15) <= keytemp(12);
                        key(16) <= keytemp(4) ;
                        key(17) <= keytemp(26);
                        key(18) <= keytemp(8) ;
                        key(19) <= keytemp(16);
                        key(20) <= keytemp(7) ;
                        key(21) <= keytemp(27);
                        key(22) <= keytemp(20);
                        key(23) <= keytemp(13);
                        key(24) <= keytemp(2) ;
                        key(25) <= keytemp(41);
                        key(26) <= keytemp(52);
                        key(27) <= keytemp(31);
                        key(28) <= keytemp(37);
                        key(29) <= keytemp(47);
                        key(30) <= keytemp(55);
                        key(31) <= keytemp(30);
                        key(32) <= keytemp(40);
                        key(33) <= keytemp(51);
                        key(34) <= keytemp(45);
                        key(35) <= keytemp(33);
                        key(36) <= keytemp(48);
                        key(37) <= keytemp(44);
                        key(38) <= keytemp(49);
                        key(39) <= keytemp(39);
                        key(40) <= keytemp(56);
                        key(41) <= keytemp(34);
                        key(42) <= keytemp(53);
                        key(43) <= keytemp(46);
                        key(44) <= keytemp(42);
                        key(45) <= keytemp(50);
                        key(46) <= keytemp(36);
                        key(47) <= keytemp(29);
                        key(48) <= keytemp(32);

end kpc2_arch;
```

Παράρτημα

Β. Η Ιστορία της Κρυπτογραφίας

Ημερομηνία	Πληροφορίες	ΠΗΓΗ
περίπου 1900 π.Χ	Ένας αιγυπτιακός γραφέας χρησιμοποίησε μη τυποποιημένα ιερογλυφικά σε μια επιγραφή. Ο Kahn το απαριθμεί αυτό ως το πρώτο τεκμηριωμένο παράδειγμα του γραπτού συστήματος κρυπτογραφίας.	Kahn σελ.71
1500 π.Χ	Μια Μεσοποτάμια επιγραφή περιέχει έναν κρυπτογραφημένο τύπο για την παραγωγή βερνικιού για αγγειοπλαστική.	Kahn σελ.75
500-600 π.Χ	Οι εβραϊκές γραφές όταν κατέγραψαν το βιβλίο του Ιερεμία χρησιμοποίησαν απλό κρυπτογράφημα αντιστροφής αλφάβητου γνωστό ως ATBASH. Ο Ιερεμίας άρχισε να υπαγορεύει στον Baruch το 605 π.Χ. αλλά τα κεφάλαια που περιέχουν αυτά τα κομμάτια κρυπτογραφημένου κειμένου αποδίδονται σε μια πηγή επονομαζόμενη «C» (που δεν αποδίδεται στον Baruch) και που ο συντάκτης του θα μπορούσε να κάποιος που γράφει μετά από την Βαβυλώνια εξορία το 587 π.Χ., κάποιος σύγχρονος με τον Baruch ή ακόμα και ο ίδιος ο Ιερεμίας.).Ο ATBASH ήταν ένας από τα εβραϊκούς τρόπους κρυπτογράφησης της εποχής.	Kahn σελ.77
487 π.Χ	Οι Έλληνες χρησιμοποίησαν μια συσκευή αποκαλούμενη «σκυτάλη», μια ξύλινη ράβδος. Όπως αναφέρει ο Πλούταρχος, η «Σπαρτιατική Σκυτάλη», ήταν μια ξύλινη ράβδος, ορισμένης διαμέτρου, γύρω από την οποία ήταν τυλιγμένη ελικοειδώς μια λωρίδα περγαμηνής. Το κείμενο ήταν γραμμένο σε στήλες, ένα γράμμα σε κάθε έλικα, όταν δε ξετύλιγαν τη λωρίδα, το κείμενο ήταν ακατάληπτο εξαιτίας της ανάμειξης των γραμμάτων. Το κλειδί ήταν το μήκος της διαμέτρου.	Kahn σελ.82
50-60 π.Χ	Ο Ιούλιος Καίσαρας (100-44 π.Χ) χρησιμοποίησε μια απλή αντικατάσταση με το κανονικό αλφάβητο (μετατοπίζοντας τα γράμματα κατά ένα συγκεκριμένο αριθμό) στις κυβερνητικές επιστολές. Αυτός ο τρόπος κρυπτογράφησης ήταν λιγότερο ισχυρός από τον ATBASH αλλά σε μία εποχή που ελάχιστοι ήξεραν να διαβάζουν ήταν αρκετά καλός.	Kahn σελ.83
0-400 μ.Χ	Το Kama Sutra του Vatsayana περιγράφει την κρυπτογραφία ως την 44η και 45η από τα 64 είδη τεχνών που πρέπει να ξέρουν και να εφαρμόζουν και οι άνδρες και οι γυναίκες. Η ημερομηνία αυτής της εργασίας είναι ασαφής αλλά τοποθετείται μεταξύ των πρώτων τέταρτων αιώνων. Ο Vatsayana λέει ότι το Kama Sutra είναι μια σύνταξη των πολύ προηγούμενων εργασιών, πράγμα που καθιστά την χρονολόγηση των αναφορών του συστήματος κρυπτογραφίας ακόμα πιο δύσκολο. Σε αυτόν τον κατάλογο τεχνών, η 44η και 45η αναφέρουν ότι η τέχνη της κατανόησης του γραψίματος με κρυπτογράφηση, η τέχνη της ομιλίας με την αλλαγή των μορφών λέξεων και το γράψιμο των λέξεων με έναν ιδιαίτερο τρόπο, είναι διάφορων ειδών, δηλαδή μερικοί μιλούν με την αλλαγή της αρχής και του τέλους των λέξεων, άλλοι με την προσθήκη περιττών γραμμάτων μεταξύ κάθε συλλαβής μιας λέξης κτλ.	Burton
200 μ.Χ	Ο αποκαλούμενος πάπυρος του Leiden υιοθετεί ένα σύστημα κρυπτογράφησης για να κρύψει τις κρίσιμες δοσολογίες των σημαντικών μαγικών συνταγών.	Kahn σελ.91
725-790 μ.Χ	Ο Abu `Abd al-Rahman al-Khalil ibn Ahmad ibn `Amr ibn Tammam al Farahidi al-Zadi al Yahmadi έγραψε ένα (τώρα χαμένο) βιβλίο για την κρυπτογραφία, το οποίο εμπνεύστηκε από τη λύση ενός	Kahn σελ.97

	κρυπτογραφήματος που ήταν γραμμένο στα ελληνικά για το βυζαντινό αυτοκράτορα. Η λύση του βασίστηκε σε ένα γνωστό(σωστά αποκρυπτογραφημένο) αρχικό κείμενο στην έναρξη των μηνυμάτων (μια τυποποιημένη κρυπτοαναλυτική μέθοδος), που χρησιμοποιήθηκε ακόμη και στον δεύτερο παγκόσμιο πόλεμο ενάντια στα μηνύματα Enigma.	
855 μ.Χ	Ο Abu Bakr Ahmad ben `Ali ben Wahshiyya an-Nabati δημοσίευσε διάφορα κρυπτογραφικά αλφάβητα που χρησιμοποιήθηκαν παραδοσιακά στον τομέα της μαγείας.	Kahn σελ.93
1226 μ.Χ	Από το 1226, ένα εξασθενημένο πολιτικό σύστημα κρυπτογραφίας εμφανίστηκε στα αρχεία της Βενετίας, όπου τα σημεία ή οι σταυροί αντικατέστησαν τα φωνήεντα σε μερικές διεσπαρμένες λέξεις.	Kahn σελ.106
περίπου 1250 μ.Χ	Ο Roger Bacon όχι μόνο περιέγραψε διάφορα κρυπτογραφήματα αλλά έγραψε: «ένα άτομο είναι τρελό όταν γράφει ένα μυστικό με οποιοδήποτε άλλο τρόπο από αυτόν που θα το κρύψει από το κοινό».	Kahn σελ. 90
1379 μ.Χ	Ο Gabrieli Di Lavinde κατά παράκληση του Clement VII, σύνταξε ένα συνδυασμό αντικατάστασης αλφαβήτου και έναν μικρό κώδικα. Αυτή η κατηγορία κώδικα/κρυπτογράφησης επρόκειτο να παραμείνει σε γενική χρήση μεταξύ των διπλωματών και μερικών πολιτών για τα επόμενα 450 έτη, παρά το γεγονός ότι υπήρξαν ισχυρότεροι τρόποι κρυπτογράφησης που εφευρίσκονται στο μεταξύ, ενδεχομένως λόγω της σχετικής ευκολίας του.	Kahn σελ. 107
Περίπου 1300 μ.Χ	Ο `Abd al-Rahman Ibn Khaldun έγραψε το "The Muqaddimah", μια ουσιαστική έρευνα για την ιστορία που αναφέρει τη χρήση των ονομάτων, των αρωμάτων, των φρούτων, των πουλιών ή των λουλουδιών σε διατάξεις διαφορετικές από τις συνηθισμένες μορφές των γραμμάτων ως μέθοδος κρυπτογράφησης μεταξύ των γραφείων φόρου και στρατού. Περιλαμβάνει επίσης μια αναφορά στην κρυπτολογική ανάλυση.	Kahn σελ. 94
1392 μ.Χ	Το "The Equatorie of the Planetis", ενδεχομένως γραμμένο από τον Geoffrey Chaucer, περιέχει τις μεταβάσεις ενός τύπου κρυπτογράφησης. Το κρυπτογραφημένο κείμενο είναι μια απλή αντικατάσταση με ένα κρυπτοαλφάβητο το οποίο αποτελείται από γράμματα, ψηφία και σύμβολα.	Kahn σελ. 182-7
1412 μ.Χ	Ο Shihab al-Din abu `l-`Abbas Ahmad ben `Ali ben Ahmad `Abd Allah al-Qalqashandi έγραψε το «Subh al-a `sha», μια αραβική εγκυκλοπαίδεια 14-τόμων που περιέβαλε ένα τμήμα της κρυπτολογίας. Αυτές οι πληροφορίες αποδόθηκαν στον Taj ad-Din `Ali ibn ad-Duraihim ben Muhammad ath-Tha`alibi al-Mausiliwho που έζησε από 1312 έως 1361 αλλά οι γραφές του στην κρυπτολογία έχουν χαθεί. Ο κατάλογος κρυπτογράφησης σε αυτήν την εργασία περιέλαβε και την αντικατάσταση και τη μετάθεση και, για πρώτη φορά, μέθοδο κρυπτογράφησης με τις πολλαπλάσιες αντικαταστάσεις για κάθε γράμμα κειμένου. Επίσης αποδίδεται στον Ibnal -Duraihim ένα παράδειγμα της κρυπτολογικής ανάλυσης, συμπεριλαμβανομένης της χρήσης των πινάκων των συχνοτήτων γραμμάτων και τα σύνολα γραμμάτων που δεν μπορούν να εμφανιστούν μαζί σε μια λέξη.	Kahn σελ.95-6
1466-7 μ.Χ	Ο Leon Battista Alberti (ένας φίλος του Leonardo Dato, ένας γραμματέας που μπορεί να καθοδήγησε τον Alberti στην κορυφή της τέχνης της κρυπτολογίας) εφεύρε και δημοσίευσε το πρώτο πολυαλφαβητικό κρυπτογράφημα, σχεδιάζοντας έναν κρυπτογραφημένο δίσκο (γνωστό σε μας ως Captain Midnight Decoder Badge) για να απλοποιήσει τη διαδικασία. Αυτή η κατηγορία κρυπτογράφησης δεν κατάφερε να παραβιαστεί έως το 1800. Ο Alberti έγραψε επίσης για την κατάσταση προόδου των κρυπτογραφημάτων, εκτός από την δικιά του εφεύρεσή. Ο Alberti χρησιμοποίησε επίσης το δίσκο του για τον κρυπτογραφημένο κώδικα. Αυτά τα συστήματα ήταν	Kahn σελ. 127

	πολύ ισχυρότερα από την ονοματολογία που χρησιμοποιούσαν οι διπλωμάτες της εποχής καθώς και των επόμενων αιώνων.	
1473-1490 μ.Χ	Ένα χειρόγραφο από τον Arnaldus de Bruxella χρησιμοποιεί πέντε γραμμές κρυπτογραφήματος για να κρύψει το κρίσιμο μέρος της λειτουργίας της παραγωγής της πέτρας ενός φιλοσόφου.	Kahn σελ. 91
1518 μ.Χ	Ο Johannes Trithemius έγραψε το πρώτο τυπωμένο βιβλίο στην κρυπτολογία. Εφεύρε ένα στεγανογραφικό κρυπτογράφημα στο οποίο κάθε γράμμα αντιπροσωπεύθηκε ως λέξη που λήφθηκε από μια διαδοχή των στηλών. Η προκύπτουσα σειρά λέξεων θα ήταν μια νόμιμη προσευχή. Περιέγραψε επίσης πολυαλφαβητικά κρυπτογραφήματα στην τώρα-τυποποιημένη μορφή ορθογώνιων πινάκων αντικατάστασης. Εισήγαγε την έννοια των μεταβαλλόμενων αλφάβητων με κάθε γράμμα.	Kahn σελ.130-6
1553 μ.Χ	Ο Giovan Batista Belaso εισήγαγε την χρησιμοποίηση μιας φράσης ως κλειδί για επαναλαμβανόμενα πολυαλφαβητικά κρυπτογραφήματα. (Αυτή είναι η τυποποιημένη λειτουργία πολυαλφαβητικής κρυπτογράφησης κακώς αποκαλεσμένη «Vigenère» από τους περισσότερους συγγραφείς.).	Kahn σελ. 137
1563 μ.Χ	Ο Giovanni Battista Porta έγραψε ένα κείμενο για την κρυπτογράφηση, εισάγοντας τον δι-γραφικό τρόπο κρυπτογράφησης. Ταξινόμησε τις κρυπτογραφήσεις ως μεταθέσεις, αντικαταστάσεις και αντικαταστάσεις συμβόλων (χρησιμοποίηση ενός παράξενου αλφάβητου). Πρότεινε τη χρήση των συνωνύμων και των λανθασμένων ορθογραφιών για να δυσκολέψει την κρυπτανάλυση. Εισήγαγε προφανώς την έννοια ενός μικτού αλφάβητου σε ένα πολυαλφαβητικό πλαίσιο.	Kahn σελ. 138
1564 μ.Χ	Ο Bellaso δημοσίευσε ένα κρυπτογράφο αυτόματου κλειδιού βελτιώνοντας την εργασία του Cardano ο οποίος εμφανίζεται να εφευρίσκει αυτή την ιδέα.	Kahn σελ. 144
1623 μ.Χ	Ο Sir Francis Bacon περιέγραψε ένα κρυπτογράφο που φέρει τώρα την επωνυμία του. Ένα διλεκτικό κρυπτογράφο, γνωστό σήμερα ως δυαδική κωδικοποίηση 5-bit.	Bacon
1685 μ.Χ	Ο Blaise de Vigenère έγραψε ένα βιβλίο κρυπτογραφίας το οποίο συμπεριλάμβανε τα πρώτα αυθεντικά συστήματα αυτόματου κλειδιού και κρυπτογραφημάτων (στα οποία τα προηγούμενα απλά κείμενα ή γράμματα των κρυπτογραφημάτων χρησιμοποιούνται για το κλειδί της τρέχουσας επιστολής).	Kahn σελ. 146
Δεκαετία 1790 μ.Χ	Ο Thomas Jefferson, ενδεχομένως βοηθούμενος από το Δρ.Robert Patterson (έναν μαθηματικό στο U. Penn), εφεύρε ένα κυκλικό κρυπτογράφο. Αυτό εφευρέθηκε εκ νέου με διάφορες μορφές αργότερα και χρησιμοποιήθηκε στον δεύτερο παγκόσμιο πόλεμο από το αμερικανικό ναυτικό, με τον κωδικό M-138-A.	Kahn σελ. 192, Cryptologia v.5 No.4 σελ. 193-208
1817 μ.Χ	Ο Συνταγματάρχης Decius Wadsworth παρήγαγε έναν μηχανικό κρυπτογραφικό δίσκο με έναν διαφορετικό αριθμό γραμμάτων στα αλφάβητα απλού και κρυπτογραφημένου κειμένου με συνέπεια έναν προοδευτικό κρυπτογράφο στο οποίο τα αλφάβητα χρησιμοποιούνται ακανόνιστα, ανάλογα με το χρησιμοποιούμενο κείμενο.	Kahn σελ. 195
1854 μ.Χ	Ο Charles Wheatstone εφεύρε αυτό που έχει γίνει γνωστό ως κρυπτογράφος Playfair και οποίο δημοσιεύθηκε από το φίλο του Lyon Playfair. Αυτός ο κρυπτογράφος χρησιμοποιεί ένα κλειδωμένο πίνακα γραμμάτων ο οποίος είναι πολύ εύκολο να χρησιμοποιηθεί. Εφεύρε επίσης τη συσκευή Wadsworth και έγινε γνωστός για αυτή του την εφεύρεση.	Kahn σελ. 198
1857 μ.Χ	Ο κρυπτογράφος του Ναυάρχου Sir Francis Beaufort (μια παραλλαγή αυτού που κάλεσε «Vigenère») δημοσιεύθηκε από τον αδελφό του, μετά από το θάνατο του ναυάρχου στη μορφή μίας κάρτας 4x5 ιντσών.	Kahn σελ. 202

1859 μ.Χ	Ο Pliny Earle Chase δημοσίευσε την πρώτη περιγραφή κρυπτογράφου (τομογραφικού).	203
1854 μ.Χ	Ο Charles Babbage φαίνεται να εφευρίσκει εκ νέου τον κυκλικό κρυπτογράφο.	Cryptologia v.5 No.4 pp.193-208
1861-1980 μ.Χ	Μια μελέτη επάνω στις ευρεσιτεχνίες στις Ηνωμένες Πολιτείες από την έκδοση του πρώτου κρυπτογραφικού διπλώματος ευρεσιτεχνίας το 1861 μέχρι το 1980 προσδιόρισε 1.769 διπλώματα ευρεσιτεχνίας που συσχετίζονται πρώτιστα με τα συστήματα κρυπτογραφίας.	Deavours
1861 μ.Χ	Ο Friedrich W. Kasiski δημοσίευσε ένα βιβλίο που δίνει την πρώτη γενική λύση ενός πολυαλφαβητικού κρυπτογράφου με την επανάληψη μιας επαναλαμβανόμενης φράσης, χαρακτηρίζοντας κατά συνέπεια το τέλος της εποχής της ασφάλειας των πολυαλφαβητικών κρυπτογράφων.	Kahn σελ. 207
1861-5 μ.Χ	Κατά τη διάρκεια του εμφύλιου πολέμου, ενδεχομένως μεταξύ άλλων κρυπτογράφων, η Ένωση χρησιμοποίησε την αντικατάσταση επιλεγμένων λέξεων από μία γραμμική-μετάθεση λέξεων ενώ η συνομοσπονδία χρησιμοποίησε τον αλγόριθμο Vigenère (η λύση του οποίου ήταν δημοσιευμένο ακριβώς από τον Kasiski).	Kahn σελ. 215
1891 μ.Χ	Ο Etienne Bazeries εξέδωσε την έκδοσή του για την κυκλική κρυπτογράφηση και δημοσίευσε το σχέδιο του το 1901 αφότου το είχε απορρίψει ο γαλλικός στρατός	Cryptologia v.5 No.4 pp.193-208
1913 μ.Χ	Ο Καπετάνιος Parket Hittre εφεύρε την κυκλική κρυπτογράφηση, σε γραμμική μορφή, οδηγώντας στο M-138-A κατά την διάρκεια του δευτέρου παγκοσμίου πολέμου.	Cryptologia v.5 No.4 pp.193-208
1916 μ.Χ	Ο Joseph O. Mauborgne επανάφερε σε κυκλική μορφή τον κρυπτογράφο του Hitt, ενδυναμώνοντας την κατασκευή του αλφαβήτου πράγμα που οδήγησε στην κρυπτογραφική μηχανή M-94.	Cryptologia v.5 No.4 pp.193-208
1917 μ.Χ	Ο William Frederick Friedman, ο οποίος αργότερα τιμήθηκε ως ο πατέρας της αμερικανικής κρυπτανάλυσης (ήταν αυτός που έπλασε αυτό τον όρο), προσελήφθει ως πολίτης-κρυπταναλυτής (μαζί με τη σύζυγό του Elizabeth) στα εργαστήρια Riverbank και έκανε κρυπτανάλυση για την αμερικάνικη κυβέρνηση, η οποία δεν είχε καμία κρυπταναλυτική εμπειρία.	Kahn σελ. 371
1917 μ.Χ	Ο Gilbert S. Vernam, ο οποίος εργαζόμενος για την AT&T, εφήυρε μία πρακτική πολυαλφαβητική κρυπτογραφική μηχανή ικανή να χρησιμοποιεί ένα κλειδί που είναι εντελώς τυχαίο και δεν επαναλαμβάνεται ποτέ. Αυτό είναι το μόνο ευαπόδεικτα ασφαλές σύστημα κρυπτογράφησης που γνωρίζουμε. Αυτή η μηχανή προσφέρθηκε στην κυβέρνηση για χρήση στον δεύτερο παγκόσμιο πόλεμο αλλά απορρίφθηκε. Τέθηκε στην εμπορική αγορά το 1920.	Kahn σελ. 401
1918 μ.Χ	Το σύστημα ADFGVX τέθηκε σε υπηρεσία από τους Γερμανούς κοντά στο τέλος του δεύτερου παγκοσμίου πολέμου. Αυτός ήταν ένας αλγόριθμος που εκτελούσε αντικατάσταση (μέσω ενός κλειδωμένου πίνακα), κατακερματίζοντας και στην συνέχεια κάνοντας αντιμετάθεση των γραμμάτων. Παραβιάστηκε από τον Γάλλο κρυπταναλυτή, υπολοχαγό Georges Painvin.	Kahn σελ. 340-5
1919 μ.Χ	Ο Hugo Alexander Koch αρχειοθέτησε ένα δίπλωμα ευρεσιτεχνίας στις Κάτω Χώρες βασισμένο σε μια μηχανική κρυπτό-μηχανή. Έδωσε αργότερα, το 1927, τα δικαιώματα του για αυτό το δίπλωμα ευρεσιτεχνίας στον Arthur Scherbius ο οποίος εφεύρε και εμπορευόταν τη μηχανή Enigma από το 1923.	Kahn σελ. 420
1919 μ.Χ	Ο Arvid Gerhard Damm υπέβαλε αίτηση για ένα δίπλωμα ευρεσιτεχνίας στη Σουηδία για μια μηχανική κρυπτό-μηχανή. Αυτή η μηχανή αναπτύχθηκε σε μια οικογένεια κρυπτό-μηχανών υπό την καθοδήγηση του Boris Caesar Wilhelm Hagelin που ανέλαβε την επιχείρηση και ήταν ο μοναδικός από τους εμπορευόμενους	Kahn σελ. 422

	κρυπτογράφους αυτής της περιόδου. Μετά από τον πόλεμο, ένας σουηδικός νόμος που επέτρεπε στην κυβέρνηση να εκμεταλλεύεται τις εφευρέσεις αισθάνθηκε την ανάγκη να μετακινήσει την επιχείρηση του προς το Zug της Ελβετίας όπου ενσωματώθηκε ως Crypto AG. Η επιχείρηση είναι ακόμα σε λειτουργία, αν και αντιμετωπίζει κατηγορίες ότι έχει αποδυναμώσει επίτηδες ένα κρυπτογραφικό προς πώληση προϊόν στο Ιράν.	
1921 μ.Χ	Ο Edward Hugh Hebern ίδρυσε την «Hebern Electric Code», μια επιχείρηση που κατασκευάζει ηλεκτρομηχανικές κρυπτό-μηχανές βασισμένες σε δρομείς που γυρίζουν, ύφος οδομέτρων, με κάθε χαρακτήρα που κρυπτογραφείτε .	Kahn σελ. 415
1923 μ.Χ	Arthur Scherbius ενσωματώθηκε στην εταιρεία «Chiffrier maschinen Aktiengesellschaft'» ούτως ώστε να κατασκευάσει και να πουλήσει την μηχανή Enigma.	Kahn σελ. 421
1924 μ.Χ	Ο Alexander von Kryha παρήγαγε τη μηχανή " κωδικοποίησης που χρησιμοποιήθηκε, ακόμη και από τα γερμανικά διπλωματικά σώματα, στη δεκαετία του '50. Ένα κρυπτογράφημα δοκιμής 1135 χαρακτήρων λύθηκε από τις ΗΠΑ από τους κρυπταναλυτές Friedman, Kullback, Rowlett και Sinkov σε 2 ώρες και 41 λεπτά. Εντούτοις, η μηχανή συνεχίστηκε για να πωλείται.	Deavours σελ..151
1929 μ.Χ	Ο Lester S. Hill εξέδωσε το βιβλίο «Cryptography in an Algebraic Alphabet» στο οποίο ένα σύνολο απλού κειμένου κρυπτογραφείτε από μια σύνθετη λειτουργία.	Kahn σελ. 404
1933-45 μ.Χ	Η μηχανή Enigma δεν ήταν μια εμπορική επιτυχία αλλά αναλήφθηκε και βελτιώθηκε επάνω για να γίνει κρυπτογραφικό εργαλείο της ναζιστικής Γερμανίας. Παραβιάστηκε από τον Πολωνό μαθηματικό Marian Rejewski.	Kahn σελ. 422
1937 μ.Χ	Οι Ιαπωνική Purple η μηχανή εφευρέθηκε σε απάντηση στις αποκαλύψεις από τον Herbert O. Yardley και στην παραβίαση από μία ομάδα στην οποία ηγείτο ο William Frederick Friedman. Η μηχανή Purple χρησιμοποίησε τις τηλεφωνικές αναμεταδόσεις αντί των δρομέων και για αυτό το λόγο είχε μια συνολικά διαφορετική αντιμετάθεση σε κάθε βήμα παρά τις σχετικές αντιμεταθέσεις ενός δρομέα στις διαφορετικές θέσεις.	18ff
δεκαετία 1930 μ.Χ	Ο Kahn αποδίδει την αμερικανική μηχανή SIGABA (M-134-C) στον William F. Friedman ενώ ο Deavours το αποδίδει σε μια ιδέα του Frank Rowlett. Βελτιώθηκε στις εφευρέσεις δρομέων των Hebern και Scherbius με τη χρησιμοποίηση μεταβαλλόμενους ψευδό-τυχαίους πολλαπλάσιων δρομέων σε κάθε βήμα κρυπτογράφησης αντί της ύπαρξης ομοιόμορφου, οδομέτρου όπως έχουμε στον Enigma. Χρησιμοποίησε επίσης 15 δρομείς (10 για τον μετασχηματισμό των χαρακτήρων, 5 πιθανώς για τον έλεγχο των σταδίων διαδικασίας) σε αντίθεση με τον Enigma's που χρησιμοποίησε 3 ή 4.	Kahn σελ.510ff, Deavours σελ.10,89-91
δεκαετία 1930 μ.Χ	Οι Βρετανική μηχανή TYPEX ήταν μια παραφυάδα του αντίστοιχου Enigma αγορασμένη από τους Βρετανούς για μελέτη στη δεκαετία του '20. Ήταν μια μηχανή 5-δρομέων με τους δύο αρχικούς δρομείς που είναι στάτες, να εξυπηρετούν το σκοπό του πίνακα συνδέσεων της Γερμανικής μηχανής Enigma.	Deavours p.144
1970 μ.Χ	Το Dr. Horst Feistelled ένα ερευνητικό πρόγραμμα στο εργαστήριο της IBM Watson Research Lab στη δεκαετία του '60 ανέπτυξε τον κρυπτογραφικό αλγόριθμο Lucifer. Αυτό το αργότερα ενέπνευσε τον αμερικανικό αλγόριθμο DES και άλλους αλγόριθμους, που δημιουργούν μια οικογένεια επονομαζόμενη αλγόριθμοι «Feistel».	Feistel
1976 μ.Χ	Ένα σχέδιο από την IBM, που βασίστηκε στον αλγόριθμο Lucifer και με τις αλλαγές (και συμπεριλαμβανομένων των s-box των βελτιώσεων και της μείωσης του βασικού μεγέθους) επιλέχτηκε από την αμερικανική NSA, για να είναι το πρότυπο κρυπτογράφησης στοιχείων	FIPS PUB-46

	(DES). Έχει καταξιωθεί παγκόσμια, κατά ένα μεγάλο μέρος επειδή έχει παρουσιαστεί ισχυρός ενάντια σε επιθέσεις 20 ετών.	
1976 μ.Χ	Ο Whitfield Diffie και ο Martin Hellman εξέδωσαν τον βιβλίο «New Directions in Cryptography», παρουσιάζοντας την ιδέα της κρυπτογραφίας δημόσιου κλειδιού. Ανέπτυξαν επίσης την ιδέα της επικύρωσης από τις δυνάμεις μια μονόδρομης διαδικασίας, που τώρα χρησιμοποιείται στην λειτουργία των κλειδιών.	Diffie
Απρίλιος 1977 μ.Χ	Εμπνευσμένοι από την εργασία και τις παρατηρήσεις επάνω στην κρυπτογραφία του Diffie-Hellman, οι Ronald L. Rivest, Adi Shamir και Leonard M. Adleman άρχισαν να μελετούν πώς να φτιάξουν ένα πρακτικό σύστημα κρυπτογράφησης δημόσιου κλειδιού. Μια νύχτα του Απριλίου, ο Ron Rivest είχε ένα μεγάλο προβληματισμό για αυτή την μηχανή και έτσι ανακάλυψε τον αλγόριθμο RSA. Το κατέγραψε για τους Shamir και Adleman και τους το έστειλε το επόμενο πρωί. Ήταν ένας πρακτικός αλγόριθμος κρυπτογράφησης δημόσιου κλειδιού και όσο για την εμπιστευτικότητα και για τις ψηφιακές υπογραφές, αυτά ήταν βασισμένα στη δυσκολία των κατασκευής μεγάλων αριθμών. Το υπέβαλαν στον Martin Gardner στις 4^{ης} Απριλίου για δημοσίευση στο επιστημονικό περιοδικό Scientific American. Εμφανίστηκε τον Σεπτέμβριο, στο τεύχος του 1977. Το επιστημονικό αμερικανικό άρθρο συμπεριέλαβε μια προσφορά να αποσταλεί η πλήρης τεχνική έκθεση. Υπήρξαν χιλιάδες τέτοια αιτήματα, από όλο τον κόσμο. Κάποιος από την NSA αντιτέθηκε στη διανομή αυτής της έκθεσης στους αλλοδαπούς και για λίγο η RS&A ανέστειλε την αλληλογραφία της αλλά όταν απέτυχε η NSA να αποκριθεί στις έρευνες ζητώντας τη νομική βάση της κίνησης αυτής η RS&A ξανάρχισε τις αποστολές. Ο Adi Shamir αναφέρει ότι αυτή η απόφαση ήταν σταθμός στην ελεύθερη διάθεση τεχνικών μελετών(Αύγουστος 1995).	Shamir
1978 μ.Χ	Ο αλγόριθμος RSA δημοσιεύθηκε στο Επικοινωνιακό τμήμα του ACM.	RSA
1984-5 μ.Χ	Ο Αλγόριθμος rot13 παρουσιάστηκε στα νέα του λογισμικού της USENET ο οποίος επέτρεπε την κρυπτογράφηση των επιστολών με σκοπό την προστασία τους από αγνώστους.. Αυτό είναι το πρώτο παράδειγμα που γνωρίζουμε στο οποίο ένας αλγόριθμος κρυπτογράφησης του οποίου το κλειδί το γνωρίζει ο καθένας ήταν πραγματικά αποτελεσματικός.	ROT13
1991 μ.Χ	Ο Phil Zimmermann κυκλοφόρησε την πρώτη έκδοση του PGP (Pretty Good Privacy δηλαδή αρκετά καλή μυστικότητα) σε απάντηση στην απειλή από το FBI στην πρόσβαση στα κείμενα επικοινωνίας των πολιτών. Το PGP προσέφερε την υψηλή ασφάλεια στο μέσο πολίτη και θα μπορούσε υπό αυτήν τη μορφή να έχει θεωρηθεί ως ανταγωνιστής στα εμπορικά προϊόντα όπως το Mailsafe από την RSADSI. Εντούτοις, το PGP είναι ιδιαίτερα ξεχωριστό επειδή δόθηκε ως δωρεάν λογισμικό και έχει γίνει ένα παγκόσμιο πρότυπο ενώ δεν υπάρχουν ανταγωνιστές του προς το παρόν.	Garfinkel
1994 μ.Χ	Ο καθηγητής Ron Rivest, συντάκτης του προηγούμενων αλγόριθμων RC2 και RC4 που περιλήφθηκαν στην κρυπτογραφική βιβλιοθήκη της RSADSI «BSAFE», δημοσίευσε έναν προτεινόμενο αλγόριθμο, τον RC5, στο διαδίκτυο. Αυτός ο αλγόριθμος χρησιμοποιεί περιστροφή των δεδομένων ως μη γραμμική λειτουργία της και παραμετροποιείται έτσι ώστε ο χρήστης να μπορεί να αλλάζει το μέγεθος των δεδομένων, τον αριθμό των κύκλων και το μέγεθος κλειδιού.	Rivest

Πίνακας Περιεχομένων

1 Εισαγωγή 2
2 Η Ιστορία της Κρυπτογραφίας 4
2.1 Πρώτη Περίοδος Κρυπτογραφίας (1900 π.Χ. – 1900 μ.Χ.) 4
2.2 Δεύτερη Περίοδος Κρυπτογραφίας (1900 μ.Χ. – 1950 μ.Χ.) 9
2.3 Τρίτη Περίοδος Κρυπτογραφίας (1950 μ.Χ. - Σήμερα) 11
3 Θεμελιώδεις Έννοιες Κρυπτολογίας 13
3.1 Κρυπτογράφηση- Αποκρυπτογράφηση 13
3.2 Κρυπτογραφικές Αρχές 14
3.3 Λειτουργίες της Κρυπτογράφησης 14
3.4 Βασικές αρχές σχεδιασμού κρυπτογραφημάτων ομάδας (block ciphers) 15
3.4.1 Τα μέτρα του Shannon 16
3.4.2 Σύγχυση (confusion) και Διάχυση (diffusion) 16
3.4.3 Δίκτυα Feistel 17
3.4.3.1 Ασφάλεια δικτύων Feistel 19
3.5 Είδη Κρυπτογράφησης 19
3.6 Σύγκριση Συμμετρικής και Ασύμμετρης Κρυπτογραφίας 21
3.7 Καταστάσεις Λειτουργίας Συμμετρικών Κρυπταλγόριθμων 22
3.7.1 Κατάσταση λειτουργίας ηλεκτρονικού βιβλίου κωδικών (ECB) 23
3.7.2 Κατάσταση λειτουργίας αλυσιδωτής σύνδεσης κρυπτ/μάτων ομάδας (CBC). 23
3.7.3 Κατάσταση λειτουργίας κρυπτογραφίας ανάδρασης (CFB) 25
3.7.4 Ανάδραση εξόδου (OFB) 26
3.8 Κρυπτανάλυση 27
3.9 Μέθοδοι Επιθέσεων 28
4 Ο Αλγόριθμος DES 30
4.1 Περιγραφή του DES 32
4.1.1 Η συνάρτηση *f* 33
4.1.2 Ανάλυση S-box 34
4.1.3 Πρόγραμμα κλειδιού του DES 37
4.2 Ασφάλεια DES 38
4.2.1 Αδύναμα και Ημιαδύναμα Κλειδιά 39
5 Μελέτη και Υλοποίηση του Αλγορίθμου DES και Triple-DES 41
5.1 Υλοποίηση του Αλγορίθμου DES και Triple-DES 42
5.1.1 Εσωτερική λειτουργία κύκλου DES (One Round Function) 45
5.1.2 Επεξεργασία Κλειδιού DES ενός κύκλου (One round Key function) 49
5.1.3 Triple DES 51
5.2 Υλοποίηση και Προσομοίωση 52
6 Συμπεράσματα 59

ΒΙΒΛΙΟΓΡΑΦΙΑ 61

Παραρτήματα 63

Α. Αποσπάσματα Κώδικα VHDL 63

Β. Η Ιστορία της Κρυπτογραφίας 81

Ευχαριστίες

Η πραγματοποίηση της παρούσας εργασίας δεν θα ήταν δυνατή χωρίς την ουσιαστική βοήθεια ορισμένων προσώπων.

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα της πτυχιακής μου εργασίας Δρ. Μηχ. Νικόλαο Πετράκη για τη συνεχή παρακολούθηση, καθοδήγηση και συμβολή του, καθώς επίσης για την υπομονή που έδειξε, καθ' όλη την διάρκεια της εκπόνησης της παρούσας εργασίας.

Επίσης, ευχαριστώ πολύ τον φίλο και συνάδελφο Παναγιώτη Μαργαρώνη (MSc) για την πολύτιμη γνώση που μου προσέφερε απλόχερα, καθώς επίσης και για την εκτενή βοήθειά του στην υλοποίηση του πρακτικού μέρους της εργασίας αυτής.

Τέλος, ένα μεγάλο ευχαριστώ στην οικογένειά μου για την αμέριστη υποστήριξη, οικονομική και ηθική, που μου παρείχε όλα αυτά τα χρόνια, η οποία συμμερίστηκε όλα τα άγχη και τις ανησυχίες μου.

Αφιερώνεται στη αδερφή μου

Μαρία

Περίληψη

Στην παρούσα Πτυχιακή Εργασία μελετώνται τόσο οι θεμελιώδεις έννοιες όσο και η ορολογία της κρυπτολογίας, προσεγγίζοντας σταδιακά τις διάφορες τεχνικές κρυπτογραφίας, τις αρχές σχεδιασμού και τα είδη κρυπτογράφησης. Δίνεται ιδιαίτερη προσοχή στην κρυπτογραφία ιδιωτικού κλειδιού και στις τέσσερις βασικές παραλλαγές του τρόπου λειτουργίας της. Έχει γίνει αναφορά σε μια πληθώρα μεθόδων κρυπτογράφησης/αποκρυπτογράφησης ξεκινώντας από την αρχαιότητα και φτάνοντας μέχρι τις ημέρες μας. Φυσικά, δίδεται μεγαλύτερη έμφαση στις σύγχρονες μεθόδους που στηρίζονται στην ψηφιακή τεχνολογία και αποτελούν πρότυπα όπως οι αλγόριθμοι DES και Triple-DES, με τις διάφορες παραλλαγές τους. Η παραλλαγή ECB των παραπάνω αλγορίθμων υλοποιήθηκε σε υλικό (hardware) χρησιμοποιώντας την γλώσσα περιγραφής υλικού VHDL και το ολοκληρωμένο περιβάλλον λογισμικού ISE (Integrated Software Environment) της Xilinx. Τέλος, έγιναν οι κατάλληλες προσομοιώσεις οι οποίες αφενός απέδειξαν την ορθότητα της σχεδίασης και αφετέρου μας βοήθησαν στην εκτίμηση των επιδόσεων.

Abstract

The present work studies the basic notions as well as the terminology of cryptology, approaching progressively the various cryptographic techniques-categories and the design principles. Special attention is given at the private key cryptography and its modes of operation. A lot of encryption / decryption methods have been mentioned, starting from the ancient till our days. Emphasis has been given to modern methods which exploit digital technology such as DES and Triple-DES algorithms. The ECB mode of operation for the previous algorithms has been implemented by hardware using hardware description language VHDL and the Xilinx's Integrated Software Environment (ISE). Moreover, the proper simulations have been done for design verification and performance estimation.